PERFORMANCE BRIEF

Machine Learning

Accelerate Machine-Learning Workloads with Intel® Math Kernel Library

Speed up algorithm performance and insights by up to 7.9x

Table of Contents

Executive Summary

Machine learning—a subset of artificial intelligence (AI)—is quickly entering the mainstream. According to Gartner, machine-learning strategy development and investment is already in the top five CIO priorities.[1] And by 2021, 30 percent of net new revenue growth from industry-specific solutions will include AI.[2] Enterprises are already seeing business value generated from their machine-learning deployments. And the use cases for machine learning are expansive, ranging from detecting fraud, making product recommendations, and automating various business processes and tasks to enabling the next generation of applications such as autonomous cars. Machine learning allows researchers, data scientists, engineers, and analysts to produce reliable, repeatable decisions and results. It can reveal hidden insights through learning from historical relationships and trends in the data. As shown in Figure 1, machine learning running on Intel® architecture speeds up model training time by up to 7.9x.[3]

Accelerating model training time means faster predictive analytics, which can provide a competitive edge in today's fast-paced business environment. The Intel® Math Kernel Library (Intel® MKL), Intel® Advanced Vector Extensions 512 (Intel® AVX-512), and Intel® Xeon® Scalable processors all contribute to machine-learning workload acceleration, without having to make any changes to Apache Spark* code or acquiring specialized hardware.

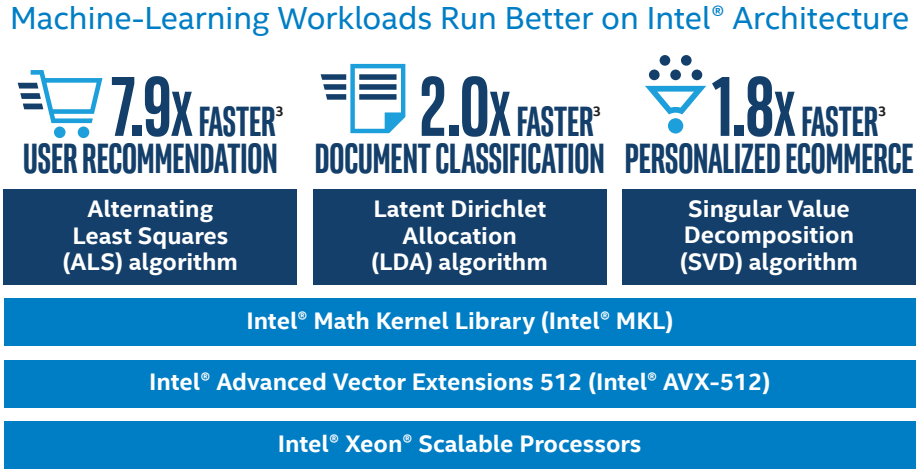

Figure 1. Machine-learning algorithms optimized by Intel® Math Kernel Library and Intel® Advanced Vector Extensions 512 run faster on Intel® Xeon® Scalable processors.

Introduction

Recent advances in machine learning, a branch of artificial intelligence (AI), are driving a surge in global productivity and economic growth. Consider these facts:

- Machine-learning patents grew at a 34 percent compound annual growth rate (CAGR) between 2013 and 2017, the third-fastest growing category of all patents granted.[4]
- IDC forecasts that spending on AI and machine learning will grow from USD 12 billion in 2017 to USD 57.6 billion by 2021.[5]
- Deloitte Global predicts the number of machine-learning pilots and implementations will double in 2018 compared to 2017, and double again by 2020.[6]

Enterprises are already deriving business value from their machine-learning projects, which span a wide variety of use cases such as fraud detection, product recommendations, business process automation and autonomous cars. The growing use of machine learning is tightly correlated with the proliferation of unstructured data and the need for enterprises to remain competitive by improving time to market, driving innovation, and generating real-time insights in a digital economy.

Enterprises of all types and sizes across industries collect vast amounts of data (typically in the petabytes range), analyze it, and make decisions based on the outcome of the analysis. Machine learning is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use this is known as predictive analytics. It is worth noting that the two go hand-in-hand as predictive models typically include a machine-learning algorithm.

