Using Clusters in Undergraduate Research: Distributed Animation Rendering, Photo Processing, and Image Transcoding

Peter Bui, Travis Boettcher, Nicholas Jaeger, Jeffrey Westphal
Department of Computer Science
University of Wisconsin - Eau Claire
Eau Claire, Wisconsin 54702
Email: {buipj,boettcta,jaegernh,westphjm}@uwec.edu

***Abstract*—With distributed and parallel computing becoming increasingly important in both industrial and scientific endeavors, it is imperative that students are introduced to the challenges and methods of high performance and high throughput computing. Because these topics are often absent in standard undergraduate computer science curriculums, it is necessary to encourage and support independent research and study involving distributed and parallel computing. In this paper, we present three undergraduate research projects that utilize distributed computing clusters: animation rendering, photo processing, and image transcoding. We describe the challenges faced in each project, examine the solutions developed by the students, and then evaluate the performance and behavior of each system. At the end, we reflect on our experience using distributed clusters in undergraduate research and offer six general guidelines for mentoring and pursing distributed computing research projects with undergraduates. Overall, these projects effectively promote skills in high performance and high throughput computing while enhancing the undergraduate educational experience.**

I. Introduction

Whether it is the rise of *Big Data* or the advent of *Cloud Computing*, it is clear that modern computer scientists must learn to harness the abundant amount of computational power available to them in order to solve large and complex problems. With the deluge of industrial and scientific data constantly increasing [1] along with the dramatic shift to multi-core and heterogeneous architectures [2] and even dynamically provisioned distributed systems [3], it is imperative that today's computer science students become familiar with the concepts and techniques required to not only utilize current high performance and high throughput systems but also to develop tomorrow's solutions for problems not yet imagined.

Unfortunately, traditional undergraduate computer science curriculums are often insufficient when it comes to exposing students to distributed and parallel computing [4]. Distributed computing in particular is regularly absent from most undergraduate computer science programs. Although topics such as concurrency and parallelism often appear in operating system or computer architecture courses, distributed computing is often alluded to in passing and rarely explored in-depth. This lack of exposure to distributed systems makes it difficult for students to appreciate and comprehend the myriad issues and challenges present in trying to scale applications across multiple independent computing systems.

978-1-4799-0898-1/13/$31.00 © 2013 IEEE

Recently, there has been a concerted effort among computer science educators to introduce parallel and distributed computing into the undergraduate curriculum [5]. At some colleges, these topics are integrated directly into existing core courses such as CS1 or CS2 [6], while at other institutions the topics may be offered in an upper level elective [7]. The focus of these efforts is not to teach any one particular technology or approach, but to instill an understanding of the fundamentals of parallel and distributed computing.

As such these classroom exposures tend to be broad and thus shallow, leaving some students wanting more. For these learners, one effective way to further their interests in distributed computing is by incorporating the use of clusters into undergraduate research. By utilizing a computer cluster in independent study or collaborative research, students gain a deeper understanding of the challenges and approaches to harnessing multiple machines to solve complex problems. Moreover, for institutions that lack an upper level elective in distributed and parallel computing or who have not yet integrated these topics into the core classes, using clusters in undergraduate research provides an effective vehicle for providing students the opportunity to learn about high performance and high throughput computing.

This paper examines the use of clusters in undergraduate research at the University of Wisconsin - Eau Claire, which is a primarily undergraduate university with approximately 220 computer science majors. It presents three undergraduate research projects conducted this year which utilized clusters and distributed computing:

1) **Animation Rendering**: In this project, an input animation file is processed to generate a series of rendering tasks that produce a collection of frames that are stitched together to form an animated video.
2) **Photo Processing**: This project involves using both a computer cluster and a cloud storage service to automatically process and archive digital photos taken at remote archaeological dig sites.
3) **Image Transcoding**: This study examined the limitations of scaling an image transcoding workflow on our local distributed computer cluster.

The remainder of the paper is as follows: The next section provides background information about using clusters in computer science undergraduate research and how UW - Eau Claire supports such endeavors. This is followed by a description of the undergraduate research projects mentioned above. Because the latter two have been recently published at the Midwest Instruction and Computing Symposium (MICS) [8], [9], we only briefly describe the technical design and implementation of all three projects and then focus mainly on examining the challenges and issues faced in each project. Afterwards, we evaluate the performance of the systems, and analyze any issues that surfaced while testing these projects. At the end, we reflect on the lessons learned from these experiences and enumerate a set of guiding principles for enhancing and promoting future undergraduate research experiences with distributed computing clusters.

II. Background

The following describes the nature of undergraduate research in computer science, the availability of clusters at primarily undergraduate institutions, and how UW - Eau Claire supports the use of clusters in undergraduate research.

A. Undergraduate Research

Undergraduate research in computer science is a common practice at both research and teaching institutions where undergraduate students are introduced to the labor-intensive process of scientific research [10], [11]. Although some students simply volunteer their time, most students participate in research for pay (hourly or stipend) or for credit (independent study, in-class projects, or capstone projects) [10]. Because undergraduates, particularly lower-division students, tend to lack the theoretical background and technical skills for more sophisticated projects, undergraduate research tends to involve either implementation-oriented projects where students program tangible software artifacts, or experimentation projects where students collect and analyze experimental data [11].

