Intersection of HPC and Machine Learning

Kadupitiya Kadupitige, UID: 2000253911, Uname: jasakadu
ENGR-E 687 IND STUDY INTEL SYS: FINAL REPORT

***Abstract*— Parallel machine learning focused on employing high performance technologies to enhance the performance of advanced machine learning algorithms in data mining frameworks. Scaling up such frameworks has been shown to enhance the performance in benchmark tasks and to enable discovery of complex high-level features. Conversely auto-tuning of multicore based cross platforms often utilizes machine learning models to obtain the best performance as well. Auto tuning explores the search space defined by code transformations, compiler flags, architectural features, optimization parameters, and etc, by engaging state of art machine learning techniques to speedup the optimization process which could otherwise take many machine months to explore exhaustively. This research presents a study focused on intersection of HPC and machine learning technologies and how those are utilized in big data applications.**

Keywords - Big data applications, machine learning, high performance computing, auto-tunning

I. Introduction

Enormous development in hardware technology and computer architecture have enabled complex machine learning methodologies to be applied to solve difficult real world problems using parallel programming models. Parallel machine learning aims to capitalize on the simultaneous execution of sophisticated machine learning algorithms using modern hardware architectures. A significant amount of effort has been put into employing High Performance Computing (HPC) technologies in to big data frameworks and salable deep learning systems which are highly focused on reliability as well as performance [1, 2].

In light of these trends, there exists a close connection between machine learning and high performance computing as machine learning algorithms has been employed to perform auto tuning of multicore based cross platforms for performance optimization [3] and HPC technologies has been utilized in many bigdata frameworks to enhance the performance and throughput [1, 4, 5, 6]. Auto tuning could be able to produce a high quality output which is many times faster than the basic implementation. The state of the art approaches involves statistical machine learning (SML) based methodologies using a single application, a single compiler, a specific set of compiler flags and homogeneous cores to perform the auto tuning [3]. On the other hand, state of the art HPC approaches in big data frame works involves both shared memory parallelization (symmetric multiprocessing) and distributed memory parallelization (communicating sequential processing) [5].

This research tries to identify how machine learning and HPC technologies are employed in corresponding frameworks to provide a comparison of existing auto tuning methodologies and big data frameworks.The organization of the paper is as follows. In section II, a literature review on intersection of HPC and machine learning is presented. Section III provides a detailed description of the existing big data frameworks appeared in the literature review. Conclusion is elaborated in sections IV.

II. Literature Review

Parallel machine learning abstractions have long been assayed and employed. Most of the research carried out in parallel machine learning area, focused on performance enhancement of bigdata or deep learning frameworks [1, 4, 5, 6]. There exits other researches which are focused on a auto tuning of multicore based cross platforms [3, 7, 8, 9]. This study investigate these parallel machine learning techniques, and hence, they are discussed in this section.

A. HPC technologies in big data frameworks

Adam et al. [1] have developed a system based on commodity off the shelf high performance computing (COTS HPC) technology. This system was a cluster of GPU servers with Infiniband interconnections and MPI [10]. As it was described, every server spawns one process for each of its GPUs and an identification number (”rank”) was assigned by MPI implementation [1]. Their approach was to divide up the computational work among the GPUs in the cluster and then organize their communication using MPI by enabling a strictly model parallel scheme as each GPU is responsible for a separate portion of the computation, but all of the GPUs compute the same mini-batch of input samples until synchronization happens. Sparse autoencoder learning algorithm was used as the deep learning algorithm on 10 million YouTube video thumbnails data set(200-by-200 pixel region) [1]. Researchers have illustrated that their system was able to train 1 billion parameter networks using only 3 machines and 11 billion parameters using 16 machines in a couple of days to claim the scalability of their system [1].

Recent research done by Peng et al. [4] has focused on implementing a novel synchronized Latent Dirichlet Allocation (LDA) machine learning system, HarpLDA+ based on Hadoop and Java for topic modeling and data analysis [11, 12]. New mechanisms have been proposed to reduce the overhead in synchronized systems considering efficiency and effectiveness of the system. Dynamic scheduling was employed to reduce the overhead in the shared memory model while Pipe-lining and timer control methodologies were introduced for model rotation framework which comes

under the distributed memory model [4]. They have explained that HarpLDA+ distributed memory model utilizes a Java collective communication library released as a plugin for Hadoop. Researchers have compared the performance of the HarpLDA+ against existing LDA algorithms such as LightLDA, F+NomadLDA, and WarpLDA and claim that HarpLDA+ was able to outperform all other compared algorithms due to the optimization of the synchronization and communication overhead [4].