For example, product recommendation, a feature on most eCommerce websites, creates models using the Singular Value Decomposition (SVD) algorithm, which learns from historical relationships and trends in the data to suggest products likely to be interesting to a particular consumer. Because machine-learning models and algorithms are compute-intensive, it is important to find ways to speed up the computing process to power faster business and scientific decision making.

Gain Value from Machine Learning with Intel® Math Kernel Library

The Apache Spark* MLlib framework is widely used in machine learning to solve high-value business use cases using its in-memory compute architecture and native linear algebra library integration. Further optimization to the computing process can be achieved using the Intel® Math Kernel Library (Intel® MKL). Intel MKL is a widely used library by enterprises of all types and sizes. It is designed to accelerate math processing routines, increase algorithm performance, and reduce development time. The library includes many features, including linear algebra, vector statistics, data fitting, and neural networks. The performance boost provided with Intel MKL is a major benefit because it does not require a modified version of Spark, or modifications to Spark application code; it also does not require procurement of extra or special hardware. Only a few steps are needed to install Intel MKL on the cluster.

This paper compares the performance of Java*-based f2jBLAS (the default MLlib linear algebra library) and Intel MKL for machine-learning algorithms provided in the Apache Spark MLlib framework.

Overview of Apache Spark* and Basic Linear Algebra Subprograms (BLAS)

Apache Spark is a unified analytics engine for big data processing, with built-in modules for SQL, streaming, machine learning, and graph processing. Apache Spark MLlib is a distributed machine-learning framework and is a leading solution for machine learning on large distributed datasets. Many common machine-learning and statistical algorithms, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives have been implemented and are shipped with MLlib, which simplifies large-scale machine-learning pipelines. Many machine-learning algorithms use linear algebra. Basic Linear Algebra Subprograms (BLAS) are routines that provide standard building blocks for performing basic vector and matrix operations.[7]

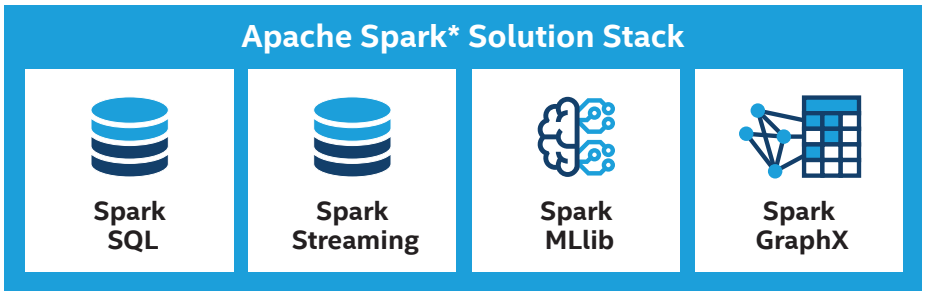

Figure 2. Apache Spark* is a unified analytics engine for big data processing.

Challenges of Apache Spark MLlib BLAS and Linear Algebra Package (LAPACK) Implementations

Figure 3 shows the software hierarchy of Spark MLlib and its integration point with different BLAS and Linear Algebra Package (LAPACK) implementations. Spark MLlib evokes BLAS and LAPACK calls provided by the high-level API which netlib-java* BLAS provides. Individual BLAS and LAPACK libraries can integrate with MLlib by implementing the function calls that netlib-java BLAS exposes to low-level libraries.

f2jBLAS is the default library for Spark MLlib and is implemented in Java; thus, it does not require the Java Native Interface (JNI) to be called from a Java Virtual Machine (JVM) process. Native implementations of BLAS and LAPACK, such as Automatically Tuned Linear Algebra Software (ATLAS), OpenBLAS, and ARnoldi PACKage (ARPACK), are naturally written in low-level languages such as FORTRAN* and C to achieve top performance and called by the JNI within a JVM. However, these libraries are targeted for general CPU use, so their implementations are not fully optimized to use all the CPU instructions provided by a given architecture.

Intel MKL is a BLAS and LAPACK implementation optimized for Intel® Xeon® processors. It offers significant performance advantages over other implementations, as discussed later in this paper. Intel MKL is free for download and is included with Cloudera Distribution for Hadoop* (CDH*).

