Integrating Big Data into the Computing Curricula

Yasin N. Silva
Arizona State University
ysilva@asu.edu

Suzanne W. Dietrich
Arizona State University
dietrich@asu.edu

Jason M. Reed
Arizona State University
jmreed3@asu.edu

Lisa M. Tsosie
Arizona State University
lmtsosi1@asu.edu

ABSTRACT

An important recent technological development in computer science is the availability of highly distributed and scalable systems to process Big Data, i.e., datasets with high volume, velocity and variety. Given the extensive and effective use of systems incorporating Big Data in many application scenarios, these systems have become a key component in the broad landscape of database systems. This fact creates the need to integrate the study of Big Data Management Systems as part of the computing curricula. This paper presents well-structured guidelines to perform this integration by describing the important types of Big Data systems and demonstrating how each type of system can be integrated into the curriculum. A key contribution of this paper is the description of a wide array of course resources, e.g., virtual machines, sample projects, and in-class exercises, and how these resources support the learning outcomes and enable a hands-on experience with Big Data technologies.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Design, Experimentation

Keywords

Databases curricula, big data management systems

1. INTRODUCTION

The development and extensive use of highly distributed and scalable systems to process Big Data is widely considered as one of the recent key technological developments in computer science [14, 10, 18]. These systems, here called Big Data Management Systems (BDMSs), have been successfully used in many application scenarios, e.g., scientific data analysis, social media data mining, recommendation systems, and correlation analysis on web service logs. BDMSs are typically composed of large clusters of commodity machines and are often dynamically scalable, i.e., nodes can be added or removed based on the workload. Examples of such systems are: MapReduce [11], Hadoop [4], HBase [5], and VoltDB [19]. The importance of BDMSs in the current landscape of database systems creates the need to integrate their study as part of the computing curricula, particularly as part of database or data management courses. This paper presents multiple guidelines to perform this integration. The main contributions of the paper are:

- The identification of three important types of BDMSs and the description of their core concepts, taxonomy, sample systems, and guidelines about when to use them.
- Guidelines to integrate each type of BDMS as a course unit and the description of the learning objectives.
- A detailed description of class resources for each unit to enable a hands-on experience with BDMSs. These resources include virtual machines, data generators, sample programs, projects, and in-class exercises.
- Files of all the resources made available to instructors. The goal is to enable instructors to use and extend these resources based on their specific needs.

The rest of the paper is organized as follows. Section 2 describes three important types of BDMSs: MapReduce, NoSQL, and NewSQL. Sections 3, 4, and 5 present the details of how each type of BDMS can be introduced as a learning unit. These sections describe many class resources and how they support the learning outcomes. These resources will be made available in [6]. Section 6 presents a discussion of various facets of integrating the proposed units into the computer science curricula. Section 7 concludes the paper.

2. BIG DATA LEARNING UNITS

While the area of BDMSs is relatively recent (the foundational MapReduce paper [11] was published in 2004), it has quickly developed and there are now dozens of BDMS-related open-source and commercial products. One of the first challenges for computer science educators is to organize these different technologies and products into well-defined learning units to be included into the computing curricula. The authors propose the following units, where each unit corresponds to a core type of BDMS:

SIGCSE '14 Atlanta, Georgia USA

- **MapReduce**. It is widely considered to be one of the core programming models for Big Data processing. MapReduce started the wave of BDMS systems. It enables building highly distributed programs that run on failure-tolerant and scalable clusters of commodity machines. Some examples of MapReduce implementations are: Apache Hadoop [4], Google MapReduce [11], and Microsoft's Dryad [13].

- **NoSQL**. This type of BDMS comprises data stores that have been designed to provide higher scalability and availability than conventional relational databases. Many NoSQL systems do not support the SQL query language and implement a simplified transaction and consistency model. Some examples of NoSQL data stores are: Apache Cassandra [3], Google's BigTable [7], MongoDB [2], and Apache HBase [5].

