Challenges and Triumphs Teaching Distributed Computing Topics at a Small Liberal Arts College

Nathaniel Kremer-Herman
Department of Computer Science and Department of Engineering
Hanover College
Hanover, USA
kremer-herman@hanover.edu

***Abstract*—Introducing undergraduate students to key concepts of distributed computing has become almost essential as the world continues to embrace cloud-based solutions to daily problems and as research continues to grow in scale requiring distributed resources. Although distributed computing is an important part of the computer science curriculum, it can be difficult to introduce at some institutions. We explore some key challenges associated with introducing distributed computing into the computer science curriculum at a small, liberal arts college. We focus on an initial failure introducing a specialized distributed computing course *too soon* and relay the successes and failures experienced over a one year span of incorporating key distributed computing concepts across multiple systems-level courses. We discuss lessons learned from our first foray into teaching distributed computing and provide recommendations for new adopters of distributed computing curriculum based on our experiences.**

***Index Terms*—computer science education, cloud computing, distributed computing, distributed systems, liberal arts**

I. Introduction

As more of our daily personal and professional activities are shifted to cloud platforms, it has become increasingly important for undergraduate students in computing to understand the fundamental principles of distributed computing and systems (e.g. clusters, clouds, and grids) including the coordination, synchronization, configuration, deployment, and debugging of distributed systems. The likelihood of students entering careers requiring them to interact with distributed resources is quickly becoming absolute. A modicum of understanding about how these large-scale systems operate will become an essential body of knowledge in many students' careers.

However, it can be difficult to integrate distributed computing (DC) topics at an institution especially if there has never been a resident specialist in the field to establish an institutional body of knowledge and curriculum. We provide an experience report of such an occurrence at a small (roughly 1,000 undergraduate students) liberal arts college in the Midwest of the United States. This report covers a one year span introducing DC topics across the computer science curriculum within systems-level courses and the creation of a distributed systems course which had no previous institutional equivalent.

We discuss the key challenges of integrating DC topics within a small, liberal arts school. Due to the size of the institution, we address scaling challenges and how the inclusion of DC topics fit the focus of the computer science program. As there had been no previous resident expertise in the field, we also discuss the challenges of shepherding students up to an appropriate level of prerequisite knowledge for the distributed systems course. These challenges led to some notable successes and failures which may inform educators in similar positions how they may introduce (or enhance) the body of DC knowledge in their institution's curriculum.

II. Motivation and Related Work

The inclusion of DC in computing curriculum is a largely solved problem at many institutions. However, there are unique challenges for schools which have yet to adopt these topics into their curriculum. As time passes, it may be that schools which have not yet introduced DC will fall further behind in their ability to *begin* adopting these topics into their curriculum (as the educational space may progress without them, harming their computing programs).

At the host institution there had been no previous specialist in DC teaching at the school, and the size of the institution *and* department of computer science made inclusion *without* a specialist infeasible. Before the specialist was hired, the computer science department consisted of three members (two of whom had split duty between CS and the mathematics department). This small size and split focus made it logistically difficult to regularly offer courses not only in DC but in systems-level courses at large. The intention of this experience report is to serve as an example for similar institutions discovering the need for adopting DC topics yet facing troubles following through with implementation.

Computer science has been a part of liberal arts curriculum at many institutions for some time. The Liberal Arts Computer Science Consortium (LACS) is an organization which, since 1984 [1], has sought to provide support for teaching CS in a liberal arts context. LACS provides model curriculum, adapting the recommendations of ACM and IEEE to fit the liberal arts context [2] and additionally considers definitions of CS in the liberal arts, its role in the liberal arts, and the key aspects which make an outstanding CS program at liberal arts institutions. The existence of organizations like this demonstrates a commitment to effectively teaching computing at liberal arts schools without compromising in quality.