Test Methodology

Our tests show that Intel MKL significantly speeds up machine-learning algorithms when used with Spark MLlib on Intel architecture, compared to f2jBLAS. We focused on how MLlib can take advantage of the Intel® Advanced Vector Extensions 512 (Intel® AVX-512) on Intel Xeon Scalable processor to accelerate BLAS computations. The elapsed time of algorithms computed using f2jBLAS serve as the baseline metric.

The tests used the spark-perf machine-learning benchmark (see Appendix B for detailed test steps).[8] We selected six popular machine-learning algorithms for our MLlib tests:

- Alternating Least Squares (ALS)
- Principal Component Analysis (PCA)
- Latent Dirichlet Allocation (LDA)
- Singular Value Decomposition (SVD)
- Logistic regression
- Linear regression

Appendix B contains information on hardware, software, workload, and CDH service configuration for our tests. Appendix C includes optimization and tuning information.

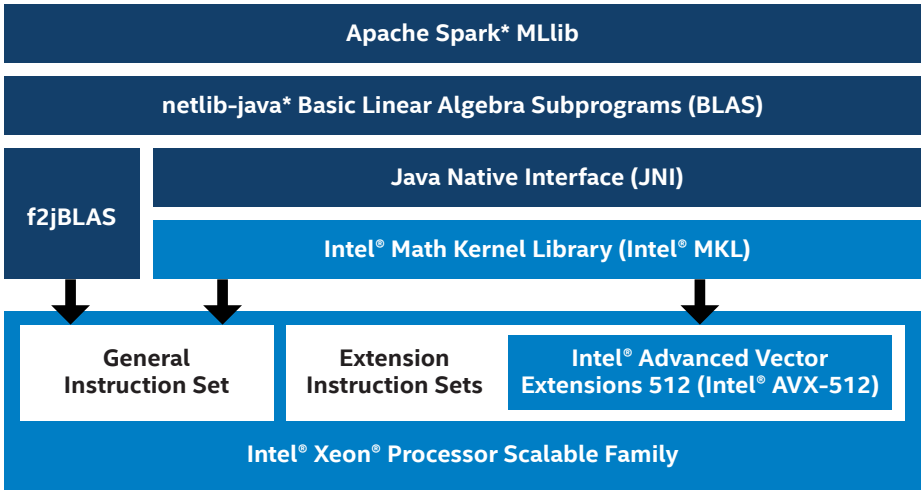

Figure 3. The Apache Spark* MLlib software stack and associated BLAS implementations can take advantage of Intel® architecture and the Intel® Math Kernel Library to accelerate machine-learning algorithms.

Results

Figure 4 shows the absolute values for training-time performance for the different machine-learning workloads we tested with f2jBLAS and Intel MKL. The y-axis shows training time in seconds (log scale) and the x-axis represents the machine-learning workload that is included in the Spark MLlib framework. The time measured is reported by the spark-perf benchmark and does not include time taken to read data from the Hadoop Distributed File System* (HDFS*). A lower training time is better.

In Figure 5, ALS using Intel MKL shows an approximate 7.9x speedup of training time compared to the MLlib default library (f2jBLAS). The other algorithms also benefit from using Intel MKL, gaining from 1.08x to 2.0x compared to f2jBLAS.

As seen in Figure 5, the training-time performance speedup varies for different algorithms. The following discussion explains what is happening with each algorithm.

ALS shows the largest speedup for training time (7.9x) from using Intel MKL compared to using f2jBLAS, because it has more than 548 million BLAS calls that are routed to Intel MKL, which takes advantage of the Intel AVX-512 instruction-set architecture available on Intel Xeon Scalable processors to accelerate vector operations.[9] PCA and SVD have approximately five million calls to the level 2 DSPR subroutine, and accordingly show a 66 to 76 percent performance gain compared to f2jBLAS, due to the matrix operation optimizations from Intel MKL and the vector operation optimizations from Intel AVX-512. LDA also invokes millions of DSPR calls and shows a 2.0x speedup from Intel MKL compared to f2jBLAS.