Regardless of the reasons why students choose to participate in undergraduate research or the exact nature of the projects, the primary goal for the faculty mentor is to provide opportunities for the students to develop new skills and to explore their interests. Furthermore, recent studies have found that undergraduate research opportunities increase student understanding, confidence, and awareness while clarifying student interests in STEM careers [12].

B. Clusters

While it is common for research institutions to have the ability to provide undergraduates access to distributed computing clusters, it is less common at smaller primarily undergraduate institutions. Due to high costs, access to such systems can be problematic and limited. Fortunately, there have been efforts to increase access to distributed systems at all levels of higher education. One example of this is XSEDE (The Extreme Science and Engineering Discovery Environment), which is a virtual system that allows scientists from around the world to share computer resources [13]. Another example is the Open Science Grid (OSG), which is a federated consortium of open distributed computing clusters available to various research and academic communities. At a smaller scale, there are efforts such as the LittleFe [14], which is a portable six node Beowulf style computational cluster for teaching parallel and distributed computing. All of these efforts emphasize the fact that while high performance and high throughput computing is growing in importance, access to computing clusters is still limited, especially for smaller institutions such as primarily undergraduate colleges and universities.

C. University of Wisconsin - Eau Claire

Despite the challenges in providing students access to a distributed computing cluster at a primarily undergraduate institution, the computer science faculty at UW - Eau Claire believe it is imperative that students be exposed to distributed and parallel computing [6]. Because of this, the primary author[1] of this paper used his startup funds to purchase a Dell D820 16-core rack server to serve as the basis of a new HTCondor cluster [15]. Virtualization was used to partition the single rack server into 3 independent nodes to form a small cluster, which is a technique employed at other teaching institutions [16]. Over time, additional nodes were added to the cluster by scavenging surplus equipment and by connecting faculty workstations to the cluster. Additionally, the primary author was fortunate enough to participate in the LittleFe buildout session during the HPC Educators program at Supercomputing 2012 and was able to add the LittleFe to the UW - Eau Claire HTCondor cluster.

To further increase access to computational resources, the primary author also contacted the HTCondor team at UW - Madison, who responded by enabling flocking (i.e. migrating) of jobs from the UW - Eau Claire HTCondor cluster to the machines in the Center for High Throughput Computing (CHTC). Additionally, a collaboration was negotiated with the chemistry department at UW - Eau Claire to enable access to their computational science cluster via HTCondor. With all these arrangements, undergraduates at UW - Eau Claire now have access to 140 cores across two local HTCondor clusters and over 1000 more cores via flocking to the CHTC.

III. Projects

Given the availability of a distributed computing cluster on campus, it was only a matter of time before undergraduates soon became interested in utilizing it. The first project described below is a distributed animation rendering system that was the result of a collaboration between the computer science and art departments. The students involved were funded by the Office of Research and Sponsored Programs (ORSP) at UW - Eau Claire and given a small stipend. The second project involves an automated photo processing pipeline that was part of an independent study by a computer science major and a member of the geology department. The final project is a study performed by a computer science undergraduate who was simply interested in distributed computing and wanted to learn more about the topic. Because the latter two projects have been published recently [8], [9], this section only briefly describes the design and implementation of each system and then focuses on discussing the underlying distributed computing challenges faced in developing each project.

[1] The first author is the faculty mentor, while the rest of the authors are the undergraduates who collaborated in the research projects.

A. Animation Rendering

The goal of the first project is 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. To solve this problem, we built DSABR (Distributed System for Automated Blender Rendering), which uses Python [17], Work Queue [18] and Blender [19] to produce computer-animated movies.

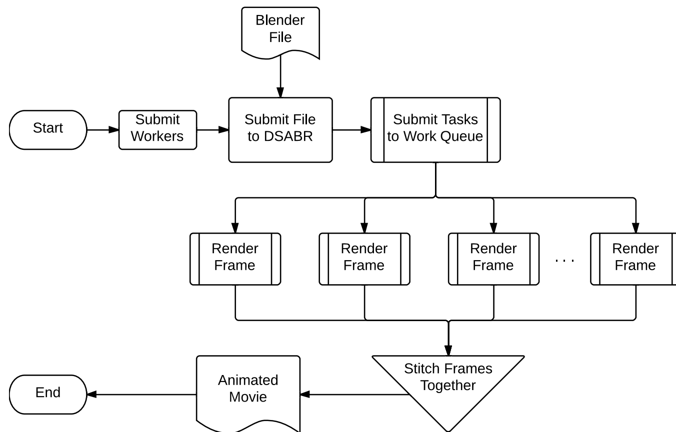

Fig. 1. Animation Rendering Architecture.

As shown in Figure 1, DSABR renders animations by submitting tasks to many computers using the Work Queue framework. Distributing the rendering is possible because Blender is capable of rendering single frames; each task submitted to Work Queue corresponds to a single movie frame.

