Implementing a GPU-based Machine Learning Library on Apache Spark

James Jia
Pradeep Kalipatnapu
Richard Chiou
Yiheng Yang
John F. Canny, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-51
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-51.html

May 11, 2016

Implementing a GPU-based Machine Learning Library on Apache Spark

James Jia

Electrical Engineering and Computer Sciences

University of California, Berkeley

Table of Contents

1.1 Problem Definition

Our capstone project is to port BIDData over onto Apache Spark, an open source engine used for large-scale data processing, so real-world developers at companies such as Yahoo, Twitter, and Facebook can leverage the 10x performance and cost benefits of GPU accelerated machines at a large scale for machine learning tasks. BIDData is a machine learning library that runs on a single machine that is GPU-optimized, outperforming other setups that run on hundreds of multi-core CPU machines due to the parallel structure of many machine learning algorithms. In fact, it holds the benchmark for a plethora of common algorithms such as k-Nearest Neighbors, Random Forests, and Latent Dirichlet Allocation, beating other setups that use hundreds of multi-core CPU computers (Canny 2015).

System	nodes/cores	nclust	Time	Cost	Energy(KJ)
Spark	32/128	256	180s	$0.45	1150
BIDMach	**1**	**256**	**320s**	**$0.06**	**90**
Spark	96/384	4096	1100	$9.00	22000
BIDMach	**1**	**4096**	**735**	**$0.12**	**140**

Running KMeans on the MNIST-8M (25GB) dataset (Canny 2015)

However, running BIDData on a single machine can only process data on the scale of hundreds of gigabytes. For comparison, many companies, like Yahoo, process petabytes of data every day (Feng 2013). Developers that want to scale up BIDData to

match their data processing needs have to handle all of the problems that arise when creating a scalable, distributed platform, such as incompatible library dependencies and data storage and synchronization issues with their database files. However, by integrating BIDData into Apache Spark, we can eliminate this overhead so developers can focus on performing actual analytics.

1.2 Task Breakdown

We split our project into two major components. Pradeep and Yiheng focused on the first component: to create an automated build management system for running BIDData on multiple platforms. Currently, developers that want to use BIDData to run machine learning algorithms on the GPU need to build the libraries themselves. This represents a huge barrier to entry as many alternative machine learning libraries have released pre-built versions that simplify the deployment process. However, as Pradeep and Yiheng will discuss in greater detail in their reports, establishing an automated build management system for BIDData is complicated by the fact that we are supporting multiple different host architectures and each of these architectures have distinct native libraries upon which BIDData depends. Meanwhile, Richard and I worked on the second component: to port BIDData on top of Apache Spark and the Hadoop Distributed File System (HDFS). Since BIDData will now be reading and writing to HDFS instead simply writing to disk, we need to extend BIDData's existing API this new requirement. Furthermore, we also need to implement several parallel machine learning algorithms in order to document the performance of running BIDData on top of Spark and compare

against other alternatives. Richard worked on implementing distributed logistic regression and I worked on extending support to reading and writing from HDFS as well as implemented the distributed K-Means algorithm.