In contrast, logistic regression and linear regression have a limited number of level 1 BLAS function invocations. This is the reason for the maximum performance gains in ALS training times. Since the ALS algorithm makes approximately 548 million BLAS calls, there is a noticeable performance improvement with respect to run time. See Table A1 in Appendix A for information on how often level 1, 2, and 3 BLAS subroutines were called in the different machine-learning workload that we tested. For detailed workload configurations, refer to Table B3 in Appendix B.

7.9x Intel® MKL can speed up machine-learning training by up to 7.9 times compared to default BLAS libraries[3]

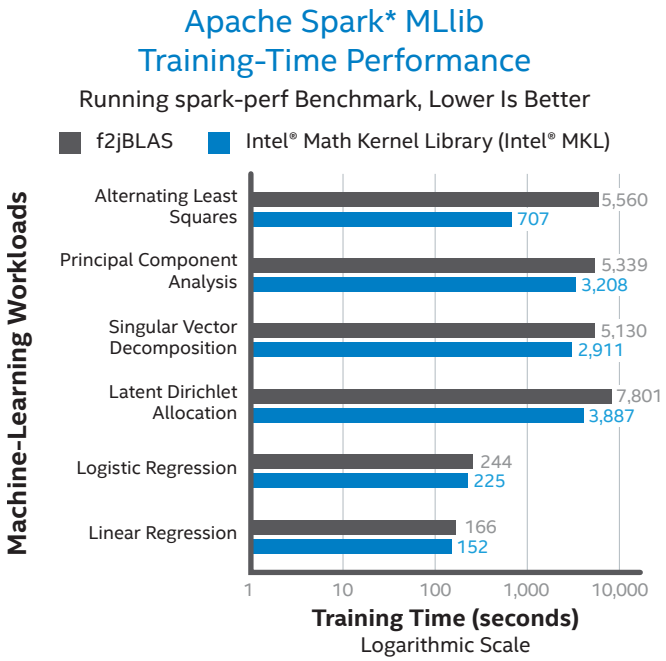

Figure 4. spark-perf benchmark results, comparing Intel® Math Kernel Library to MLlib's default Basic Linear Algebra Subprograms (BLAS) implementation.

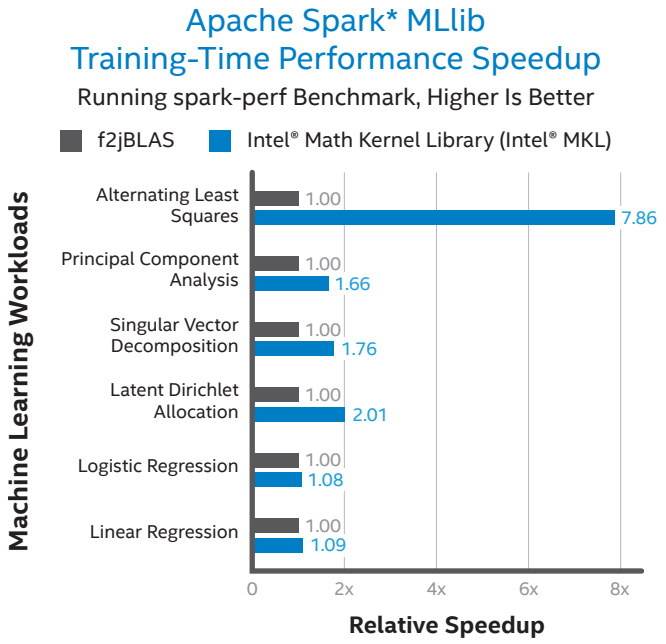

Figure 5. Intel® Math Kernel Library speedup for various machine-learning algorithms.

7.9x Increase in Machine-Learning Performance with Intel MKL

Modern, successful enterprises require real-time solutions that can keep pace with the accelerating volume of big data. Machine-learning systems have recently demonstrated superhuman performance in domains as diverse as recognizing objects in images, detecting fraud, diagnosing disease, making product recommendations, and even playing poker.[10] In our tests, the ALS recommendation engine algorithm demonstrated an exceptional 7.9x speedup of training time when using Intel MKL, compared to using f2jBLAS.[3] Faster training time means faster analytics—all of which lead to near-real-time or real-time insights that can enhance the customer experience and drive new revenue.

