WISEngineering: Achieving Scalability and Extensibility in Massive Online Learning

Xiang Fu[1(✉)], Tyler Befferman[1], Jennie Chiu[2], and M.D. Burghardt[1]

[1] Hofstra University, Hempstead, NY 11549, USA
{Xiang.Fu,M.D.Burghardt}@hofstra.edu, tbeffe1@pride.hofstra.edu
[2] University of Virginia, Charlottesville, VA 22904, USA
jlchiu@virginia.edu

Abstract. Massive Open Online Courses (MOOCs) have raised many unique challenges to online learning platforms. For example, the low teacher-student ratio in MOOCs often means lack of feedback to students and poor learning experiences. We present WISEngineering, a MOOCs platform that provides a rich set of features for overcoming these challenges. The system embraces social media for fostering student reflection. Its automated grading system adopts an open-architecture and uses stack generalization to blend multiple machine learning algorithms. A Zookeeper based computing cluster runs behind auto-grading and provides instant feedback. A behavior tracking system collects user behavior and can be later used for learning outcome analysis. We report the design and implementation details of WISEngineering, and present the design decisions that allow the system to achieve performance, scalability and extensibility in massive online learning.

Keywords: Online learning platform · Automated grading · Web application · Scalability · Extensibility

1 Introduction

For thousands of years, humankind has been trying to lower the cost of education, for making it more accessible. Massive open online courses (MOOCs) [18] are the latest attempt. MOOCs have great potential to revolutionize how people learn and how people teach. Using MOOCs in engineering classes, however, faces several challenges. Distance learning lacks face to face social interaction and larger class size can often contribute negatively to learning outcomes [1]. These problems are magnified in engineering education when learners need to be deeply engaged in hands-on environments and the feedback from peer learners and teachers is important.

We present WISEngineering [3,4], a distributed and web-based MOOCs platform that intends to address the above challenges. WISEngineering embraces social media computing for encouraging learner engagement. Its mobile portal, running as a Google Chrome application, provides easy access in a hands-on

J. Wang et al. (Eds.): WISE 2015, Part I, LNCS 9418, pp. 323–337, 2015.
DOI: 10.1007/978-3-319-26190-4_22

lab environment using 7 inch Android tablets. In particular, WISEngineering offers features that are available in a typical social media website, for learners to scaffold engineering design.

Automated and instant feedback is the key to providing quality learning experiences in WISEngineering. The system adopts an open architecture that blends a variety of automated grading algorithms and modules. It can be trained by providing manual grading samples, and be further calibrated at run time. A learning outcome system is built upon the automated grading system. All questions are tagged with learning outcome goals, and a weighted sum formula can be defined to take into account various aspects of a learning process. A reporting system is available for performing comparative study of learning outcomes of any selected learner(s).

WISEngineering adopts component based software engineering [13], and it builds the features set by integrating components from a number of open source traditional and MOOCs web platforms. For example, its course and user management system is centered around the WISE system from UC Berkeley [2,22]. Its automated grading module uses the EASE module from edX [5]. Its mobile portal uses the responsible style template from Twitter [17]. Its instant user feedback module is built upon Apache HDFS and Zookeeper [9]. The video processing uses Google cloud and it is optimized by a local load balancer.

To construct a heterogeneous software system like WISEngineering is challenging. An open and extensible architecture is required for accommodating a wide variety of web and mobile application technologies and languages in one general framework. Caution has to be exercised when wrapping up the components, while at the same time, it has to provide authentication/security, atomic transaction, synchronization, and aggregation of data.

Efficiency and scalability are the key requirements of WISEngineering. For instance, to support hands-on engineering experiments, the system has to process large quantity of video data. For another example, to provide instant feedback to users, multiple auto-graders have to run alive, with each consuming large amount of RAM resources. The system has to address these issues by leveraging parallel and distributed computing techniques.

This paper reports the rich feature set as well as the design trade-offs and implementation details of WISEngineering. Section 2 introduces the system features of WISEngineering. Section 3 presents general architectural decisions of the system. Section 4 discusses how extensibility is accomplished in design. Section 5 addresses system performance and scalability. Section 6 discusses related work, and Section 7 concludes.

2 System Features

To better understand the architectural and design decisions in developing WISEngineering, we present a number of its important system features. A working copy of the system is available at [3]. Section 2.1 first presents the Web-based Inquiry Science Environment (WISE) system from the University of California,

Berkeley [2], upon which WISEngineering is built. Then, the rest of the section discusses the new features.

2.1 Existing Features Provided by WISE

WISE [2], like other educational platforms such as Blackboard [11] and Moodle [14], provides core functions such as user registration, curriculum development and manual grading. The WISE environment has been developed based on extensive research in classroom instruction. It uses rich online interactive plug-ins (such as PhET [25]) which are used by students to experiment with scientific concepts in hands-on exploration.