The initial step in rendering a movie with DSABR is to submit workers to a distributed computing cluster. In this case, we would start workers on our HTCondor cluster. The number of workers submitted depends on the file to be rendered and the desired animation length, but generally the more the better as we will discuss in the next section. Because Work Queue is designed to handle dynamic worker pools, we can scale the number of machines working on the project at run-time.

The next step is to use DSABR to submit the Blender project to be rendered. All DSABR needs to know is the location of the project and the number of frames to be rendered. Optionally, the user may supply arguments that allow for debugging and logging, specify the output file's name and type, or set the intermediate image type. Upon submission, DSABR divides the project up into a series of frames. Each frame constitutes a Work Queue task which is sent off to the workers started in the previous step. Those workers render the frame, send the resulting image back to DSABR, and receive another task. When all images have been rendered, DSABR stitches them together into a movie using FFmpeg.

Although this system appears straightforward, several issues and challenges arose during the development of DSABR. One initial problem encountered was not being able to use every cluster node to render the Blender file. This was because not every machine had Blender or one of its library dependencies installed. The solution to this problem involves telling each task to download the Blender archive from a central server to the local worker machine, extract it, and create an executable wrapper to the Blender program. This extracted archive and wrapper is then cached so that the current worker can jump right to rendering the next frame on subsequent tasks.

Another challenge faced was straggling workers. If a worker takes too long to complete its task, it can hold up the entire rendering process. For example, if a file would under ”normal” conditions render in one minute, straggling workers could cause that rendering run-time to exceed ten minutes. This was solved by using a feature of Work Queue called fast-abort. This feature allows DSABR to remove a worker taking longer than the average rendering time multiplied by some given multiplier, and resubmit the task to another machine. Finding the ideal multiplier was tricky - if it is too small, DSABR culls almost all the workers because the average never has time to grow; on the other hand, if the multiplier is too large, the program behaves as though fast-abort was off.

The final set of problems involved stitching the frames together to form the final movie. The first issue dealt with how DSABR saves the intermediate image files. Initially, all the frames were saved in one directory. Once we began to render longer movies, however, it became apparent that this was not going to work - as the number of frames grew into the thousands and tens of thousands, the file system would either dramatically slow down or fail altogether. The solution involves *sharding* the image files in subdirectories such that each subdirectory contains only a limited number of files. This alleviates the stress on the filesystem and eliminates the issue.

The second final challenge was telling FFmpeg the files to stitch together. Our first attempt used Python's subprocess module and shell commands to pipe the files as command line arguments to FFmpeg. While at first this seemed like it was going to work, when DSABR was used to render longer movies (10,000+ frames or 7+ minutes), DSABR would fail. This is because the total size of command line arguments on Linux is limited. The final solution still makes use of Python's subprocess module, but uses an additional Python script to stream the contents of each image file directly to FFmpeg.

B. Photo Processing

The second project involves creating a system for automatically processing and archiving archaeological photographs at remote archaeological dig sites. To support a team of field scientists, we developed DP3 (Distributed Photo Processing Pipeline), which utilizes Dropbox for initial storage, Work Queue to coordinate the distributed computation, and HTCondor as the cluster environment. Once again, the whole application was written in the Python programming language.

An overview of DP3 is provided in Figure 2. The first phase of the system requires users to transfer their photos to personal laptops and then upload them to Dropbox [20]. A daemon running at UW - Eau Claire monitors the Dropbox repository for incoming files. When new photos are detected, the daemon dispatches a series of photo processing tasks to the local HTCondor cluster using the Work Queue framework. These processing tasks typically involved extracting meta-data and reformatting the images for different uses. When these tasks are completed the original photos and the generated artifacts are archived to multiple storage systems for redundancy. The daemon also serves web portal that allows users to track and

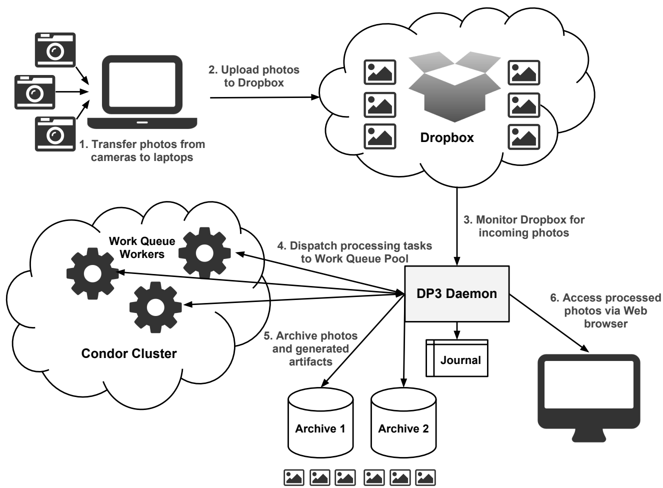

Fig. 2. Photo Processing Architecture.

access the source images and the generated outputs from any web-capable device.

