A Web Portal For An Animation Render Farm

John Rankin, Travis Boettcher, and Peter Bui
Department of Computer Science
University of Wisconsin - Eau Claire
Eau Claire, WI 54702
{rankinjn, boettcta, buipj}@uwec.edu

Abstract

Last year, we developed a collaborative animation rendering system called DSABR (Distributed System for Automated Blender Rendering) [2]. Due to the growing demand for increasingly complex digital 3D modeling and animation, our software application harnesses the computing resources of multiple networked machines to speed up the rendering of digital animated videos. Although we have working system, we wanted deliver DSABR this year as a *Software as a Service* platform. By creating a web portal, we have hope to provide an easy user interface where users can sign in, upload their blender file, set a few parameters, and be able to render their videos from the comfort of any web browser.

1 Introduction

The primary goal of our collaborative animation rendering system, DSABR [2], was to make it possible for art students to produce longer, and more detailed 3D animated movies, which generally require large amounts of compute time to render. Because Blender [1], an open source modeling tool, is capable of rendering single frames independently, we were able to distribute the rendering workload across multiple machines using Python [9] and Work Queue [3]. To actually execute the work, we utilized computing resources at both UW-Eau Claire and the Center for High Throughput Computing at UW-Madison, we were able to reduce the rendering time for a video that took 72 minutes on a single machine down to 5 minutes using our system. Despite the success of our initial system, however, it is still not widely used by the target art students because it requires knowledge of cluster computing and Linux terminal commands.

To address this lack of usability and remove this barrier, this research project focuses on building and testing a web portal that serves as a front-end to DSABR and thus allows users to easily schedule, render, and view animated videos from their web browser. This paper describes the design and implementation of our web portal, discusses the challenges we faced in building our system, and evaluates the new web portal's functions in the context of enabling novice users to utilize distributed computing to generate animation videos.

2 Design

An overall view of the DSABR web portal is shown in Figure 1. As can be seen, the design of the DSABR web application is relatively straightforward: a web portal hosted in a virtual machine that receives user requests such as scheduling an animation rendering job. Behind the scenes, the system utilizes our existing DSABR command line tool to coordinate Work Queue workers which perform the task of rendering the videos the users specified.

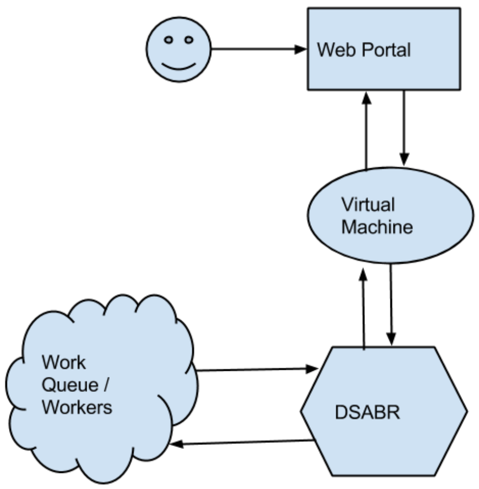


Figure 1: Web Portal Overview

2.1 User Accounts

Before users can utilize the system, they must first register or signup for an account. In the web portal, user accounts hold information such as which source Blender files did a user upload, which jobs have they submitted, and what videos have they produced.

