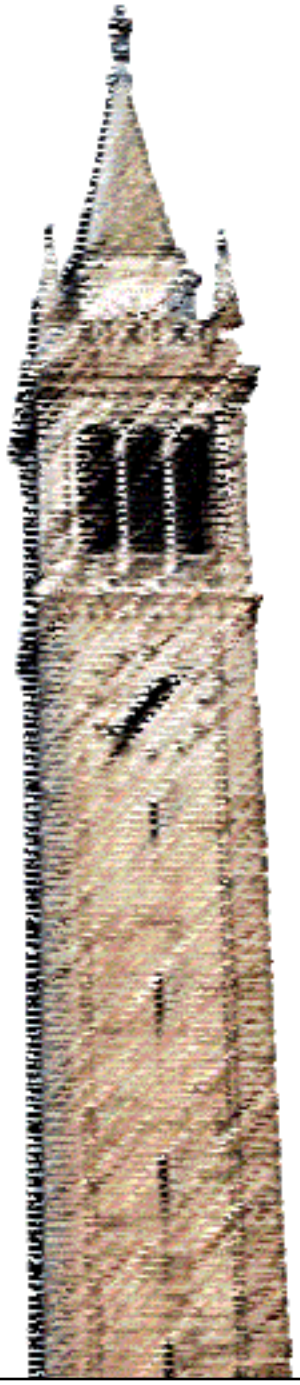
Building a Distributed, GPU based Machine Learning library



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

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-52
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-52.html

May 11, 2016

Pradeep Kalipatnapu

(prad@berkeley.edu)

Capstone Report

Building a Distributed, GPU based Machine Learning Library

Chapter 1: Technical Contributions

Pradeep Kalipatnapu

Introduction: BIDData Project

BIDData is a fast, low-cost Machine Learning library, setting impressive benchmarks for numerous standardized problems (Canny, 2013). BIDData achieves these impressive benchmarks this by making use of the Graphical Processing Unit (GPU) as opposed to traditional libraries which execute exclusively on the Central Processing Unit (CPU). A CPU comprises a single powerful chip, or increasingly up to a dozen, that can perform millions of operations each second. A GPU in contrast consists of hundreds of chips, albeit weaker ones. GPUs are now the most widely available massive parallel platforms available (J. Nickolls and W. J. Dally, 2010). As discussed later in Chapter two, it should come as no surprise that strategically using GPUs for parallelizable computation is gaining popularity. BIDData epitomizes the possible gains from such a shift.

In spite of this, as a library that runs on a single machine BIDData's usability to big data problems is severely limited. Big data ranges in the order of terabytes to petabytes these days, but the size of hard disk on a single machine is capped manyfold smaller at a few hundred gigabytes. This leaves single machines incapable of storing large datasets. As a result, Parallelizing or Distributing data storage and processing, especially using cloud computing, is a common design pattern for dealing with large datasets (Hashem, Yaqoob et. al, 2015).

My team aims to restructure BIDData along this distributed design pattern. This will increase its usability to big data problems. Another important result of such a design change, is the

computational speedup offered by processing on multiple computers. Thus, we will be able to improve upon BIDData's already impressive benchmarks.

There are several parts to achieving this goal. Obviously, we need to make BIDData read/write to a distributed file system, and continue to improve upon its speed by using efficient communication protocols between computers. However, this alone will not lead to widespread adoption of BIDData. We also need to make it accessible, and package it in such a way that it is easy to use. This paper discusses my technical contributions towards all of these ends.

My contribution has centered around converting BIDData from an academic project, to a product that the industry can be interested in. The work i've done, and this paper as a result, can be divided into three broad sections:

Firstly, I discuss using Maven to build BIDData. In order to popularize BIDData we need to make it easy to use, both by individual users and large companies. A large part of this problem is the build system which converts code to an executable that actually runs on computers.

Yiheng and I worked on this together, and we are sharing the reporting responsibilities.
His paper covers in detail, why and how we decided to use Maven as our build system. His paper also talks about how Maven works. I have summarized some of his talking points here.
My paper covers our efforts to adapt Maven for BIDData. I discuss native dependencies, and the challenges of building code that relies on these dependencies.

Secondly, I discuss my contributions towards a distributed setup for BIDData. In the fall semester; I along with my team, learnt about hadoop, a popular file system for large datasets among the machine learning community. We then configured hadoop to work with BIDData. While the team as a whole worked on the early aspects of making BIDData a distributed library, I am reporting on the file system aspect of our work.