In developing and using this system, we came across a couple of challenges. As with the previous project, DP3 ran into the problem of different machines in the cluster not having the appropriate software for processing the photos. To solve this, we used Starch [21] to create self-extracting application archives (SAAs) which contained all the necessary dependencies of the photo processing tools. When we needed to perform a photo processing task, we simply shipped the SAAs to the remote workers instead of the original executables.

Because of the complexity of the system, we often had to stop and restart the application to fix and debug problems. Since we were doing this during a live field test, it was necessary that we ensured that system functioned properly at all times and maintained a consistent state. To increase reliability and to guard against incorrect behavior due to system stops and restarts, we implemented a transaction log that tracked the state of each photo as it progressed though our pipeline. The details of this transaction journal are explained in our MICS paper [8]. With the transaction log in place, we were able to confidently stop and restart our daemon, knowing the system would function properly on restart.

Finally, because this was a long running project, we did not want to unnecessarily tie up the cluster resources. Since Work Queue supports dynamic allocation of workers, we employed the `work_queue_pool` utility to automatically maintain a set of workers. With this tool, users set limits on the minimum and maximum amount of workers to allocate to a project. The pool manager then monitors the statistics of the workflow and automatically scales the number of workers up and down depending on the current need. In the context of this project, this meant that if there were no photos to be processed, the worker pool manager would remove workers until it reached the minimum number. When more photo processing tasks were scheduled, the manager would detect a backlog of tasks and would dispatch more workers until the either the limit is reached or there were no more tasks to be executed. As such, we were able to avoid unnecessarily using cluster resources by taking advantage of Work Queue's elastic capabilities.

C. Image Transcoding

The third research project is an empirical study of the limits of scaling an image transcoding workflow. In this project, we used Python and Work Queue to implement an image transcoding application that utilized the HTCondor cluster to transcode images in parallel. The focus of the project was to determine the effects of input file sizes and number of workers on the performance of the application.

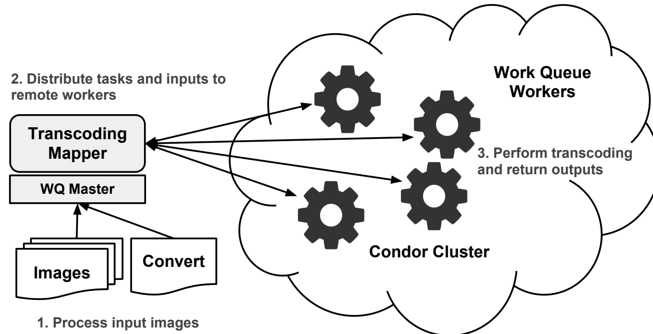

Fig. 3. Image Transcoding Architecture.

Figure 3 provides an overview of the transcoding application. First our application reads in the set of images and the specified conversion utility from the user. Next, the program maps each image to a corresponding transcoding task and uses Work Queue to dispatch these tasks to workers running in the HTCondor cluster. The remote workers perform the transcoding and return the generated output files back to the application. Further description of the system can be found in our MICS paper [9].

Once again, this project, although rather straightforward, ran into a few challenges. First, as with the other projects, the problem of software dependencies popped up again. As with DP3, this project utilized Starch to create self-contained executables. Another major problem that arose in this project is the fact that Work Queue automatically caches the input files on the remote workers. This means that transcoding the same input image N times would not be the same as transcoding N files since in the former case the image file is only transferred once. To workaround this problem in our benchmarking, we had to create directories of N input files. Finally, the last problem our study encountered was ensuring that our workers connected to our master in a timely fashion. Because Work Queue workers will slowly back off when disconnected from the master, it proved difficult to re-use workers across multiple runs of the benchmarking and get consistent timings. To work around this problem, we simply killed off each set of workers after each benchmark run and started a fresh batch before the next test. Doing so ensured that the new set of workers would connect to the master immediately and reduced the amount of variability in our timing results.

IV. Evaluation

In this section, we evaluate each of the described projects and discuss any challenges faced in testing the applications on distributed computing clusters.

A. Animation Rendering

To evaluate our animation rendering application, we measured the execution time for an increasing number of frames and Work Queue workers. Although DSABR vastly improves the rendering time of the user's Blender project, the exact speedup depends on the complexity of the animation and number of frames being rendered. The first Blender project used to test DSABR, `dolphin.blend`, is intentionally simple so that render times would be short, but the improvement over rendering on a single machine is shown in Figure 4.

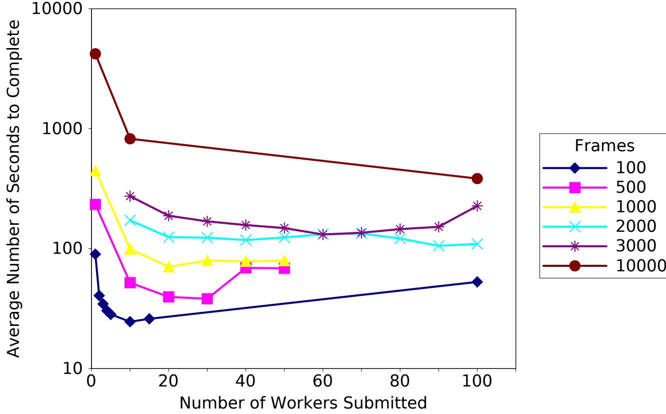

Fig. 4. Average Render Time of dolphin.blend With Respect to the Number of Submitted Workers.