Work has been conducted investigating the underlying philosophy of computer science in the liberal arts [3]. Some

institutions view CS as an extension of mathematics, others from the natural sciences, and some from engineering. Noting that these perspectives are not entirely disjoint, computer science often acts as a discipline at the boundary of many fields of study which forwards the ideal of the well-rounded liberal arts student. Organizations like LACS provide curricular suggestions for CS programs such that they do not feel lesser than those offered by non-liberal arts institutions which allow for more focus on CS coursework over other disciplines. One work written by LACS members explicitly notes DC's place within the CS curriculum at liberal arts schools [4] though also shows the curricular suggestion from LACS that systems-level courses are considered electives. This recommendation closely (though not entirely) matches the degree requirements of the host institution's computer science department.

Teaching DC has factors which make it unlike other computer science courses. It necessitates discussing and accessing systems spanning tens, hundreds, and thousands of machines. However, given the complexities of these systems, it can be difficult for students to become accustomed to writing software which runs on multiple cores and multiple machines. The OnRamp [5] web portal was designed to introduce students to DC concepts while abstracting away some of the more painful experiences associated with installing and configuring distributed systems software. We provide our experience introducing students to DC *without* any abstracting tools like OnRamp. We note tools like this may have helped us avoid initial failures in our DC curriculum.

Even so, teaching a DC course to students who do not have much helpful prerequisite knowledge is not unknown. One work discusses details of an offering in DC at the second year of a university program [6]. The course is frequently ranked highly by students. The topics introduced are covered at a more introductory level and introduced to significant effect. Our experiences offering a DC course and integrating DC into our curriculum would have been better informed had we used this course as a starting template for teaching DC to students with little to no prior knowledge of the subject rather than making presumptions about our students' prior knowledge and capabilities to effectively use distributed resources.

Beyond work investigating how to teach DC courses, there are also resources available to aid instructors in adopting and enhancing DC curriculum at their institution. The Center for Parallel and Distributed Computing Curriculum Development and Educational Resources (CDER) [7] is an organization dedicated to developing curricula, gathering instruction materials, providing access to hardware and software useful for teaching parallel and distributed computing, and organizing events and training sessions. Their resources include a heterogeneous cluster of machines for educational use, educator-provided instruction materials, and curriculum guidelines.

CSinParallel [8] is another organization dedicated to providing instructors with resources for teaching parallel and distributed computing. Their resources include educator-provided web modules for teaching particular concepts, online discussion boards, and teaching workshops. These two organizations' resources can significantly reduce the course preparation load for instructors teaching a DC course for the first time.

We note that a key obstacle to providing a DC course at an institution is offering the course without a resident specialist. However, both CDER and CSinParallel would be especially helpful for institutions with no resident expert in distributed systems or a related field. Since the institution hired a specialist in distributed systems to teach the DC course discussed in this experience report, we note this key obstacle was avoided by the institution.

TABLE I
KEY STATISTICS FOR INSTITUTION

Inst. Size	Dept. Faculty	Num. Majors	Num. Minors
Approx. 1,000	4 (3 dual dept.)	27	8

III. DISTRIBUTED COMPUTING AT A SMALL INSTITUTION

Table I summarizes key information about the institution. This small, liberal arts college hosts roughly 1,000 undergraduate students. At the time of writing, 27 were computer science majors and 8 were CS minors across all graduating classes (with an average of 24 majors, a minimum of 11, and a maximum of 36 since the year 2000). The size of the department was average for the institution but considered quite small relative to other schools at 4 faculty members. Of those, 3 had dual appointments within other departments (2 being computer science/mathematics and the other computer science/engineering). Dual appointments were not unusual for the institution given its size.

The environment of the institution creates some key challenges for incorporating DC curriculum. We discuss factors of scale, the institutional body of knowledge, and the novelty effect of adding new content into established computing curriculum. We attempted to overcome these challenges over the course of a calendar year (across 3 semesters) and culminated in both noteworthy successes and failures which may prove insightful for educators facing similar challenges adding DC for the first time into their curriculum.

A. Key Challenges Inhibiting Curriculum Changes

One of the greatest challenges for incorporating any new curriculum at an institution is scale of faculty and students in the department. Without enough faculty, there is not much availability in teaching slots each term to deviate from existing curricula or the specializations of the faculty. Without enough students, it becomes difficult to justify to the institution offering potentially low enrollment courses in specialized topics (such as DC) when offering additional sections of higher enrollment classes is feasible.

