2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)

# The Vision of a HeterogeneRous Scheduler

Ioannis Mytilinis, Constantinos Bitsakos, Katerina Doka, Ioannis Konstantinou and Nectarios Koziris
Computing Systems Laboratory, National Technical University of Athens, Greece
{gmytil,kbitsak,katerina,ikons,nkoziris}@cslab.ece.ntua.gr

***Abstract*—Modern Big Data processing systems, scheduling platforms and cloud infrastructures employ specialized hardware accelerators such as GPUs, FPGAs, TPUs, ASICs, etc. to optimize the execution of resource intensive workloads such as Machine Learning, Artificial Intelligence or generic Data Analytics tasks. Nevertheless, this support is mostly a user-dependent, manual process that requires careful and educated decisions on both the amount and type of required resources to exploit the underlying hardware and achieve any user-defined higher level policies. In this work we present the initial design of the HeterogeneRous Scheduler (HRS), an intelligent scheduler that can make automated decisions on both how and where to map arbitrary data analytics tasks to underlying cloud hardware that may consist of a mix of hardware accelerators and clusters with general purpose CPUs. We experimentally evaluate the performance trade-offs between hardware accelerators and CPUs where we show that there are cases where one technology outperforms the other. We finally present an initial architecture of HRS where we depict its different components and their interactions with the Big Data framework and the cloud infrastructure.**

## I. Introduction

In the past years, we have witnessed an explosion in the amount of data created and consumed worldwide, with an estimation of the world's digital universe data to exceed 44 zetabytes by 2020 [1]. On the one hand the three Vs of the Big Data landscape (i.e., volume, velocity and veracity) along with the need to extract meaningful insights from the collected data has resulted in a number of different big data analytics software systems. These systems are able to scale to thousands of nodes and perform resource-hungry data intensive tasks ranging from simple descriptive OLAP-like analysis (e.g., Spark SQL [2], Hive [3], Presto [4], Impala [5], Flink [6] etc.) to more advanced predictive analytics such as Machine Learning (ML) and Artificial Intelligence (AI) (e.g., Google's Tensorflow [7], Spark MLlib [8], etc.) in a fast and accurate manner.

On the other hand, the fact that very popular algorithms in the Machine Learning (ML) and Artificial Intelligence (AI) domains can be efficiently executed over specialized hardware has increased the popularity of hardware accelerators such as GPUs, FPGAs and Google's TPUs [9]. These devices combine their massive execution parallelism capabilities with the inherently data parallel nature of the aforementioned workloads and they are capable of "crunching" large amounts of data in lightning speed, outperforming general purpose CPU-based implementations in ML and AI workloads both in terms of speed and accuracy [10].

The need to access and exploit the computing capabilities of heterogeneous hardware in a massive manner has also changed the typical model of cloud infrastructures in two ways: First, most modern cloud vendors offer access to a suite of specialized hardware accelerators on a pay-as-you-go manner alongside general purpose CPUs (e.g., Amazon's EC2 Elastic GPUs[1] or FPGA instances[2], Google's TPU[3]). Second, typical cloud scheduling software systems such as Apache Yarn [11] and Mesos[4] that are employed to coordinate complex task execution pipelines are starting to offer support for heterogeneous hardware through their API over bare metal, Virtual Machines or even Docker containers [12].

Most modern Big Data systems like Apache Spark and Apache Hadoop [11], Flink [13], TensorFlow [7], etc. offer the opportunity to leverage accelerators and benefit from their increased performance. They are able to execute different implementations of typical data intensive calculations over various hardware including GPUs, FPGAs and TPUs in the form of different *kernels*, i.e., routines compiled for high throughput accelerators, separate from (but used by) a main program. Kernel examples used in Big Data settings include algorithms such as a Stochastic Gradient Descent optimization or a matrix multiplication, typically encountered in a Machine Learning Training. Kernel code must be implemented (i.e., ported) for the specific available hardware using accelerator-dependent software development frameworks such as OpenCL and CUDA, as opposed to the rest of the CPU-based code, developed using higher level languages supported by the chosen Big Data framework (typically Java, Scala or Python).