A research done by Saliya et al. [5] have studied the parallel machine learning on the Java Virtual Machine (JVM) using an Intel Haswell HPC cluster with 24 or 36 cores per node. They have mainly focused in employing bulk synchronous parallel (BSP) model in to existing Long Running Jobs Fork Join (LRT-FJ) to java shared memory parallel computing in order to optimize the Java parallel programming model so that it can match the performance of traditional programming model in C [5]. Researchers have considered three major factors such as thread models (2 models), affinity patterns (6 patterns) and communication mechanisms(2 mechanisms). LRT-FJ and Long Running Threads Bulk Synchronous Parallel (LRT-BSP) were considered for the thread models and an interesting observations are made such as context switches, CPU migrations, dTLB misses are comparatively lower in LRT-BSP [5]. Core, Socket and All were considered for process affinity variations while Inherit and Explicit per Core were included as thread affinity variations allowing these researchers to have 6 different affinity variations. They have also implemented a memory maps-based communication layer to improve the Java interprocess communication [5]. Researchers have focused on two parallel machine learning logarithms : K means clustering (6 variations) and Multidimensional Scaling (MDS - 2 variations) for the performance analysis [5]. They have concluded the research by elaborating that their LRT-BSP approach could be implemented in to big data frameworks such as Apache Beam, Flink and Spark to improve the existing collective communications [5, 13, 14, 15].

A framework for parallel machine learning (GraphLab) was designed and implemented by Yucheng et al.[6] for aiding machine learning experts to avoid from repeatedly solving the same design challenges. Researchers have developed GraphLab to identify and utilize the common pattern in machine learning such as MapReduce by compactly expressing asynchronous iterative algorithms with sparse computational dependencies while maintaining data consistency and achieving a high degree of parallel performance[6]. As they have explained their approach contained a data graph which represents the data and computational dependencies, a update functions which describe local computation, a Sync mechanism for aggregating global state, a data consistency model which determines the extent to which computation can overlap and a scheduling primitives which express the order of computation [6]. Researchers have implemented parallel versions of belief propagation, Gibbs sampling, Co-EM, Lasso and Compressed Sensing to show that GraphLab performs really well with realworld large scale problems [6].

B. Machine Learning for auto tuning of multicore based cross platforms

A case for machine learning to optimize multi-core performance has studied by Ganapathi et al.[3] to enhance the state of art auto-tuning approaches as hand-tuning is neither salable nor portable. As researchers claimed existing auto tuning approches are scalable, automatic, and produce high-quality code but it suffers from two major drawbacks such as size of the parameter space to explore (only a single application, a single compiler, a specific set of compiler flags, and homogeneous cores would have 40 million configurations) and most existing auto-tuners only focus to minimize overall running time not the efficiency (ex: energy and power consumption) [3]. They have proposed a solution which utilizes statistical machine learning (SML) approaches to infer models from large quantities of data as SML based methodologies do not rely on domain or application specific knowledge. Kernel Canonical Correlation Analysis (KCCA) was used as the SML algorithm that effectively identifies the relationship between a set of optimization parameters and a set of resultant performance metrics [3]. Researchers have reported that they were able to reduce the six month long search time to two hours on 7-point and 27-point stencil code [3].

Bergstra et al. [7] have employed machine learning for predictive auto-tuning of the Filterbank correlation kernel with boosted regression trees. They have identified two major state of art approaches to auto-tuning such as empirical auto tuning which is generic approach that works by measuring runtimes of implementations, and model-based approach which determines those runtimes using simplified abstractions [7]. They have used three sets of variables such as task description (argument shapes, physical layout), implementation description (auto-tuning parameters), platform description (capabilities, micro-benchmarks) inside their auto-tuning routine. Researchers have claimed that their approach is not specific to filterbank correlation, nor to GPU kernel auto-tuning, and can be extended to support most code optimization task such as wide variety of problem types, kernel types, and platforms [7].