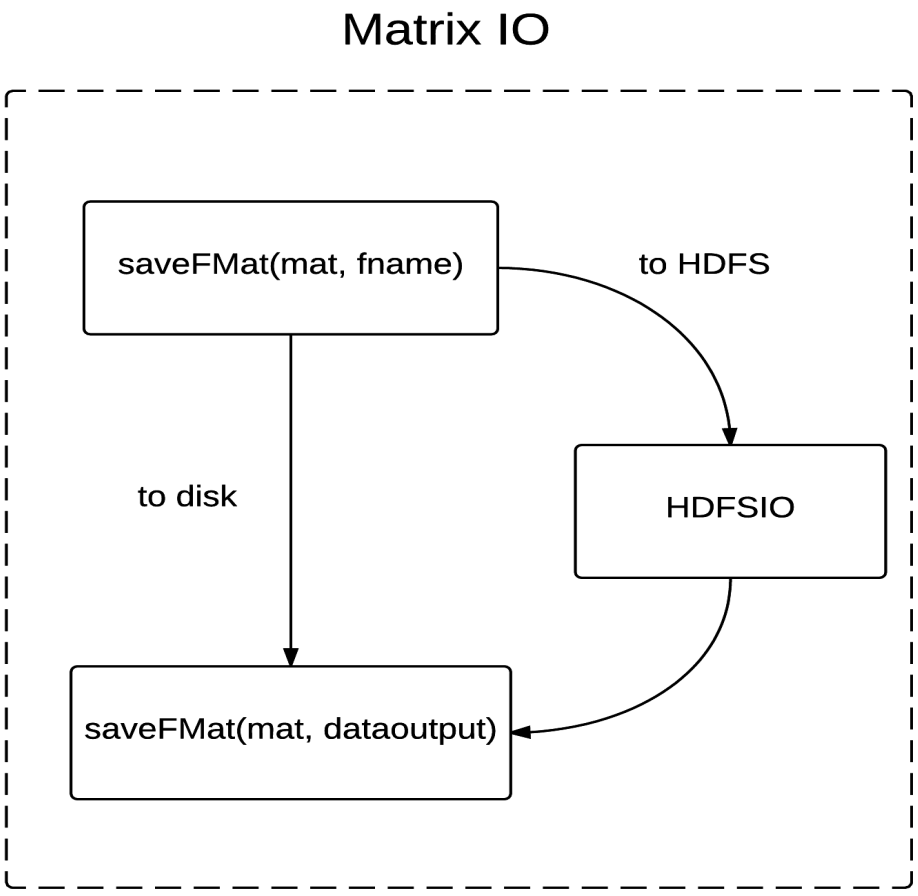
1.3 Setting up EC2

My first priority was to set up a sample distributed system as a mock environment that we could use for future development. Without such an environment, we would not be able to benchmark the parallel variants of common machine learning algorithms to compare against the serial versions. We chose to use Amazon EC2 as it is the predominant cloud computing platform used in both academia and industry to power distributed computation. This entailed setting up Apache Spark on the master and slave nodes with the appropriate configuration and hooking Spark up to the Hadoop Distributed File System (HDFS) in which we house our data. The BIDData libraries and Apache Spark both depend on native libraries that are specific to the machine that one is running on. This meant that running the BIDData libraries on top of Spark requires additional configuration steps beyond downloading and building Spark and Hadoop on the EC2 instances – we needed a mechanism to specify all the library modules that BIDData requires to be distributed across all the nodes in our cluster. Traditionally, users who want to run BIDData on a single machine run a shell script that downloads all the library dependencies. The Spark guide on configuration and deployment recommended adding in the appropriate library paths to the default configuration file located in SPARK_HOME/confs/spark-defaults.conf, so I created a shell script that deploys the

libraries from the master node to the slave nodes and included the file path in the configuration file on all nodes. Although this solution is not robust to changes in library dependencies from version upgrades in Spark or BIDData, it removed the bottleneck that prevented us from parallelizing our work distribution. Since we also want to allow developers to build Spark with BIDData for themselves, I documented the deployment process as well as the command line invocations to import the BIDData libraries when running Spark as a standalone cluster.

1.4 Supporting Reading and Writing from HDFS

BIDData has its own specialized matrix data types that characterize the different primitives that are stored within the matrix as well as the representation of the matrix itself (sparse or dense). As the abstractions for saving and loading matrices are the same, I will detail the process for saving a matrix to disk for brevity. Currently, to save a matrix to disk, the BIDMach API exposes an overloaded *saveMat* function that, at runtime, calls the appropriate save function based on the matrix type, writing it to an OutputStream. In order to support reading and writing from the Hadoop Distributed File System, we needed to revamp the matrix I/O routines to be based on DataInputStreams and DataOutputStreams classes rather than their current generic variants and implement a wrapper over these custom matrix datatypes in order to allow Hadoop to serialize the data for transmission across the network.



To update the matrix I/O routines, Professor Canny rewrote the layer that writes the matrix data to disk to operate on DataOutputStreams instead of OutputStreams. This ensures that the underlying data is formatted in a platform independent way and adheres to HDFS's abstraction for I/O. I then added a middle layer that checks the matrix's file path. If the path is addressed to the HDFS, I invoke the new HDFS function to wrap the matrix in a serializable format for Hadoop.