As can be seen, rendering times are generally reduced with an increasing number of submitted workers until we reach a certain limit: 40 workers. After this point, the workers must come from the CHTC rather than the local HTCondor cluster. Because of the variability in the startup times for those workers and the increased network distance, workers in the CHTC are often volatile and exhibit mixed performance.

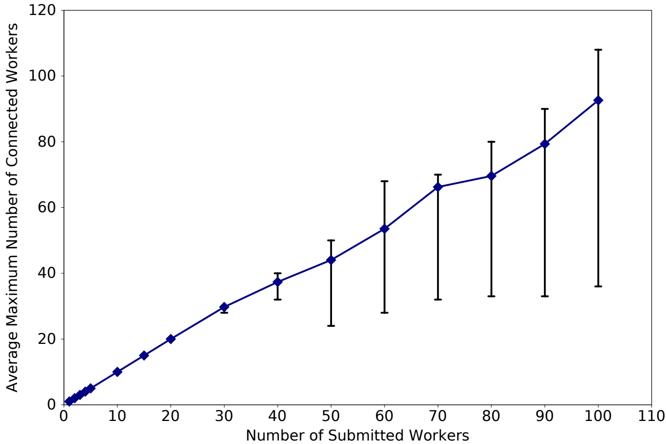

Fig. 5. Average Maximum Number of Connected Workers With Respect to the Number of Submitted Workers.

Moreover, the maximum number of workers that would connect to the master was one of the problems faced in testing DSABR. In testing a simple project, even if 100 workers are submitted for a given project, there is no guarantee that all 100 will connect. Because of this, even if there is an improvement to the render time with more workers, there was a greater observable variation in the render time when using flocking compared to only using the local campus cluster.

This phenomenon is evident in Figure 5, which plots the average maximum number of connected workers against the number of submitted workers. The minimum and maximum amount of workers connected for that number of submitted workers are shown with the whiskers. This clearly shows the soft limit of between 32 and 36 maximum connected workers (i.e. the number of local HTCondor cores).

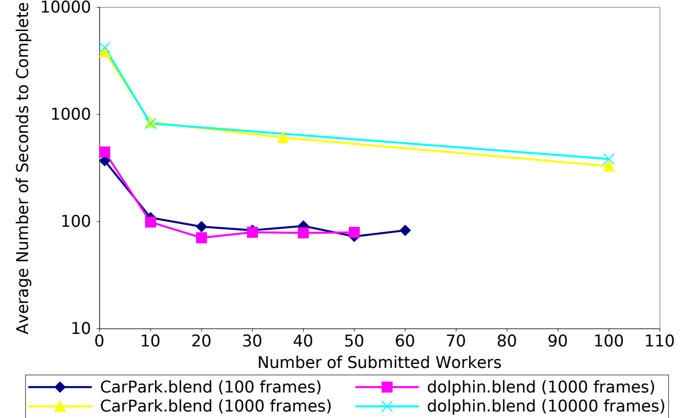

Fig. 6. Average Render Time of Two Different Files With Respect to the Number of Submitted Workers.

Figure 6 illustrates how complexity can impact the render time of a project. The Blender project used for testing, `dolphin.blend`, had no light sources and so all that needed to be rendered was the silhouette of the dolphin. On the other hand, `CarPark.blend` made use of many light sources and reflective surfaces. As a result, similar render times were seen between DSABR projects where frame totals were a magnitude different (i.e. 1000 frames of dolphin.blend rendered in relatively the same time as 100 frames of CarPark.blend). This may also imply that more workers may be of more use for projects where it takes proportionally longer to render one frame.

B. Photo Processing

For our second project, we evaluated DP3 by utilizing the system in a live field test. Working with collaborators in geography and archeology, we configured the system to process images from archaeological dig sites in Israel.

Number of Batches	448
Maximum Batch Size	1385
Minimum Batch Size	1
Average Batch Size	16.4
Number of Tasks Submitted	7372
Number of Tasks Failed	104

TABLE I. DP3 Statistics

Table I provides a summary of the results from this live field test. In total, DP3 executed 448 batches of tasks where the largest such group consisted of 1385 photos, while the smallest consisted of just a single image. The average number of tasks in each batch was 16. Overall, 7372 photo processing tasks were submitted. Of those, 104 resulted in failures (1.4%). Some of these failures were due to user error while other

failures were because our tasks executed on machines without the necessary libraries for our processing tasks. Because of our transaction journal, however, we able to ignore these errors or temporarily suspend the application, fix the problem, and restart it without duplicating work or losing data.

The biggest performance limitation turned out to be the upload speeds available to the researchers in the field. Because the digital images were quite large and the researchers only had access to slow Internet connections, getting the images to Dropbox proved to be a serious bottleneck. Moreover, not all of the users understood how to configure the Dropbox settings, so the upload speeds were further throttled by the Dropbox software, which only took advantage of 75 percent of the available upload speeds. Because of this, the system remained largely under-utilized due to the lack of data to process. More details about the performance of DP3 can be found in our MICS paper [8].