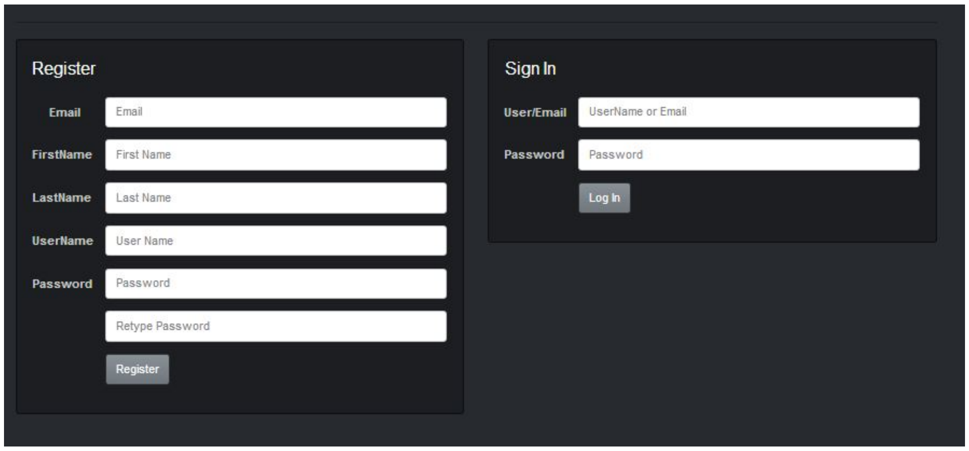


Figure 2: Registration For DSABR

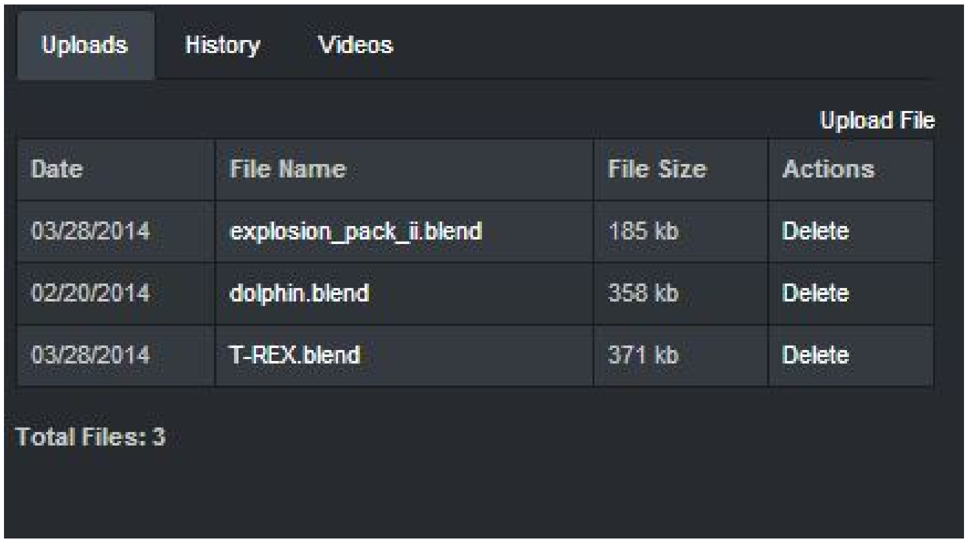
To facilitate signing up for the portal, we created a simple and efficient registration and login page. All a user needs to do is supply a few fields to register and then he or she will be able to login as shown in Figure 2.

Once they have registered for an account, users can then login to the website and access their own profile page. From this page, they can easily navigate to all of the information that is required or needed from using the DSABR application. The page is divided into two columns:

1. The left column is a tabbed navigation which allows a user to click on the tabs that show information regarding their *Uploads*, *History*, and *Completed* videos.
2. The right column of the page shows the status of their most recently launched DSABR executions.

2.2 Uploads Tab

When a user is focused on the *Uploads* tab, an Upload File link is displayed. Clicking on this link brings the user to a page where he or she can quickly upload a Blender file that can be used with DSABR. When the file is added successfully, the user will see the file listed in the table. The user is also given an option to delete the file from the Web Portal if needed as shown in Figure 3.



