J. Parallel Distrib. Comput. 118 (2018) 166–176

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A course on big data analytics

Joshua Eckroth

Stetson University, DeLand, FL, United States

HIGHLIGHTS

- A course designed for undergraduate junior and senior computer science students.
- Curriculum is project-oriented and focuses on modern tools such as Hadoop and Spark.
- Course projects are explained in detail.
- On-premise and cloud computing infrastructure are evaluated and cost-compared.

ARTICLE INFO

Article history:
Received 1 August 2017
Received in revised form 28 January 2018
Accepted 26 February 2018
Available online 5 March 2018

Keywords:
Curriculum
Undergraduate education
Big data
Cloud computing

ABSTRACT

This report details a course on big data analytics designed for undergraduate junior and senior computer science students. The course is heavily focused on projects and writing code for big data processing. It is designed to help students learn parallel and distributed computing frameworks and techniques commonly used in industry. The curriculum includes a progression of projects requiring increasingly sophisticated big data processing ranging from data preprocessing with Linux tools, distributed processing with Hadoop MapReduce and Spark, and database queries with Hive and Google's BigQuery. We discuss hardware infrastructure and experimentally evaluate the cost/benefit of an on-premise server versus Amazon's Elastic MapReduce. Finally, we showcase outcomes of our course in terms of student engagement and anonymous student feedback.

1. Introduction

In 2015, Stetson University introduced a Data Analytics interdisciplinary minor for undergraduate students. A new four credit hour course focused on *big data* analytics was created to serve as an elective for this minor as well as an upper-level elective for computer science majors. This report documents the curriculum, infrastructure, and outcomes of our big data analytics course.

The landscape of big data tools, techniques, and application areas is vast [15]. Popular tools include the software frameworks Hadoop, Spark, and Hive, as well as cloud-based services like Google's BigQuery. Popular techniques include MapReduce, relational and non-relational data stores, and computations represented as directed acyclic graphs of functions. Sometimes, these tools and techniques include processing overhead that results in slower processing on small data than traditional approaches such as a single-threaded application. Thus, it is important that students understand not only how but also when and when not to use big data technology.

Our course is designed to give students realistic, hands-on practice with big data. The course is project-focused and the projects are organized so that later projects require more sophisticated data processing techniques. Our projects and the parallel and distributed computing topics that they address are listed in Table 1. Each project is described in more detail in the corresponding sections of this report. Our projects may change in the future as the popularity and appropriateness of various technologies change over time.

Students are given access to a "virtual cluster" running on a moderately-sized server as well as cloud computing resources. Each project employs a publicly-available dataset and challenges students to answer simply-stated queries about the data. While the queries may be simple, the steps to extract the answers from the data may be considerably more complex. We emphasize to students that the data processing steps are a means to an end and the ultimate goal of each project is to provide clear, insightful answers to the project's queries. Students are required to produce a short report documenting these answers with supporting statistical analysis and plots as appropriate. They are expected to explain their findings with language intended for non-experts and to hide all implementation details in an appendix separate from the report.

This course is designed for juniors and seniors who have completed prior courses in programming (Java, C++, and Python). Our students have also used Linux in prior courses, and as such the

E-mail address: jeckroth@stetson.edu.

https://doi.org/10.1016/j.jpdc.2018.02.019
0743-7315/

Table 1
Outline of projects in our course, with corresponding sections in this report as well as topics relating to parallel and distributed computing (PDC).

Section	Project task	PDC topics
3.1	Summarize Backblaze data	Performance analysis
3.2	Merge and analyze StackExchange datasets	Distributed computing; MapReduce; Hadoop; Hive
3.3	Visualize NYC taxi trips	Cloud computing; BigQuery; Stream processing; Storm
3.4	Detect and count stars	Spark; Threads; GPU computing
3.5	Capstone project	(Variable)

big data course makes extensive use of Linux on-premise and in cloud environments. We do not require that students have a strong statistics background, thus our projects involve only a small degree of statistics such as correlation analysis and hypothesis testing.

Teachers who wish to use some or all of the projects described below should be very comfortable with Linux and learning new processing frameworks written in Java, C++, and/or Python. Typically, the popular frameworks support multiple languages equivalently, so one may use Python, for example, for all projects. Prior experience with Hadoop, Spark, Storm, and OpenCV will help for the projects described.

We used both a virtual cluster and Google's cloud computing environment. An effective big data course needs some kind of cluster environment, virtual, physical, or cloud-based. A university or college's IT staff may be able to support these needs. Our virtual cluster technology is made available as a set of Varant and Ansible configuration scripts.[1] Additionally, education grants are available from Google, Amazon, and Microsoft for cloud computing. In Rabkin et al.'s [22] experience, such grants are able to cover the computing needs of an entire course.

This course is the only big data analytics course available to our computer science majors. We are not proposing at this time to integrate PDC topics in first- or second-year programming courses, as explored by Bogaerts [2] and Grossman et al. [12], or throughout the curriculum as done by Swarthmore College and reported by Newhall et al. [19], among other institutions. We agree with these other authors that an integrated curriculum has advantages for student learning and skill development. However, we did not have the opportunity at our university to change the curriculum of multiple courses. Instead, we introduced an upper-level course. This report is targeted at educators who are interested in introducing a single course that covers big data analytics.

This report builds off prior works by the author [8,9], which document the same big data analytics course at our institution. Specifically, in the present report, the course projects are described in more detail with learning objectives, "résumé notes" to be adopted by the students for their résumés, and project variations including experimental comparisons of different technology choices to solve the same project. The present report includes an updated experimental evaluation of a virtual cluster environment versus a cloud computing environment. Finally, the present work includes more recent student feedback and outcomes.

Our big data analytics course differs from some other courses that similarly focus on big data processing paradigms and tools rather than low-level methods like MPI. DePratti et al.'s course on big data programming at Eastern Connecticut State University [6] most closely matches our course. Their institution is comparable in size to our own and also considers itself a liberal arts college. The authors determined that the best way to introduce big data tools and techniques to their students was to introduce a single upper-level project-focused course covering Hadoop, MapReduce, and Spark. However, unlike our course, they do not focus on using big data technology to provide insight to non-experts. Their course also requires that students use their own computers to simulate a cluster environment thus limiting the apparent usefulness of big data techniques.

Ngo et al. [20] describe a Hadoop MapReduce module in a broader parallel and distributed computing course. The assignments related to this module focus on developing MapReduce jobs to answer to a query such as "which job in Google Data Center's system log has the most resubmissions?" Their assignments do not appear to include broader analysis or visualization components. Additionally, their MapReduce module does not cover other big data technologies such as Spark and Hive as we do in our course. Rabkin et al. [22] similarly focus on computation without broader analysis or communicating findings. However, unlike Ngo et al., they include projects and labs that compare the performance characteristics of algorithms implemented in C and Hadoop and run locally and on a cluster. Likewise, in this report we describe alternative implementations for each of our projects and experimentally evaluate the performance of several of these alternatives. We also describe how students may be challenged to experimentally evaluate their intuitions about which data processing techniques are most performant.

Mullen et al. [18] describe a MOOC course that focuses on the technical aspects of building and executing parallel and distributed processing jobs. Like our course, they emphasize practical skill development with programming activities. However, possibly due to the limitations of evaluation in MOOC courses or possibly by design, their course curriculum does not appear to ask students to write reports that summarize their data processing outcomes. Instead, their course focuses on the theory and mechanics of parallel and distributed computing rather than on big data analysis for the sake of providing insight to non-experts.