While ALS showed the greatest performance improvement, the other machine-learning algorithms we tested also benefited, with a range of 1.08x to 2.0x speedup of training time with no extra cost associated with cluster resources.

By taking advantage of Intel AVX-512, Intel MKL accelerates machine learning without requiring any modifications to MLlib source code or the purchase of specialized hardware. Intel MKL makes it possible to train with larger data sets, explore a larger range of the model hyperparameter space, and train more models. Enterprises can use Intel MKL to find new and interesting patterns, retrain existing models in real time as new data becomes available, increase development agility, and shorten time to insight.

Find the solution that is right for your organization. Contact your Intel representative or visit **Intel® AI Academy**.

Real-World Use Cases for Machine-Learning Algorithms

Being able to train machine-learning models faster can provide a competitive edge. Consider the following examples for LDA, ALS, and SVD:

LDA can be applied to many areas, including brand monitoring, customer insights, retail marketing analytics, and social media analysis. For example, Algorithmia offers LDA as part of its "Analyze Tweets*" service: **Introduction to Twitter Topic and Sentiment Analysis.**

ALS is a recommendation algorithm that helps make automatic predictions about customers' interests—a cornerstone of modern eCommerce, eBanking, and e-almost-anything-else. See how Spotify is using ALS: **Music Recommendations at Scale with Spark**.

SVD is a collaborative filtering algorithm that uses matrix factorization; it has broad applicability across many disciplines. For example, a team of researchers used SVD to **predict product duration for adaptive advertisement**.

Learn More

You may find the following resources useful:

- Reference Architecture: Machine Learning-Based Advanced Analytics Using Intel® Technology
- Intel® Xeon® Scalable Processors
- Intel® Math Kernel Library
- Intel® Advanced Analytics
- BLAS Information
- spark-perf benchmark

Appendix A: Libraries for Comparison

We compared the following two libraries:

- **f2jBLAS**, which is the default BLAS library for MLlib. The primary use for f2j is to provide numerical linear algebra software originally written in FORTRAN* as Java*-class files. The numerical libraries are distributed as class files produced by an f2j translator. The f2j translator is a formal compiler that translates programs written using a subset of FORTRAN 77 into a form that can be executed on JVMs.
- **Intel® Math Kernel Library**, which optimizes code with minimal effort for future generations of Intel® Xeon® processors. It is compatible with a wide variety of compilers, languages, operating systems, and linking and threading models. It features highly optimized, threaded, and vectorized math functions that maximize performance on each processor family. In addition, it uses industry-standard C and FORTRAN APIs for compatibility with popular BLAS, LAPACK, and Fastest Fourier Transform in the West (FFTW) functions, with no code changes required. Intel Math Kernel Library dispatches optimized code for each processor automatically without the need to branch code and is optimized for single-core vectorization and cache utilization.

Table A1. Function Calls Profiled from Fommil_Netlib_Java*

Workload	BLAS Functions	Number of Times Invoked
Alternating Least Squares (ALS)	DSPR	149,554,032
	SSCAL	36,895,513
	DAXPY	150,885,368
	DDOT	51,881,970
	SNRM2	41,648,800
	DPPSV	117,335,569
Principal Component Analysis (PCA)	DSPR	4,999,989
	DAXPY	2,214
Singular Value Decomposition (SVD)	DSPR	4,999,971
	DAXPY	1,107
Latent Dirichlet Allocation (LDA)	DSPR	62,616,544
	DGEMV	603,201
	DAXPY	488,123,049
	DCOPY	1,427,614,037
Logistic Regression	DAXPY	111,650
Linear Regression	DDOT	12,480,819
	DAXPY	2,233

Appendix B: Test Configuration Information

This appendix describes the test steps and provides configuration details.

Test Steps

We used the following methodology for our Apache Spark* MLlib tests:

1. Build spark-perf once.
2. Configure hyper-parameters in workload settings (see Table B3).
3. Generate data and store it in HDFS*.
4. For each of the two libraries (f2jBLAS and Intel® Math Kernel Library):
   a. Run three iterations of each workload.
   b. Record training time from benchmark output.
   c. Collect and process data collected by the Performance Analysis Tool (PAT). (PAT automates benchmark execution and data collection and uses Linux* performance counters.)