1.5 Parallelizing BIDData

1.5.1 BIDData Architecture Overview

Within the BIDData architecture, the Model class is an abstraction that implements the specific machine learning algorithm. For example, there is a KMeans model that implements the KMeans model update and prediction methods. The Datasource and Datasink classes encapsulate the logic for reading from and writing to the various data formats that we support. Professor Canny created a datasource called *IteratorSource* designed to work with the Iterators class provided by Spark. Finally, the Learner class orchestrates the flow of model training and prediction. It iteratively updates the model and outputs the predictions to the datasink. The learner also has a reference to an options class, which as the name implies, stores the configurations that the developer provide. The developer creates an instance of a learner, passing in the data source, the data sink, and the model that is appropriate for their machine learning task, trains the learner using a subset of the data, and makes a prediction with the remaining data (Canny 2015).

```
val (mm, opts) = KMeans.learner(data_path)
mm.train
val (pp, popts) = KMeans.predictor(mm.model, test_data_path)
pp.predict
```

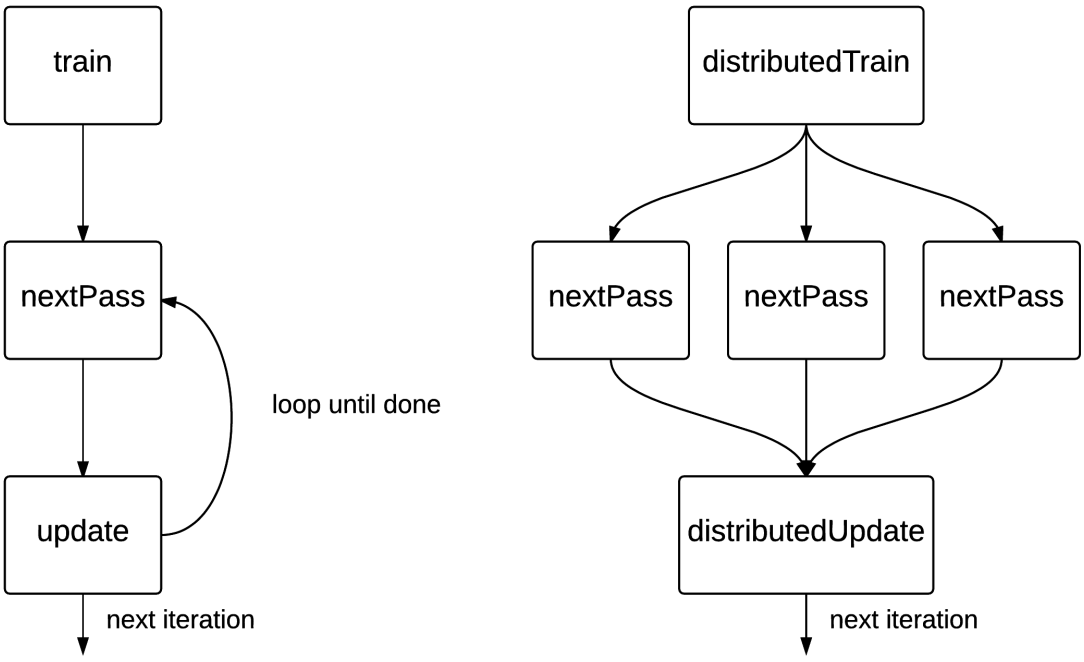
Sample code for running KMeans on a single machine (Canny 2015)

1.5.2 Extending the Learner

The learner class previously assumed that the model is run on a single machine, and thus runs through the iterations of training all at once upon invocation. However,

when running BIDData on a distributed platform, we need to synchronize the models within the learners that are running on the separate executors after each pass through the dataset.

I abstracted away the logic that handles one pass through the dataset into two functions, *firstPass* and *nextPass*, and instead have *train* call the appropriate *pass* function based on the current iteration index. This more fine-grain control over the learner logic lets the distributed learner to perform model synchronization after each pass through the dataset, while still allowing the single-machine variant to iterate through all at once. The distinction between *firstPass* and *nextPass* is based on the fact that, in certain algorithms, the first pass through the dataset is fundamentally distinct from the remaining passes. As an example, the K-Means algorithm instantiates the centroid clusters in the first pass, whereas the remaining passes improve the centroids.



Comparison between iterative training and distributed training

1.5.3 API for Distributed Machine Learning

We distribute our model and our data across the executors. One benefit of this approach is that we can train models that do not fit into the memory of a single GPU. However, this also brings additional challenges of needing to synchronize the model after each pass through the dataset. At a high level, we want to create a learner for each executor and iterate through the data that exists locally on that executor on each pass, synchronizing the learner's model between the passes. Although we later want to utilize Kylix, a butterfly all-reduce communication network to synchronize the model for optimum network performance, I currently use Spark's implementation of treeReduce as a baseline for comparison.

I created an application programming interface (API) that abstracts away the implementation details for running BIDData on top of Spark. The API takes in several parameters: the spark context of class SparkContext, the learner of class Learner, and the data of class RDD[(SerText,MatIO)] on which to run the learner, and the number of executors in the cluster. The SparkContext variable is required to operate on the RDD abstraction, the learner and data are necessary to run the machine learning algorithm on the dataset, and the number of executors is passed in so the program can load balance across the executors.