- **NewSQL**. This type of BDMS aims to have the same levels of scalability and availability of NoSQL systems but still maintains the ACID properties (Atomicity, Consistency, Isolation, Durability) and SQL query language of traditional relational databases. Some examples of NewSQL systems are: VoltDB [19], NuoDB [16], Google Spanner [9], and Clustrix [8].

As discussed in Section 6, the proposed units can be integrated into introductory and advanced database and distributed system courses.

In general, the student learning outcomes of these units are that students are able to:

- recognize the key properties, strengths and limitations of each type of BDMS (**LO1**),

- build applications using specific instances of each type of BDMS (**LO2**), and

- understand when to use each type of BDMS (**LO3**).

3. DATA PROCESSING WITH MapReduce

3.1 The MapReduce Framework

MapReduce is an extensively used programming framework for processing large data. MapReduce jobs run on clusters of commodity machines that are dynamically scalable. The MapReduce framework quickly processes massive datasets by splitting them into independent chunks that are processed in parallel. The framework works by dividing the processing task into two main phases: *map* and *reduce*. The framework user is required to provide two functions (*map* and *reduce*) with the following general form:

```
map: (k1,v1) → list(k2,v2)
reduce: (k2,list(v2)) → list(k3,v3)
```

Multiple *map* tasks process independent input chunks in parallel. Each *map* call is given a key-value pair `(k1,v1)` and produces a list of `(k2,v2)` pairs. The output of the *map* calls is transferred to the *reduce* nodes (*shuffle* phase). All the intermediate records with the same intermediate key `(k2)` are sent to the same reducer node. At each *reduce* node, the received intermediate records are sorted and grouped (all the intermediate records with the same key form a single group). Each group is processed in a single *reduce* call.

```
Algorithm WordCount

Map (line_number, line_contents)
   for each word in line_contents
      emit(word, 1)

Reduce (word, values[])
   sum = 0
   for each value in values
      sum = sum + value
   emit(word, sum)
```

Figure 1: MapReduce algorithm of WordCount.

3.2 MapReduce Learning Resources

A popular open-source implementation of the MapReduce framework available to use in support of hands-on in-class exercises and project assignments is Apache Hadoop [4]. Unfortunately, due to the nature of the MapReduce framework, a computer cluster would be needed to directly enable such interaction. Many institutions, however, do not have such clusters available for teaching. Another option is the use of Hadoop's pseudo-distributed mode which allows running MapReduce jobs on a single node where each Hadoop daemon runs on a different Java process. This approach, however, still requires a significant installation and configuration effort. One solution that the authors utilize and contribute as part of the resources in [6] is the development of a virtual machine (VM) with all the required packages already installed and configured. This VM has the following components: (1) Linux as the operating system, (2) Hadoop 1.0.4 using pseudo-distributed mode (this includes the Hadoop MapReduce programming framework and the Hadoop distributed file system - HDFS), (3) the Eclipse development environment and the Hadoop Eclipse plug-in [1], (4) data generators and sample datasets, and (5) source code of multiple sample MapReduce programs.

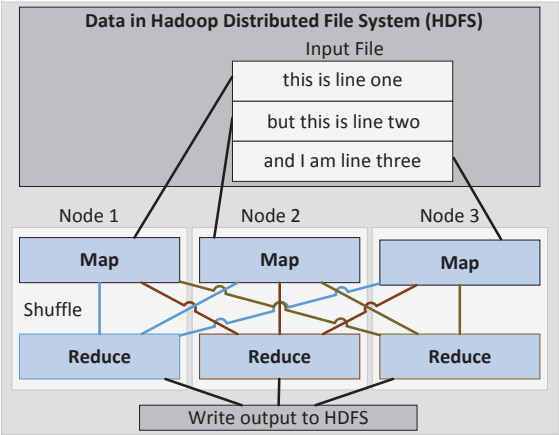
3.2.1 Data Generators and Datasets

The MapReduce programs that will be presented in the next sections make use of two datasets: MobyShakespeare and MStation. MobyShakespeare contains the text of the complete unabridged works of Shakespeare [17]. MStation, a synthetic dataset prepared by the authors, contains meteorological station data. Each line of the MStation dataset contains the following tab-separated attributes: station ID, zip code, latitude, longitude, temperature, precipitation, humidity, year and month. MStation's data generator will be made available to instructors [6].