The rest of this report is organized as follows. Section 2 gives a brief overview of the course curriculum and schedule. Section 3 details five projects that cover a range of parallel and distributed computing tools and techniques and make use of public datasets. We require that students provide a rationalization for their choice of tools and techniques for their capstone project using a "decision matrix" detailed in Section 4. The hardware and software infrastructure used to support our course is detailed in Section 5, including benchmarks comparing on-premise and cloud-based computing resources. Section 6 summarizes outcomes of the course and the students' anonymous feedback. Finally, Section 7 offers recommendations and concluding remarks.

2. Curriculum

Our big data analytics course has three high-level learning objectives:

- Determine whether a data analysis task requires "big data" tools and techniques, or can be done with traditional methods, and identify appropriate means to perform the analysis.
- Design and build processing pipelines to efficiently extract relevant data.
- Communicate the outcomes of data analysis with convincing arguments involving, as appropriate, text, tables, and plots.

For the purposes of distinguishing big data from small data processing tasks, we define big data as follows, adapting a definition from Russom [23]:

> A data analysis task may be described as "big data" if the data to be processed have such high volume or velocity that *more than one* commodity machine is required to store and/or process the data.

[1] https://github.com/StetsonMathCS/hadoopvirtualcluster.

In regard to our first learning objective, we define a "commodity" machine as one that may be found in a typical small business data center. By today's standards, such a machine may contain a 2–8 core CPU, 8–16 GB memory, 1–4 TB hard disk, and gigabit ethernet. These numbers are vague approximations but sufficient to make the point: big data is at least several TB of data volume or millions of records per second data velocity.

Big data tools like Hadoop and Spark are designed to distribute a computation across more than one machine. They may be used on a single machine with either a single worker thread, a virtual cluster (as we detail below in Section 5) or, in the case of Spark, multiple worker threads on a single machine (though Spark may also be used on a Hadoop cluster). These tools introduce overhead in time and memory for managing the computation, even when the computation is not necessarily distributed but only runs on a single machine. Thus, if one needs to process small data rather than big data, i.e., data that may be processed on a single machine, traditional techniques like multithreaded algorithms may result in more efficient processing. Our first learning objective addresses this point so that we ensure students choose their data processing tools appropriately and do not approach every data processing task with big data technology unless such technology will actually result in more efficient processing or is otherwise absolutely necessary due to the size of the data. Students are required to fill out a decision matrix and argue their choice of technologies for their capstone project. This decision matrix is detailed in Section 4.

Our second learning objective addresses our intent to give students practical, hands-on experience with big data processing. As much as possible, we use big datasets and ask meaningful questions about the data. Depending on the infrastructure available for a big data analytics course, the size of the datasets may not constitute "big data" as defined above. Some degree of illusion may be required in order to simulate distributed data processing, as we do with our virtual cluster.

Finally, our course emphasizes that the role of data processing is to provide insights, often to non-experts. For each project, we require that students produce a report of their findings in non-technical English and enhanced with plots and tables as appropriate.

Students are evaluated primarily based on their projects (90% of the grade overall). We do not have exams in our course. The last 10% of their grade is based on two required class demonstrations of two tools or techniques not covered in the course lecture material but still relevant to the course topics or projects. The students' notes from their demos are collected and added to the course website in the form of "cookbook" examples. For example, one cookbook example shows how to query a SQL database with R code, while another shows how to run a MapReduce job over all files in a directory that satisfy a certain filename pattern. These demos encourage students to explore the topics on their own and allow them to benefit from the findings of their peers.

2.1. Schedule

Our course spans thirteen weeks. While there are many big data tools available, and more each year, we can only expect students to understand and practice with a small number of them. During the most recent offering of our course, we covered Hadoop and HDFS, MapReduce, Spark, Google's BigQuery, and supporting tools R, ggplot, and OpenCV. In a previous offering, we covered Hive and Mahout but did not include Spark. As the number of big data tools grows, their relative popularity evolves as well. For example, in future offerings, it is likely we will no longer include Hadoop's version of MapReduce given its relative unpopularity vis-à-vis Spark, and we plan to consider introducing a stream processing technology like Storm, detailed in the NYC taxi trips project (Section 3.3).

Table 2
Course schedule.

Weeks	Topics	Project
1–4	Data cleaning, statistical hypothesis testing, performance analysis	Summarize Backblaze data
5–6	Hadoop architecture, HDFS, MapReduce paradigm, Hive	Merge and analyze StackExchange datasets
7–8	Relational databases and SQL queries, Google BigQuery	Visualize NYC taxi trips
9–10	Computations represented as DAGs, Spark, GPU computing, OpenCV	Detect and count stars
11–13		Capstone project

The course schedule from our most recent offering is shown in Table 2. In the early weeks, we cover supporting tools R and ggplot for help summarizing, analyzing, and visualizing small data produced by big data processing. During this time, students are required to complete several small assignments to practice with R and ggplot. These assignments are not detailed in this report. The remainder of the course is concerned with five major projects, each of which is described in turn in the next section.

3. Projects

In this section, we describe the five projects included in our most recent offering of the course. In different offerings, the projects may change depending on availability of new datasets and tools. Students were allowed to pair together on each project if they wished. All but the capstone project spanned two weeks of class time; the capstone project spanned three weeks. At the end of the course, we provided example résumé notes for each student to adapt in their own words and include on their résumé. We believe that these notes help students realize that these projects are realistic and the skills they develop during the course may be useful in their professional lives.

For each project, we list the related parallel and distributed computing (PDC) topics, describe the project in detail, list learning outcomes, discuss possible variations, and provide the aforementioned résumé note.

3.1. Summarize backblaze data

PDC Topics: Performance analysis

Backblaze, a cloud storage service, maintains data warehouses with thousands of hard drives. They have released daily hard drive metrics since 2013 [1]. These metrics include drive information such as manufacturer, model, and capacity, ninety different S.M.A.R.T. metrics (Self-Monitoring, Analysis and Reporting Technology) for each drive, and a boolean value indicating if the drive failed on the given day. The dataset is made available in daily CSV files, totaling 14 GB with 60 million records.

The students are asked to respond to the following prompts about the data:

- "Some manufacturer produced particularly bad drives of a certain size (in TB). Find this manufacturer by graphing annual failure rate (AFR) for all manufacturers and drive sizes".
- "Drives of a certain TB size-range (e.g., 3–4 TB) have a higher proportion of failures than other sizes".
- "The hard disk S.M.A.R.T. metric #187, which measures read errors, is strongly positively correlated with disk failure".

Our learning objectives for this project are as follows.

- Apply filtering to efficiently transform big data into small data.
- Analyze (small) data in R.

Table 3
Time and memory required to process and load the Backblaze data by various techniques. In all cases, the fread function from R's data.table package was used to read the CSV files.

Data preparation method	Time	Peak memory
Linux utilities: cut and pipes, then load into R	3 min	3.45 GB
CSVKit: csvstack & csvcut, then load into R	105 min	3.45 GB
Load all data into R, then filter	9 min	27.3 GB

Table 4
Comparison of R and MySQL for loading the filtered dataset and aggregation queries. The MySQL table included indexes where appropriate but was not otherwise optimized specifically for this use case.