Users seem to have all the puzzle pieces: Big Data frameworks that support the use of hardware-accelerated kernels, cluster management tools that can allocate them to the underlying hardware and cloud offerings that provide such heterogeneous infrastructures. In reality, however, putting the pieces together is not enough to always satisfy the user requirements for increased performance and low cost. This is due to the lack of an intelligent decision making process that can advise users to make the best out of the available resources. Indeed, as we observe in Section II, offloading parts of the code to specialized hardware is not always the "silver bullet" that can "automagically" solve fast and efficiently any typical Big Data workload calculation: factors such as data transfer time between RAM and GPUs, different algorithmic characteristics (one-pass vs multi-pass), dataset sizes etc. play a very impor-

[1] https://aws.amazon.com/ec2/elastic-gpus/
[2] https://aws.amazon.com/ec2/instance-types/f1/
[3] https://cloud.google.com/tpu/
[4] https://www.nvidia.com/object/apache-mesos.html

978-1-5386-7899-2/18/$31.00 ©2018 IEEE
DOI 10.1109/CloudCom2018.2018.00065

tant role in the end-to-end execution time or monetary cost, rendering the scheduling decision a cumbersome process.

In this work we present the initial design of the *HeterogeneRous*[5] *Scheduler (HRS)*, an intelligent, hardware-aware scheduler that can make educated decisions on how to map a given workflow of data analytics tasks to the appropriate underlying hardware for execution in order to optimize for any user-defined policy (e.g., maximize performance, minimize cost). The contributions of our work are the following:

- We experimentally evaluate the performance trade-offs in a heterogeneous environment consisted of CPUs and GPUs. Our study unravels a wide space for optimization and validates the need of HRS, a scheduler capable of allocating heterogeneous resources in an efficient manner.
- We present the preliminary architecture of our approach, describing in detail the internal components of HRS as well as its interaction with the Big Data framework, the resource scheduler and the heterogeneous infrastructure.

In the remainder of this paper, Section II motivates our problem with specific workload executions and algorithm implementations over heterogeneous devices, Section III presents a preliminary architecture of our proposed scheduler, Section IV summarizes related work and, finally, Section V concludes the paper.

## II. Motivation

In this Section, we describe a real-life scenario that can benefit from HRS and discuss the importance of heterogeneous systems for data processing. The described application is driven by actual business needs and has been specified in the context of the EU-funded E2Data project[6].

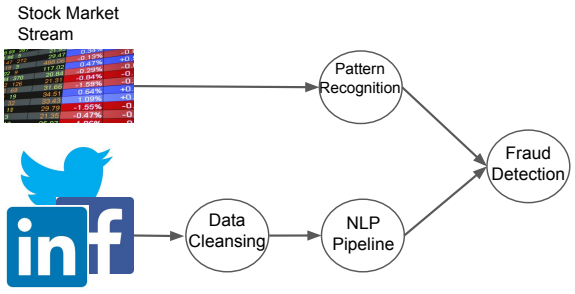



Fig. 1: Fraud-detection application graph

Let us consider a stream-processing engine that is used for real-time fraud detection in a stream of stock market orders. The identification of a fraud is performed based on the recognition of certain patterns in the stock time-series, correlated also with relevant information in articles and textual content in the social media. Such an application involves various algorithms from different domains like data cleansing, text-analytics and pattern recognition. As the output of an algorithm may be needed as input for another one, different tasks interact with each other through well-defined data-dependencies. This way, we can imagine the application as a graph of tasks where data flow among its vertices.

Figure 1 presents the application graph for the fraud detection case. Data is gathered from social media sources and is fed to a data cleansing operator. Subsequently, the cleaned text is processed by a series of natural language processing (NLP) operators. At the same time, a stream of stock market time-series is mined for patterns of interest. The results of the pattern recognition task are joined with the annotated text and a proprietary algorithm is applied for detecting frauds.