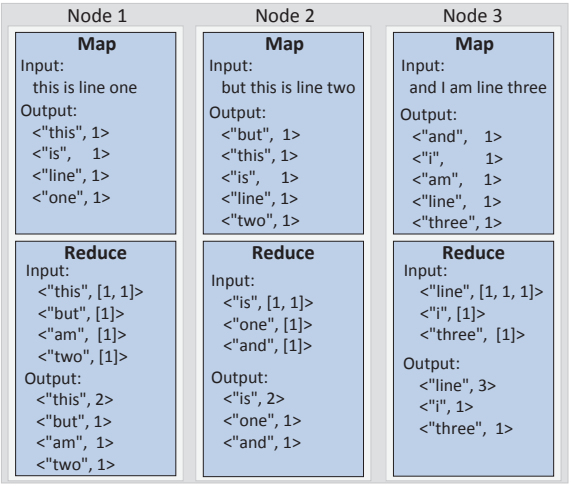
3.2.2 MapReduce Program Template

To facilitate the process of writing MapReduce programs in Hadoop, students could use a provided MapReduce template (MapReduceShell.java). This template imports all the required libraries, provides empty definitions of the Map and Reduce methods, and includes a generic version of the run method, which creates, configures, and runs a generic Hadoop MapReduce job.

One of the first activities the authors suggest after introducing the MapReduce model is to describe the WordCount problem. This is a very illustrative problem where the goal is to count the number of times each word appears in the input dataset (text files). Fig. 1 shows the pseudo-code of the MapReduce WordCount algorithm and Fig. 2 shows the



a) Map, shuffle and reduce stages



b) Input and output of Map and Reduce tasks

Figure 2: WordCount - MapReduce Processing.

details of how this program is executed on the MapReduce framework. Fig. 2.a provides a high-level representation of how different chunks of the input dataset are processed by different map tasks and the way intermediate data is sent to the reduce tasks in the shuffle stage. This figure also shows that the input and output datasets are stored in the distributed file system. Fig. 2.b shows the distributed processing details in this MapReduce job. Particularly, this figure shows the input and output of each map and reduce task. Since the WordCount code has been built using the template described previously, this is also a good example to show students how to extend the template to solve specific problems using MapReduce. The WordCount example can be run using the MobyShakespeare dataset or any other text input file.

3.2.3 *In-class Exercises and Assignments*

The MStation dataset provides a basis for additional in-class exercises for illustrating MapReduce programs [6].

- **AverageTempByStation.java**. This program aggregates the data by station ID and outputs the average temperature for each station. The pseudo-code of this MapReduce algorithm is presented in Fig. 3.

```
Algorithm AverageTempByStation

Map (line_number, MStation_line)
   load MStationRecord from MStation_line
   emit(MStationRecord.stationID,
        MStationRecord.temp)

Reduce (stationID, temp_values[])
   sum = 0
   count = 0
   for each value in temp_values
      sum = sum + value
      count = count + 1
   average = sum / count
   emit(stationID, average)
```

Figure 3: MapReduce algorithm of AverageTempByStation.

- **AverageTempByZipCode.java**. This MapReduce program groups the data by zip code and outputs the average temperature at each zip code.

- **SortByStationID.java**. This code sorts the input data by station ID. The output data still contains all the attributes of the input dataset.

Also, the MStation dataset can be used for class assignments and projects to design and implement MapReduce programs, such as: (1) the computation of the total rainfall grouping by zip code and year, (2) the computation of the average rainfall and average temperature, grouping by zip code and year while considering only years greater than 1950, and (3) the identification of the highest 10 temperature readings reporting the temperature, zip code, month and year.

Achieving learning objectives.