Task	R	MySQL
Import filtered CSV data	2 min	11 min
Select sum(capacity) group by date	<1 s	4.5 min
Select avg(capacity)	<1 s	38 s

As the first project in our course, this project is designed to challenge students to answer fundamental data processing questions: (1) how are the data formatted and how can they be read? (2) Are all attributes of the data required or can the data be filtered first to speed up processing? (3) What is the best (most efficient) technique for preprocessing the data? Students have not yet been exposed to paradigms such as MapReduce or dataflow graphs, so they invariably attempt to load the entire dataset in memory. We provide the class with access to a server that has sufficient memory for the whole dataset for a few (but not all) students at a time. On more constrained environments like laptops, students would be required to find an alternative approach. Even so, preprocessing and loading the dataset may require a significant amount of time depending on the technique used, which we explain in more detail below.

Since the only S.M.A.R.T. metric that is relevant for this project is metric #187, the dataset may be filtered before further processing by eliminating all other S.M.A.R.T. metric values, thus reducing the dataset to about 3 GB. Students are not told about this optimization ahead of time. Rather, we hope they will discover the optimization on their own, or at least hit a roadblock attempting to process the full dataset using the tools that have been covered so far in the course (namely, Microsoft Excel, R, and Linux utilities). One option is to stack the daily CSV files into a single file using csvstack and then drop unneeded columns with csvcut, both tools from the CSVKit package, which is commonly discovered by students who search Google for CSV processing techniques. Another option is to use Linux utilities such as cut and pipes in a script. Finally, one could load the entire dataset into memory and then eliminate unneeded attributes. These options are compared experimentally in Table 3. Different students took different approaches, and after the completion of the assignment we reviewed these approaches and emphasized the importance of writing benchmarks to find the best solution.

Once the dataset has been filtered, it may be loaded into R or a traditional database like MySQL. Table 4 shows the time required to load the dataset into R and MySQL and thereafter perform data aggregation queries. It is clear that R has significantly less overhead since the data is kept entirely in memory. Functions from R's dplyr package were used for efficient operations. We do not ask students to decide on their own whether to use R or MySQL — we instead directed them to R for its statistical analysis features. However, we point out the findings in Table 4 for further evidence that tools exhibit different performance characteristics.

At this point in the course, students have not been exposed to database tools like Hive and Google's BigQuery. However, these distributed databases may be used to store and query the Backblaze data. A variation of this project could focus on performance characteristics of distributed vs. on-disk vs. in-memory databases, e.g., comparing Hive vs. MySQL vs. R. Of course, with only 14 GB of data, or 3 GB after filtering, such a comparison may not be representative of larger datasets that truly require distributed processing. Ultimately, as a first project, we elected to keep the performance analysis simple, just focusing on various ways to preprocess the data. A later project, "Visualizing NYC taxi trips" in Section 3.3, is better suited for distributed database technologies.

Résumé note.

Hard Drive Failure Analysis & Visualization (Linux, R, ggplot)
I analyzed 60 million records of daily hard drive performance statistics provided by Backblaze to discover daily and yearly total storage capacity, yearly failure rate per manufacturer, and a strong positive statistical correlation between the SMART 187 hard drive metric and imminent drive failure. I preprocessed the data with Linux utilities to extract a relevant subset, then analyzed and visualized these findings using R and ggplot.

3.2. Merge and analyze StackExchange datasets

PDC Topics: Distributed computing; MapReduce; Hadoop; Hive
StackExchange is a collection of topic-oriented websites in which users post questions and answers and earn "reputation" by such activities. Questions may include tags (i.e., keywords) describing the focus of the question. Recently, all user data and question and answer posts were archived [25]. The dataset consists of 116 GB of XML files.

This project corresponds to lectures covering Hadoop and MapReduce. Students are asked to build MapReduce jobs to answer the following queries:

- "What is the age distribution of users across all sites as a whole?"
- "What are the top 10 tags (by frequency of posts) for each site?"
- "How strong is the correlation between user reputation and number of posts (questions or answers) by that user?"
- "Do high-reputation users answer questions faster?"

Our learning objective for this project is straightforward:

- Merge two big data sets (StackExchange posts & users) via multiple MapReduce jobs.

This project is valuable as it requires students to make sense of the MapReduce paradigm and apply the paradigm to extract and merge data stored across numerous files and machines. The MapReduce paradigm makes merging large datasets somewhat problematic. In the StackExchange archive, user data is separate from post data. Each Map task receives one line of an XML file representing, say, a single post, so the Map task cannot connect user attributes like reputation and age with post attributes like tags and votes. During lecture, we explain the benefits of MapReduce and Hadoop's design for robust distributed computing, but students find it difficult to implement a solution in the MapReduce paradigm. Though less flexible than alternative approaches like arbitrary dataflow processing with Spark, we believe it is valuable for students to learn to solve problems in constrained paradigms that are specifically designed for distributed or high performance computing.

This project is much simpler to solve in Hive or another relational database tool. The only difficulty is importing the XML data into Hive. This can be accomplished with Hive's "serde" (serializer/deserializer) library and regular expressions to extract the relevant data out of the XML syntax.

We believe Hive is best introduced after students have practice with writing their own MapReduce jobs. With the MapReduce backend, Hive generates one or more jobs to execute a normal SQL query. It is informative to show students this translation process, at least at a high level, so that they understand that their own custom-written MapReduce jobs will (likely) have a similar structure and therefore their mental model of MapReduce is correct. Consider the SQL query "select site, count(*) as users, avg(age) as avgage from stackexchange_users group by site order by avgage desc limit 10". This query computes the number of users and their average age for each StackExchange site and returns only the top ten sites with the highest average age. For example, stackoverflow.com has 420,009 users with an average age of 29.6 years in this dataset, and the site with the oldest average age is genealogy.stackexchange.com. Hive creates two MapReduce jobs, or "stages" in Hive nomenclature, to compute this query. In the first stage, the map function returns pairs (site, row data) and the reduce function receives all row data per site and computes the count of rows (count of users) and average age. The stage 2 map function is the identity function and the reduce function limits the output to the top 10 records. For students knowledgeable in SQL, this Hive query is significantly easier to formulate than writing custom MapReduce jobs. Hive returns the answer in 77 s when running on our virtual cluster environment.

Although Hive makes query writing easy for students with a background in SQL, it may hide the intrinsic complexity of merging large datasets. Students must grapple with this complexity if they are asked to write their own MapReduce jobs. Depending on the available time, one may wish to modify this project so that students have to solve it both ways: with custom MapReduce jobs and Hive queries.

Résumé note.

> *Social Q&A Site Analysis & Visualization (Java, MapReduce, R, ggplot)*
> I developed a parallel and distributed data processing pipeline using Apache Hadoop and the MapReduce architecture to study various aspects of all StackExchange sites, including StackOverflow. The data consisted of 6.5 million users and 27.5 million posts stored across hundreds of XML files in an HDFS cluster. I found that most users are between 20 and 30 years old, users with more posts have higher reputation, and the most common tag on StackOverflow is "Java", among other findings.

3.3. Visualize NYC taxi trips

PDC Topics: Cloud computing; BigQuery; Stream processing; Storm

The New York City Taxi & Limousine Commission released Yellow taxi trip data that includes "fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts" [21]. The dataset includes about 1.4 billion records across 7.5 years. The dataset is formatted as CSV files and totals 213 GB.