WISE provides a hosting platform for many science and engineering educational initiatives, such as the WISE Guys & Gals project (WGG) [3]. WGG introduces middle school age youth to innovative and engaging blended STEM based engineering design activities. Each activity is framed to expose youth to an engineering discipline (e.g. Mechanical, Electrical, and Civil).

WISE provides authoring tools for developing curriculum materials. In WISE, each activity is structured as a tree view of learning steps, where a step can either be an HTML page that presents a scientific concept, or an assessment step that collects a student's feedback and reflection. WISE supports typical assessment approaches such as short answer, multiple-choice, match and sequence, and discussion. Assessment steps such as multiple-choice can be automatically graded by WISE, however, at this moment, short answer questions still have to be manually graded by teachers.

2.2 WISEngineering Mobile Portal

Powered by WISE technologies [2], the WISEngineering system has incorporated a number of important system features for meeting the challenge of massive online learning. The first addition is a mobile portal for students.

The mobile portal is a Google Chrome Web application that ports the major functions of WISE to tablets. Students no longer have to walk between their computers and workbench for data entry and analysis. For example, students can use data plotting and tabulating tools on tablets to analyze and visualize data on the spot. In particular, WISEngineering embraces social media computing to engage students in reflection and collaboration. Tools such as the design journal and design wall are used to share ideas and designs. The mobile tools allow students to take live pictures and videos, which puts demands on the system to process large video uploads efficiently.

2.3 Automated Grading

WISEngineering uses an open architecture to embrace automated grading technologies such as the edX EASE engine [19]. This section presents the user interface of the system. Later, Sect. 4.2 presents the details of design and implementation.

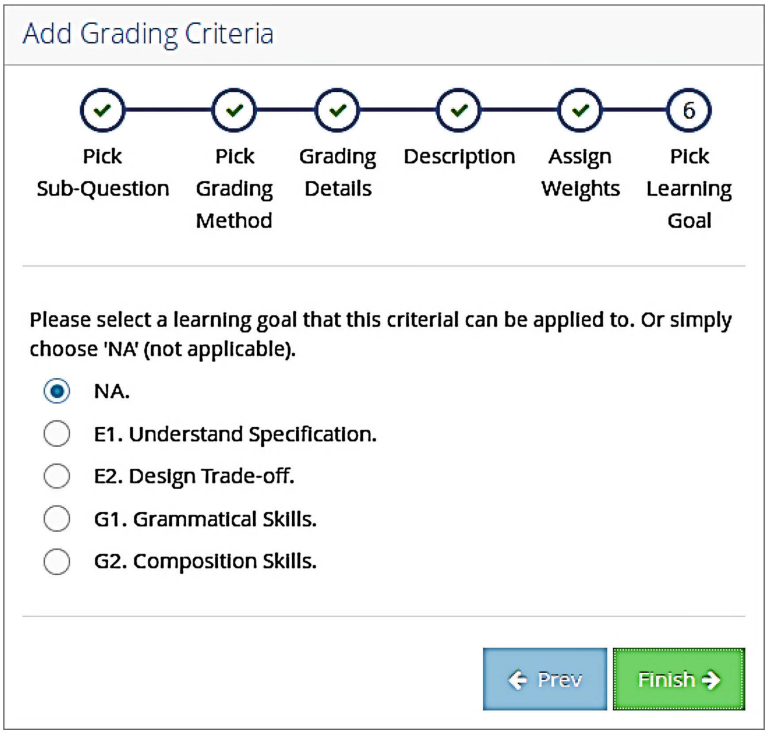

Fig. 1. Grading Criteria

Learning Goals and Grading Criteria. WISEngineering supports outcome-based education [24]. Before the curriculum is developed, a set of learning outcome goals can be entered into the system. Later, they are associated with assessment questions in activities. As an example, Fig. 1 presents the user interface for defining a grading criteria. A curriculum developer can choose to use the question as an indicator for selected learning outcome goals.

Training. Each automated grading criteria has to be trained and calibrated. At any moment, the system keeps two sets of data: one training set and one calibration set. Each training/calibration sample consists of three parts: a student response, the grade assigned by a human grader, and a grade assigned by the AI grader (in calibration data only). The training set is used to generate the grading model, while the calibration set is used to measure the quality of the model.

It is recommended that for each grading level, at least ten samples are entered for the grading engine to function correctly. WISEngineering provides tools for creating training samples, and student response samples can be retrieved directly from the database of the WISE system. Figure 2 displays the training statistics of a grading criteria. It shows the progress of data entry (of samples), and the current quality/precision of the model. At the bottom of the page, it also displays the details of those mismatched records in calibration set.