Recent research done by Yigitbasi et al. [8] have focused on employing machine learning-based auto-tuning for diverse MapReduce applications and cluster configurations in Hadoop framework. Researchers have shown that support vector regression model (SVR) has good accuracy and is also computationally efficient for performance modeling of MapReduce applications [8]. Researchers have compared the exiting Starfish auto-tuner (a cost-based model) to the SVR based approach and reported comparable and in some cases even better performance [8]. Liao et al. [9] also have focused on optimizing MapReduce in Hadoop framework using a search-based machine learning model (instead of the existing cost-based approach) called Gunther. Their approach utilizes a Genetic algorithm designed to identify parameter settings contributes to near-optimal job execution time [9].

As described above, interaction between machine learning

and HPC can be studied under two categories such as adaptation of machine learning techniques for code optimization in big data frameworks and HPC techniques used in Big data frameworks [1, 4, 5, 6, 3, 7, 8, 9]. This research is focused on studying these techniques in the identified big data frameworks such as Hadoop (Harp for collective communication), Apache Beam, Apache flink, Apache Spark according to the literature review [11, 12, 13, 14, 15].

III. Review of Big Data Frameworks

A. Hadoop and Harp

The Apache Hadoop is a framework that allows for the distributed processing of big data across clusters of computers using simple programming models [11]. It is modeled and implemented to scale up from single node to thousands of nodes, each offering local computation and storage. This frame work offers Hadoop MapReduce library to support parallel processing of large data sets [11]. Harp is a collective communication library which is plugged into Hadoop in order to trasfer MapReduce jobs to Map-Collective jobs to enhance the efficient in-memory collective communication directly in map tasks [12]. Simlar to Hadoop, Harp also consist of three levels such as application, framwork and resource manger as shown in figure 1. BSP model (Map-Collective) in the Harp library is shown in the figure 2. Pig K-means and Pig PageRank are both compared against Hadoop MapReduce and Harp Map-Collective to report the comparatively better results obtained using collective communications [12, 16].

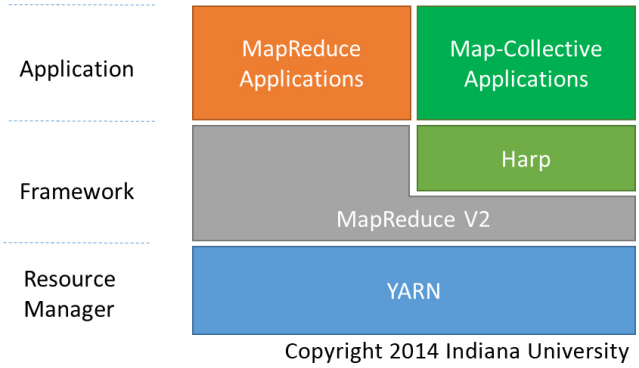

Fig. 1. Architecture of Harp

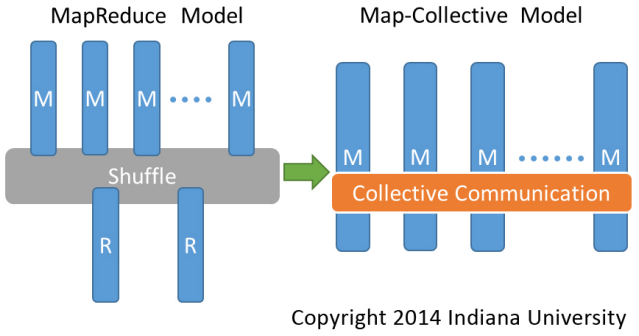

Fig. 2. Parallelism Model for Harp

B. Apache Beam

According to Apache documentation, Beam is an unified model for defining both batch and streaming data-parallel processing pipeline [13]. Apache Beam supports distributed processing back-ends such as Apache Apex, Apache Flink, Apache Spark, and Google Cloud Dataflow [14, 15, 17]. Beam currently supports the Java Python SDKs and there are three main data processing pipelines such as Apache Spark, Apache Flink, or Google Cloud Dataflow representing back-ends. The Beam model is a successor to MapReduce and is focused on providing a unified solution for batch and stream processing. The Apache Beam architecture is shown in figure 3. Programming model in beam consist of three major sections such as Pipelines, PCollections and Transforms. Pipelines refers to the data processing step while PCollections represents the data inside the Pipeline. Transform is a operation such as core transforms, composite transforms and IO transforms [13].