C. Image Transcoding

For the final project, we benchmarked our image transcoding application to test for scalability and performance. To do this, we used three sets of three images with different sizes: 15 KB, 1 MB, and 10 MB. These images were organized into sets of 10, 100, and 1,000 files of each size. For each combination of image size and group size, we utilized our application to transcode the images and measured the execution time necessary to convert each set of images using a given number of Work Queue workers, which we varied according to the following increments: 1, 2, 4, 8, 16, 24, and 30.

File Size	Set Size	# of Workers						
		1	2	4	8	16	24	30
15KB	10	1x	1.47x	1.56x	2.13x	1.85x	2.00x	**2.40x**
	100	1x	1.60x	2.80x	4.43x	5.96x	6.42x	**6.44x**
	1000	1x	1.65x	3.12x	5.02x	7.97x	9.27x	**9.31x**
1MB	10	1x	1.65x	2.40x	2.78x	3.05x	3.73x	**3.87x**
	100	1x	2.10x	3.87x	6.55x	**9.56x**	7.65x	8.27x
	1000	1x	2.17x	4.28x	7.75x	11.2x	10.5x	**12.12x**
10MB	10	1x	1.84x	2.46x	2.88x	**4.48x**	3.43x	3.27x
	100	1x	1.98x	3.90x	4.95x	**7.34x**	4.61x	4.76x
	1000	1x	1.74x	3.97x	5.63x	**6.26x**	4.75x	4.93x

TABLE II. TRANSCODING BENCHMARK SPEEDUPS RESULTS

Table 3, which comes from our MICS paper [9], summarizes the results of our benchmarking experiments. Based on our benchmarking, it is clear that the time required to transcoding a set of images files is generally reduced as the number of Work Queue workers is increased, although not linearly. For moderate amounts of files and medium sized files, we see a good amount of speedup as the number of workers increases. For small file sizes and small number of files we see only slight improvements. Large file sizes and large number of files, however, display erratic behavior most likely due to network transfer bottlenecks. That is, because Work Queue is based on the master-worker paradigm, all data must be transferred to and from the master. For larger files the master quickly becomes a bottleneck and thus limits the overall system throughput.

Although all of the undergraduate research projects in this paper experienced challenges in evaluating and testing their performance, each system reached high-levels of functionality and demonstrated some amount of performance improvements. The animation rendering system was able to demonstrate greatly reduced rendering times, while the photo processing application proved reliable and effective in a live field test. Finally, the transcoding study was able to demonstrate the limitations of scaling a naturally parallel application.

V. REFLECTION

Based on our experience in utilizing distributed computing clusters in undergraduate research projects, we offer the following guiding principles for promoting and encouraging future endeavors:

1) ***Develop applications not infrastructure:*** As mentioned in the introduction, most computer science undergraduates are unfamiliar with distributed computing. Because of this lack of familiarity, we find it easier to attract students by focusing on developing distributed applications rather than low-level systems. This means that rather than constructing distributed systems infrastructure, students instead build applications that utilize distributed computing to take advantage of clusters. Since students may not initially be interested in the low-level aspects of distributed systems, aiming for high-level applications such as the animation rendering system or the digital photo processing pipeline described in this paper helps keep the students engaged and motivated. In this context, distributed systems is not the end, but the means for solving sophisticated problems. This is reasonable since the purpose of undergraduate research is not necessarily to develop new distributed systems, but instead to expose students to distributed computing and support the exploration of their interests.

2) ***Utilize high-level frameworks:*** Building off the previous principle, we recommend that students develop their applications by using high-level frameworks that abstract some of the low-level details of distributed computing from them. Because the focus is for students to become familiar with the challenges involved in coordinating multiple autonomous machines to achieve a common goal, it is not necessary that students to delve into low-level details. Instead, students should be introduced to frameworks such as MapReduce [22] or Work Queue [18], which allow them to take advantage of distributed systems without all the complications of distributed computing. This is important because once students can get over the initial challenges of getting their application to run on multiple machines, they can focus on more interesting problems.

As described previously, all of the projects in this paper utilized Work Queue as the underlying distributed computing framework and yet the students still had to overcome many of the common problems found in distributed applications: dependency management, local system limitations, data transfer bottlenecks, and imperfect scaling, etc.. Our reasoning is that high-level frameworks provide scaffolding that enables students to quickly get started with distributed computing, while not completely removing all of the interesting problems. The students will still have to solve complex problems; it is just that these problems will be issues that appear when there is a somewhat functional distributed application rather than merely trying to get the application to execute on a cluster.

3) ***Recognize that nothing is straightforward:*** With the focus on developing applications rather infrastructure and the use of high-level frameworks, it may seem that the research

projects should be relatively straightforward. This, however, has not been our experience. When constructing distributed applications a multitude of issues can appear even when you have a functional program. For instance, in all three projects we had to deal with the issue of dependency management. Because the machines used in our cluster had different configurations, simply transporting our executable from the local machine to the remote node was insufficient since the remote site may not have the appropriate libraries required for execution. This was particularly the case in the animation rendering project which used machines in two different clusters and thus had to deal with a large heterogeneous mix of machines.