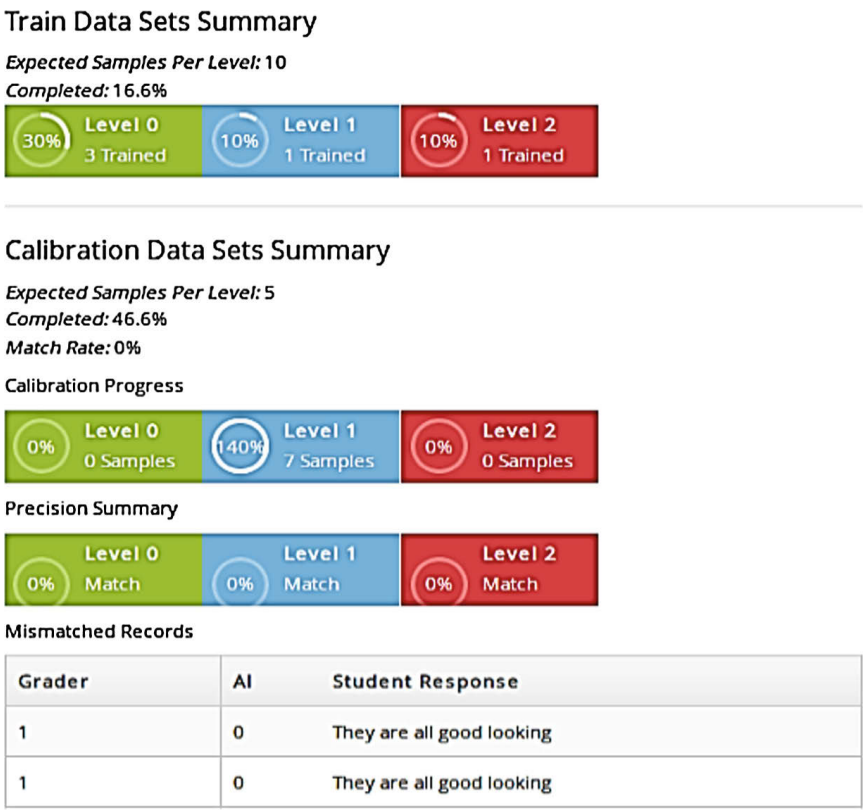

Fig. 2. Training Grading Criteria

Grading. After a model is generated for all criteria, a project can be graded. An automated grading script runs on each server overnight. The script loads each grading model, performs the grading, calculates the weighted sum of the grade, and pushes the grade into the database of the original WISE system. Students can view their grade and the automatic hint/response generated by the AI grader the second day. The default automated grading module is *not* instant because model generation and loading is expensive in both time and space. We present the details of the instant feedback system in Sect. 5.2.

2.4 Learning Outcome Analysis and Reporting System

The system periodically generates learning outcome reports for all groups of learners. Figure 3 shows a part of one sample report. Student activity data are generated by aggregating data from the WISE database. Learning outcomes are computed using the weighted sum formula associated with each question, which serves as an indicator of learning goals. Histograms of learning outcomes and the achievement of each individual learning outcome by each club can be generated and included in the report. The reporting function allows the curriculum developer to perform comparative study of learning outcomes on selected learners over the time.

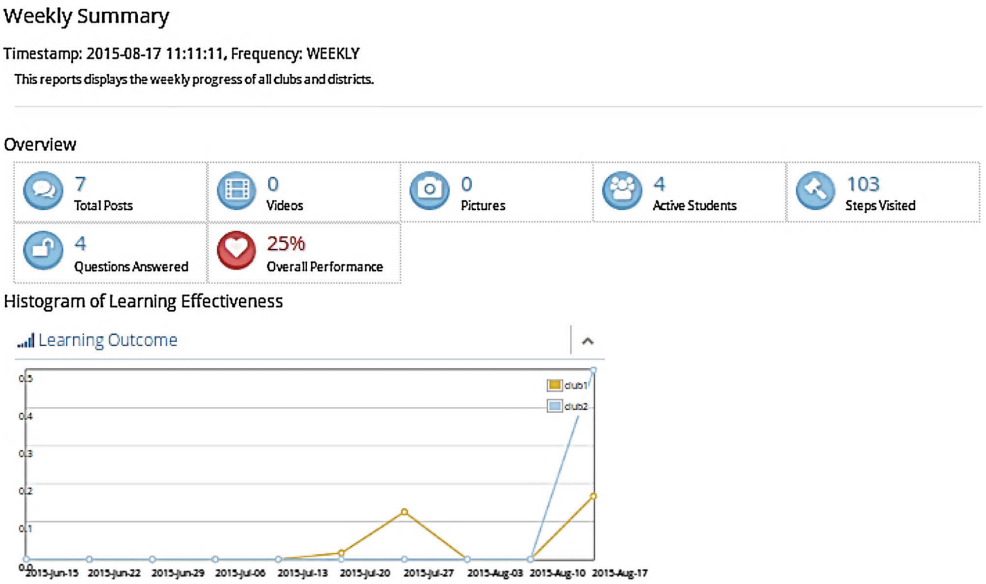

Fig. 3. Learning Outcome Report

3 System Architecture

In addition to those introduced in Sect. 2, WISEngineering has provided many other features, such as a user avatar system, a user behavior tracking system, a video processing cluster, and an instant feedback cluster. To include these heterogeneous software components, while at the same time achieving performance and scalability requires careful design decisions in its software architecture.