This project asks students to simulate building a retrospective report of taxi trips over the last seven years. Students are shown a variety of plots that were prepared ahead of time on the same dataset. These plots show frequency of trips per hour of day and day of week, total passenger counts per month and year, average distance traveled, and starting and ending points overlaid on a map of Midtown Manhattan. Students were tasked with reproducing the plots without being told how to process the data or how to build the plots.

Our learning objectives, listed below, emphasize that the goal of this project is to transform big data into small data and produce sophisticated plots.

- Utilize cloud computing or stream processing to efficiently transform big data into small data.
- Use ggplot to produce sophisticated plots.

This project may be customized along two paths. First, the project may be treated as a batch processing job in which all the data lives on disk or in a cluster and must be aggregated to produce summary data for plotting. Second, the project may be treated as a simulated stream processing job in which the plots are updated in "real-time". Each approach is detailed in turn.

Batch processing. When treated as a batch processing job, students may use MapReduce, Hive, or other techniques to extract summary statistics for plotting. However, one may elect to introduce students to cloud computing services like Google's BigQuery, where the taxi data is already publicly available. This is the approach we took in our recent offering of the course. We used a Google Cloud Platform Education Grant to provide students free access to this service.

An alternative approach may include using Hive or Cassandra or Presto or some other relational database tool designed to handle big data. We benchmarked Hive on our virtual cluster with the taxi dataset and found that Hive computed the average cost per number of passengers in 10 min. It is worth noting that Litwintschik [16] found that Presto with Hive as a backend store produced an order-of-magnitude speedup compared to Hive without Presto on the NYC taxi dataset. BigQuery, on the other hand, computed the average cost query in two seconds. Presumably, Google employs multiple layers of caching for such queries, especially considering that the taxi data is pre-loaded into BigQuery as an example dataset and therefore likely popular among users experimenting with BigQuery.

A teacher must be cautious in providing students access to BigQuery or a similar service. Such services enable students to complete the data processing task by using SQL queries without necessarily understanding how the distributed processing works. We see two solutions to this potential problem. (1) Require that students implement similar processing using another tool, like Hive on a local or cloud Hadoop cluster, and compare performance with BigQuery; or (2) require that students upload the taxi data to their own cloud storage accounts before processing while ensuring they incur less than a specific maximum of cloud computing charges. The pedagogical benefit of (1) is clear because it helps students understand the advantages of Google's scale, but students still learn how to build their own similar though less performant solution. The benefit of (2) is that students will be required to develop a budget for cloud computing resources. This budget must include data storage as well as processing time. In realistic distributed and high performance computing tasks, cloud computing costs must be understood and bounded before such resources are utilized.

Stream processing. Although the NYC taxi data is available in whole, we can simulate a stream of data and treat the project as a stream processing job. Using Apache Storm, students may be asked to develop a topology that begins with the CSV input (using Storm's RawInputFromCSVSpout from the Predictive Model Markup Language (PMML) library) and ends with inserts into a relational database. The database may be queried periodically by a script that generates plots based on summary statistics produced in the stream processing pipeline.

The NYC taxi data includes about 10 million rides per month, or four per second. If the data were actually arriving in real time, one would likely wish to use a windowing technique to summarize groups of data over a specific time range. For example, a Storm topology may be designed that records counts of trips per hour and day of week, starting and ending points according to a discrete grid of city blocks, and so forth, for each window of, say, 250 records. After each window is processed, the summary statistics may be inserted into the database.

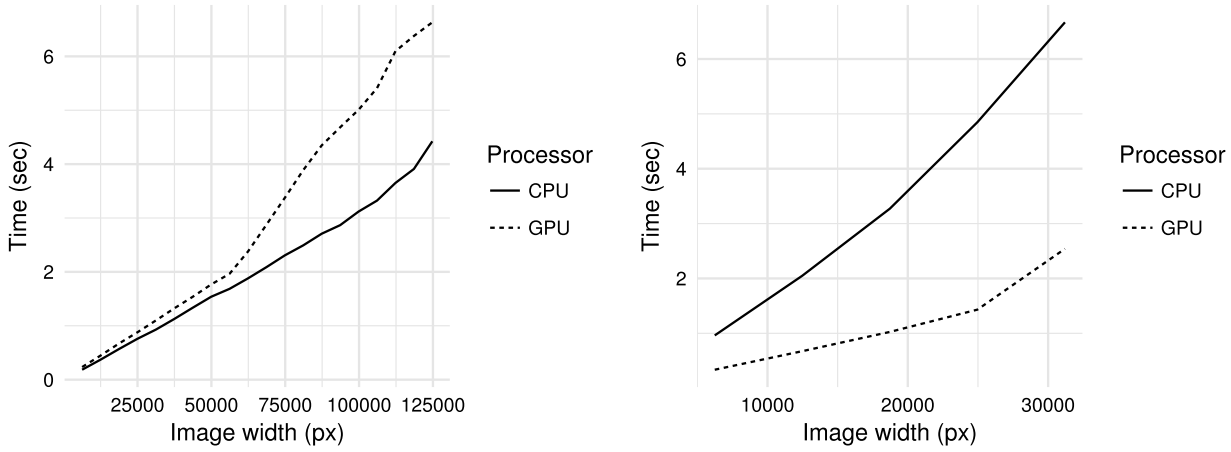

Fig. 1. *Left figure*: Time to apply convolutions (dilate, erode, invert) on star images of various sizes on both CPU and GPU; GPU is slower due to overhead of uploading and downloading the image to/from the GPU. *Right figure*: Time to apply convolutions and detect Hough circles on star images of various sizes; GPU is faster due to GPU-optimized Hough circle algorithm and no need to download the processed image from the GPU. Memory limitations of the GPU prevented testing larger images.

Résumé note. Naturally, this résumé note should be customized according to whether this project was approached as a batch processing or stream processing job.

> *NYC Taxi Usage and Revenue Analysis & Visualization (Google Cloud Platform, SQL, R, ggplot)*
> I performed a deep analysis on more than a billion rows of NYC Yellow Taxi data to visualize every taxi ride for morning and evening commutes drawn on a map of Midtown Manhattan. The most frequent origin and destination points were highlighted on the map and each trip was drawn as a line connecting the starting and ending points. I also produced plots representing the frequency of trips for each hour of each day of the week, average distance traveled each month of each year, and the overall revenue trend over the last decade.

3.4. Detect and count stars

PDC Topics: Spark; Threads; GPU computing

In this project, students are asked to find the largest 100 stars in a dataset of star images, where "largest" is defined as the greatest area in pixels in the images. The dataset consists of about 16,000 6000 × 6000 pixel images of the night sky from the Pan-STARRS1 data archive [11] totaling 535 GB and more than one-trillion pixels. This dataset is only a small subset of the Pan-STARRS1 archive that we downloaded to our server.

In order to detect the stars and measure their size, students must make use of a technique like Hough circle detection or contour detection from OpenCV [3]. Each image may be processed independently and hence in parallel if sufficient resources are available. The stars and their sizes from each image must be aggregated and sorted before filtering to the top 100. We encouraged students to use Spark to parallelize the task of processing each image and to aggregate and filter the results. We also encourage students to use our server's GPU, an Nvidia Titan Xp, for some or all of the star detection steps. Depending on how they combine CPU and GPU code, however, GPU usage may result in a less efficient solution. We explain our experimental results regarding GPU usage below.