As the task described above should take place in real-time, it is highly sensitive to latency issues. For optimizing its performance, the development team conducted an experimental analysis in order to identify the compute intensive tasks that can potentially comprise a bottleneck in the data flow and tried to offload their computation to hardware accelerators. The employed NLP algorithms are embarrassingly parallel and need only one-pass over the data. Moreover, calls to the GPU can be asynchronous, masking out data transfer delays to and from the device. As such, a GPU implementation could bring a considerable speedup. Some preliminary results verified this speculation and proved GPU to be the ideal target platform for the specific task.

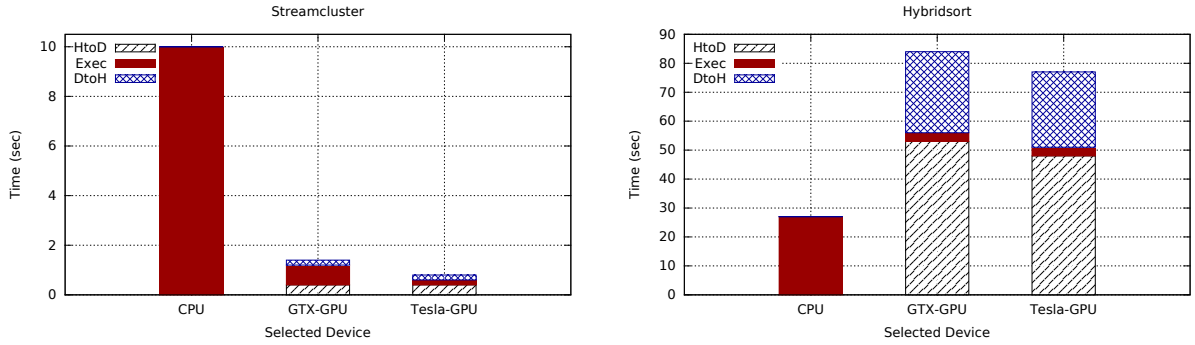
As implicitly mentioned, apart from compute efficiency, data transfers also affect the task-device mapping decision. In order to showcase our point, we experimented with the Rodinia [14] benchmark. Rodinia is highly referenced in the literature and contains OpenCL and OpenMP[7] implementations for a plethora of algorithms. We conducted experiments on three different platforms:

- a multicore CPU system with 40 × Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz processors and 256 GB of main memory
- a Tesla V100-SXM2 GPU with 32GB of memory
- a GeForce GTX 1060 GPU with 6GB of memory

Results for the Streamcluster and Hybridsort algorithms are presented in Figure 2. Streamcluster is a clustering algorithm for n-dimensional points, while Hybridsort sorts an array of floats by performing a combination of the bucket-sort and merge-sort algorithms. *HtoD* denotes the time needed to copy data from the host's memory to the device, while *DtoH* denotes the data copy to the opposite direction. We observe that both algorithms can be efficiently parallelized, and execution-wise a GPU can offer great speedups. However, we point out that as I/O operations can have a severe impact, processing time by itself does not guarantee optimal end-to-end performance.

In the case of Streamcluster, processing dominates total execution time and leads the fastest GPU to be the device of preference. On the other hand, in the Hybridsort case, there is a considerable amount of time spent on copying data. Although the pure execution time is much smaller in the GPUs, the user is going to experience a better end-to-end performance if the multicore CPU system is selected.

[5]Pun intended.
[6]https://e2data.eu/
[7]https://www.openmp.org/




(a) Streamcluster (b) Hybridsort

Fig. 2: Performance of Rodinia Hybridsort and Streamcluster for various devices

TABLE I: Heterogeneous Data Processing Systems

| System | Devices | Application |
|---|---|---|
| Caffe [15] | CPU/GPU | ML |
| Torch7 [16] | CPU/GPU/FPGA | ML |
| Theano [17] | CPU/GPU | ML |
| Tensorflow [7] | CPU/GPU/Mobile/TPU | ML |
| MXNet [18] | CPU/GPU/Mobile | ML |
| Dandelion [19] | CPU/GPU | Ananlytics |
| LINQts [20] | Mobile/FPGA | Analytics |
| Spark [2] | CPU/GPU | ML/Analytics |