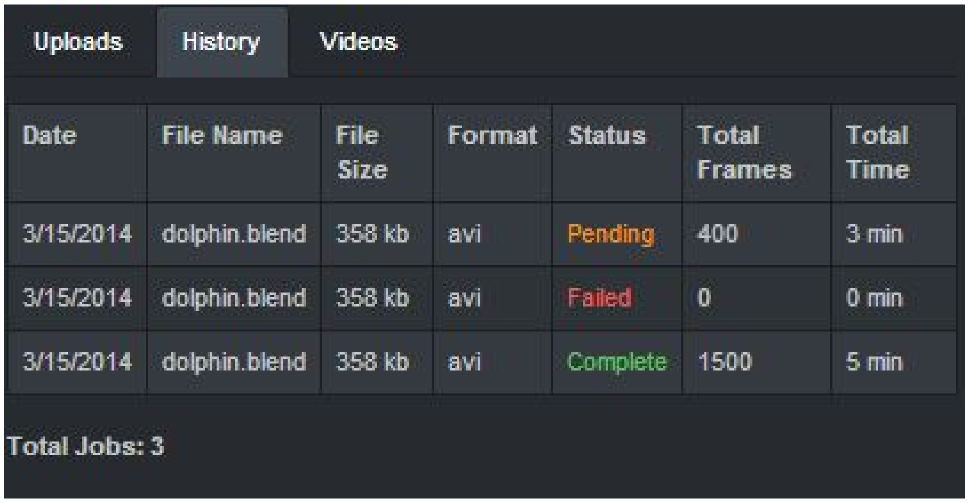
Date	File Name	File Size	Actions
03/28/2014	explosion_pack_ii.blend	185 kb	Delete
02/20/2014	dolphin.blend	358 kb	Delete
03/28/2014	T-REX.blend	371 kb	Delete

Total Files: 3

Figure 3: Uploads Tab For DSABR

2.3 History Tab

When focused on the *History* tab, the user is able to take a brief overview of all the submissions to DSABR that has been executed. This table shows a few statistics on what parameters were used in the DSABR programs such as the date when DSABR was executed, the Blender file that was used, how long it took to complete the rendering, and how many frames were executed. This table also shows the current status of the executed task such as "Completed", "Failed", or "Pending" as shown in Figure 4.



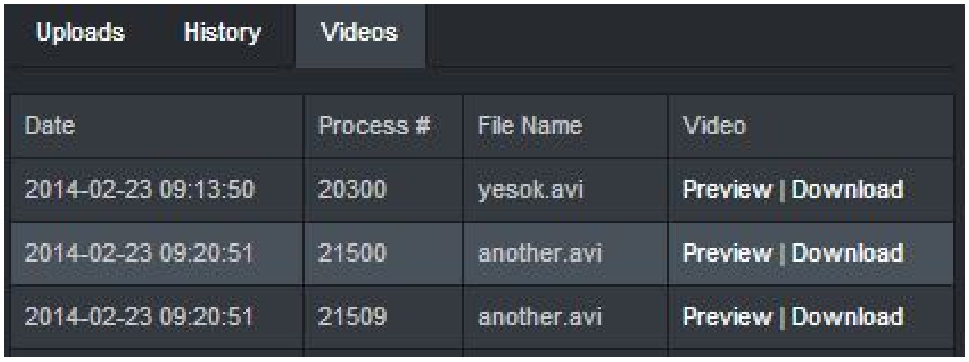
Date	File Name	File Size	Format	Status	Total Frames	Total Time
3/15/2014	dolphin.blend	358 kb	avi	Pending	400	3 min
3/15/2014	dolphin.blend	358 kb	avi	Failed	0	0 min
3/15/2014	dolphin.blend	358 kb	avi	Complete	1500	5 min