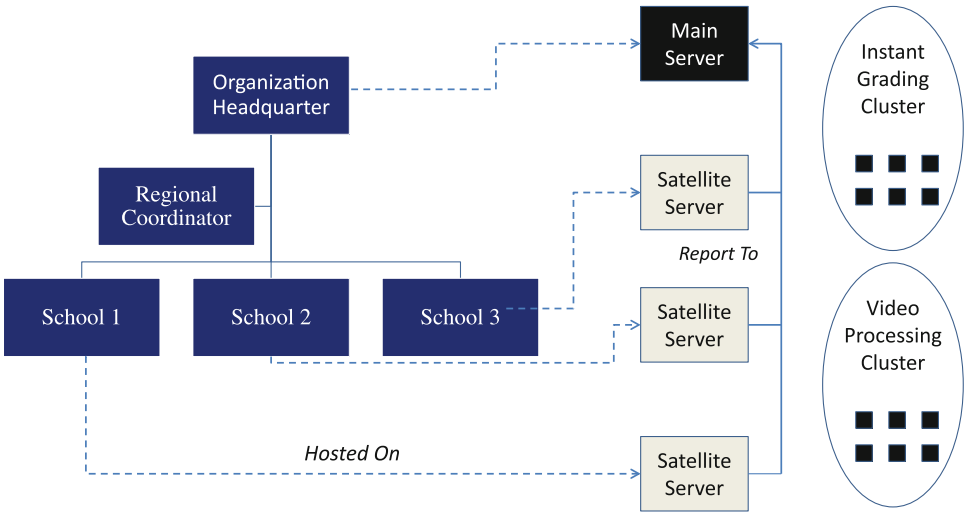

Fig. 4. WISEngineering Network Topology

3.1 Network Topology

The deployment of WISEngineering requires multiple loosely coupled servers and several clusters. Figure 4 displays the WISEngineering network topology.

To all users of the system, the main access point of WISEngineering is the *main server*. It has the identical software stack as all other satellite servers. The only difference is that the main server has a load balancer distributing the incoming traffic, and it is equipped with a report aggregation system.

WISEngineering has the ability to support large-scale educational activities nationwide (e.g., [3]). Such activities are usually organized by national organizations with a hierarchical structure. The main server records the information and utilizes this information to generate aggregated report. For example, to retrieve the number of active students in one particular region, the main server will first query the organizational structure and submit data retrieval requests to the related satellite servers for aggregation.

All servers are supported by two back-end clusters. The video processing cluster accepts video uploads from mobile devices, pre-processes/compresses video files, and stores the media file in the Google Cloud. The design of the instant grading cluster will be described in details in Sect. 5.2.

3.2 Software Stack

Figure 5 displays the software component stack at each satellite server. It addresses the challenge of integrating a wide variety of components and web technologies in one general framework.

The left side of Fig. 5 shows the structure of the original WISE system [2], upon which WISEngineering is built. WISE adopts a typical three tier structure. At the bottom, sits the data tier. It consists of two MySQL databases that store user and session information. Student responses and teacher feedback are also

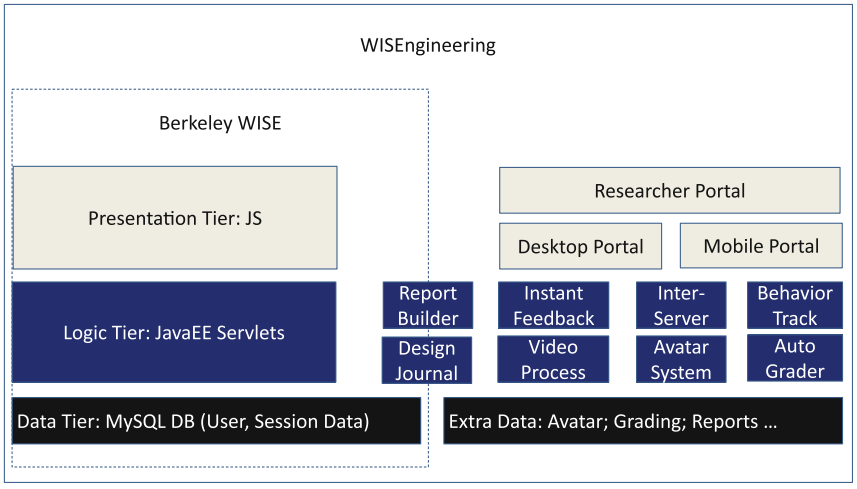

Fig. 5. WISEngineering Software Stack

stored there. The logic layer is implemented as a collection of JavaEE servlets. Then the presentation layer (customized for desktop browsers) renders the data generated by the logic layer using a combination of JQuery and CSS.

The main design goal of WISEngineering is to keep all the original functions of WISE, and extend it with other major functions such as automated grading and a mobile portal. This is achieved using a similar three-tier structure, and making minimal changes to the original WISE source code.