Our learning objectives are as follows.

- Utilize Spark for parallel processing and OpenCV for image processing.
- Compare CPU vs. GPU processing time for various workloads.

To best detect stars in these images, the images should be subjected to a sequence of convolutions (dilate, erode, invert) before applying Hough circle detection or contour detection. Multiple images may be stitched together before processing to more efficiently make use of the GPU. OpenCV may be used for all of these steps. GPUs are typically more efficient than CPUs for convolutions, and OpenCV has an optimized implementation of Hough circle detection for GPUs (using CUDA). OpenCV does not yet have a GPU-specific implementation for contour detection. Thus, one must decide if image convolutions should be applied by the CPU or the GPU, and if contour detection (CPU-only) or Hough circle detection (GPU or CPU) should be performed. Students are encouraged to use the GPU wherever they can as long as it is the more efficient solution. After they have explored various approaches, we show them an experimental study of the tradeoff.

Shown in Fig. 1, we see in the left plot that performing only image convolutions but no Hough circle detection on the GPU is more costly than performing the same operations on the CPU due to the overhead of uploading and downloading the image to/from the GPU. However, using the CUDA-based Hough circle detector is significantly more efficient than the CPU-based Hough circle detector, as shown in the right plot. In this latter approach, one does not need to download the image from the GPU. Only a vector of detected circle coordinates is downloaded. On the other hand, the edge detection step of the Hough circle detection algorithm requires as much memory as the original image, so the maximum image size that can be uploaded to the GPU for Hough circle detection is limited by the amount of memory on the graphics card (in our case, 12 GB).

Besides the tradeoff between CPU and GPU image processing, one also must decide how to use multithreaded processing. In our case, the images were stored on disk in either a RAID array of hard disk drives (HDD) or a single solid state drive (SSD). As one increases the number of processing threads (a simple parameter in Spark), processing time is impacted differently depending on properties of the underlying storage. The performance of HDD vs. SSD is shown in Fig. 2. The figure shows that the SSD is significantly faster than HDD for image storage since the images are large and a significant amount of processing time is spent loading each image into memory. The figure also shows that for HDD storage, the number of Spark worker threads may impact the overall processing time as more threads cause more competition for the hard drives' (or RAID controller's) limited IO bandwidth and random access seek time. It is worth noting that if the GPU is used to process the images, then a kind of parallelism may be achieved by stitching multiple images together, but multithreaded operation may

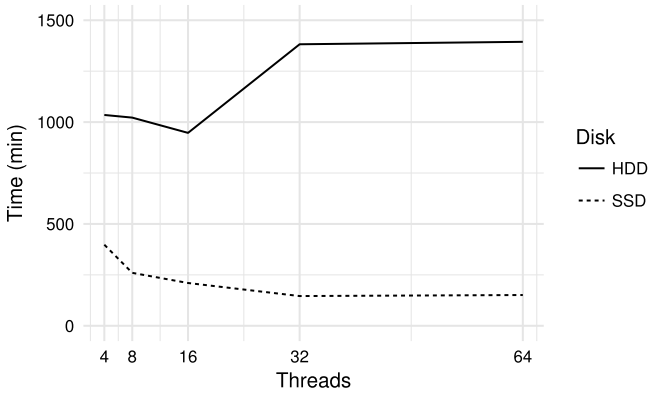

Fig. 2. Time to complete the computations of the star detection and counting project on our class server for different numbers of Spark worker threads with star images stored on a hard disk (HDD) or solid-state drive (SSD). The GPU was not used in this experiment.

provide no benefits since the GPU will be busy and its memory exhausted by the stitched image.

The star detection and counting assignment exposes multiple tradeoffs in processing efficiency with respect to computing hardware (CPU vs. GPU, HDD vs SSD), algorithms (Hough circle vs. contour detection), and parallelism (number of stitched images or number of threads). All of these concerns may be too much for students to grapple with in a single project. We advise teachers to pick a subset of these concerns with which to challenge their students depending on hardware available and their pedagogical goals.

Résumé note.

> *Star Identification from Wide-Field Astronomical Imaging (Apache Spark, OpenCV, Python)*
> I built a parallel processing pipeline using Apache Spark to analyze nearly 16,000 images totaling more than one trillion pixels provided by the Panoramic Survey Telescope and Rapid Response System. Using Python and OpenCV, I wrote an algorithm that eliminated noise from each image and detected each star. Ultimately, I discovered more than 10 million stars in the images and reported the "Right Ascension/Declination" coordinates for the top 100 brightest stars in the survey.

3.5. Capstone project

The final project requires individual students or pairs of students to formulate their own data analysis problem. This ensures the students are engaged in a novel task that they personally find interesting. They must find relevant datasets that total at least one million records or 1 GB of data. They must produce a report that hides all code and answers the original question with supporting evidence. In addition to this report, they must also complete the decision matrix described in Section 4. Finally, they must prepare a 5–10 min presentation that poses the question, summarizes findings, and offers a conclusion.

During the most recent offering of this course, students posed the following data analysis problems.

- "Do Hacker News posts about programming languages correlate with an increase in questions related to that programming language on Stack Overflow?"
- "Can we group Reddit users by certain attributes such as comment frequency, comment length, subreddit subscriptions, etc.?"
- "Do highly controversial subreddits experience more growth than less controversial subreddits? And are certain words in titles correlated with controversy?"
- "Can we predict Bitcoin price based on sentiment in Twitter messages that include the word 'bitcoin'?"
- "What are the most common video tags for the top YouTube channels?"
- "Generate a 2016 'annual report' for the San Francisco Bike Share program".

Our learning objectives for this project are straightforward.

- Formulate a question about big data that the student personally finds interesting.
- Collect data that helps answer the question.
- Formulate and implement an appropriate high performance data processing strategy after evaluating alternative approaches.
- Report findings in a written document and oral/visual presentation.

In our experience, only a fraction of the proposed capstone project problems are answered in the positive. For example, students found it is not the case that highly controversial subreddits experience more growth than less controversial subreddits. We emphasize to the students that they are not evaluated (for grading purposes) according to whether their problem statement can be answered positively, but instead whether they engaged in an appropriate data processing strategy, argued the appropriateness of their strategy with a decision matrix, and clearly and effectively presented their findings.

4. Decision matrix

In order to help students identify the best approach to a data processing task and avoid a common tendency to simply attack every problem with a trendy but inappropriate technology, e.g., applying Spark to every data processing task, we developed a "decision matrix" that allows students to connect tools and techniques with data processing subtasks. An example decision matrix for solving the star detection and counting project is shown in Table 5, and accompanying explanation follows below. Depending on the task, the rows and columns may list different tools and subtasks. The decision matrix is an enhanced version of a decision tree tool developed previously [7] to help determine if a data processing task required big data technology and whether batch or stream processing was required.

We only asked students to complete their own decision matrix for the final capstone project. We recommend instead that students complete a decision matrix for every project. We will take this approach in future offerings of the course. As described in each project above, there are often alternative approaches that have their own characteristics in terms of ease of implementation and efficiency.