The aforementioned examples highlight the need for systems of heterogeneous hardware. This need has already been recognized and there are many systems that give the user the option of which hardware platform should be employed for each task. Table I shows the available compute platforms for some well-known machine-learning and Big Data analytics frameworks. However, in the majority of existing systems, we meet one of the following problems:

- It is the user who needs to specify the target device for each task.
- Scheduling ignores the application graph and works at the task level.
- There is no information on the way scheduling works.

In this work, we argue that the manual tuning of an application can be a really costly and time-consuming process. Thus, we opt for an open-source scheduler that can automatically identify the optimal mapping between tasks and devices in order to minimize the execution time of the whole workflow. Given the code of an application and a set of available resources, HRS will be able to understand which tasks best fit the execution model of each hardware platform and will schedule them according to a global optimization function.

As HRS does not come with a specific Big Data framework, it is important to be easily integrated with any state-of-the-art data processing platform. In the following Section, we present a modular design that will ensure the interoperability of HRS with systems like Apache Spark, Flink, etc.

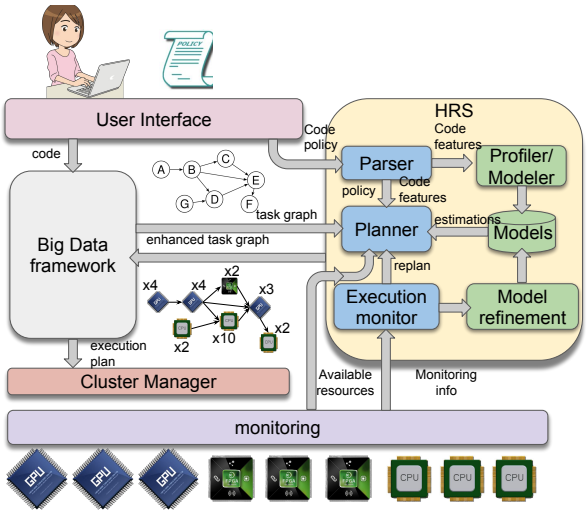



Fig. 3: Components of the Heterogeneous Resource Scheduler and their interactions.

## III. HRS Preliminary Architecture

HRS aims to automatically optimize and plan the execution of complex Big Data workflows containing both compute and data intensive operators over a pool of heterogeneous resources. In a nutshell, the business logic of the workflow is reflected in code through the use of any Big Data framework of choice. Given the Directed Acyclic Graph (DAG) of involved tasks and based on performance and cost estimations for each one of them, HRS optimizes for any user-defined optimization policy by mapping tasks to the available heterogeneous infrastructure and by making informed decisions on the type and amount of resources to be allocated. The final execution plan is communicated to the Big Data framework, which enforces it.

Figure 3 depicts the preliminary architecture of HRS, including its internal components and their interaction with other internal or external system components.

At an abstract level, the user submits through a user

interface the code developed using her favorite Big Data framework, along with the desired optimization policy (e.g., minimize execution time, minimize power consumption or both). The code is passed to the Big Data execution engine, where tasks are internally organized in a graph structure which we will henceforth call *task graph*. At this point, it is the responsibility of HRS to instruct the Big Data framework where to execute each task of the task graph. Having information about: (a) the code itself along with the optimization objectives form the user interface, (b) the task graph from the Big Data framework and (c) monitoring information about the available infrastructure, HRS can produce the optimal execution plan, which allocates each task to the most beneficial set of hardware resources among the available ones.

The basis of this workflow optimization process lies in the utilization or detailed models of the cost and performance characteristics of Big Data tasks over various underlying hardware, be it CPUs, GPUs or FPGAs. The models are stored and updated in a model library. Whenever a new workflow is run atop HRS, these models are used in order to intelligently assign and orchestrate workflow parts to the available hardware according to the user optimization policy.