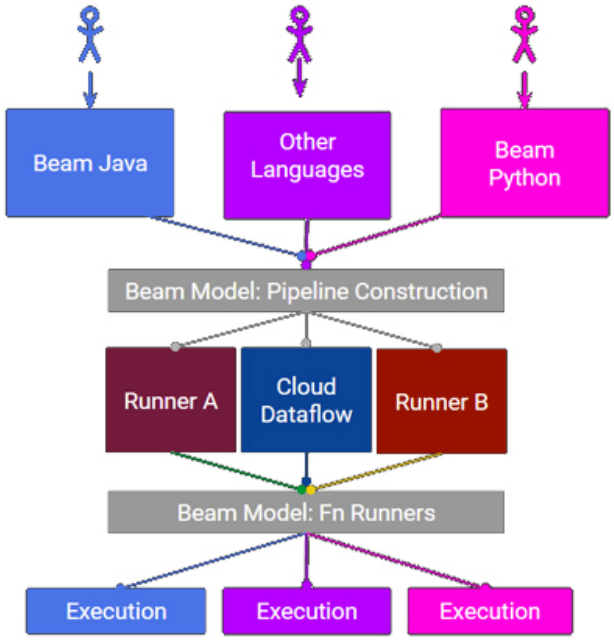

Fig. 3. Architecture of Apache Beam

C. Apache Flink

Apache Flink is an open-source stream processing engine which supports distributed in-memory data processing for big data applications. Flink currently supports programming APIs in Java, Scala, Spargel, and python for both Unbounded (infinite datasets that are appended to continuously) and Bounded (finite, unchanging datasets) data models [14]. Flink supports two execution models such as streaming : executes continuously as long as data is being produced, and batch: executes and runs to completeness in a finite amount of time, releasing computing resources when finished. Flink

provides few unique advantages for data processing such as it provides results that are accurate, even in the case of out-of-order or late-arriving data, it is stateful and fault-tolerant and can seamlessly recover from failures and it performs at large scale [14]. Components stack of Apache Flink is shown in figure 4. A Flink program Can be studied under three major states such as data source: incoming data that Flink processes, transformations: data processing step, and data sink: where Flink sends data after processing [14]. Apache Flink feature data flow model and the execution model also happens in data flow style.

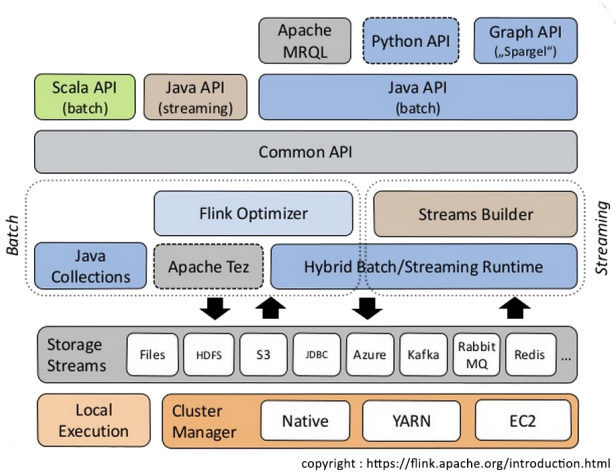

Fig. 4. Components stack of Apache Flink

D. Apache Spark

Apache Spark is a fast and general engine which perform tasks up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk for high speed large-scale streaming, machine learning and SQL workloads tasks [15]. Comparison of runtimes of Logistic regression in Hadoop and Spark is shown in figure 5. Spark offers to program the applications employing over 80 high-level operators using Java, Scala, Python, and R. Components stack of Apache Spark is shown in figure 6. As shown in the figure, Spark powers the combined or standalone use of a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming [15]. Spark can be utilized in standalone cluster mode, on EC2, on Hadoop YARN, or on Apache Mesos and it allows data access in HDFS, Cassandra, HBase, Hive, Tachyon, and any Hadoop data source [15].