As shown on the right of Fig. 5, the extra WISEngineering components are also structured in three layers. Several more MySQL databases are added to the system to store information for automated grading system, design journal/wall, and a user avatar system. Then an extra logic layer is added, using a variety of platforms such as PHP and Python, depending on the components being integrated. At the presentation layer, three portals are provided: a desktop portal which enriches the original WISE system, a mobile portal which uses the Bootstrap [17] responsive templates for mobile devices, and a special desktop portal for educational researchers to train automated grading system and generate system reports.

Notice that some WISEngineering components, such as the report generator, do have to interact with the data tier of the original WISE environment. The system is designed in a way so that tight coupling is minimized.

4 Achieving Extensibility

This section presents a number of design decisions we made to achieve extensibility in WISEngineering. When new components are "plugged" into WISEngineering, we use a service-oriented approach to wrap-up the standard interface of a component so that it can be invoked by others.

4.1 Inter-Server Communication

In many cases, satellite servers have to exchange information with the main server. One typical example is to generate aggregated reports. For instance, at the main server, when a user requests the total number of video posts in a specific date range, the data has to be retrieved from all satellite servers and then aggregated by the main server.

Report generation is time consuming, mostly caused by huge join statements in SQL queries. To simply retrieve data via an HTTP request can often time out. Instead, the system has to provide a mechanism similar to restful web services [20]. The detailed communication protocol is described below.

We assume $E(k, m)$ is an encryption operation which uses key k to encrypt message m and it is always true that $E^{-1}(k, E(k, m)) = m$. Let the requester be R (e.g., the main server) and the server be S (e.g., a satellite server). If R has access to an operation P on S, there is a common secret $s_{R,P,S}$ shared between R and S. The interaction follows the steps below:

1. $R \rightarrow S$: `request` for operation P.

2. $S \rightarrow R$: (`id`, `n`). Here `id` is a unique service request identifier and `n` is a nonce (random number). In the database of S, a new entry is established for service request `id` where the field of `operation_result` is left blank. A timeout process is started simultaneously to kill the request operation if it is timed out. At time out, the database entry is removed as well, to avoid denial of service attack.
3. $R \rightarrow S$: $(E(s_{R,P,S}, (\texttt{id}, \texttt{n}))$ to prove that R has the access. Here, the use of nonce `n` is to avoid replay attack.
4. S verifies the access right of R using E^{-1} and starts the operation P. When P completes, it writes the data into `operation_result`.
5. R will periodically check the status of request `id`, until the data is available or time out.

On each server, an administrator application is provided for managing keys. We use this lighter weight authentication protocol, instead of Kerberos [12], to avoid failure of single point.

4.2 Open Architecture of Automated Grading

The extensibility of automated grading module is a similar but separate problem. In this case, we would like to make the system extensible, in the sense that, new machine learning algorithms can be added to the system in the future, to further improve the precision of grading.

We use an open architecture and stack generalization algorithm to repeatedly train and blend the results of a collection of automated grading algorithms. Details are given below.

Automated grading is essentially a machine learning problem. Take edX EASE [19] as an example. To train EASE, a user has to provide a training set of samples, where each is a pair of string (student answer) and a number (trainer assigned score). EASE, based on the number of samples (whether greater than 5), takes one of the regression or classification approaches. Using sklearn [21], a machine learning package in Python, EASE builds a classifier for each training set. Then, the classifier (also called the model), can be used for grading, i.e., it generates a numeric score for any string input.

The vector of features extracted for a model and the training algorithm used usually determine the quality of a classifier. For example, the feature set of EASE includes the bag of words (n-grams), length feature (counts of words, punctuation etc.), the number of spelling errors, and the number of grammar errors. This is suitable for essay grading, but may not be ideal for grading short answer questions in a specific technical context in WISEngineering. For another example, the sklearn package provides many different learning algorithms such as support vector machine, nearest neighbors, and Gaussian Processes. A great number of factors can determine the quality of auto-grading. Our framework tries to use one more level of training to find an optimized combination of auto-grading algorithms. We formalize our algorithm below.

Let Σ be the English alphabet, 2^{Σ} is the domain of input. Let $\mathcal{N}$ be the target range of scores. A classifier $\mathcal{C}$ is a function from 2^{Σ} to $\mathcal{N}$. Let $2^{\mathcal{C}}$ be the domain of all classifiers. A training algorithm $\mathcal{T}$ is modeled as $\mathcal{T} : 2^{2^{\Sigma} \times \mathcal{N}} \rightarrow 2^{\mathcal{C}}$, which given a training set T, generates a classifier.