The following tables provide information about the hardware, software, workload, and Cloudera Distribution for Hadoop* configuration used in our Spark MLlib machine-learning tests. Click here for additional hardware configuration details.

Table B1. Hardware Configuration

Component	Management Node	Master Node	Worker Nodes
Number of Nodes	1	1	6
Processor	Intel® Xeon® Platinum 8168 processor (2.70 GHz)	Intel Xeon Platinum 8168 processor (2.70 GHz)	Intel Xeon Platinum 8168 processor (2.70 GHz)
Sockets	Dual-socket	Dual-socket	Dual-socket
Core Count		48	48
Thread Count		96	96
Base Frequency		2.7 GHz	2.7 GHz
Turbo Frequency		3.7 GHz	3.7 GHz
Hyper-Threading Enabled	Yes	Yes	Yes
Memory	256 GB - 8x 32 GB DDR4 2,666 MHz RDIMM	384 GB - 12x 32 GB DDR4 2,666 MHz RDIMM	384 GB - 12x 32 GB DDR4 2,666 MHz RDIMM
Network	Intel® Ethernet Connection X722 for 10G BASE-T, Speed: 10000 Mb/s	Intel Ethernet Connection X722 for 10G BASE-T, Speed: 10000 Mb/s	Intel Ethernet Connection X722 for 10G BASE-T, Speed: 10000 Mb/s
OS Drive	Intel® SSD DC S3700 Series (800 GB, 2.5-inch SATA 6 Gb/s, 25nm, MLC)	Intel SSD DC S3700 Series (800 GB, 2.5-inch SATA 6 Gb/s, 25nm, MLC)	Intel SSD DC S3700 Series (800 GB, 2.5-inch SATA 6 Gb/s, 25nm, MLC)
Data Drive	n/a	NameNode: 2x 2.5" Seagate ST2000NX0403* HDD 2 TB SATA 6 Gb/s 7200RPM 128 MB buffer	DataNode: 8x 2.5" Seagate ST2000NX0403 HDD 2 TB SATA 6 Gb/s 7200RPM 128 MB buffer
Tiered Storage (Block I/O)	n/a	n/a	Intel SSD DC P3520 Series (2 TB, 1/2-Height PCIe* 3.0 x4, 3D NAND G1, MLC)

DC = Data Center **HDD = hard disk drive** **MLC = multi-level cell** **SSD = solid state drive**

Table B2. Software Configuration

Component	Management Node
OS	CentOS Linux* release 7.3.1611 (Core) (inclusive of the patch for Spectre and Meltdown)
Kernel	3.10.0-693.11.6.el7.x86_64
Java* Development Kit (JDK)	Java HotSpot* 64-bit Server VM (build 25.131-b11, mixed mode)
Apache Spark*	1.6.0
Cloudera Distribution for Hadoop* (CDH*)	CDH 5.11.0, Parcels
Apache Hadoop*	Hadoop 2.6.0-cdh5.11.0
Apache Hive*	Hive 1.1.0-cdh5.10.0
Intel® Math Kernel Library	2018.0.082
spark-perf	Version 1 - github.com/databricks/spark-perf