Once the optimal execution plan is available, the Big Data framework enforces it through a cluster management framework that can handle heterogeneous resources (e.g.,YARN, Mesos). During runtime, the workflow execution is being monitored for failures and/or performance degradation. In that case, HRS dynamically adapts to the current conditions by creating a new execution plan for the remaining tasks.

Delving into the internal of the proposed system, HRS consists of two layers: (1) the *Intelligence* layer, depicted in green, a Machine Learning modeling framework that derives hardware- and elasticity-related knowledge from both offline profiling and dynamic runs; and (2) the *Hardware-Aware Planner*, depicted in blue, that communicates with the Intelligence layer and performs the decision making of how, where, and when to schedule code for execution on the available hardware resources.

In a nutshell, the modules of HRS cooperate towards the optimization of Big Data workflow executions with respect to the policy provided by the user. The **Planner** determines in real-time where each task is to be run, under what amount of resources provisioned and whether data need to be moved to/from their current locations and between processing units. Such a decision must rely on the characteristics of the involved tasks, derived as code features by the **Parser**, and the underlying hardware they are executed upon. These are modeled and stored within the **Models library**. The initial task models result from the offline profiling through the **Profiler/Modeler** module that directly interacts with the pool of physical resources and the monitoring layer in-between. Moreover, while the workflow is being executed, the initial models are refined in an online manner by the **Model Refinement** module, using monitoring information of the actual run. This mechanism allows for dynamic adjustments of the models and enables the Planner to base its decisions on the most up-to-date knowledge. Next we describe each of the internal HRS components in a more detailed manner.

### A. Parser

This module parses the user-provided code and extracts code features such as density and computational or memory intensity that will be used as input for the model training process. Moreover, it validates the user-defined policy.

### B. Profiler/Modeler

This module produces a number of different ML models that describe the behaviour of each hardware processing unit, in terms of performance, cost, energy efficiency etc., when executing Big Data tasks. The descriptive models are initially trained using profiling in an offline mode, as well as machine learning over actual runs.

The profiling input includes: (a) data-specific parameters which describe the data to be used for the operator profiling (e.g., the type of data and its size) and (b) resource-related parameters, which define the resources to be considered during profiling (e.g., type of hardware, number of cores, amount of available disk/memory, etc.).

The output of each run is the profiled performance and cost, such as completion time, throughput, power consumption, etc., under each combination of the input parameter values. The collected metrics along with the code-specific parameters, i.e., the features parsed by the Parser, are then used to create estimation models, making use of neural networks, SVM, interpolation and curve fitting techniques.

### C. Models Library

This module consists of a number of different ML models that describe the behaviour of each hardware processing unit, in terms of performance, cost, energy efficiency etc., when executing Big Data tasks.

### D. Model Refinement

This module exploits the experience collected during runtime to augment the existing performance models and increase their efficiency. To that end, monitoring information is fed back to the existing models to dynamically adapt them to the current infrastructure conditions that might include changes in the underlying hardware like hardware upgrades or temporal degradation due to various reasons including collocation of competing tasks, surges in load etc. Thus, the accuracy of the models remains high regardless of possible changes.

### E. Planner

The Planner is the core component of HRS which intelligently explores all the possible execution plans over the available heterogeneous infrastructure and discovers the most beneficial one with respect to the user-defined optimization objectives. The Planner takes as input (a) the task graph from the Big Data framework, (b) the code features of each task of the task graph through the Parser, (c) the available resources through the infrastructure monitoring layer and (d) the cost and performance estimation for each task over the various available

hardware setups consulting the machine learning models of the Models library. To find the optimal execution plan, the planner relies on exhaustive search to explore all possible allocations of individual tasks to hardware resources to find the plan that optimizes the global optimization policy. The optimization policy can include a single or multiple objectives, meaning that one or a set of Pareto execution plans will be discovered respectively. While an exhaustive search would provide optimal solutions, it would only be practical for small workflow instances, since the size of combinations to be checked grows exponentially with the number of task graph nodes. Thus, to be able to accommodate large and complex workflow instances within a reasonable time-frame, HRS can employ heuristics based on genetic algorithms to find near-optimal solutions.