The authors have found that the use of diagrams, like the one in Fig. 2, helps students to understand the steps that take place at the various stages of a MapReduce job (learning objective LO1). Also, the use of the described VM provides students with hands-on experience with Hadoop MapReduce (LO2) and reinforces the understanding of the MapReduce features (LO1). The VM enables students to focus on designing and implementing MapReduce programs rather than installation and tedious configuration. Furthermore, the described data generator can be modified by instructors to generate datasets with different sizes or data distribution properties that can be used in additional exercises or projects.

To give students a better understanding of when to use MapReduce, it is important to contrast it with other data processing systems (LO3). For instance, in a database class, MapReduce can be compared with traditional relational databases. It should be highlighted that MapReduce systems do not make older data processing technologies obsolete. Instead, different systems might be more suitable for different tasks. For instance, in the scenario of a retail store, database systems could be the right tool to process selling transactions since these systems provide consistent and durable data processing. On the other hand, a MapReduce system may be the right tool to analyze and get useful insights, e.g., aggregations, trends and correlations, from data that contains selling operations for the last five years.

4. BIG DATA ANALYSIS WITH NoSQL

4.1 NoSQL Data Stores

NoSQL is a broad class of data stores that aims to provide higher scalability and availability compared to conventional relational databases. In many cases, these systems do not adhere to the relational database model, do not support SQL for data manipulation, and implement a simplified transaction/consistency model. Some of the key properties of NoSQL data stores are as follows: (1) they usually have a distributed, fault-tolerant architecture where more machines can be easily added, (2) data is partitioned among different machines, (3) data replication is used for fault tolerance, and (4) NoSQL data stores commonly provide a simplified consistency model, for instance *eventual consistency*: given a sufficiently long period of time in which no changes are made, all the previous updates are expected to eventually propagate through the system.

Several types of NoSQL systems are presented next:

- **Document stores**. Store documents that contain data in some format (XML, JSON, binary, etc.). Examples: MongoDB, Oracle NoSQL Database, CouchDB, and SimpleDB.
- **Key-Value stores**. Store the data in a schema-less way (commonly key-value pairs). Data items could be stored in a data type of a programming language or an object. Examples: Cassandra, Dynamo, Riak, and MemcacheDB.
- **Graph databases**. Stores graph data. For instance: social relations, public transport links, road maps or network topologies. Examples: AllegroGraph, InfiniteGraph, Neo4j, and OrientDB.
- **Tabular data stores**. Stores tabular data with potentially many columns and rows. Examples: HBase, BigTable, and Hypertable.

4.2 NoSQL Learning Resources

Similarly to the case of MapReduce, one of the best ways to introduce NoSQL data stores is to enable students to interact with real NoSQL systems. To accomplish this, it is important to incorporate the use of a specific NoSQL database like Apache HBase [4], which is also included in the provided VM. HBase is an open-source NoSQL distributed database system written in Java and modeled after Google's BigTable [7]. It runs on top of HDFS and provides a fault-tolerant way of storing and processing large amounts of sparse tabular data. HBase provides efficient random reads and writes on top of HDFS, which does not support random writes. HBase has been used by companies like Yahoo! and Facebook [15].

One of the key lessons about NoSQL systems is learning about the different ways in which developers and applications can interact with the database. This can be accomplished with the following in-class activity where students create an HBase table to maintain blog posts and interact with the system using both shell commands and a Java application. First, students use the HBase shell interface to run the *create* and *put* commands to create an HBase table and insert the blog posts data. Next, students use the commands *scan*, *get*, and *delete* to print all the content of the table, get the content of individual records, and delete records, respectively. Fig. 4 shows sample shell commands for this activity. In the second part of this activity students programmatically access the HBase table created in the first part. To this end, students first establish a connection with the blog posts table, and use instances of the *Put*, *Scan*, and *Get* classes to insert additional rows, list all rows, and retrieve the content of single rows, respectively. Fig. 5 shows a fragment of the Java code that performs the described tasks. In addition, complementary resources to incorporate NoSQL using MongoDB [2] are described in [18].