1.5.4 Implementing Distributed KMeans

To test out the framework to run machine learning algorithms on Spark using BIDData, I implemented a distributed variant of the existing KMeans algorithm in the BIDData machine learning library. This required adding a general reduction operator,

combineModels, to the Model class, which the KMeans model overrides with its model-specific reduction operation. Since the first iteration of a model update may be different from the other iterations, the function also takes in the iteration index.

In the first iteration of distributed KMeans, we want to randomly sample *n* clusters from our dataset with equal probability. I do so by counting the number of data points that have been processed by each model, and sample *n* clusters from the two models' clusters with probability proportional to the number of processed datapoints. In the other iterations of model reduction, I add up all the centers' features and average them post-reduction.

1.5.5 KMeans Benchmarks

256 clusters, batch size of 10000, 10 epochs

	Total Time	Per Epoch
Sequential BIDMach	349 seconds	37 seconds
Distributed BIDMach with 16 clusters	45.6 seconds	1.7 seconds
Spark with 16 clusters	263.1 seconds	26 seconds

5000 clusters, batch size of 10000, 10 epochs

	Total Time	Per Epoch
Sequential BIDMach	1605 seconds	169.2 seconds
Distributed BIDMach with 16 clusters	235 seconds	14.4 seconds
Spark with 16 clusters	5779 seconds	576 seconds

These benchmarks were obtained from running on AWS g2.2xlarge instances, each with one 1,5360 CUDA cores NVIDIA GPU and eight Intel Xeon processors (Amazon 2016). From the benchmarks we see that running BIDMach on Spark vastly outperforms both BIDMach running on a single machine as well as Spark running on the same EC2 setup. As expected, there is some overhead for reducing the model at the end of each epoch and redistributing the updated model back to the workers at the beginning of the following epoch.

Chapter 2: Engineering Leadership

James Jia

Pradeep Kalipatnapu

Richard Chiou

Yiheng Yang

2.1 Introduction

As data storage becomes increasingly commoditized, companies are collecting transactional records on the order of several petabytes that are beyond the ability of typical database software tools to store and analyze. Analysis of this "big data" can yield business insights such as customer preferences and market trends, which can bring companies benefits such as new revenue opportunities and improved operational efficiency (Manyika et. al 2011). To analyze this big data, companies typically build or use third party machine learning (ML) tools.

Professor John Canny of UC Berkeley's EECS department has developed BIDData, a set of GPU-based ML libraries that is capable of completing common ML tasks an order of magnitude faster than rival technologies, when run on a single machine. However, most "big data" tasks require greater computational power and storage space than that offered by a single machine.

Our capstone project integrates BIDData with Apache Spark, a fast, open-source big data processing framework with rapid adoption by the software industry. Integration of BIDData with Spark will enable more complex big data analysis and generate time, energy, and cost savings.

2.2 Trends and Market

The market opportunity for big data is astronomical. Due to the convenience of consumer electronics, more and more daily services such as banking and shopping are being conducted online. These transactions generate a gargantuan amount of user and

market data that companies can benefit from. As an example, the social networking and search engine industries often use machine learning in order to increase their profits on selling advertising space to various companies, optimizing the pricing model for each ad spot as well as determining the best advertisement placement to maximize the likelihood of user clicks (Kahn 2014:7). Determining the optimal pricing and placement strategy is especially pivotal to Google and Yahoo's operations, since most of these ad spaces are paid for through a pay-per-click model and constitute 98.8% of the total revenue of $11.2 billion in 2014 (Kahn 2014:14).

Processing gigantic amounts of information within sub-second time intervals is computationally demanding. Modern computer processing units (CPUs) can process data at 1 billion floating point operations per second, but typical big datasets are on the order of quadrillions of bytes (Intel 2016). Traditional CPUs would take hours to process a single big dataset, and will become increasingly inefficient as dataset sizes continues to grow. Thus, there exists a great opportunity for companies that focus on cost-efficient data analytics tools which provide easy integration and process data quickly. For instance, McKinsey Global Institute estimates that retailers that use efficient big data tools could increase their operating margins by more than 60 percent (Manyika et. al 2011).