Total Jobs: 3

Figure 4: History Tab For DSABR

2.4 Videos Tab

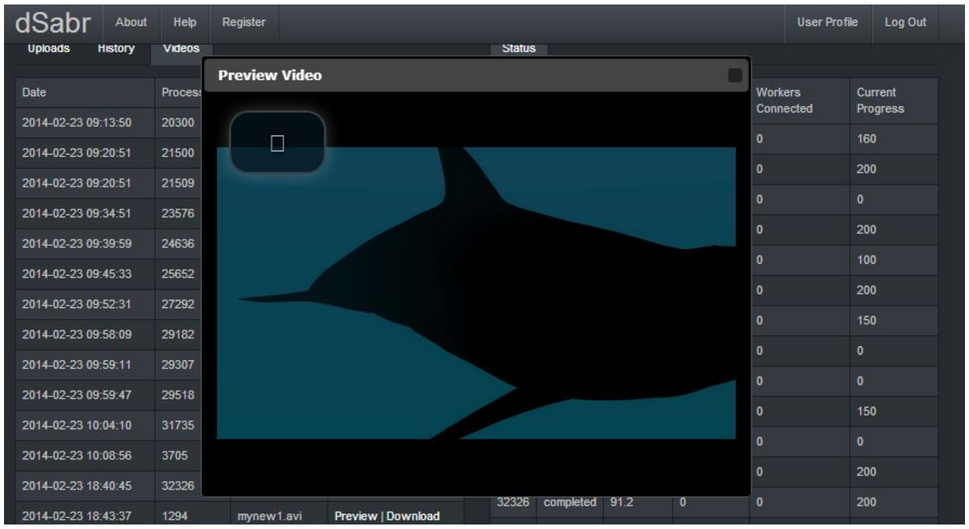
When the *Videos* tab is active, the user is able to view all of the completed videos that DSABR has successfully rendered. The user is given two links to either *Preview* the video in the web browser or *Download* the video to the local machine as shown in Figure 5.



Date	Process #	File Name	Video
2014-02-23 09:13:50	20300	yesok.avi	Preview | Download
2014-02-23 09:20:51	21500	another.avi	Preview | Download
2014-02-23 09:20:51	21509	another.avi	Preview | Download

Figure 5: Videos Tab For DSABR

When the user clicks the *Preview* button, a modal dialog opens on the current page and the user is able to watch the video through their browser as demonstrated in Figure 6.



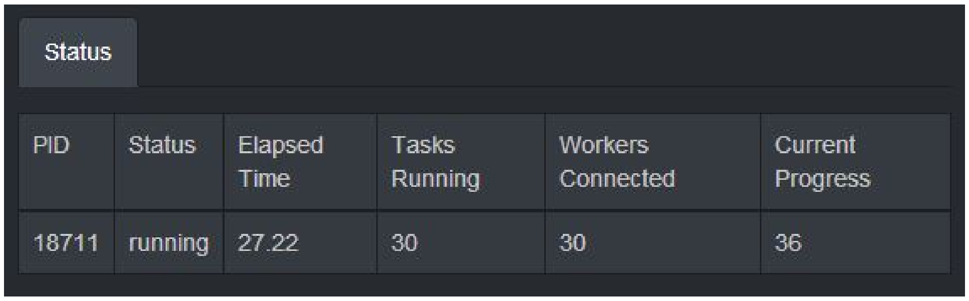
Date	Proces		
2014-02-23 09:13:50	20300		
2014-02-23 09:20:51	21500		
2014-02-23 09:20:51	21509		
2014-02-23 09:34:51	23576		
2014-02-23 09:39:59	24636		
2014-02-23 09:45:33	25652		
2014-02-23 09:52:31	27292		
2014-02-23 09:58:09	29182		
2014-02-23 09:59:11	29307		
2014-02-23 09:59:47	29518		
2014-02-23 10:04:10	31735		
2014-02-23 10:08:56	3705		
2014-02-23 18:40:45	32326		
2014-02-23 18:43:37	1294	mynew1.avi	Preview | Download

				Workers Connected	Current Progress
				0	160
				0	200
				0	0
				0	200
				0	100
				0	200
				0	150
				0	0
				0	0
				0	150
				0	0
				0	200
32326	completed	91.2	0	0	200

Figure 6: Video Preview For DSABR

2.5 Status Updates

As mentioned previously, right column of the user's profile page shows the status of their most recently launched DSABR jobs. During the execution of a job, the user can see live status updates (updated every 3 seconds) of DSABR's progress as shown in Figure 7. A user can watch the total time elapsed, how many tasks are currently running, how many workers are connected, and how many frames have been actually been completed. A user can wait patiently until the status is changed from “Pending” to “Completed” and then go to the *Video* tab in the left column to see their video as described above.