We struggled with both facets of this challenge in particular. At only four faculty members, there was a lack of redundancy in specializations (i.e. each faculty member had their specialized courses with little to no overlap). This meant only one faculty member taught most systems-level courses and the DC course. Additionally, due to the scale of the institution

as a whole, there were no on-campus distributed resources, and there was little to no internal funding available to procure cloud resources for classroom use.

Nobody in the department's history had specialized in DC, and previously there had not been the bandwidth to offer the course. This meant the introduction of a DC course upon the hiring of a specialist led to the introduction of completely novel content into the institution's curriculum. The course had to match the institution's liberal arts model of education while also providing relevant experiences for students such that they could call upon their experience in their careers.

A related challenge to introducing DC was the existing focus of the CS program. The department had a strong, successful focus in software engineering. However, many systems-level courses focus more closely upon foundational principles and theory rather than software development. Introducing a course in DC as part of the systems curriculum meant introducing a course which deviated from the existing focus of the CS program meaning it would likely deviate from student expectations for a computer science course and would at best only indirectly advance the existing departmental objectives and learning outcomes.

Because a separate DC course was novel at the institution (compounded by the scale and focus of the program), this meant the first cohort of students to take the course would be under-prepared. Indeed, most students in the first offering of the course lacked useful prerequisite knowledge typically obtained from courses like Computer Networks and Operating Systems (neither were prerequisites for the offered course). A passing familiarity with how a single machine operates at the software level (Operating Systems) and the means by which multiple computers may interact (Computer Networks) are essential for achieving a thorough understanding in a standalone DC course. Otherwise, the course would have significant backtracking to introduce these concepts (thus leading to cutting out some DC content).

B. The First Distributed Computing Course Offering

The initial offering of the specialized DC course took place during a special month-long term in the academic year (associated with the winter semester). The course was called Cloud Computing, though we note this name was a misnomer used to make the course more identifiable by students, other faculty, and institutional administration. The sole prerequisite for the course was the introductory computer science course. Taking Data Structures (the second course in the major requirements sequence) was highly recommended. All enrolled students (being juniors and seniors) had taken Data Structures prior to the course. During this term, the DC course (with an enrollment of 8) was the only class the registered students took.

There were 4 lab assignments throughout the month with one due at the end of each week of the course. A cumulative final exam was given at the end of the month. The topics of the labs covered certain distributed applications: distributed video rendering, genomics using a cluster, large-scale text analysis, and genomics using serverless functions. Each lab required students to deploy software in a distributed context (whether on a cluster, cloud, or grid), using some dataset, to generate some final output they would turn in alongside some reflection questions. The video rendering assignment required students to establish a parallel video rendering pipeline capable of rendering a video composed of O(10,000) frames. The two genomics assignments used an 8GB dataset and performed the Smith-Waterman Alignment Algorithm in parallel. In the text analysis lab, students had to write software to perform some preprocessing on the text of a large number - O(100,000) - of public domain books.

TABLE II
DISTRIBUTED COMPUTING SPECIALIZED COURSE TOPICS

Topic	Papers Read	Assignment
Distributed Systems Landscape	[9]–[12]	X
Distributed Systems Principles	[13]	X
Grid Computing & HTCondor	[14]	✓
Scientific Workflows	-	✓
Execution Engines	-	✓
MapReduce Paradigm	[15], [16]	X
Hadoop Ecosystem	[17], [18]	X
Distributed Data Analysis	[19]	X
Serverless Computing	[20]–[22]	✓
Scaling & Consistency	[23], [24]	X
Internet of Things	-	X
Coordination & Configuration	[25]	X
Performance & Troubleshooting	[26], [27]	X
Cloud Gaming	[28]	X

Student readings were a combination of academic papers and online videos. Students were required twice per week to submit copies of their notes taken for given readings (which went toward a participation grade). Table II shows the list of course topics, references to academic papers used as readings for that topic, and whether there was a dedicated lab assignment for the topic.