Scaling applications to tackle large problems will also lead students into the limits of local systems. For instance, the animation rendering project had to overcome Linux's limitations on the number of files that can be effectively stored in a single directory and on the number of arguments that can be passed through the command line. Likewise, the transcoding project eventually encountered the problem of the master node reaching the limits of its network bandwidth capacity.

Even though the students were focused on developing applications and utilized high-level frameworks, they were not completely alleviated from the challenges of effectively utilizing clusters. Despite appearing straightforward, the projects encountered non-trivial problems that surfaced in implementing and testing the programs. This is because even with the aim of constructing high-level programs and the use of scaffolding abstractions, developing reliable and effective distributed applications is challenging due to a wide range of issues that surface when scaling beyond a single machine.

*4) **Practice incremental development:*** Because of these challenges, we recommend the practice of incremental development during the course of the research collaboration. For us, this research process means the following:

- **Iterative Development**: Rather than try to build the whole system at once, students work on a single focused task at a time with the goal of always having a functional application. This means that the first task for the students is to simply to write an application that dispatches jobs to the cluster. Once this program is working, the students move on to other tasks such as collecting and processing the results of the remote tasks. After each iteration, the students identify the challenges they currently face and brainstorm different approaches for solving their problems. Such an approach naturally decomposes a larger project into smaller tasks which prevents students from becoming overwhelmed and helps them stay on track.
- **Version Control**: Because applications are developed incrementally, the students use a version control system (VCS) to manage their source code. For the projects described in this paper, we used Mercurial as the underlying VCS and Bitbucket to host our repositories. Using a VCS fits naturally with the iterative development process since it encourages developers to periodically commit changes. Additionally, since the projects involve multiple people collaborating on the same codebase, the VCS also facilitates distributed development. Finally, the VCS commit log also provides faculty mentors a way of tracking the progress of the students without directly inquiring them.
- **Blogging**: Since research involves not only developing interesting systems, but also communicating the results and discoveries, students are required to maintain a blog that serves as an online journal of the students' progress. Each week, students describe the progress they have made and any issues they have faced. This serves as a record of their work and helps them elucidate their problems and clarify their thoughts.

The three practices above all come together during the weekly meetings we have with our undergraduate researchers. Before this meeting, faculty mentors review the students' blog posts and check out the latest version of their project code. This allows mentors to review the students' progress and to anticipate any questions that may come up during the meeting. At the meeting, students recount their progress and current issues and the faculty mentors try to guide the students towards different solutions and possible resources.

Overall, this incremental development process has been effective for us in terms achieving successful research projects and keeping the students motivated and engaged. Encouraging our students to use a hosted version control system such as Bitbucket and to keep track of their progress on a blog has the side benefit of developing an online digital portfolio that they can use in applying to internships, jobs, and graduate schools.

*5) **Seek interdisciplinary collaboration:*** Another crucial aspect of a successful undergraduate research experience is to expand the horizons of our students. Beyond fostering their interests in distributed computing, we also strive to show how their computer science knowledge and skills can be applied in other disciplines to solve interesting and relevant problems. As Fred Brooks noted [23]:

> If the computer scientist is a toolsmith, and if our delight is to fashion power tools and amplifiers for minds, we must partner with those who will use our tools, those whose intelligences we hope to amplify.

Therefore, we seek collaborators in different disciplines whenever possible. The animation rendering project, for instance, is a collaboration with between computer science and art; a computer science student is developing the system described in this paper while an art student is constructing an animated film that will take advantage of increased rendering capabilities. Likewise, the digital photo processing project is a collaboration between computer science and members of the geography department. Such interdisciplinary collaborations allow our students to learn how to work with people beyond their major and to see how the knowledge and skills they are learning can be applied to solve real world problems.

*6) **Take advantage of internal and external resources:*** Our final recommendation is to harness both internal and external resources. For us, this means taking advantage of the funding opportunities provided our school's Office of Research and Sponsored Programs (ORSP). With ORSP's support, we were able to provide a small stipend to some of our students and pay for their travel and registration fees at conferences such as the Midwest Instruction and Computing Symposium. Additionally, ORSP hosts an annual research symposium

called CERCA (Celebration of Excellence in Research and Creative Activity), which provides a platform for students to present their work to the local campus community.

An example of harnessing external resources is how we connected our local HTCondor cluster to the Center for High Throughput Computing (CHTC). Because our local cluster only consists of around 40 CPU cores, we reached out to the HTCondor team at the University of Wisconsin - Madison who agreed to let us migrate some of our jobs to the machines in the CHTC via HTCondor's flocking mechanism. Access to the CHTC enables us to do sophisticated and large scale projects. For instance, the animation rendering system took advantage of hundreds of CPU cores and has demonstrated significant reductions in rendering times by utilizing machines in both the UW - Eau Claire cluster and those at the CHTC.