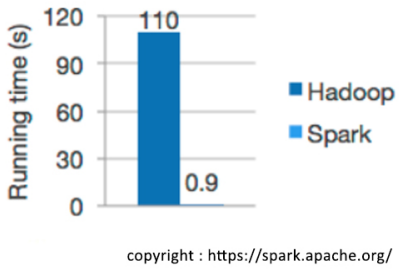

Fig. 5. Logistic regression in Hadoop and Spark

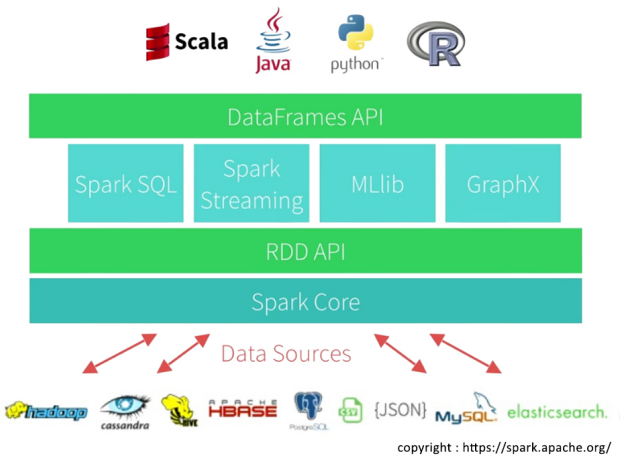

Fig. 6. Components stack of Apache Spark

IV. Conclusion

This research focused on identifying intersection of machine learning and HPC technologies in big data applications. Most of the big data frameworks are using data flow execution model while others utilizes BSP model (MPI). According to the literature review, it is evident that adaptation of BSP model in to big data frameworks for parallel machine learning has enhanced the performance than the data flow execution model. Many researchers have reported the inefficiencies of famous big data frameworks (Spark and Flink) such as inability to support nested loops, lack of all-to-all collective operations (using reduce operations followed by a broadcast operation creates additional overheads), etc. Factors like thread models, affinity patterns and communication patters also plays a major role in performance when it comes to parallel machine learning. Even though, most of these big data processing framworks utilizes HPC concept, there is a plenty of room for improvement in terms of communication and synchronization overheads.

References

[1] Adam Coates et al. "Deep learning with COTS HPC systems". In: *International Conference on Machine Learning*. 2013, pp. 1337–1345.

[2] Philip K Chan and Salvatore J Stolfo. "Toward Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection." In: *KDD*. Vol. 1998. 1998, pp. 164–168.

[3] Archana Ganapathi et al. "A case for machine learning to optimize multicore performance". In: *First USENIX Workshop on Hot Topics in Parallelism (HotParfffdfffdfffd09)*. 2009.

[4] Bo Peng et al. "HarpLDA+: Optimizing Latent Dirichlet Allocation for Parallel Efficiency". In: ().

[5] Saliya Ekanayake et al. "Java thread and process performance for parallel machine learning on multicore hpc clusters". In: *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE. 2016, pp. 347–354.

[6] Yucheng Low et al. “Graphlab: A new framework for parallel machine learning”. In: *arXiv preprint arXiv:1408.2041* (2014).

[7] James Bergstra, Nicolas Pinto, and David Cox. “Machine learning for predictive auto-tuning with boosted regression trees”. In: *Innovative Parallel Computing (InPar), 2012*. IEEE. 2012, pp. 1–9.

[8] Nezih Yigitbasi et al. “Towards machine learning-based auto-tuning of mapreduce”. In: *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*. IEEE. 2013, pp. 11–20.

[9] Guangdeng Liao, Kushal Datta, and Theodore L Willke. “Gunther: Search-based auto-tuning of mapreduce”. In: *European Conference on Parallel Processing*. Springer. 2013, pp. 406–419.

[10] William Gropp et al. “A high-performance, portable implementation of the MPI message passing interface standard”. In: *Parallel computing* 22.6 (1996), pp. 789–828.

[11] Tom White. *Hadoop: The definitive guide*. ” O’Reilly Media, Inc.”, 2012.

[12] Bingjing Zhang, Yang Ruan, and Judy Qiu. “Harp: Collective communication on hadoop”. In: *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE. 2015, pp. 228–233.

[13] *Apache Beam: An advanced unified programming model*. `https://beam.apache.org/`. Accessed: 2017-12-24.

[14] Paris Carbone et al. “Apache flink: Stream and batch processing in a single engine”. In: *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36.4 (2015).

[15] Apache Spark. *Apache Spark: Lightning-fast cluster computing*. 2016.

[16] Tak-Lon Wu, Abhilash Koppula, and Judy Qiu. “Integrating Pig with Harp to support iterative applications with fast cache and customized communication”. In: *Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds*. IEEE Press. 2014, pp. 33–39.

[17] SPT Krishnan and Jose L Ugia Gonzalez. “Google cloud dataflow”. In: *Building Your Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 255–275.