The course served as an introduction to DC topics. Since it was only a month long, it was infeasible to go into the depth possible within each topic as in a traditional semester-long offering. We note later that the scaled-back depth covered was perhaps still too much. The course learning objectives were:

- Communicate clearly the core concepts of distributed computing.
- Analyze the differences between distributed system architectures and implementations.
- Utilize three separate distributed systems to perform large-scale computations.
- Implement four distinct distributed applications.

C. A Summarized List of Lessons Learned

We learned many lessons from both the roadblocks and successes offering the DC course. Before discussing each in depth, we provide a summarized list of key lessons learned:

- Commit to thorough preparation *before* offering a specialized DC course along the following considerations: student skill level and comfort for relevant tasks, prerequisite knowledge and placement of a DC course in the

department curriculum, and course fit with departmental objectives and student expectations.
- Develop internal resources to support students who may not be familiar with how to effectively use external software documentation and train adequate support for the instructor (e.g. faculty peers and IT staff).
- Consider public distributed systems resources (rather than potentially costly private clouds), especially for smaller or resource-constrained institutions.
- Integrate academic papers into DC course readings to promote student understanding while also providing content from its source.
- Automate setup and configuration steps of distributed system deployment (may help or hinder a DC course based on its focus).

D. Initial Failure of the Distributed Computing Course

Though the rollout of DC content occurred over a calendar year (within 3 semesters), the DC course was introduced as a first step of the project. This was a tactical blunder due to the lack of institutional prerequisite knowledge. Students' programming and problem solving skills were strong, having been honed through the software development focus of their prior coursework. However, they were unfamiliar with commonly used languages for systems-level programming (such as C), were unfamiliar with Linux and the command line, had little experience with real world systems outside their personal computers, and had little to no experience writing networked or multithreaded applications. Systems-level courses like Operating Systems and Computer Networks, where these skills are often refined, had not regularly been taught at the time the DC course was offered meaning few were eligible to have taken either course before taking the DC course. Computer Networks was offered for the first time in years in the winter semester just before the month-long special term, but not all students in the DC course were able to take it.

While having experience with *all* these topics is ideal, having familiarity with at least a couple was necessary for success in the course. Essentially, by introducing the DC course too early, we made the error of misjudging the body of knowledge the students had and overestimated their level of comfort in becoming rapidly familiar with new skills. Instead, it would have been best to have introduced the DC course in the final semester of the yearlong rollout as the culmination of laying the foundational skills and knowledge in other courses. Instead, we offered the course *first* without meaningful prerequisites and learned after the fact what knowledge was necessary for success in the course (as structured at the time) for future offerings. This led to the majority of students encountering stressful bouts during the course which were reported in the course evaluations. These stressors were compounded by a lack of preconfiguration for each assignment. The presumption that students were familiar and comfortable working with Linux machines in a terminal led to instances of students becoming lost when installing software, passing the proper command line options on execution, and navigating to their application's results and debug logs.

The mismatched assumptions of student skill in the rollout of the DC course was compounded by a lack of providing institutional support for student work in the DC course. While other systems-level courses had strong support from the institution's information technology team, the DC course was novel and as such had no institutional expert besides the instructor available to assist students. This led to a reliance on external distributed resources and relying upon their documentation and the official documentation of any software used on the systems. If a student was not well versed in how to read documentation and troubleshoot in a terminal session (which we note was not a skill explicitly introduced in the curriculum), they would get stuck in their assignment and would have to wait until the instructor was available to assist. This single point of local support created a bottleneck which became a point of frustration as students grappled with a set of brand new skills and often needed significant hands-on guidance (often simultaneously).