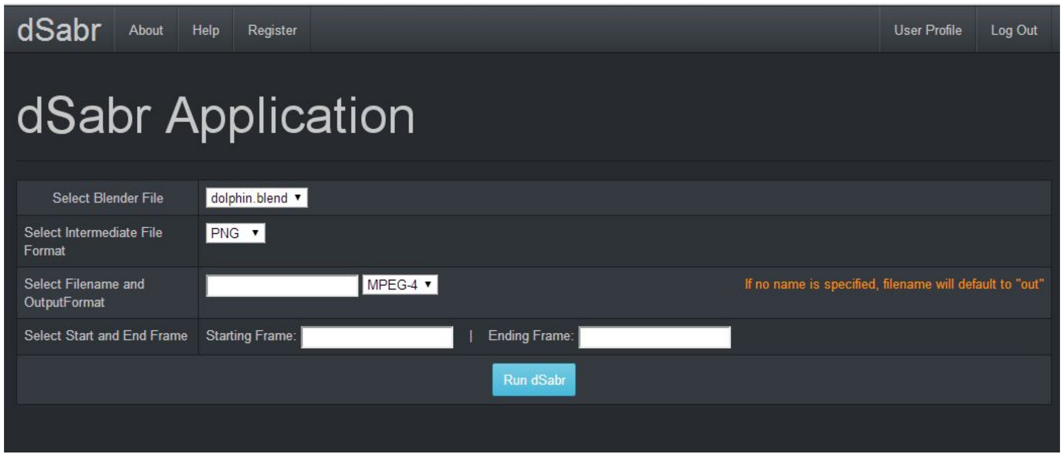


PID	Status	Elapsed Time	Tasks Running	Workers Connected	Current Progress
18711	running	27.22	30	30	36

Figure 7: Status Updates For DSABR

2.6 Job Submission

In order to begin a job, a user must go to their profile page and click on the button that says “Go to DSABR”. When this button is clicked, the user is directed to another page where the DSABR Application form is displayed. The user many select various options and then simply presses, “Run dSabr” as shown in Figure 8.



Select Blender File	dolphin.blend
Select Intermediate File Format	PNG
Select Filename and OutputFormat	MPEG-4 — If no name is specified, filename will default to "out"
Select Start and End Frame	Starting Frame: | Ending Frame:

Figure 8: Job Submission For DSABR

This page provides the user a graphical interface to DSABR. In the form, the first option allows a user to select a Blender file. A dropdown menu is populated from the database from a list of all the Blender files the user has uploaded to the server. The next option the user is given is the “Intermediate File” format. Currently, DSABR only supports `PNG`, `BMP`, and `JPG`. Next, the user is given an option to specify an output file name and an output format for which the video will be rendered as. If no output name is specified, it defaults to “out.avi”. Last, the user is given two input fields to specify the range of frame numbers to generate. The user only has to input their start frame, and their end frame, and DSABR will handle the rest.

The behind the scenes details on how the parameters in the job submission form are utilized to execute a distributed animation rendering task is described in the next section, which focuses on the back-end implementation details of our system.

3 Implementation

Behind the scenes of our web portal, the system utilizes our original DSABR command line tool to create an animated video using the options specified and while taking advantage of different distributed computing resources. This section describes the tools and techniques we used to implement our web portal, discusses any challenges we encountered while developing the system, and examines our solutions to these obstacles.

3.1 Software Stack

Last year, we created the DSABR command line application that utilizes the Python-WorkQueue framework to harness distributed computing resources in order to expedite the rendering of Blender animation files. Essentially, what DSABR does is opens request to receive workers on a specified port from WorkQueue. When these workers are accepted by the DSABR program, DSABR extracts each frame from the Blender file and sends it to the specified worker. After the worker receives the frame, it renders it into an image file, and sends it back to DSABR. When DSABR collects all of these rendered frames, and uses FFMPEG to encode all of the images into a final video file.