Table B3. Workload Configuration

Component	Dataset Configuration Size	Dataset Size	Configuration Parameters
Alternating Least Squares (ALS)	num-users=100M num-products=100M num-ratings=50M rank=400	1.7 GB	inter-trial-wait=10 random-seed=5 rank=400 reg-param=0.1 num-partitions=1140 num-iterations=1
Latent Dirichlet Allocation (LDA)	num documents=60M num-vocab=60K num-topics=80 document-length=500	140.6 GB	inter-trial-wait=10 random-seed=5 num partitions=1140 num-iterations=5 optimizer=online
Singular Value Decomposition (SVD)	num-rows=5M num-columns=8K rank=4K	286.6 GB	inter-trial-wait=10 random-seed=5 num-partitions=1140 num-iterations=5 optimizer=online
Principal Component Analysis (PCA)	num-rows=5M num-columns=8K rank=50	286.6 GB	inter-trial-wait=10 random-seed=5 num-partitions=1140 num-iterations=5 optimizer=online
Logistic Regression	num-examples=1.2M num-features=100K	1.1 TB	inter-trial-wait=10 random-seed=5 num-partitions=1140 num-iterations=50 step-size=0.001 reg-type=l1 reg-param=0.1 optimizer=sgd per-negative=0.3 loss=logistic elastic-net-param=0.01 feature-noise=0.1
Linear Regression	num-examples=10M num-features=10K	711.3 GB	inter-trial-wait=10 random-seed=5 num-partitions=1140 num-iterations=50 step-size=0.001 reg-type=l1 reg-param=0.1 optimizer=sgd per-negative=0.3 loss=logistic elastic-net-param=0.01 feature-noise=0.1

Table B4. Cloudera Distribution for Hadoop* (CDH*) Service Configuration

Service	Management Node 1	Master Node	Worker Nodes (1-6)
HDFS* NameNode		✓	
HDFS Secondary NameNode		✓	
HDFS DataNode			✓
YARN* MR2 Included Resource Manager		✓	
YARN MR2 Included Job History Server		✓	
YARN MR2 Included Node Manager			✓
Spark* History Server		✓	
Spark Gateway		✓	✓
Hive* Server2		✓	
Hive Metastore Server		✓	
Hive Gateway		✓	✓
ZooKeeper*		✓	
Cloudera Management Service Activity Monitor	✓		
Cloudera Management Service Alert Publisher	✓		
Cloudera Management Service Event Server	✓		
Cloudera Management Service Host Monitor	✓		
Cloudera Management Service Report Manager	✓		
Cloudera Management Service Service Monitor	✓		

Appendix C: Optimizations and Tunings

The following sections describe BIOS, OS, Apache Hadoop*, and Apache Spark* tunings.

BIOS Tuning

We chose "Performance" for the CPU Power and Performance Policy.

OS Tuning

- **Disable Transparent Huge Pages**
  - `echo never > /sys/kernel/mm/transparent_hugepage/defrag`
  - `echo never > /sys/kernel/mm/transparent_hugepage/enabled`
- **Disable Swapping**
  - `swapoff –a`
- **Set Scaling Governor to Performance**
  - `/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor`
- **Directory Mount Options**
  - `Set Directory Mount Options ext4  defaults,noatime`
- **Enable Jumbo Frames.** The OS uses the maximum transmission unit (MTU) to control the maximum size of a packet or frame sent over TCP. By default, MTU is set to 1500 but you can adjust its value upwards to a maximum of 9000. When the MTU value is greater than its default value, this is called "Jumbo Frames." To change the MTU value, add `MTU=9000` in `/etc/sysconfig/network-scripts/ifcfg-eth0` or whatever your eth device name is. You must restart the network service before the change takes effect.
- **Open File Handles and Files.** By default, the maximum number of open files is set to 1024 for each user. However, we found the default value resulted in a java.io.FileNotFoundException (Too many open files) and our jobs failed. To avoid this scenario, we set the open file limit to 32832, for both hard and soft limits, as follows:
  - `ulimit -Sn 32832`
  - `ulimit -Hn 38232`

Apache Hadoop Tuning

We used most of the default YARN* and Spark settings, recommended by Cloudera Distribution for Hadoop* (CDH*), except for the ones listed in Table C1.

Table C1. Apache Hadoop* Tuning Settings

YARN*	Property	Value
Resource Manager	yarn.scheduler.maximum-allocation-mb	380 GB
	yarn.scheduler.minimum-allocation-mb	1 GB
	yarn.resourcemanager.scheduler.class	Fair Scheduler
Node Manager	Container memory	380 GB
	yarn.nodemanager.resource.cpu-vcores	96
Gateway	mapreduce.map.memory.mb	3.9 GB
	mapreduce.reduce.memory.mb	3.9 GB
	Client Java Heap Size in Bytes	3 GB
	mapreduce.map.java.opts.max.heap	3 GB
	mapreduce.reduce.java.opts.max.heap	3 GB
	mapreduce.task.io.sort.mb	512 MB
	mapreduce.map.output.compress.codec	org.apache.hadoop.io.compress.SnappyCodec
	mapreduce.output.fileoutputformat.compress.codec	org.apache.hadoop.io.compress.SnappyCodec
Hive* Gateway	Client Java Heap Size in Bytes	2 GB

Apache Spark Tuning

Table C2 shows the general tuning we used for Spark, as well as settings related to Intel® Math Kernel Library.