Our framework is a 2-level application of the stack generalization by D. Wolpert [27]. The input of the algorithm is a collection of auto-grading (training) algorithms $\mathcal{T}_1, ..., \mathcal{T}_n$. They will be the "ensemble" training algorithms at level 0. Let $\theta = \{e_1, ..., e_k\}$ be the training samples, where each sample is a tuple $e_i = (s_i, n_i)$ where s_i is a string and n_i is the numeric score. Given a training sample, let $\texttt{input}(\texttt{e}_\texttt{i})$ be its s_i and let $\texttt{score}(\texttt{e}_\texttt{i})$ be its n_i. At level 1, we use k-NN as a selection algorithm (written as $\mathcal{K}$) that selects the result generated by level 0 algorithms. The training process works as follows:

1. Partition: build a set of partitions over θ. For each partition p_i, it is constructed by picking one training sample from θ. More formally each p_i is a tuple $(p_{i,1}, p_{i,2})$ where $p_{i,2}$ is a singleton element $\{e_i\}$, and $p_{i,1} = \theta - p_{i,2}$.
2. Train level 1 selector $\mathcal{K}$: the classifier generated by $\mathcal{K}$ will be a mapping from $\mathcal{N}^n \rightarrow [1, n]$ (where n is the number of level 0 training algorithms). Intuitively, given a vector of numeric scores produced by all level 0 trainers, the level 1 selector chooses the result produced by one of the level 0 trainers.
   Now we discuss how $\mathcal{K}$ is trained. For each partition p_i, use its $p_{i,1}$ as the training set for each level 0 $\mathcal{T}_j$, we obtain a classifier. Then using the level 1 classifier, we compute the output of $\mathcal{T}_j$ on $p_{i,2}$. Given n level 0 trainers, we have a vector of n output scores, and then we use $\texttt{score}(\texttt{p}_\texttt{i,2})$ as the ground fact for training. Officially, one training sample of $\mathcal{K}$, derived from partition p_i is defined as below: Let $\mathcal{C}_j$ be the classifier generated by $\mathcal{T}_j$ on $p_{i,1}$, i.e., $\mathcal{C}_j = \mathcal{T}_j(p_{i,1})$. We have a training vector:

$$((\mathcal{C}_1(\texttt{input}(\texttt{p}_\texttt{i,2})), ..., \mathcal{C}_\texttt{n}(\texttt{input}(\texttt{p}_\texttt{i,2}))), \texttt{score}(\texttt{p}_\texttt{i,2}))$$

Now given any question text q, let $\mathcal{C}_\mathcal{K}$ be the classifier generated by the above algorithm. Let $(v_1, ..., v_n)$ be the vector of results generated by level 0 classifiers. Feed the vector to $\mathcal{C}_\mathcal{K}$ we get an index number between 1 and n, let it be i, then the corresponding output is v_i.

In summary, the two level stack generalization algorithm [27] allows the system to extend with an arbitrary number of auto-grading algorithms.

5 Design for Scalability and Performance

System performance and scalability are the key to user satisfaction. This section presents the related implementation details.

5.1 Video Processing

WISEngineering uses a loosely distributed server cluster for handling video upload. The nodes of this cluster do not even have to belong to the same local area network (LAN). Load balancing is achieved using a simple scheme.

The main server keeps track of a list of servers capable of handling video uploads. Each video processing server provides a set of web servlets, implemented using PHP, for uploading video files. Once a file arrives, a video processing servlet compresses the file, and uploads it to the Google/Youtube cloud service. All video processing servers use the same Google/Youtube account. When a user requests a video upload, e.g., in creating a design journal post, an AJAX request is sent to the main server, to ask for a random decision on the video server to use. This random video server is then used for upload and processing. Clearly, this scheme is easily scalable by adding more processing servers to the pool.

5.2 Instant Feedback System

The preliminary automated grading system introduced in Sect. 4.2 is not appropriate to serve as an instant feedback system. To grade a question, the grader has to load a pre-compiled classifier model, which takes at least 30 seconds on a typical server. The grading itself, however, only takes less than a fraction of a second.

One naive solution is to make the grader running as a service/daemon process in OS, and responding to grading request. This solution does not work for WISEngineering because there are many graders and each consumes at least 10 MB of RAM. One single server cannot host them all. On the other hand, grading requests (due to how class sections are run) usually come in bursts. That is, in a short period of time, the majority of grading requests can be for a small subset of questions. We need a flexible way to create a large number of graders and put down unused graders for saving system resources. This solution has to be scalable to add new hardware resources.

General Architecture. The entire instant grading/feedback system (IFS), as shown in Fig. 6, can be deployed in a local area network (LAN) different from the main server. However, all cluster nodes of IFS have to be located in the same LAN, for the convenience of setting up HDFS and Zookeeper services.