The goal of our project is to use a build a web portal to allow users to easily upload files, schedule DSABR jobs, and view the generated videos. To accomplish this goal, we decided on the following software stack:

1. **LAMP**: To implement our web portal, we utilized a pretty standard LAMP (Linux, Apache, MySQL, PHP) stack. First, we setup an Ubuntu 12.04 LTS virtual machine dedicated to running DSABR and our web site. Next, we configured a MySQL server to easily and quickly store information pertaining to our web portal. Afterwards, we setup the Apache webserver and configure it to allow for dynamic websites using PHP.

2. **CodeIgniter**: To ensure that the web portal would be maintainable and flexible enough to meet our requirements, we decided to use CodeIgniter [5], a PHP MVC (Model-View-Controller) Framework. We chose this system because it was relatively straightforward setup and the students had some familiarity with it prior to

starting this project (from an internship and job experience). We believe that this MVC Framework would not only enable us to rapidly build our system, but also allow other students in the future to easily modify and make enhancements to the web portal in the future.

3. **Bootstrap**: To allow for a *responsive* experience, we used a themed version of Bootstrap to help give us a quick and decent looking design. After extracting the template and making our base template for CodeIgniter, we created a few pages to fill out most of the application.

3.2 Job Execution

We initially designed the site first by focusing on what types of data we wanted to display to the user, and how we wanted the user to interact with DSABR. Once we had a way for users register and login, and our testing data in place, all we needed to do is connect the web portal to DSABR.

Before we started on getting DSABR to run, we had to figure out how we can fork off a process in PHP, store its PID, and let the user know whether a process was running, completed, or failed. This ability is required since DSABR was a command line application and did not provide any programmatic interfaces. To implement this mechanism, we created what was essentially a shell wrapper for executing a Unix process This mechanism allowed us flexibility to do other tasks such as calling the `curl` command on when a process failed as we will explain later. Luckily for us, we learned that PHP is very nice about forking off processes and the PHP code for executing our wrapper is shown in Listing 1.

```
$command = "nohup /var/www/codeigniter/dsabr.sh \
    $blender_file $blender_name $intermediate_format \
    $output_file $starting_frame $ending_frame >> \
    /var/www/codeigniter/scripts.txt \
    2>/var/www/codeigniter/err.txt & echo $!";
exec($command, $output);
$pid = (int)$output[0];
```

Listing 1: PHP Execution Wrapper

Unfortunately, before we could test running DSABR within the PHP environment, we first had to get it to run in our Ubuntu development machine, which proved problematic. First, DSABR was created and tested on CentOS and RHEL 6 machines and had strict dependencies for tools and libraries that it required. Getting DSABR to execute on the Ubuntu virtual machine required figuring out these dependencies and installing them. Some libraries such as Work Queue had to be installed manually since no package existed for it in the Ubuntu repositories. Other applications, such as FFMPEG had to also be built from source because the Ubuntu version was not built with multimedia codecs we needed.

```
#!/bin/bash

#Parameters for this script.
BLENDER_FILE=$1
BLENDER_NAME=$2
INTERMEDIATE_FORMAT=$3
OUTPUT_FILE=$4
START_FRAME=$5
ENDING_FRAME=$6

set -e

BASE=/var/www/codeigniter/dsabr
CATALOG_HOST=dpl.cs.uwec.edu
TCP_LOW_PORT=9000
TCP_HIGH_PORT=10000
SANDBOX=/var/www/codeigniter/runs/dsabr.$$

export CATALOG_HOST TCP_LOW_PORT TCP_HIGH_PORT
export BLENDER_FILE BLENDER_NAME INTERMEDIATE_FORMAT
export OUTPUT_FILE STARTING_FRAME ENDING_FRAME

mkdir ${SANDBOX}

cp -fr ${BASE}/{dsabr.py,pipefiles.py,setup_blender.sh} ${SANDBOX}
cp -fr ${BLENDER_FILE} ${SANDBOX}

# Execute DSABR
(cd ${SANDBOX};
    ./dsabr.py -b -d -o ${OUTPUT_FILE} \
        ${BLENDER_NAME} ${START_FRAME} ${ENDING_FRAME} > log 2>error.log;
    ffmpeg -i ${OUTPUT_FILE} -f flv -b 1024 -s 640x375 flashvid.flv;
    ffmpeg -i ${OUTPUT_FILE} -ss 0  -vframes 1  preview.jpg;
)

if [ "$?" = "0" ]; then
        echo "Done"
else
        # Something went wrong, so Update the failure to the DB
        curl http://localhost/run/failure/$$
        exit 1
fi
```