Recent technology trends show that in order to accelerate the speed of big data ML algorithms, CPU technology is being replaced by the graphic processing unit (GPU). Because GPUs are optimized for mathematical operations, they are orders of magnitudes faster than CPUs on tasks related to big data analysis, and both industry and

academia are moving towards using GPUs (Lopes and Ribeiro 2010). As a GPU-based ML library, BIDData also offers numerous improvements compared to rival technologies. In terms of single-machine processing speed and electricity expenditure, BIDData already beats the performance of other competitors, including distributed ML libraries, by an order of magnitude, assuming the dataset can be housed under a single machine (Canny 2015).

However, single-instance ML libraries such as BIDData are currently not widely used in industry, as they lack the scalability to handle increasing sizes and complexities of big datasets. Instead, most companies are turning towards efficient distributed computing platforms to process the data (Low et. al 2012). Thus, our capstone project aims to integrate BIDData with the distributed framework of Spark, a leading machine learning software in the market today. Moreover, the project also incorporates Amazon Web Services for big data storage, another platform which many companies are already leveraging today (Amazon 2016). By using this underlying infrastructure, BIDData is more likely to be viewed as a highly desirable big data analysis tool by the market.

2.3 Industry Analysis

2.3.1 Value Chain Analysis

Figure 1: The above value chain covers how users and companies alike benefit from big data. As consumers use products, technology companies are able to collect data on their habits and preferences. Analytics firms and third-party tools process this data and sell their findings. Ultimately, big data analytics can lead to improved products and better user experiences.

In the big data industry, firms processing user data to gain insights into customer habits and preferences. The statistics and trends that they discover can be used to refine existing products and services or be sold to advertisers. For example, users either buy a product (e.g. FitBit) or sign up to use a free ad-supported product (e.g. Gmail) from various tech companies. These companies collect data from their users based on their usage patterns: FitBit provides anonymized, aggregated data for research purposes, and Gmail provides relevant information on users to the Google Ads team. This data is passed onto organizations that specialize in big data analysis. After obtaining insights on the data, these organizations then sell their findings back to the tech companies or to firms such as advertisers. Ultimately, big data analytics can be used to improve products and user experiences for consumers.

While these big data analysis companies have access to lots of data, they may not have the understanding or the resources to create every appropriate tool for analyzing all of their big data, which can come in various formats. Subsequently, these

big data analysis organizations must turn to third-party customers to process some of their data. Because its machine learning libraries record the best possible benchmarks amongst their peers, BIDData has a strong case for becoming one of these leading third-party tools. Subsequently, a startup with expertise in BIDData on Spark could act as both a supplier and a consultant for these big data organizations.

2.3.2 Porter's Five Forces Analysis

According to Michael Porter, all companies face five forces of competition within their industry. Despite its benchmark-leading performances, BIDData faces potential competition from existing firms and future entrants in the big data industry.

2.3.3 Threat of Substitutes

Current machine learning libraries mostly run on CPUs, but as discussed in the Trends and Markets section, the industry has shifted away from them. As evidenced by Netflix's recent switch to using GPUs, the industry has noticed that GPU-based software solutions record higher benchmarks than traditional CPU-based tools (Morgan 2014). Thus in the long term, the threat of substitutes is relatively weak, as CPU-based software is gradually replaced.

2.3.4 Bargaining Power of Suppliers

Typically, programmers are the main individuals involved in the creation of software libraries. Currently, BIDData is open source, allowing any interested independent programmers to collaborate and make unpaid contributions to the project. Thus, the bargaining power of suppliers with regard to wages is extremely weak. As an

additional benefit, the open source model can potentially lead to high-quality products at a fraction of the cost (Weber 2005).

2.3.5 Bargaining Power of Consumers

On the other hand, the bargaining power of consumers in this space is strong. Although lots of research on GPU-based ML techniques has been conducted in recent years, most computers used by consumers and companies alike still only use CPUs. Subsequently, customers are more likely to choose from a wide variety of CPU-based ML tools to use. Moreover, the biggest and most profitable customers (e.g. Google) have the resources to create their own data analysis tools for internal use, which can further depress demand for third-party machine learning tools in this space. While newer computers come with GPUs, it may take some time before users and companies fully invest in and transition to GPU-based ML tools.

2.3.6 Threat of New Entrants

In general, the software industry has an extremely low barrier to entry, as it only takes a single programmer with one computer to create fully-functional software. Additionally, software undergoes lots of iterations quickly. Although BIDData's underlying GPU technology is still relatively new, startups and existing firms alike are growing more interested in GPU-based solutions. Subsequently, the industry faces a very strong threat from new entrants.