Table C2. Apache Spark* Tuning

Property	Value
Overall Apache Spark* Tuning	
spark.executor.memory	15850 MB
spark.driver.memory	340 GB
spark.executor.cores	5
spark.memory.fraction	0.9
spark.driver.maxResultSize	256 GB
spark.scheduler.mode	FAIR
spark.yarn.executor.memoryOverhead	4096
spark.scheduler.allocation.file	/etc/spark/conf/fairscheduler.xml
spark.executor.extraJavaOptions	-XX:+UseG1GC -XX:MaxGCPauseMillis=100 -XX:ParallelGCThreads=51
spark.driver.extraJavaOptions	-XX:+UseG1GC -XX:MaxGCPauseMillis=100 -XX:ParallelGCThreads=51
spark.serializer	org.apache.spark.serializer.JavaSerializer
Settings Related to Intel® Math Kernel Library	
spark.executor.extraClassPath	/opt/intel/mkl/wrapper/mkl_wrapper.jar
spark.driver.extraClassPath	/opt/intel/mkl/wrapper/mkl_wrapper.jar
spark.executor.extraJavaOptions	-Dcom.github.fommil.netlib.BLAS=com.intel.mkl.MKLBLAS -Dcom.github.fommil.netlib.LAPACK=com.intel.mkl.MKLLAPACK
spark.driver.extraJavaOptions	-Dcom.github.fommil.netlib.BLAS=com.intel.mkl.MKLBLAS -Dcom.github.fommil.netlib.LAPACK=com.intel.mkl.MKLLAPACK
spark.executorEnv.OMP_NUM_THREADS	1
spark.driverEnv.OMP_NUM_THREADS	1
spark.yarn.appMasterEnv.OMP_NUM_THREADS	1

Solution Provided By:

[1] Gartner, October 2017, "Top 10 Strategic Technology Trends for 2018." gartner.com/ngw/globalassets/en/information-technology/documents/top-10-strategic-technology-trends-for-2018.pdf

[2] Gartner, 2017, "The Business Impact and Use Cases for Artificial Intelligence." gartnerinfo.com/apacemergingtechtaipei/TheBusinessImpactandUseCasesforAI_TracyTsai.pdf

[3] For test configuration details refer to Appendix B.

[4] IFI Claims Patent Services, January 2018, "8 Fast Growing Technologies." ificlaims.com/rankings-8-fast-growing.htm

[5] IDC, September 2017, "IDC Spending Guide Forecasts Worldwide Spending on Cognitive and Artificial Intelligence Systems to Reach $57.6 Billion in 2021." idc.com/getdoc.jsp?containerId=prUS43095417

[6] Deloitte, 2018, "2018 Global TMT Predictions." deloitte.com/global/en/pages/technology-media-and-telecommunications/articles/tmt-predictions.html#

[7] For more information on Basic Linear Algebra Subroutines (BLAS) and BLAS wrappers, refer to the following sources of information: Data Science Made Simpler; GitHub's netlib-java page; Netlib.org's BLAS page

[8] For information about all the algorithms in the spark-perf suite, refer to spark.apache.org/docs/1.6.1/mllib-guide.html

[9] For more information about using Intel® Advanced Vector Extensions 512, refer to "Instruction Set Specific Dispatching on Intel® Architectures" at software.intel.com/en-us/mkl-linux-developer-guide-instruction-set-specific-dispatching-on-intel-architectures

[10] Financial Times, July 2018, "Machine learning will be the engine of global growth." ft.com/content/133dc9c8-90ac-11e8-9609-3d3b945e78cf

1018/BGOW/KC/PDF 337180-001US