Listing 2: DSABR Shell Script

When we were to able to run DSABR from the command line, we then focused on getting DSABR to be executed from the website. To do accomplish this task, we created a wrapper script for the DSABR program as shown in Listing 2.

This wrapper script takes in 6 parameters, essentially all of the values the user specified on the website. We tried to initially run DSABR with the Blender files from another directory but that seemed to cause problems with Work Queue. Our solution to this problem to create a sandbox directory where we essentially copied all of the files that were going to be used for the execution of the DSABR program, including the user's Blender file. In this directory we also log any errors and save the JSON status log. This latter file is used for the *Status* column on the `Profile` page and is read asynchronously by the web client using AJAX. The bash script also quickly renders a Flash version of the video to allow a user to use the Preview functionality of the website, and also creates a preview GIF for the flash video when DSABR finishes. Finally, the bash script also checks to see if DSABR exited without failure. If something happened along the way, we let the user know by `POSTing` a URL using `curl` that sets that process status as *failed*.

4 Evaluation

As we built our web portal, we performed continuous testing and integration to ensure we always had a working website. This meant developing a small feature and then immediately testing it to determine whether or not it worked and to see how it fit into the rest of the application. Since the web portal was meant to be used by a variety of users will different levels of computing knowledge, we had to do extensive testing on multiple factors.

1. **User registration**: We have database calls in place so only one person can register with a distinct email address and a distinct username. For flexibility we allow users login with either using their username or their email address as long as the password was correct. Since we didn't want to exactly know everyone's password, we hashed each password in the database. Once the registration system was in place, we tested for various error messages when registering and logging into the system by creating multiple accounts.

   One future possibility is to avoid the registration process by integrating with UW-Eau Claire's central LDAP system. This, however, would require coordinating with the campus IT group and receiving permission from them.

2. **File uploads**: We tested the system by uploading different Blender animation files to the system. When a new user uploads a file, we have made sure that the system creates a folder with their user identification number inside the uploads directory and places the file inside of the appropriate user directory. We tested the ability to upload a file with the same name, and will give an error message to the user that notifying them that user either needs to delete the file from first, or simply rename their file before they upload.

   One issue we have not yet resolved is what to do when storage space is completely used up. While we can detect this situation, it is not clear or obvious to use how to react to the problem.

3. **Concurrent Jobs**: We tested the ability to run multiple DSABR rendering jobs at once. Unfortunately, if there is an instance of DSABR currently running, the user

will have to wait for that DSABR job to completely end before a new DSABR application can make progress. This is because while it is possible for multiple DSABR masters to execute simultaneously, our current Work Queue worker pool implementation is not very intelligent.

As noted previously, in Work Queue, workers locate a master process and requests work to do. In our current implementation, we use Work Queue's default service discovery mechanism which is to simply flock to the first available master. Because of this, workers tend to clump towards one DSABR master, starving any other DSABR jobs. Once the current master finished, however, the workers will migrate to another master, allowing the next job to progress and complete.

In our next iteration of our web portal, we hope to address this load balancing problem with a more intelligent flocking mechanism.

4. **Status Updates**: We tested our *Status* update mechanism by simply submitting DSABR jobs from the web portal, and watching the live status updates from within the Profile page. When it showed "Complete" we knew we could then click on the *Preview* tab, and our video should play in the new modal dialog box.

   We also manually killed some running jobs to ensure that our wrapper scripts we able to detect and report failed or incomplete jobs. In our testing, we were able to successfully handle all of these interruptions. Moreover, the use of a sandbox for each job proved quite useful for not only sharing the final video file, but also viewing any of the logs generated during the job execution.