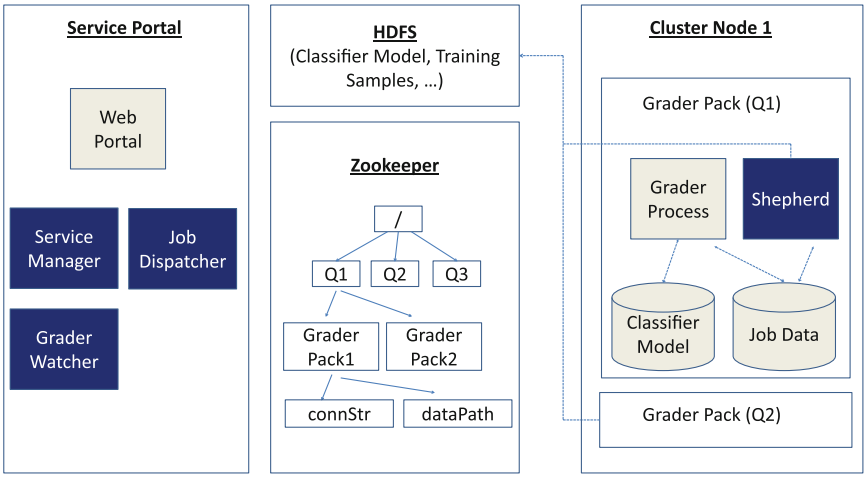

Fig. 6. HDFS/Zookeeper Cluster for Instant Feedback

The only externally visible component of IFS is its web portal. When a student user submits to WISEngineering, an AJAX request is sent to IFS web portal for instant feedback. Once the question is graded, a callback Javascript function is invoked at the client side to display the auto-grader feedback.

Running behind the web portal is the entire IFS cluster. Figure 6 displays its structure. IFS utilizes two Apache cloud frameworks: the HDFS distributed file system and the Zookeeper synchronization service [9]. We use Zookeeper as both a centralized directory service of all of our auto-grader services and a synchronization service for providing atomic and mutual exclusive access of shared data.

IFS Service Portal. The IFS web server and several Java applications (such as IFS Service Manager and Job Dispatcher) are hosted in a physical server called "Service Portal" (as shown on the left of Fig. 6).

The Service Manager is a daemon process. It examines the request/performance statistics of the IFS web portal periodically and decides to start new grading services or decommission unused ones. As the directory information of all grading services are contained in Zookeeper, the Service Manager has write access on the Zookeeper nodes.

The Job Dispatcher, once receiving a grading request (the question ID and a student response) from the WISEngineering main server, generates a new service ticket ID, and locates an available grading service by querying the directory information in Zookeeper. The Grader Watcher service is also a daemon process. It periodically checks the status of all auto-graders by sending them heartbeat messages.

HDFS and Zookeeper. The IFS cluster relies on HDFS and Zookeeper. HDFS serves as a shared file system among all computing nodes. It stores the classifier models for grading, training data, and other pertinent information.

Zookeeper is used for synchronizing the requester processes (such as the Job Dispatcher) and the service processes (such as the auto-graders) that are distributed on different servers.

More importantly, Zookeeper stores a central service directory of all services. As shown in Fig. 6, the information is structured as a tree that embodies the location and model data related to the graders for all questions in the system.

For example, when a new grading job is submitted to Job Dispatcher. It will first create a job node in Zookeeper (documenting its status, unique ticket ID). It then searches for an available grader in the pool of graders and updates its status. When the grader finishes its job, it will update the Zookeeper central directory (e.g., removing the job information and updating the grader status).

Grading Services. For each question, there is one or more grading packs. Each grading pack consists of two processes: (1) an automatic grader process, and (2) a shepherd process.

The grader process is a daemon process. Depending on the grading module it uses, it can be implemented using different languages (e.g., Python). The main body of the process basically runs as an infinite loop, waiting for requests from network sockets, which are sent from the shepherd process. When a request signal arrives, it reads the details of the request, which is prepared by the Shepherd process in advance.

The atomicity/mutual exclusive access of shared local files and classifier models are guaranteed by the Shepherd process (and Zookeeper). When the grader process finishes processing, it writes the output to a local pipe. Then the Shepard process reads the output and transfers it to HDFS, and then updates Zookeeper to wake up the waiting requester.

In summary, the use of HDFS and Zookeeper makes IFS a highly scalable, reliable, and robust system that can handle in-burst requests with reasonable consumption of hardware resources.

6 Related Work

Traditional web based educational platforms, such as Blackboard [11] and Moodle [14] provide standard course management functions, however, lack the support for massive online learning. Emerging platforms such as edX [5] and OpenMooc [16], embrace social media, mobile portal and cloud based video processing services such as YouTube.

WISEngineering has the above features developed in parallel with the aforementioned platforms and it has several unique features. First of all, WISEngineering serves a special sector of massive online learning - large scale implementation of a same curriculum with the support of a national organization (e.g., running Engineering educational activities through Boys & Girls Clubs [3]). In WISEngineering, the same curriculum could be potentially used in hundreds of classrooms in distant geographical regions. These classes are usually small, in contrast to huge classes hosted on edX, which could have thousands of students in one class section. The application context of WISEngineering raises many interesting problems. For example, for the same curriculum, why does one region have better learning outcomes than another? This demands that WISEngineering provides a well-defined learning outcome structure and correlate it with assessment questions and user behavior. It also determines the loosely coupled network topology because organization units may be hosted on different servers. This further leads to the need to aggregate reports from satellite servers and hence the remote information exchange protocol among servers.