Thirdly, in order to convince users of our superior performance, we need to benchmark our competitors. I helped benchmark Petuum, a distributed Machine Learning library from CMU. Petuum, as elaborated later, is in many ways a close competitor to BIDData.

Maven

Maven: Synopsis

Build systems convert programmer written code to deliverables such as a library or an executable. Complex software projects have numerous dependencies. Keeping track of these dependencies is an additional burden that is eased by the use of a build system. It is estimated that a build system setup and maintenance imposes a 12% overhead on developers (Shane McIntosh et. al., 2011).

The choice of programming language affects the choice of build system. BIDData is written in a combination of two popular programming languages, Scala and Java. Maven and Scala Build Tool are the prevailing build systems for these languages. We chose to move from Scala Build Tool to Maven during the course of our project. Section 1.3 of Yiheng Yang's paper discusses the reasons behind our selection of Maven in detail.

Maven has a central repository that is accessible via the internet and serves as a library of dependencies. Some of BIDData's dependencies are already present in the central repository, and we can configure Maven to download them during the build process. For other dependencies, we created our own Maven repository using Bintray. Section 1.4 of Yiheng's paper deal with this process.

Once we specify an exhaustive list of dependencies, Maven is able to build an executable BIDData file that can run on the machine where it was built. Apart from this, our challenge is also to build an executable that can run on other machines, thereby unburdening our users of having to undergo the build process.

Native dependencies

Most people with a personal computer are aware that when downloading some programs, it is important to download the version appropriate for your computer architecture. For example, there are specific versions of softwares for Windows, and MacOS. This is because different computers understand different instruction sets. Java and Scala fix this problem through a layer of indirection. Code written in these languages does not run on the machine directly. Instead it runs on a Java Virtual Machine (JVM), which needs to be installed on the machine. This JVM converts Java code to the appropriate instruction set.

Ideally, we would just build BIDData code for this JVM and it would work for all computer architectures. However, BIDData depends on a set of libraries that are not written in Java and unavailable on the JVM. Such dependencies are called native dependencies, i.e. native to the underlying architecture. There are a variety of reasons for doing so. For example, IOMP is a threading library provided by Intel that is part of its Math Kernel library. This library is about 1.4 gigabytes in size and is not available for free. We cannot reasonably expect our users to have it pre-installed. However, the actual IOMP dependency is only 2.3 megabytes in size, and we have appropriate licenses to distribute it. It is obvious that we need to distribute the iomp dependency in some fashion, preferably using Maven, and make it available to BIDData during execution.

Building with Maven

We have successfully managed to configure maven to handle all of the BIDData compile time dependencies. And we are in the process of dealing with runtime native dependencies.

Prior to this effort, the user would need to download the source code, and run some scripts. These scripts would download the appropriate dependencies, both native and otherwise, after figuring out the machine architecture. The script would then finally build the executable. This executable had to then be run in-place and could not be copied to a different location, because some dependencies were not located inside the executable.

Now, maven automatically downloads appropriate dependencies and builds the BIDData code. The following snippet of maven xml is an example of how the dependency is identified.

```
<dependency>
    <groupId>jline</groupId>
    <artifactId>jline</artifactId>
     <version>2.11</version>
</dependency>
```

Some repositories are already hosted by their authors on maven, in which case the dependency is automatically located. Otherwise, we host it on our own repository, Yiheng Yang's paper explains this process in detail.

```
<repository>
```

```
    <snapshots>
        <enabled>false</enabled>
    </snapshots>
    <id>bintray-biddata-BIDData</id>
    <name>bintray</name>
    <url>http://dl.bintray.com/biddata/BIDData</url>
</repository>
```

Moreover, for each system architecture, we created a compressed (jar) file consisting of all the native dependencies and hosted it on bintray along with other self-hosted dependencies.

```
<profile>
    <id>mac</id>
    <activation>
        <os>
            <family>mac</family>
        </os>
    </activation>
    <dependencies>
        <dependency>
            <groupId>biddata</groupId>
            <artifactId>natives</artifactId>
            <classifier>natives-mac</classifier>
            <version>0.1</version>
        </dependency>
    </dependencies>
</profile>
```

The snippet above adds an additional dependency natvies.jar containing files such as libJCudaRuntime-apple-x86_64.dylib. The profile is automatically detected by maven, so that windows architectures download windows appropriate natives etc.