5. **Video Playback**: Finally, we tested the generated video by using a variety of web browsers and video players on different operating systems. Because our web preview was encoded as a `FLV` Flash video, it was generally available to any system that had Adobe Flash installed. Viewing the generated `AVI` or `MP4` file, however, turned out to be more problematic as some systems lacked the appropriate codecs to view the generated video.

   We are currently working with an Art faculty member on what are the best options for encoding videos and hope to provide users will a list of high level options that they can select. This will ensure that all the hard work of distributing and rendering the Blender file is not lost simply because the produced video file is not playable on the desired machines.

Despite some rough edges, our web portal is overall quite functional and is ready for public testing. We are currently in discussion with an Art faculty member about using our web portal for some animation classes and hope to receive feedback and comments from the art students and other novice users.

5 Related Work

As noted previously, our web portal is a front-end to our distributed collaborative rendering system, DSABR [2], which utilizes Blender [1], Python [9], and Work Queue [3] to distribute the rendering workload across multiple machines. The system is inspired by previous research which also used Blender to develop an open source render farm [8], but differs in implementation details. For instance, by using Work Queue, our system is both scalable (i.e. the performance generally increases with additional workers) and elastic (i.e. the system tolerates workers leaving and joining the pool of resources during run-time). More details about the implementation of DSABR can be found in our previous paper [2]. The addition of a web portal front-end to DSABR is similar to ongoing efforts in building easy-to-use interfaces to sophisticated scientific applications. As noted by Ian Foster, scientific tools tend to be difficult to deploy, configure, and utilize effectively. An alternative to forcing users to become distributed system experts is to provide different service abstractions and in particular web service applications [6]. One prominent example of this is the Nanohub portal [7], which is a web site that allows users to access sophisticated scientific applications related to researching nanoscale materials and devices. Similarly, Biocompute is a web portal that allows bioinformaticians to perform research using different bioinformatics tools in an easy-to-use web interface [4]. Our research project in this paper, then, follows the lead provided by these projects and continues the efforts in providing novice users convenient access to complex tools via comfortable and straightforward web applications.

6 Conclusion

Because a tool is only useful when it can be used, we built a web portal for our animation rendering system. With this new web interface to DSABR, we hope to finally enable art students to explore their imaginations by building high quality and sophisticated animations that were once impractical or infeasible due to local computing constraints. As shown in this paper, the new web portal is functional and relatively straightforward to use. We look forward to larger scale testing and more user feedback to further improve our system.

References

[1] Blender. http://www.blender.org/, 2013.

[2] P. Bui, T. Boettcher, N. Jaeger, and J. Westphal. Using Clusters in Undergraduate Research: Distributed Animation Rendering, Photo Processing, and Image Transcoding. In *IEEE CLUSTER*, 2013.

[3] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications. In *Workshop on Python for High Performance and Scientific Computing at SC11*, 2011.

[4] R. Carmichael, P. Braga-Henebry, D. Thain, and S. Emrich. Biocompute 2.0: An Improved Collaborative Workspace for Data Intensive Bio-Science. *Concurrency and Computation: Practice and Experience*, 23(17):2305–2314, 2011.

[5] EllisLab. CodeIgniter. http://ellislab.com/codeigniter, 2014.

[6] I. Foster. Service-oriented science. *Science*, 308(5723):814–817, May 2005.

[7] K. Madhavan, L. Zentner, V. Farnsworth, S. Shivarajapura, M. Zentner, N. Denny, and G. Klimeck. Nanohub. org: Cloud-based services for nanoscale modeling, simulation, and education. *Nanotechnology Reviews*, 2(1):107–117, 2013.

[8] M. Patoli, M. Gkion, A. Al-Barakati, W. Zhang, P. Newbury, and M. White. An open source grid based render farm for blender 3d. In *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*, pages 1–6. IEEE, 2009.

[9] Python Programming Language. http://www.python.org/, 2010.