WISEngineering uses but goes one step beyond the edX EASE automated grading module [19]. It adopts the stack generalization algorithm [27] for an open architecture to blend multiple machine learning algorithms for grading. Instant feedback system is built upon the auto-graders, and can handle the grading requests in burst. It leverages the highly robust Zookeeper cluster [9] for synchronizing and coordinating distributed auto-grader processes. This is in line with the practice of big cloud systems such as the Eclipse Communication Framework [10] and the Apache HBase [8].

Design for performance and scalability is always one of the greatest challenges faced by web application developers [26]. The REST (Representational State Transfer) pattern [6,7], allowing redundancy and caching, has been followed for decades to scale up services. The design of WISEngineering adopts most of its principles (e.g., the stateless request property), for serving client requests by randomly selecting cluster nodes at run time. WISEngineering also tries to maximize the interoperability of components by standardizing their communication interface, following the principles outlined in [15]. In general, the distributed architecture allows WISEngineering to scale up easily by adding more hardware resources. Its performance can be further tuned using Software Performance Engineering [23].

7 Conclusion

This paper has presented WISEngineering, a novel web application addressing the needs of massive online learning. The system provides many interesting features such as a powerful and extensible automated grading and instant feedback system, upon which a learning outcome analysis and reporting module is constructed. To achieve extensibility, an open software architecture is adopted to integrate heterogeneous software components. To improve service performance and scalability, back-end cloud clusters are used and interoperability is achieved using a simple service invocation protocol. The WISEngineering system is a promising platform that supports both learning and the research on learning. Future directions include investigating data mining analysis techniques that associate user behavior data with learning outcomes.

Acknowledgments. This work is partially supported by the National Science Foundation Grants DRL-1422436 and DRL-1253523. The instant grading service cluster is hosted by Hofstra Big Data Lab, funded by grant ESD CFA 29409.

References

1. Barwick, D.: Does Class Size Matter? Inside Higher Education. http://www.insidehighered.com/views/2007/12/06/barwick
2. UC Berkeley. Berkley WISE System. https://wise.berkeley.edu/
3. Hofstra STEM Research Center. Research and Development: Advances in Wise Guys & Gals - Boys and Girls as WISEngineering STEM Learners. http://www.hofstra.edu/Academics/Colleges/SEAS/CTL/wise/research-development.html
4. Hofstra STEM Research Center. WISEngineering Web Portal. http://wgg1.hofstra.edu
5. edX Inc., OpenEdx educational platform. https://open.edx.org/
6. Fielding, R.T., Taylor, R.N.: Principled design of the modern web architecture. ACM Trans. Internet Technol. **2**, 115–150 (2002)
7. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
8. Apache Foundation. Apache HBase. http://hbase.apache.org/

9. Apache Foundation. Apache Zookeeper. http://zookeeper.apache.org/
10. The Eclipse Foundation. Eclipse Communication Framework Project Home. http://www.eclipse.org/ecf/
11. Blackboard In.c. Blackboard Learning Platform. http://www.blackboard.com
12. Kohl, J., Neuman, C.: The Kerberos Network Authentication Service (V5). http://tools.ietf.org/html/rfc1510
13. McIlroy, C.: Mass produced software components. In: Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, October 1968
14. Moodle.org. Moodle: Modular Object-Oriented Dynamic Learning Environment. http://www.moodle.org
15. Nelson, D.: Next gen web architecture for the cloud era. In: SATURN 2013 Software Architecture Conference (2013)
16. openmooc.org. OpenMooc: A fully open source MOOC solution. https://openmooc.org/
17. Otto, M.: Say Hello to Bootstrap 2.0. https://blog.twitter.com/2012/say-hello-to-bootstrap-2
18. Pappano, L.: The Year of MOOC. http://www.nytimes.com/2012/11/04/education/edlife/massive-open-online-courses-are-multiplying-at-a-rapid-pace.html?pagewanted=all&_r=0
19. Paruchuri, V., Huang, D., Jarvis, J., Tauber, J., Aune, N., Kern, J.: EASE Auto-Grading Module. https://github.com/edx/ease
20. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly (2007)
21. scikit-learn developers. Scikit-learn: machine learning in python. http://scikit-learn.org/stable/
22. Slotta, J., Linn, M.: WISE Science: Web-based Inquiry in the Classroom. Teachers College Press, New York (2009)
23. Smith, C., Williams, L.: Building responsive and scalable web applications. In: 26th International Computer Measurement Group Conference, pp. 127–138 (2000)
24. Spady, W.: Outcome Based Education: Critical Issues and Answers. American Association of Schol Administrators, Arlington Virginia (1994)
25. The PhET Team. PhET: Interactive Simulation for Science and Math. http://phet.colorado.edu/
26. Williams, L., Smith, C.: Web application scalability: a model-based approach. In: Software Engineering Research and Performance Engineering Services, pp. 215–226 (2004)
27. Wolpert, H.D.: Stacked generalization. Neural Networks **5**, 241–259 (1992)