We use a maven plugin (maven-natives) to unpack these native dependencies and copy them inside the final executable.

```
<plugin>
    <groupId>com.googlecode.mavennatives</groupId>
    <artifactId>maven-nativedependencies-plugin</artifactId>
    <version>0.0.7</version>
    <configuration>
        <nativesTargetDir>natives/</nativesTargetDir>
        <separateDirs>false</separateDirs>
    </configuration>
    <executions>
        <execution>
            <id>unpacknatives</id>
            <phase>generate-resources</phase>
            <goals>
                <goal>copy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

During execution, library loading coding in BIDData copies the native dependency to a temporary location as a separate file and loads the dependency. However, to parse the native files, BIDMat needs to the class definitions of the native dependencies along its classpath. Previously, we would do this at runtime, by downloading the class definitions to a specific location and using a script to run bidmat that would add these dependencies to the class path. Because of this BIDData binaries could not be moved around on the same machine, or to other similar machines. To overcome this problem, we used maven to create an uber jar. The uber jar contains the class definitions inside the jar itself. The following snippet of code does this.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>1.5</version>
    <executions>
        <execution>
        <phase>package</phase>
        <goals>
```

```
            <goal>shade</goal>
        </goals>
        <configuration>
            <artifactSet>
                <excludes>
                    <exclude>biddata:natives:*:*</exclude>
                </excludes>
            </artifactSet>
        </configuration>
        </execution>
    </executions>
</plugin>
```

The final directory structure inside the BIDMat jar looks something like this.

```
BIDMat.jar
----lib
--------libJCudaRuntime-apple-x86_64.dylib
----jcuda
--------runtime
--------.....
----edu
……..
----BIDMat
……...
```

The edu and BIDMat directories contain the java and scala code. The native dependencies are in a subdirectory lib/ and the class definitions are all in their individual directories and can be found along the class path.

Using this process, we come close to distributing BIDData as a single executable file.This grants it a plug and play nature that can be an important factor for a user looking to evaluate our product. It also makes it easier to distribute the executable automatically to numerous production machines in large scale companies.

Ideally, we would prefer not to have different builds for different environments and have a single executable that works on all architectures. One way to achieve this is to write machine independent code. This could happen once more libraries are open source and we can compile their code directly. It could also happen if some of our native dependencies become popular and are supported in the JVM directly (e.g. JVM support for GPUs).

Hadoop

Choice of Hadoop

Most computation in the big data industry is done on commodity machines. Our own project is expected to run on Amazon's EC2 offering. These are machines that fail, regularly and unpredictably. This is a problem for any software, and it presents itself with two possible solutions.The first solution is redundancy. We store the same data on multiple machines. This guarantees that there is no single point of failure, at the cost of more machines or disk space. The second solution is reproducibility. We guarantee that any data or analysis that is lost due to a system crash, can be reproduced. While reproducibility may be viable in certain situations, our "big data" datasets are far too large to be reconstructed quickly. Moreover, commodity machines are cheap at a few cents an hour. Because of these reasons we are looking for a file system that gives us redundancy.

The ubiquitous nature of this problem has resulted in a some ready made solutions. We have decided to use a particularly popular product, the Hadoop File System (also known as HDFS). HDFS offers protection against Hardware Failure, streaming access to data files, supports large data sets and is compatible with various hardware and software platforms (Borthakur 2007).

HDFS architecture can be spatially visualized as a tree. There is a single namenode, which sits at the top of the tree. All communication goes through the namenode. The namenode stores metadata information. The datanodes are scattered throughout the rest of the tree, the store the

actual data. When reading a file from HDFS, the namenode is first contacted. The namenode stores information about which datanodes contain the file we want to read. I use datanodes in plural, because multiple datanodes may contain this file for redundancy reasons.

We have successfully accomplished having BIDData read from HDFS. Currently, we can read and write datasets to HDFS using BIDData.

Reading and Writing from Hadoop

While HDFS can store files in any format, we choose a particular format called *“sequence files”.* These files are easily understood by Spark, and can be partitioned among multiple machines trivially. However, sequence files can only be read linearly, or in sequence, thus the name. For example, In order to access the 10th line of this file, one must read the 10 lines that preceded it. However, the sequence files are partitioned among the Datanodes, and each partition can be read and processed independently for some operations.