E. Notable Successes of the Distributed Computing Course

Although the first offering of the DC course encountered some troubles, there were a few notable successes as well. Foremost, students were given the opportunity to interact with live, scalable - O(1,000) - systems *without* having to pay any additional fees. Given the lack of institutional funding for execution time on a cloud platform, this was a considerable success. We used XSEDE (Extreme Science and Engineering Discovery Environment) [29] resources which consist of large-scale systems across the USA funded by the National Science Foundation and accessible for both research and educational purposes. Students used the Open Science Grid and the Stampede2 cluster to run bioinformatics scientific workflows, highly parallel text analysis, and video rendering at scale. They also executed a serverless genomics application using the public Jetstream cloud. While these systems were not those typically encountered in contemporary cloud contexts (e.g. Amazon Web Services, Microsoft Azure, or Google Cloud Platform), they provided an excellent first introduction for students in a safe, *no cost* environment with dedicated support personnel [30] from within academia.

The use of XSEDE systems meant students were able to run applications at varying scales. They were instructed to run toy applications before running applications at scale. Although each student had finite, pre-allocated execution time on each system, there was no financial disincentive against exploration. They were encouraged to make mistakes to figure out how their applications worked before running them at scale with real world datasets (such as a genomics workflow).

Another success in the DC course was the inclusion of foundational academic papers to complement running applications on large-scale systems. No other course in the CS curriculum at the institution included academic papers in their assigned readings, so this was a novel experience for the students. Though they did not initially have the skills to effectively get

the most out of reading each paper, they learned over time to pick out the key problem(s), solution(s), and analyses within each academic work and discuss them in class.

This had an anecdotal lasting effect for other systems courses students took after the DC course: when relevant they would cite a paper covered in the DC course unprompted. The students who took further systems-level courses would connect the concepts learned in the DC course and ask how a more foundational concept (such as network protocols or CPU scheduling) related to a paper they previously read. Although the inclusion of academic papers in course readings was a gamble, it anecdotally appeared to pay off in dividends in later coursework for the students. The positive feedback received from including academic papers inspired their occasional inclusion across the systems-level curriculum.

F. A Note on XSEDE

While we see using XSEDE to run large-scale applications as a success, it is worth pointing out that forgoing popular cloud platforms (e.g. Amazon Web Services, Microsoft Azure, or Google Cloud Platform) could be considered a notable failure of the DC course. Using XSEDE allowed students to experience firsthand the foundational concepts of DC covered in the course. However, it is quite unlikely that many students entering the cloud space after graduation would ever interact with the research-focused infrastructure offered by XSEDE.

The DC course would have been more useful to students in the short term had it incorporated at least one of the popular, contemporary cloud platforms. Students entering a profession using cloud resources would be more likely to assimilate quickly into their role since they would have familiarity with popular cloud platforms. However, given the scale of the institution and focus of the course, we found XSEDE to be a perfect fit.

G. Distributed Computing Topics in Systems-Level Courses

After the initial hardships experienced by offering the DC course too soon, we recognized a stronger foundation needed to be laid not only in DC topics but in systems-level curriculum as a whole. As such, we began to offer systems-level courses (particularly Operating Systems and Computer Networks) with higher frequency and began to include DC content across the systems-level courses.

This served to shore up students' lacking prerequisite knowledge needed for a standalone DC course. Additionally, the courses helped students gain experience and confidence interacting with Linux machines, the command line, and paradigms which exchanged abstraction for direct interaction with operating system libraries and hardware (such as coding in C and X86 assembly). Courses where DC content was included were: Computer Organization, Computer Architecture (taught within the engineering department), Computer Networks, Cybersecurity, and Operating Systems. DC focused activities were only included in these courses if they were relevant for particular classroom topics rather than being standalone course modules or topics listed in the syllabus.

TABLE III
DISTRIBUTED COMPUTING TOPICS ACROSS SYSTEMS-LEVEL COURSES

Course	Relevant Distributed Computing Topics
Computer Architecture	Mem. Hierarchy, Pipelining, High Perf. Comp.
Computer Networks	Protocols, Sys. Abstraction, Timing/Races
Computer Organization	Mem. Hierarchy, Pipelining, Scheduling
Cybersecurity	Protocols, Authentication, Timing/Races
Operating Systems	Multithreading, Scheduling, Virtual Machines

Table III shows topics in systems-level courses which are relevant to DC and were explicitly connected to distributed systems in class. For example, in Computer Organization, some attention was paid to including networked storage and remote accesses in the traditional memory hierarchy. Computer Architecture introduced students to the hardware powering supercomputers and datacenters. This culminated in students giving presentations on their choice of high performance systems of the past. In Operating Systems, discussions of multithreading and scheduling dovetailed cleanly into introductory discussions of race conditions in distributed systems and scheduling algorithms for clusters, clouds, and grids.