The following explanation accompanied the marked-up table:

- Data acquisition: "The images were acquired using a Perl script (aka Linux tool) that iterated through URLs for each image".
- Exploratory analysis: "My exploratory analysis consisted of examining the file listing and writing sample code with OpenCV to process a single image. I did not use any big data tools for this. In another situation, like the Taxi data, I might well use big data tools (BigQuery) to perform exploratory analysis since BigQuery is quick for small queries".
- Plotting: "I plotted the stars with their RA/Dec coordinates in ggplot since ggplot is easier to code (for myself) than Excel".

Table 5
A sample decision matrix that relates features of a data analysis task with various tools. This decision matrix is filled in for the 'Detect and count stars' project. Explanations of each bullet may be found in the text.

Feature	Linux tools	Excel	R	BigQuery	MapReduce	Spark	OpenCV
Data acquisition	•						
Exploratory analysis	•						•
Plotting			•				
Distributed workers						•	
Numeric/string processing						•	
Image processing							•

- Distributed workers: "For processing all the images, I used Spark in a distributed manner to pipe image filenames to a C++ program".
- Numeric/string processing: "For sorting and finding the top stars (i.e., numeric/string processing), I also used Spark. R would have worked as well but would be less efficient since I would then have to load all of the star data into R after saving from Spark. By keeping all the processing in Spark, I avoided an expensive operation".
- Image processing: "Image processing was accomplished with OpenCV. In simpler cases, Linux tools from the ImageMagick suite might have sufficed".

The following section details the hardware infrastructure used to support our big data analytics course as well as an evaluation of alternative options.

5. Infrastructure

A course on big data or high performance computing strongly benefits from a hardware infrastructure that allows students to actually process big data or compute at high performance. Such a realistic infrastructure is not necessarily required, as one could focus more on theoretical aspects or use software designed for large systems on relatively small systems or virtual machines.

Large institutions typically have an advantage in providing large infrastructure for courses and research. At a small school like Stetson University, we felt the need to be innovative and to evaluate alternative approaches. In this section, we present several approaches and discuss their benefits and drawbacks. As infrastructure has an associated cost, we detail costs or estimates for each approach. Ultimately, we decided to make use of a moderately-sized server (32 processing cores, 128 GB of memory, about 30 TB of disk, Nvidia Titan Xp GPU) to host a virtual cluster environment using virtual machines for Hadoop-based projects, Spark in multithreaded "local" mode (thus avoiding the overhead of the virtual cluster), and the Google Cloud Platform for BigQuery. Our experiments below include Amazon Elastic MapReduce for a comparison of the virtual cluster and a cloud-based cluster. This section is an extension to our previous work on this subject [8].

5.1. Prior work

Recently, several researchers have documented various approaches to providing infrastructure for a big data analytics or similar course.

For example, St. Olaf College built both virtual and physical clusters using KVM and Beowulf platforms [4,13], respectively. They examine energy usage and the costs of cooling a room full of servers. In a more recent report by Brown and Shoop [5], the authors report that virtual clusters are less performant than physical clusters, but the penalty depends on the workload. They do not compare physical and virtual cluster costs.

Ngo et al. [20] report on a variety of physical clusters for teaching Hadoop. They neither address the costs of building these clusters nor analyze their performance characteristics. Kaewkasi and Srisuruk [14] address the other end of the hardware spectrum by documenting low-powered clusters and experimentally evaluating various configurations of ARM-based clusters. Although consuming very little power, their ARM-based clusters are also significantly less performant than traditional x86-64-based server architectures.

Rabkin et al. [22] report their experiences using Amazon Web Services rather than on-premise infrastructure. Because cloud computing is an ongoing expense, typically either students or the institution must cover these costs. Rabkin et al. obtained a grant from Amazon for the offering of the course documented in their work. They found that students utilized an average of $45 worth of cloud resources during the semester. Because code efficiency is variable, processing time can be hard to predict and thus cloud computing costs can be hard to predict.

Eickholt and Shrestha [10] recently looked at several physical cluster configurations in an evaluation that closely mirrors our prior work in this area [8]. As such, we will review their findings after discussing our virtual cluster environment.

5.2. Virtual cluster

When our big data analytics course was initially developed, our department already managed a moderately-sized server running Linux. We upgraded this server and repurposed it to support this course. Given that the server is only one machine and therefore cannot form a physical cluster on its own, we opted to develop a virtual cluster environment with LibVirt virtualization and the KVM/QEMU hypervisor. Students access the virtual cluster by connecting to the server with a common SSH client and submitting their code via typical Hadoop commands. Spark is also installed on this server but does not access the virtual cluster. Instead, Spark is configured to run in multithreaded "local" mode to best take advantage of the system's multiple CPU cores and memory.

We authored Vagrant and Ansible scripts to create a virtual cluster environment with any number of worker nodes in addition to Hadoop's Name Node, Resource Manager, and MapReduce Job History nodes. Our scripts update the server's local Hadoop configuration to point to these virtual machines for running jobs. The server's Apache webserver configuration is also updated to allow full access to the Hadoop web interfaces, silently proxying all requests to the appropriate virtual machine. These scripts are available online[2] and licensed under the MIT open source license. Our server is managed by the faculty and IT department and the virtual cluster scripts are maintained by the faculty. Other institutions have explored student-managed clusters [5,17] in both virtual and physical environments.

Our server without the GPU is equivalent to a Dell PowerEdge R430 with dual Intel Xeon E5-2640 processors, 128 GB memory, hard drives totaling 28 TB, and SSD drives totaling 3 TB. At the time of writing, the cost of these components totals about $8400. Assuming the system is operational for three years and uses 400 W of power continuously during these years, power costs total about

[2] https://github.com/StetsonMathCS/hadoopvirtualcluster.

$1000 at the US national average electricity rate. Thus, the total cost for three years is about $9400.

To estimate the cost per student per submitted job, we assume that each of 20 students submits an average of five processing jobs per project. This amounts to 500 jobs submitted per semester, or 3000 jobs over the lifetime of the cluster. Thus, the cost per job for our proposed server is $3.13. There are a lot of assumptions in forming this number, the most egregious of which is that the server will be replaced in its entirety every three years. Even so, these assumptions allow us a point of comparison with other platforms such as cloud computing.

Eickholt and Shrestha [10] built several physical clusters: Cluster A had 24 CPU cores, 64 GB memory, and 16 TB disk; Cluster B had 16 CPU cores, 128 GB memory, and 16 TB disk; and Cluster C consisted of retrofitted computers provided by students with different processor speeds, at least 32 GB memory overall, and 8 TB disk overall. Given prices of comparable hardware as of August 2016, the cost per job for these clusters is $3.08 for Cluster A, $1.68 for Cluster B, and $0.80 for Cluster C. The authors do not include an evaluation of the performance for these clusters, so it is not clear if these costs are reasonable. In any event, the costs are close to our own virtual cluster cost per job and the costs of cloud computing, detailed next.

5.3. Cloud cluster

Cloud computing has completely changed the landscape of acquiring computing resources. Now any individual or institution alike may acquire a cluster of arbitrary size in just minutes, and pay for only the time the cluster is running. Several large organizations offer cloud computing resources including Amazon, Google, and Microsoft. In this section, we evaluated Amazon Web Services, specifically their Elastic MapReduce (EMR) systems which automatically install and configure Hadoop, Spark, and related software.

Cloud computing resources are typically billed by the hour, and minutes are rounded up to the nearest hour. In our evaluation, we computed the cost of Amazon EMR without rounding up to the nearest hour. We also factored out the time involved in setting up the cluster (a process automated by Amazon), which typically required 5–10 min.