Most machine learning approaches (including BIDData) use matrices for storing data. These are files are optimized for random access reads. This enables quickly looking up a small amounts of data, which is critical for BIDData performance. Sequence files make it impossible to do random reads.

We overcame this challenge by using Spark for our cluster computing software. Spark stores significant amount of data in memory using Resilient Distributed Datastores (RDDs). So while we continue to use sequence files on disk, the data that is most actively being read by BIDData, is cached in memory, where it can be randomly accessed.

However, as discussed earlier HDFS only support sequence files. Thus we still need to convert matrix file format into sequence files. Moreover, datasets generally contain more than one matrix files, with implicit correspondence. Corresponding lines of two different matrix files are somehow related to each other. However, HDFS does not guarantee how many partitions will be created when writing a file. This means that two related matrix files could be broken up differently, and stored on different machines.

Converting matrix files to sequence files required some out of the box thinking. We wrote a single sequence file wrapped around all the matrix files. HDFS does not parse the data of the sequence files, it treats them as a series of bytes. Therefore BIDData code needed to be modified to deal with creating this wrapped file, and unwrapping it when reading the sequenced file.

Another important challenge overcame while using Hadoop was the speed of serialization. Since we want to benchmark our end to end performance, we need to read and write to hadoop fast. However, the builtin libraries for serialization in Java are quite slow. Moreover, we needed to work in tandem with the compression algorithms LZ4, Bzip etc that we were already supporting. A creative approach led us to compressing the data before serializing, thus decreasing the amount of time we spent on serialization.

Petuum

Petuum: Synopsis

Petuum is distributed machine learning library from CMU. In many ways it is one of the closest competitor to BIDData. Petuum targets big data problems, and promises to run on commodity machines like EC2 and Google Cloud. It also uses GPU, although this is restricted to their implementation of Neural Networks (a machine learning model not supported by BIDData).

Petuum's alpha version has been available on GitHub since December 2013. One can gauge its popularity from the fact that it has been forked and starred on Github almost twice as much as BIDData's machine learning tool.

Competitive Analysis

While Petuum publishes their own benchmarks, we cannot use them for a direct comparison with BIDData. In many cases, Petuum does not publish results for the same algorithm on the same dataset. For example, our first distributed algorithm is KMeans which we benchmark on MNIST, a handwritten digit classification problem. Petuum only has benchmarks for this algorithm using Logistic Regression. Moreover, we cannot be sure if they were using similarly powerful machines, or the same number of them. For these reasons, we need to benchmark Petuum ourselves.

Since Petuum does not offer a ready made Amazon instance image, or scripts to setup a cluster, we needed to start an ubuntu image from scratch and install Petuum. While BIDData intends to use Apache Spark as underlying infrastructure to run in a distributed fashion, Petuum uses Hadoop Yarn. We setup and configured a yarn cluster comprising of instances with Petuum installed. These machines were identical to the ones running BIDData. Petuum and BIDData accept input in different formats, but we used hadoop to store both these different kinds of files.

We are in the process of working out the exact criteria for the analysis: speed, throughput, cost etc. As well as optimizations that we should be doing on Petuum to make it a fair comparison since we intend to publish the analysis.

References

Ibrahim Targio Hashem, Abaker; Yaqoob, Ibrar; Badrul Anuar, Nor; Mokhtar, Salimah; Gani, Abdullah; Ullah Khan, Samee. “The rise of “Big data" on cloud computing: Review and open research issues”, Information Systems, volume 47, pages 98–115. 2015.

Canny, John and Zhao H. “Big data analytics with small footprint: Squaring the cloud”. 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 95–103. 2013.

Borthakur, Dhruba. ”The hadoop distributed file system: Architecture and design”. Hadoop Project Website, page 21. 2007.

Nickolls, John, and William J. Dally. "The GPU computing era." IEEE micro 2: 56-69. 2010.

McIntosh, Shane; Adams, Bram; Hassan, Ahmed E.. “ The evolution of Java build systems”. Empirical Software Engineering, vol. 17, no. 4, pp. 578-608. 2011.

Chapter 2: Engineering Leadership Analysis

James Jia, Pradeep Kalipatnapu, Richard Chiou, Yiheng Yang

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.

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).