Create an HBase table

```
Create 'blogposts', 'post', 'image'
```

Adding data to a table

```
put 'blogposts', 'post1', 'post:title', 'The Title'
put 'blogposts', 'post1', 'post:author', 'The Author'
put 'blogposts', 'post1', 'post:body', 'Body of a post'
put 'blogposts', 'post1', 'image:header', 'image1.jpg'
put 'blogposts', 'post1', 'image:bodyimage', 'image2.jpg'
```

List all the tables

```
list
```

Scan a table (show all the content of a table)

```
scan 'blogposts'
```

Show the content of a record (row)

```
get 'blogposts', 'post1'
```

Deleting (all the cells of a given row)

```
deleteall 'blogposts', 'post1'
```

Drop a table

```
disable 'blogposts'
drop 'blogposts'
```

Figure 4: Accessing HBase with shell commands.

Achieving learning objectives.

In the authors' experience, the use of a VM with HBase and the described in-class activity have been very important tools to enable students to get a more concrete understanding of the features of a NoSQL system (LO1), and to gain hands-on experience about the different mechanisms to interact with an HBase data store (LO2).

One of the key lessons about NoSQL data stores is when they should be used (LO3). For this, it is important to highlight that NoSQL databases should be considered when there is a need to manage very large amounts of data (currently in the order of terabytes or petabytes), the application scenario does not use a fixed schema, and performance is more important than consistency (the application does not require full ACID guarantees).

It is also important to observe that features and data types supported by NoSQL systems differ, particularly in how the system supports the various levels of consistency (C), availability (A), and network partition tolerance (P). In fact, the CAP theorem states that a system has to pick two of these three properties and that a system cannot have all three simultaneously while maintaining an acceptable latency [12]. For instance, HBase [5] prioritizes consistency and partitioning tolerance (CP), while Cassandra [3] values availability and partitioning tolerance (AP).

NoSQL should also be contrasted against other types of data processing systems (LO3). Traditional relational databases for instance may be the right tool for a banking application where all the data fits in a single server since they

```
//Use HTable object to connect to the blogposts table.
Configuration config = HBaseConfiguration.create();
HTable table = new HTable(config, "blogposts");

//Add a row (post2) to the table
Put p = new Put(Bytes.toBytes("post2"));
p.add(Bytes.toBytes("post"), Bytes.toBytes("title"),
      Bytes.toBytes("Title2"));
p.add(Bytes.toBytes("post"), Bytes.toBytes("author"),
      Bytes.toBytes("Author2"));
table.put(p);

//Retrieve the data of a row using Get
Get g = new Get(Bytes.toBytes("post2"));
Result r = table.get(g);
byte [] valTitle = r.getValue(Bytes.toBytes("post"),
                   Bytes.toBytes("title"));
byte [] valAuthor = r.getValue(Bytes.toBytes("post"),
                   Bytes.toBytes("author"));
System.out.println("GET: " + Bytes.toString(valTitle)+
                   " " + Bytes.toString(valAuthor));

// Retrieve multiple rows using Scan (all titles)
Scan s = new Scan();
s.addColumn(Bytes.toBytes("post"),
            Bytes.toBytes("title"));
ResultScanner scanner = table.getScanner(s);
for (Result rr : scanner)
  System.out.println("Title: " + rr);
```

Figure 5: Accessing HBase table from Java.

provide full ACID guarantees. On the other hand, a NoSQL system like HBase could be the right tool for web-scale messaging or blogging systems since they deal with massive amounts of data and prioritize performance over full consistency. In fact, HBase is used to support the messaging subsystem in Facebook [15]. While MapReduce and NoSQL systems can process massive amounts of data in a highly distributed and scalable fashion, each system may be the best option in different scenarios. Furthermore, it is important to consider the specific features of NoSQL systems. For instance, in the scenario of the messaging system, HBase may be the right tool to run a query that quickly retrieves the messages of a user (random access). In the same scenario, the MapReduce framework could be the best tool to run a job that processes the entire dataset to aggregate the number of messages by sender location.

5. BIG DATA PROCESSING WITH NewSQL

5.1 NewSQL Database Systems

NewSQL is a class of database systems that aims to provide the same levels of scalability and availability of NoSQL systems while still maintaining the ACID guarantees and SQL query language of traditional relational databases. Some of the key properties of NewSQL systems are as follows: (1) they are based on a distributed shared-nothing architecture that can dynamically scale horizontally (adding cluster nodes), (2) they support the relational data model, (3) they provide ACID properties for transactions, and (4) they support SQL as the primary query language.