Fig. 3 shows the average time (plus/minus the standard error) required to perform a word frequency count on the 37 GB Westbury Lab Usenet Corpus [24]. The various cases compared along the *x*-axis are as follows:

- On-Premise Spark: Spark running on our server in multithreaded "local" mode, with varying number of worker threads. Hadoop is not involved and the virtual cluster was disabled.
- On-Premise Hadoop: Hadoop running in our virtual cluster environment. The 37 GB corpus was uploaded to HDFS on the virtual cluster before processing.
- EMR Spark: Amazon Elastic MapReduce running Spark on Hadoop with m3.xlarge EC2 instances (8 vCPU, 15 GB memory, 80 GB SSD storage). Different cluster sizes from 2 to 8 nodes were tested and their various timings produced the average and standard error bars shown.
- EMR Hadoop: Similar to EMR Spark, except Spark was not used, only Hadoop MapReduce.
- On-Premise Non-Parallel Shell Pipes: For comparison's sake, we also coded a word frequency count using the shell command:

```
time cat file.txt | tr ' ' '\n' | sort | uniq -c
```

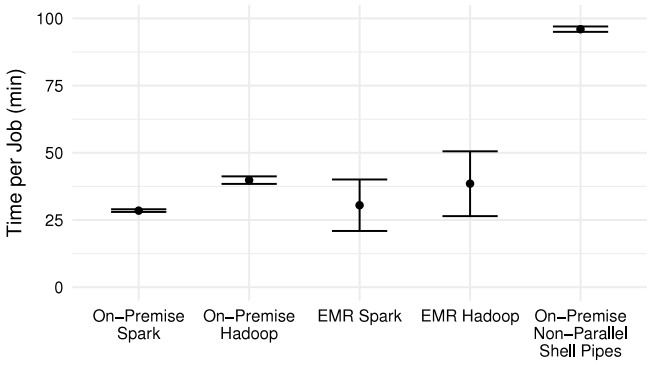

Fig. 3. Average time to complete a 37 GB word frequency count job for various hardware configurations. The categories along the *x*-axis are explained in the text.

We see from the figure that On-Premise Spark is the most efficient for the word frequency count task. The virtual cluster, indicated by On-Premise Hadoop, is still comparable to the cloud computing offering. Overall, Spark is faster than Hadoop (i.e., MapReduce), which underscores the reason for Spark's continued rise in popularity. On-premise or in the cloud, distributed computing performs better than the On-Premise Non-Parallel Shell Pipes. This is to be expected.

Amazon is naturally interested in selling higher performance cloud computing beyond the m3.xlarge systems with which we tested. In our experience, their costs are generally aligned with their performance. E.g., paying twice as much generally means the data processing task will finish twice as quickly. Our costs for the various Amazon EMR cluster configurations we tested stayed within a small range of $1.06 per job for Hadoop MapReduce and $0.85 for Spark on Hadoop. Spark costs less since it is more efficient on the same hardware. These costs compare favorably to our virtual cluster estimated cost of $3.13 per job. However, the numbers are difficult to compare because cloud computing costs are ongoing while the on-premise server is a fixed cost, modulo periodic upgrades, maintenance, and power costs.

Whether a big data analytics course should use an on-premise or cloud computing cluster depends on several factors unique to the institution. If a moderately-sized server or collection of free machines is already available, then establishing a virtual cluster environment will likely be the most cost-effective solution without significant penalty in performance, as illustrated by our own virtual cluster setup. However, if such resources are not available, cloud computing is likely the only reasonable choice. The choice is somewhat more complicated across a second dimension. On-premise hardware is mostly an upfront, fixed cost that the students do not usually bear. Cloud computing, on the other hand, is an ongoing cost that either the students must bear, or the institution, or covered by a grant which must be periodically renewed.

Cloud computing may well be the future of big data processing, so there is an advantage in introducing students to the mechanics of cloud computing. Even if the students do not have to pay for cloud computing resources themselves, a course could be designed that models the ongoing cost by giving students "credits" to spend on their data processing jobs. Students may start the course with a fixed number of credits and be asked to "budget" their credits across their data processing jobs. If a job is implemented poorly and crashes before producing output or runs too long, a student may exhaust far more credits than he or she was intending. The lesson might be painful but mirrors the reality outside of the classroom that many businesses and other organizations face when engaged in big data processing and analytics.

Table 6
Anonymous student ratings responding to various queries about the course. Possible scores range from 0 (“no apparent progress”) to 5 (“exceptional progress”). Data are shown for Spring 2015, $N = 17$ out of 20 enrolled students; and Spring 2017, $N = 8$ out of 12 enrolled students.

Prompt	Spring 2015		Spring 2017	
	Mean	Std. dev.	Mean	Std. dev.
Gaining factual knowledge	4.29	0.89	5.00	0.00
Learning fundamental principles	4.18	0.86	5.00	0.00
Learning to apply course material	4.18	0.78	5.00	0.00
Developing specific skills needed by professionals	4.19	0.96	5.00	0.00
Learning how to find and use resources	4.24	0.81	5.00	0.00
Acquiring an interest in learning more	4.18	0.92	5.00	0.00

6. Outcomes

The success of our big data analytics course may be viewed through multiple lenses. First, we find that students are engaged with the material. Across two semesters, students demonstrated multiple tools and techniques and ultimately contributed 17 cookbook examples related to R, 7 cookbook examples related to ggplot, 3 additional data sources, and 2 cookbook examples related to big data processing in Python. The course website, which was the sole source of course material in lieu of a textbook, was heavily used during the course. Between course offerings and up to the time of writing, the website is accessed about 80 times a month by returning visitors (presumably mostly students), with about 2 min spent looking at each page. The most popular pages are those that describe RStudio (our R code editor of choice) at 34% of page accesses, the website front page (26%), notes about setting up a development environment on Linux (7%), notes about Hadoop, HDFS, and MapReduce (5%), notes about ggplot (4%), and notes about Spark (3%).

Student anonymous ratings about the course are summarized in Table 6. It is clear from this table that students find the course to be valuable and that they are encouraged to learn more about big data processing.

We are able to get more insight about the students’ experiences by looking at their anonymous comments responding to the query, “Which aspects of this course helped you learn the most?”

- “[...] mostly the projects and how they were structured, they allowed us to take the content from the class and apply it”.
- “The hands on labs and extremely well put together projects”.
- “The practice with the various tools and software used in the class”.
- “Hands on experience with current trends in the industry”.
- “The projects that involved finding our own solution to a data analytics problem were the most challenging and I learned a lot from them. It was also nice to learn new techniques from the student presentations”.
- “Constantly added new and innovative content throughout the year”.
- “Slowly building up to bigger data sets made me understand big data clearly. Having multiple small and big project with different tools was the best part of this class. I can say that I have learned a lot from this course, and projects were really challenging and really fun”.
- “The bulk of the grades in this course come from our projects, which really allow us to get familiar with the tools we have learned in the class”.

These unfiltered comments conclusively show that the project-oriented curriculum appealed to the students. They appreciated the hands-on nature of the course and the gradually increasing level of difficulty of the projects.