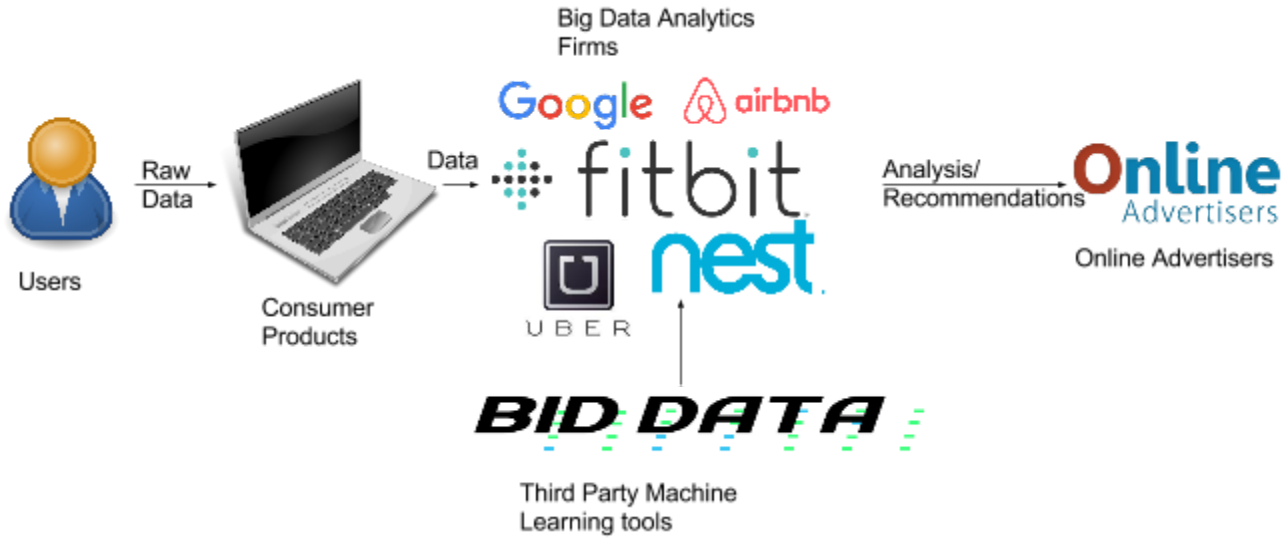
Processing gigantic amounts of information within sub-second time intervals is computationally demanding. Modern computer processing units (CPUs) can process data at 1 billion floating point operations per second, but typical big data sets are on the order of quadrillions of bytes (Intel 2016). Traditional 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.

Industry Analysis

Value Chain Analysis



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.

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.

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.

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).

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.

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.

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.

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.

References

Amazon. “All AWS Customer Stories.” (https://aws.amazon.com/solutions/case-studies/all/). 2016, accessed 5 Mar. 2016.

Canny, John. “BIDData: Benchmarks.” BIDData GitHub repository (https://github.com/BIDData/BIDMach/wiki/Benchmarks). 2015, accessed 10 Feb. 2016.

Canny, J., & Zhao, H. “Bidmach: Large-scale learning with zero memory allocation,” BIGLearn Workshop, NIPS 2013.

Intel. “6th Generation Intel Core i7 Processors.” (https://www-ssl.intel.com/content/www/us/en/processors/core/core-i7-processor.html). 2016, accessed 5 Mar. 2016.

Kahn, Sarah. “2014 IBISWorld Industry Report 51913a: Search Engines in the US,” accessed October 16, 2015.

Lopes, Noel, & Ribeiro, Bernardete. “GPUMLib: An Efficient Open-Source GPU Machine Learning Library,” International Journal of Computer Information Systems and Industrial Management Applications, 2010.

Low, Yucheng, et al. "Distributed GraphLab: a framework for machine learning and data mining

in the cloud," pp. 716-727, P*roceedings of the VLDB Endowment* 5.8, 2012.

Manyika, James, et al. "Big data: the next frontier for innovation, competition and productivity," McKinsey Global Institute, 2011.

Morgan, Timothy Prickett. "Netflix Speeds Machine Learning With Amazon GPUs." EnterpriseTech, 2014.

Porter, Michael E. "The Five Competitive Forces That Shape Strategy." Harvard Business Review, 2008.

Weber, S. *The success of open source* (Vol. 368). Cambridge, MA: Harvard University Press, 2004.