Several NewSQL systems have been built from scratch to operate on distributed clusters of shared-nothing nodes. Some of these systems, e.g., Google Spanner, NuoDB, and Clustrix, are general purpose databases that support any type of query. Other systems, e.g., VoltDB, are in-memory

```
C1: CREATE TABLE votes(phone_number bigint NOT NULL, state
 varchar(2) NOT NULL, contestant_number integer NOT NULL);

Q1: SELECT contestant_number, count(*) FROM votes
 GROUP BY contestant_number ORDER BY contestant_number;

Q2: SELECT state, COUNT(*) cnt FROM votes GROUP BY state
 order by cnt DESC;

Q3: SELECT contestant_number, state, COUNT(*) FROM votes
 GROUP BY contestant_number, state;
```

Figure 6: VoltDB SQL Queries.

databases that focus on supporting short-lived and repetitive queries that touch small fractions of the data. Other NewSQL systems are based on optimized storage engines for MySQL, e.g., TokuDB, MemSQL, and Akiban. While applications interact with these systems as if they were standard MySQL databases, they have better scalability properties. There are also some NewSQL systems that are based on middleware components that automatically partition and distribute existing databases, e.g., dbShards and ScaleBase.

5.2 NewSQL Learning Resources

VoltDB [19] is a well-documented, open-source NewSQL system that can be used to support the introduction of NewSQL technologies with the integration of hands-on activities into the curriculum. VoltDB is an in-memory and ACID-compliant database system that uses a shared nothing architecture. VoltDB is supported on Linux and Mac OS X and provides client libraries for Java, C++, C#, Python, PHP, and Node.js.

The use of a VM is also a great way to enable a hands-on interaction with NewSQL systems like VoltDB. In this case, instructors can use the VM available at VoltDB's web site [19]. Besides the VoltDB software, this VM contains a set of sample applications and tools for application development. The sample applications are particularly useful resources to teach about NewSQL and VoltDB. These applications include: (1) a system that simulates a telephone based voting process, (2) the implementation of a Key-Value store backed by VoltDB, (3) an example that shows memcache-like cache implemented using VoltDB, and (4) an application that shows the use of flexible schema and JSON.

Instructors can use the following in-class activity working with the voting process simulator. Students first analyze and execute the provided scripts that start the database server, create tables, and run a client application that simulates a call center receiving the votes. Fig. 6 shows the SQL (C1) that creates table *votes*. During the execution of the client application, students use a provided web-based interface to monitor in real-time the voting results and the current winner in each state. Next, students use VoltDB Studio to write and run SQL queries. For instance Fig. 6 shows queries prepared by the authors to compute the total number of votes obtained by each contestant (Q1), the total number votes in each state (Q2), and the number of votes for each candidate in each state (Q3). This activity also provides a good opportunity to highlight some of the key properties of this type of NewSQL system, e.g., high transaction throughput (transactions/second) and low latency (time to process a vote).

Achieving learning objectives.

Similarly to the previous units, the use of the VM enables

students to get not only hands-on experience with VoltDB (LO2) but also a better understanding of key features of NewSQL systems (LO1), particularly the support of the relational model and SQL.

It is also important to describe scenarios, beyond the voting system, where NewSQL systems can be the right type of system to use (LO3). While NoSQL and NewSQL systems can handle datasets with massive size and high velocity, NewSQL systems are particularly suitable for applications that require support of the relational model, ACID guarantees, and SQL. For instance, while an HBase (NoSQL) can be the right data store for a web-scale blogging system, a NewSQL system like VoltDB can be the right database to process the rapid stream of stock exchange operations while ensuring transactional consistency [20].