It is important to remember that primary purpose of the undergraduate research experience is to develop the students' knowledge in a specific field and allow them to constructively explore their interests. While some of the guiding principles above apply to undergraduate research collaborations in general, we offer these six guidelines based on our experience in promoting and supporting the use of distributed computing clusters in undergraduate research projects.

VI. Conclusion

Due to the growing importance of distributed and parallel computing in both industry and academia, it is imperative that computer science educators expose their students to high performance and high throughput computing. Although there is a concerted effort to integrate some of the concepts of distributed and parallel computing into the core computer science curriculum, distributed computing is still lacking in coverage. This paper presents undergraduate research and independent study as a viable alternative or supplement to course-based instruction for introducing students to cluster computing.

As demonstrated by the projects examined in this paper, utilizing a cluster in undergraduate research provides students an opportunity to tackle issues prevalent in distributed computing and to further their understanding of computer science. Practicing incremental development, focusing on developing applications, and utilizing a high-level framework to abstract some of the complexity of distributed systems helps keeps the students motivated and engaged in their research project. Furthermore, taking advantage of local and external resources provide students with increased opportunities and while seeking interdisciplinary collaborations helps students realize the role computer science can play in solving real world problems.

Acknowledgment

This work was supported in part by the Office of Research and Sponsored Programs (ORSP) at the UW - Eau Claire. Portions of this research was performed using resources and the computing assistance of the UW-Madison Center For High Throughput Computing (CHTC) in the Department of Computer Sciences.

References

[1] A. J. Hey, S. Tansley, and K. M. Tolle, *The fourth paradigm: data-intensive scientific discovery*. Microsoft Research Redmond, WA, 2009.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[3] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, no. 4, pp. 16–25, Dec. 2007.

[4] D. A. Patterson, "Computer science education in the 21st century," *Commun. ACM*, vol. 49, no. 3, pp. 27–30, Mar. 2006.

[5] S. K. Prasad, A. Chtchelkanova, S. Das, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A. La Salle, R. LeBlanc, M. Lumsdaine, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, and J. Wu, "Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing: core topics for undergraduates," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, ser. SIGCSE '11. New York, NY, USA: ACM, 2011, pp. 617–618.

[6] D. J. Ernst and D. E. Stevenson, "Concurrent cs: preparing students for a multicore world," in *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, ser. ITiCSE '08. New York, NY, USA: ACM, 2008, pp. 230–234.

[7] S. Rivoire, "A breadth-first course in multicore and manycore programming," in *Proceedings of the 41st ACM technical symposium on Computer science education*, ser. SIGCSE '10. New York, NY, USA: ACM, 2010, pp. 214–218.

[8] N. Jaeger and P. Bui, "To the Cloud and Back: A Distributed Photo Processing Pipeline," in *Midwest Instruction and Computing Symposium*, 2013.

[9] J. Westphal and P. Bui, "Scalable Distributed Image Transcoding using Python-WorkQueue," in *Midwest Instruction and Computing Symposium*, 2013.

[10] K. Ward, "Research with undergraduates: a survey of best practices," *J. Comput. Sci. Coll.*, vol. 21, no. 1, pp. 169–176, Oct. 2005.

[11] A. Koeller, "Experiences with student research at a primarily undergraduate institution," *J. Comput. Sci. Coll.*, vol. 20, no. 3, pp. 181–187, Feb. 2005.

[12] S. H. Russell, M. P. Hancock, and J. McCullough, "Benefits of undergraduate research experiences," *Science(Washington)*, vol. 316, no. 5824, pp. 548–549, 2007.

[13] S. Gordon, "Advancing computational science education through xsede," *Computing in Science Engineering*, vol. 15, no. 1, pp. 90–92, 2013.

[14] C. Peck, "Littlefe: parallel and distributed computing education on the move," *J. Comput. Sci. Coll.*, vol. 26, no. 1, pp. 16–22, Oct. 2010.

[15] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, A. Hey, and G. Fox, Eds. John Wiley, 2003.

[16] R. A. Brown, "Hadoop at home: large-scale computing at a small college," *SIGCSE Bull.*, vol. 41, no. 1, pp. 106–110, Mar. 2009.

[17] Python Programming Language, http://www.python.org/, 2010. [Online]. Available: {http://www.python.org/}

[18] 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.

[19] Blender, http://www.blender.org/, 2013. [Online]. Available: {http://www.blender.org/}

[20] "Dropbox," http://www.dropbox.com/, 2013. [Online]. Available: {http://www.dropbox.com/}

[21] A. Thrasher, R. Carmichael, P. Bui, L. Yu, D. Thain, and S. Emrich, "Taming complex bioinformatics workflows with Weaver, Makeflow, and Starch," in *Workflows in Support of Large-Scale Science (WORKS), 2010 5th Workshop on*, 2010, pp. 1 –6.

[22] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Operating Systems Design and Implementation*, 2004.

[23] F. P. Brooks, Jr., "The computer scientist as toolsmith ii," *Commun. ACM*, vol. 39, no. 3, pp. 61–68, Mar. 1996.