IV. DISCUSSION

Noting these successes and failures, we discuss the impact of the DC rollout at the institution. We provide the primary assessment of the DC course's success: course evaluations. We also discuss key lessons learned from the DC rollout and how other institutions first starting to incorporate DC into their curriculum may fare better.

A. Brief Assessment of the Distributed Computing Course

Our primary source of assessment for the DC course were the course evaluations. These were collected by the institution and consisted of a standardized online form used for all courses at the institution. Since the enrollment for the course was eight, we note that the results from the course evaluations are not of sufficient scale to make statistically significant statements about the effectiveness of the course or student learning. However, we demonstrate some of the positive and negative feedback which correspond to the key challenges outlined previously.

Table IV shows the prompts students were asked about the course (omitted are questions asked about the instructor). The prompts were associated with a Likert scale (1 being strongly disagree and 5 being strongly agree). We provide the count of respondents for each question and how they rated the course.

TABLE IV
STUDENT RESPONSES REGARDING DISTRIBUTED COMPUTING COURSE

Prompt	1	2	3	4	5
I am satisfied with the amount of coursework.	2	3	1	1	1
The course taught me valuable skills.	0	0	1	5	2
The readings helped me learn the material.	0	1	1	4	2
Assignments contributed to my understanding.	0	0	0	6	2
This course challenged me to do my best work.	0	0	0	7	1
Overall I rate this an excellent course.	0	0	1	5	2

The feedback on the amount of coursework reinforces our observation that we misjudged the level of comfort and expertise students had with a Linux terminal (and additionally is an indication the scope of assignments may have been too great for a month-long course). However, seeing the high ratings for the assignments contributing to student understanding indicates the chosen topics used for assignments may be spot on: deploying specific distributed applications spanning different domains. Additionally, the mostly positive feedback regarding the course readings aligns with our considering the use of academic papers a success of the DC course. While there were many student frustrations which caused us to rate the initial DC course offering as a failure, we note that student satisfaction was higher than we anticipated.

B. Fit into Liberal Arts Curriculum

Since the DC course was taught at a liberal arts college, it was important that it fit into the philosophy of the institution. As such, we chose to emphasize the theory of distributed systems. The use of academic papers as the primary source of readings was driven by this emphasis since classroom discussion would focus on the design and architecture of distributed systems over the span of recent decades rather than the particulars of the most popular contemporary platforms and systems. Two guest lectures from industry were also hosted during the course. One focused on the Internet of Things as applied to manufacturing and automotive contexts (outside the scope of the typical software developer experience) while the other focused on the daily experience of a senior software developer building cloud software. This breadth of focus (foundational architectures, DC as applied to diverse industry contexts, and some insight into contemporary cloud software development) helped the course more closely fit the liberal arts philosophy of the school.

If the institution were *not* a liberal arts school (and assuming funding was available), the focus of the course could have shifted from foundational theory to hands-on practice across a wider variety of systems. Using XSEDE was a boon for the course since the systems provided fit the goal of the course: a broad understanding of distributed systems. However, it would be hard to argue in favor of using XSEDE if one or more of the popular cloud platforms were more readily feasible to use since these platforms are *far* more likely to be encountered by graduates in their early careers. If the institution had less of a liberal arts focus and were more professionally oriented, this would have been the direction of the course. More frequent, smaller assignments sampling the *wide* variety of services offered by the popular cloud platforms would supplant the bulkier, week-long lab assignments used to explore the XSEDE systems.