2.3.7 Competitive Rivalry

Because big data comes from a variety of sources and in a multitude of formats, consumers require many different ML techniques for analysis. If BIDData does not contain an implementation of a specific ML algorithm, consumers could simply use another tool that offers it. As a result, there exists an intense feature-based rivalry in the third-party tools space, in which several firms offer a multitude of services. For instance, one firm may specialize in data classification, while another firm may provide a product optimized for data regression. Nonetheless, this feature-based rivalry could allow more players to co-exist in this space.

2.4 Go to Market Strategy

Although GPU acceleration has been identified as a promising development, it is largely still in its infancy and has not seen widespread adoption across multiple sectors. Rather than focusing on profitability as in traditional models, we will focus on a strategy that helps us gain market share. To do so, we are going to target a particular subset of companies that have big data problems, specifically companies that have the ability to collect large amounts of data, but not necessarily access to the computational power or resources to obtain business intelligence from it. As we have discussed earlier, Fitbit is a prime example: they have a great capacity to gather information from their devices as an auxiliary effect of their product, but it would require an extraordinary amount of technical expertise as well as infrastructure to sift through the vast sea of data.

With this in mind, our go to market strategy has three main emphases. First, we are utilizing a “plug and play” model that emphasizes fluid software integration to encourage early adoptions. Since we are also going to remain open source, this will also inspire developers to contribute back to our codebase. The stronger advocates could also serve to evangelize within their companies, giving us stronger leverage over our competitors. Lastly, the nature of our product is inherently scalable. Once we have written code ready for production, the cost to have an additional developer use our codebase is negligible.

While customers may cite a lack of technical support as an obstacle to adoption, there exists an opportunity for startups like Databricks to gain customers through their consulting services. Subsequently, we could proactively establish potential partners with companies like Databricks that operate on a consulting model, segmenting our solution as the leading GPU-accelerated machine learning library.

Bibliography

Amazon. (n.d.). All AWS Case Studies. Retrieved March 5, 2016, from https://aws.amazon.com/solutions/case-studies/all/

Amazon. (n.d.). EC2 Instance Types. Retrieved May 7, 2016, from https://aws.amazon.com/ec2/instance-types/

Apache. (n.d.). Spark Programming Guide. Retrieved April 14, 2016, from http://spark.apache.org/docs/latest/programming-guide.html

Apache. (n.d.). Clustering - spark.mllib. Retrieved February 9, 2016, from http://spark.apache.org/docs/latest/mllib-clustering.html

Canny, J. F., & Zhao, H. (2013). BIDMach: Large-scale Learning with Zero Memory Allocation. *Big Learning*. Retrieved from http://biglearn.org/2013/files/papers/biglearning2013_submission_23.pdf

Canny, J. F. (2015, August 19). Benchmarks. Retrieved February 10, 2016, from https://github.com/BIDData/BIDMach/wiki/Benchmarks

Canny, J. F. (2015, September 27). BIDMach Tutorials. Retrieved February 14, 2016, from https://github.com/BIDData/BIDMach/wiki/BIDMach-Tutorials

Intel. (n.d.). 6th Generation Intel® Core™ i7 Processors. Retrieved March 5, 2016, from https://www-ssl.intel.com/content/www/us/en/processors/core/core-i7-processor.html

Kahn, S. (2014). *Search Engines in the US*. *IBISWorld Industry Report 51913a.* Retrieved from IBISWorld database.

Lopes, N., Ribeiro, B., & Quintas, R. (2010). GPUMLib: A new Library to combine Machine Learning algorithms with Graphics Processing Units. *2010 10th International Conference on Hybrid Intelligent Systems*. doi:10.1109/his.2010.5600028

Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., & Hellerstein, J. M. (2012). Distributed GraphLab. *Proc. VLDB Endow. Proceedings of the VLDB Endowment, 5*(8), 716-727. doi:10.14778/2212351.2212354

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Byers, A. H. (2011, May). *Big data: The next frontier for innovation, competition, and productivity* (Rep.). Retrieved http://www.mckinsey.com/business-functions/business-technology/our-insights/big-data-the-next-frontier-for-innovation

Morgan, T. P. (2014, February 11). Netflix Speeds Machine Learning With Amazon GPUs. Retrieved from http://www.enterprisetech.com/2014/02/11/netflix-speeds-machine-learning-amazon-gpus/

Porter, M. E. (2008). The Five Competitive Forces That Shape Strategy. *Harvard Business Review*.

Weber, S. (2004). *The success of open source*. Cambridge, MA: Harvard University Press.