6. DISCUSSION

The three proposed modules can be integrated into several introductory and advanced computer science courses. For instance, an introduction to MapReduce could be integrated into introductory databases or distributed systems courses. In this case, instructors can incorporate the in-class exercises and activities suggested for the MapReduce module. A high level exposure to NoSQL and NewSQL could also be integrated into an introductory database course through an assignment focused on exploring data management technologies beyond relational databases. The instructor could complement the assignment with an in-class discussion that highlights the key properties of the explored systems and scenarios where they are the right systems to use. A more extensive exposure of the three types of BDMSs can be incorporated in an advanced database class. In this case, instructors could incorporate the different hands-on activities and projects described in this paper. MapReduce could also be integrated into a programming course. This integration should focus on presenting the MapReduce programming model as a simple but effective approach to distribute the execution of a program among multiple computers. The authors have specifically incorporated the three modules into an advanced database course and the MapReduce module into an introductory distributed systems course.

To assess the integration of the three proposed modules and the extent to which the learning objectives were achieved, the authors used an end-of-class survey in the advanced database class. While this was a small class, the results are promising. The score distributions for the three modules were very similar. The average scores are presented next (1: Strongly agree - 5: Strongly disagree). To the question "*I am able to recognize the key properties, strengths and limitations of MapReduce/NoSQL/NewSQL*" (LO1), 50% of students strongly agree and 50% agree. To the question "*I understand when to use MapReduce/NoSQL/NewSQL systems*" (LO3), 67% of students strongly agree and 33% agree. To the question "*I am familiar with Hadoop/HBase/VoltDB and can build applications that interact with these systems*" (LO2), 42% of students strongly agree, 42% agree, and 16% gave lower scores. In the open ended questions, students gave positive comments about the use of hands-on exercises, VMs, and illustrative diagrams, e.g., "*using the VMs really captured my attention when using hands-on exercises and working directly with the systems*", "*the diagrams really helped visualize relationships and comparisons between different technologies*", and "*the hands-on exercises were great*".

7. CONCLUSIONS

Many application scenarios require processing massive datasets in a highly scalable and distributed fashion and different types of BDMSs have been designed to address this challenge. Graduating students gain employment in companies that already use BDMSs or are eager to use them to get better insights from the large datasets they manage. Thus, it is important to integrate the study of these systems as part of the computing curricula. This paper presents a set of guidelines and a wide array of class resources to integrate the study of three core types of BDMSs: MapReduce, NoSQL, and NewSQL. The paper also reports results of integrating the proposed units into a database course.

8. REFERENCES

[1] Hadoop eclipse plug-in. `http://wiki.apache.org/hadoop/EclipsePlugIn`.

[2] 10gen. Mongodb. `http://www.mongodb.org/`.

[3] Apache. Cassandra. `http://cassandra.apache.org/`.

[4] Apache. Hadoop. `http://hadoop.apache.org/`.

[5] Apache. Hbase. `http://hbase.apache.org/`.

[6] ASU. Big data management course resources. `http://www.public.asu.edu/~ynsilva/iBigData`.

[7] 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 Trans. Comput. Syst.*, 26(2):1–26, 2008.

[8] Clustrix. Clustrix. `http://www.clustrix.com/`.

[9] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed database. In *USENIX*, 2012.

[10] A. Cron, H. L. Nguyen, and A. Parameswaran. Big data. *XRDS*, 19(1):7–8, Sept. 2012.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. In *OSDI*, 2004.

[12] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[13] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[14] D. Kumar. Data science overtakes computer science? *ACM Inroads*, 3(3):18–19, Sept. 2012.

[15] K. Muthukkaruppan. The underlying technology of messages. `https://www.facebook.com/note.php?note_id=454991608919`.

[16] NuoDB. Nuodb. `http://www.nuodb.com/`.

[17] M. Project. Moby shakespeare. `http://icon.shef.ac.uk/Moby/`.

[18] A. Sattar, T. Lorenzen, and K. Nallamaddi. Incorporating nosql into a database course. *ACM Inroads*, 4(2):50–53, June 2013.

[19] VoltDB. Voltdb. `https://voltdb.com/`.

[20] VoltDB. Voltdb application gallery. `http://voltdb.com/no-limits/apps-gallery.php`.