C. Suggestions for New Adopters

Our greatest issues in the DC course stemmed from inaccurate presumptions of student experience and knowledge of relevant systems-level curricula. Had more time been spent planning when to roll out the DC course, we may have realized it would not have been possible for some eligible students to have taken Computer Networks (one of the most useful preparatory courses). Additionally, it was impossible for *any* eligible student to have taken Operating Systems (the other incredibly useful prior course) since it had not been offered in nearly four years. With this in mind, the first suggestion we can make would be to take stock of the current abilities of students eligible to take a DC course (perhaps through a brief in-class survey in a systems-level course).

Another suggestion we would make would be to determine upfront how a DC course fits within the current focus of the computer science program (and perhaps evaluate the fit of all systems-level courses). In hindsight, it would have more closely matched the departmental learning outcomes to have shifted focus from foundational principles of distributed systems and instead had students programming distributed applications (more closely related to the departmental focus on software engineering). This would have more closely matched the expectations of a typical computer science course for students and would have been more attuned to their preexisting body of knowledge. Depending on the focus of other CS programs, it may be more appropriate to offer a DC course more akin to a software development course rather than a more theoretical systems-level course.

Related to this, we found it useful to provide small experiences with DC across all systems-level courses. In part this was to better prepare students for subsequent offerings of the specialized DC course through brief asides during relevant course topics, DC-related guest lectures, and *brief* student-driven presentations about distributed systems. It also served as a reminder to students that we live in a highly interconnected world of distributed systems. The course of study at the systems-level often focuses on the details within a single machine, but they can also be abstracted to discuss machines working in concert to achieve a shared goal. This acknowledgement will hopefully lead to an increased awareness of DC among students at the institution whereas before they had little to no prior exposure before graduation.

Regarding the DC course in particular, our use of academic papers as course readings was well-received. However, it became apparent the number of readings led to increased stress and some burnout throughout the course. In our haste to cover a broad swath of DC topics, we risked cognitive overload for many students (especially since almost all had never read academic papers before the course). Since we found papers to be a valuable source of class discussion, we would suggest simply reducing the number of papers covered rather than cut them altogether. Some trickier papers may be better supplanted by guest lectures (perhaps by an author if possible) or videos about the topics. In a more traditional semester-long course it may also be more appropriate to use a textbook.

A key pain point for our students were the setup and configuration steps needed to complete assignments. Since many were not confident in their terminal skills, it would have led to a better experience if these steps were automated where possible. Providing scripts to install and configure software

and systems would have allowed students to see the steps in action without having to struggle completing them on their own. Depending on the focus of the course, this struggle may very well be a feature of the assignments worth experiencing rather than circumventing. However, if the goal is to have students effectively run distributed applications and write distributed software, automating these steps may be critical in reducing the time between them starting the assignment and them beginning to run applications at scale.

A final recommendation we can make based on our experience would be to teach DC concepts to someone other than the DC course instructor (*before* teaching the specialized course) who can realistically provide support for students. This may be a teaching assistant, support staff, or another faculty member. While difficult at the scale of the host institution, somewhat larger schools may have an easier time finding a volunteer (or compensating a participant) to help support students in the DC course with troubleshooting and advice when the instructor is busy or unavailable. Distributed systems are complex, and the troubleshooting process is, in many ways, fundamentally unlike debugging a program running on only one machine. For our students, troubleshooting a distributed system was an entirely unknown prior experience. Having extra assistance during the course would have been incredibly helpful when it came to coaching students how to effectively troubleshoot their applications.

V. Conclusions

Our introduction of DC to the body of knowledge at a small, liberal arts college led to noteworthy successes and failures. By offering a standalone DC course *before* most students had enough prerequisite knowledge, we uncovered flaws in our assumptions of student self-confidence and their body of knowledge in systems-level topics. Although the initial DC course resulted in many teachable failures causing us to reassess our assumptions, it also provided some successes such as the effective use of academic papers (which some students referenced in later systems-level courses), the use of XSEDE resources, and the feedback that students *did* have rewarding experiences running distributed applications at scale.

We believe our experiences can serve as inspiration for other institutions who have not yet adopted DC topics in their curriculum. As the prevalence of distributed systems approaches ubiquity, it is imperative our students be introduced to these concepts before graduation. Our recommendations address pitfalls encountered in our rollout of distributed computing into the computer science curriculum. We hope they help other institutions avoid the same pitfalls and have a smoother rollout of distributed computing topics.