It is worth noting two negative student comments. One student mentioned that the course should have more servers for cluster computing. As noted above, we used a virtual server environment, so performance sometimes suffered when many students were using the hardware at the same time. Our second offering of the course included new material that helped students make use of cloud computing resources. A significant benefit of cloud computing is that students do not compete for resources. Thus, we recommend teachers strongly consider cloud computing options and seek out educational grants offered by Google, Amazon, and Microsoft.

Second, one student mentioned that the projects required they learn a significant amount of material on their own that was not presented in lecture. Appropriately balancing lecture exposition and student self-guided discovery can be challenging when developing new curricula. We advise that teachers who adopt some of the projects reported here take time to show students the various APIs and libraries required to build Hadoop, Spark, and OpenCV projects.

Direct student outcomes of this course include the following: two students now work as data scientists at the Pacific Northwest National Laboratory (PNNL), two students work as data analysts at GEICO; one student participated in a Facebook summer internship in a data analytics role; one student participated in a Microsoft summer internship in a data analytics role; one student participated in a Florida Institute of Technology (FIT) summer research experience applying data analysis techniques; and one student participated in a Carnegie Mellon summer research experience applying data analysis and machine learning techniques.

7. Conclusion

This report detailed a big data analytics course that serves as an elective course for upper-level computer scientists at Stetson University. The course is project-oriented and engages students with realistic, hands-on practice using modern big data tools and techniques. Different options for supporting hardware infrastructure were explored and experimentally evaluated. Student feedback and academic and professional outcomes conclusively show that the course is a success and the high-level learning objectives were met.

A course like the one described here is necessarily continuously evolving. New technologies are introduced while others go out of favor. For example, in the first offering of this course, we did not cover Spark. Now, such an omission is unjustifiable. Likewise, we believe it is important for the projects to stay relevant and timely. As new big datasets are made available, projects should be updated to make use of those datasets. For example, at the time of writing, the NYC Yellow Taxi dataset is well known and several blog posts have been authored detailing different ways to analyze the data. The novelty of a NYC taxi data analysis project is rapidly waning, indicating that a different project might be more appropriate in the future.

We expect that cloud computing will become more ubiquitous than it already is and more appropriate for students to practice with than a small- or moderate-scale on-premise cluster environment. Cloud computing also allows instructors to develop a curriculum that includes larger data processing tasks than those illustrated in this report, e.g., analyzing the Google Books Ngrams corpus (2.2 TB) or the Common Crawl corpus (multiple petabytes). Both of these corpora are already available on the Amazon Web Services platform.

Ultimately, it is important that students not only learn the various data processing paradigms and tools but also develop a mature understanding of when to use certain approach to solve specific data processing tasks. Our various projects and decision matrix are designed to help students meet these goals. As larger datasets are made available and more varied and sophisticated paradigms and tools are developed, the importance of these goals is only reinforced.

Acknowledgments

A portion of this work was supported by the Brown Center for Faculty Innovation & Excellence at Stetson University. We also wish to acknowledge alumni Ou Zheng, Nathan Hilliard, and Katie Porterfield for generously donating SSDs for the server that hosts the virtual cluster.

References

[1] Backblaze hard drive data and stats. https://www.backblaze.com/b2/hard-drive-test-data.html.
[2] S.A. Bogaerts, One step at a time: Parallelism in an introductory programming course, J. Parallel Distrib. Comput. 105 (2017) 4–17.
[3] G. Bradski, A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc., 2008.
[4] R.A. Brown, Hadoop at home: Large-scale computing at a small college, in: ACM SIGCSE Bulletin, Vol. 41, no. 1, ACM, 2009, pp. 106–110.
[5] R. Brown, E. Shoop, Teaching undergraduates using local virtual clusters, in: IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2013, pp. 1–8.
[6] R. DePratti, G.M. Dancik, F. Lucci, R.D. Sampson, Development of an introductory big data programming and concepts course, J. Comput. Sci. Colleges 32 (6) (2017) 175–182.
[7] J. Eckroth, Foundations of a cross-disciplinary pedagogy for big data, J. Comput. Sci. Colleges 31 (3) (2016) 110–118.
[8] J. Eckroth, Teaching big data with a virtual cluster, in: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, ACM, 2016, pp. 175–180.
[9] J. Eckroth, Teaching future big data analysts: Curriculum and experience report, in: IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW, IEEE, 2017, pp. 346–351.
[10] J. Eickholt, S. Shrestha, Teaching big data and cloud computing with a physical cluster, in: Proceedings of the 48th Technical Symposium on Computer Science Education, ACM, 2017, pp. 177–181.
[11] H. Flewelling, E. Magnier, K. Chambers, J. Heasley, C. Holmberg, M. Huber, W. Sweeney, C. Waters, T. Chen, D. Farrow, et al., The Pan-STARRS1 Database and Data Products, 2016. ArXiv Preprint arXiv:1612.05243.
[12] M. Grossman, M. Aziz, H. Chi, A. Tibrewal, S. Imam, V. Sarkar, Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level, J. Parallel Distrib. Comput. 105 (2017) 18–30.
[13] E. Johnson, P. Garrity, T. Yates, R. Brown, et al., Performance of a virtual cluster in a general-purpose teaching laboratory, in: IEEE International Conference on Cluster Computing, CLUSTER, IEEE, 2011, pp. 600–604.
[14] C. Kaewkasi, W. Srisuruk, A study of big data processing constraints on a low-power Hadoop cluster, in: IEEE International Computer Science and Engineering Conference, ICSEC, IEEE, 2014, pp. 267–272.
[15] K. Kambatla, G. Kollias, V. Kumar, A. Grama, Trends in big data analytics, J. Parallel Distrib. Comput. 74 (7) (2014) 2561–2573.
[16] M. Litwintschik, A billion taxi rides in Hive & Presto, 2016. http://tech.marksblogg.com/billion-nyc-taxi-rides-hive-presto.html.
[17] P. López, E. Baydal, On a course on computer cluster configuration and administration, J. Parallel Distrib. Comput. 105 (2017) 127–137.
[18] J. Mullen, C. Byun, V. Gadepally, S. Samsi, A. Reuther, J. Kepner, Learning by doing, high performance computing education in the MOOC era, J. Parallel Distrib. Comput. 105 (2017) 105–115.
[19] T. Newhall, A. Danner, K.C. Webb, Pervasive parallel and distributed computing in a liberal arts college curriculum, J. Parallel Distrib. Comput. 105 (2017) 53–62.
[20] L.B. Ngo, E.B. Duffy, A.W. Apon, Teaching HDFS/MapReduce systems concepts to undergraduates, in: IEEE International Parallel & Distributed Processing Symposium Workshops, IPDPSW, 2014, pp. 1114–1121.
[21] NYC TLC Trip Data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.
[22] A.S. Rabkin, C. Reiss, R. Katz, D. Patterson, Experiences teaching MapReduce in the cloud, in: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, ACM, 2012, pp. 601–606.
[23] P. Russom, Big data analytics, TDWI Best Practices Report, Fourth Quarter, 2011.
[24] C. Shaoul, C. Westbury, A reduced redundancy usenet corpus (2005–2011), vol. 39, University of Alberta, 2013. http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html.
[25] Stack Exchange Data Dump. https://archive.org/details/stackexchange.

Joshua Eckroth is an assistant professor at Stetson University and chief architect of i2k Connect. His research interests lie in computer science education and artificial intelligence such as abductive reasoning and belief revision. Eckroth holds a Ph.D. in computer science from the Ohio State University.