VI. Acknowledgements

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562. We thank Mats Rynge for his assistance which was made possible through the XSEDE Extended Collaborative Support Service (ECSS) program.

References

[1] K. B. Bruce, R. D. Cupper, and R. L. S. Drysdale, "A history of the liberal arts computer science consortium and its model curricula," *ACM transactions on computing education*, vol. 10, no. 1, pp. 1–12, 2010.

[2] L. A. C. S. Consortium, "A 2007 model curriculum for a liberal arts degree in computer science," *Journal on educational resources in computing*, vol. 7, no. 2, pp. 2–es, 2007.

[3] H. M. Walker and C. Kelemen, "Computer science and the liberal arts: A philosophical examination," vol. 10, no. 1, pp. 1–10, 2010.

[4] D. Baldwin, A. Brady, A. Danyluk, J. Adams, and A. Lawrence, "Case studies of liberal arts computer science programs," *ACM transactions on computing education*, vol. 10, no. 1, pp. 1–30, 2010.

[5] S. S. Foley, D. Koepke, J. Ragatz, C. Brehm, J. Regina, and J. Hursey, "Onramp: A web-portal for teaching parallel and distributed computing," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 138–149, 2017, keeping up with Technology: Teaching Parallel, Distributed and High-Performance Computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517300205

[6] R. Sakellariou, "Experiences with teaching a second year distributed computing course," in *Euro-Par 2016: Parallel Processing Workshops*, F. Desprez, P.-F. Dutot, C. Kaklamanis, L. Marchal, K. Molitorisz, L. Ricci, V. Scarano, M. A. Vega-Rodríguez, A. L. Varbanescu, S. Hunold, S. L. Scott, S. Lankes, and J. Weidendorfer, Eds. Cham: Springer International Publishing, 2017, pp. 28–37.

[7] NSF/IEEE-TCPP. CDER center. NSF/IEEE-TCPP. [Online]. Available: https://tcpp.cs.gsu.edu/curriculum/?q=node/21183

[8] CSinParallel. Parallel computing in the computer science curriculum. NSF. [Online]. Available: https://csinparallel.org/index.html

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[10] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[11] N. Savage, "Going serverless," *Communications of the ACM*, vol. 61, no. 2, pp. 15–16, 2018.

[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[13] S. C. Kendall, J. Waldo, A. Wollrath, and G. Wyant, "A note on distributed computing," 1994.

[14] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the grid," *Grid computing: Making the global infrastructure a reality*, pp. 299–335, 2003.

[15] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[16] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[18] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on management of data*, ser. SIGMOD '08. ACM, 2008, pp. 1099–1110.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

[20] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[21] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2017, pp. 405–410.

[22] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 1063–1075.
[23] E. Brewer, "Cap twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.
[24] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.
[25] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "Zookeeper: Wait-free coordination for internet-scale systems," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.
[26] N. Kremer-Herman, B. Tovar, and D. Thain, "A lightweight model for right-sizing master-worker applications," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 504–516.
[27] N. Kremer-Herman and D. Thain, "Log discovery for troubleshooting open distributed systems with tlq," in *Practice and Experience in Advanced Research Computing*, 2020, pp. 224–231.
[28] W. Cai, F. Chi, X. Wang, and V. C. Leung, "Toward multiplayer cooperative cloud gaming," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 70–80, 2018.
[29] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "Xsede: Accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, Sept.-Oct. 2014. [Online]. Available: doi.ieeecomputersociety.org/10.1109/MCSE.2014.80
[30] N. Wilkins-Diehr, S. Sanielevici, J. Alameda, J. Cazes, L. Crosby, M. Pierce, and R. Roskies, "An overview of the xsede extended collaborative support program," in *High Performance Computer Applications - 6th International Conference, ISUM 2015, Revised Selected Papers*, ser. Communications in Computer and Information Science, vol. 595. Germany: Springer Verlag, 1 2016, pp. 3–13.