### *F. Execution Monitor*

The responsibility of this module is to inspect the smooth execution of the optimal plan through the monitoring of the underlying infrastructure. Failures and performance degradation must be detected early on to make the necessary adjustments in a timely manner. The detection of such anomalies trigger the re-planning of task executions according to the new circumstances in order to make amends. Remedial actions include the compilation of the code for execution over a different type of hardware and elastic contraction/expansion of provisioned resources.

## IV. Related Work

Over the last years many systems that exploit heterogeneous architectures have emerged. The related work is organized along two axes: The first concerns Big Data processing frameworks that exploit accelerators and support execution over heterogeneous hardware, while the second includes scheduling approaches that work in heterogeneous environments.

The well-known Google's TensorFlow [7], a system used for large-scale machine learning purposes, supports heterogeneous execution. TensorFlow represents computations and operations as a dataflow graph of tasks, where each task can be executed on a system's device (CPU, GPU or TPU). It is open-source and its only limitation is that it is specifically used for machine learning applications.

GFlink [13] is an in-memory computing architecture on heterogeneous CPU-GPU clusters, an extension on the original Apache Flink framework [6]. GFlink manages to exploit the high performance of GPUs and to deal with the issue of data transfer between CPU and CPU by proposing the GDST a framework that automates the communication of the JVM based Flink and the GPU device. Furthermore it introduces the GWork Sheduler, a scheduler which uses a locality-aware scheduling algorithm and a locality-aware work stealing algorithm. GFlink is not open-source.

HeteroSpark [21], a GPU-accelerated heterogeneous architecture integrated with Apache Spark [22] and GPU-Spark [23], a CPU-GPU hybrid data analytics system that enables Spark to utilize GPUs parallel processing ability manage to demonstrate sizeable performance improvements to original Spark implementation. However they both struggle with data transfer and communication between the heterogeneous system devices, since data that need to be processed by GPUs must be transferred from JVM memory to native memory and finally to GPU memory.

LINQits: [20], Dandelion [19] support heterogeneous execution on CPU, GPU or FPGAs. The issue that occurs though is how to find the optimal scheduling to map application tasks to available resources.

In [24] the authors propose the HCI scheduler that accepts as input the application DAG (Directed Acyclic Graph). Expressing applications as dataflow graphs is the most common programming model in cloud systems, where nodes represent computations and edges data dependencies between them. HCI uses task runtime estimations, I/O estimations and also an estimation of future resource availability to find the optimal scheduling by testing all possible variations of task to resource mapping.

The authors of TetriSched [25] tested a different approach. They formulated the task to resource mapping and their execution time as a MILP (Mixed Integer Linear Programming) problem, in which the solver finds the optimal job scheduling. TetriSched is DAG-oblivious though, meaning that it is not aware of the task relations in the application.

In [26] the authors formulated the problem of scheduling using a machine learning approach. They do not represent the application code as a task graph, but rather they extract features from the application code, which they use to feed an SVM [27] model that after trained is able to predict execution times for unseen programs and place their execution on the most suitable device.

Qilin [28] is an experimental heterogeneous programming system that uses *adaptive mapping*, a fully automatic technique to map computations to heterogeneous devices. It uses the application DAG to build an analytical performance model based on static analysis. It stores information that it has collected in its lifetime which it then uses to make a projection of a new program's estimated execution time.

The work in [29] proposes a static solution build on the OpenCL framework. After the compilation of a program the system extracts static code features which it then uses to feed an SVM model which predicts execution times for different devices.

## V. Conclusions

In this work we presented the initial design of HRS, a HeterogeneRous scheduler that can automatically select the appropriate type and size of resources to execute data analytics tasks over heterogeneous cloud platforms. Our motivation stems from both real-life scenarios and experimental observations. These observations showcased that there is no clear winner in the choice of hardware architecture between general purpose CPUs and hardware accelerators, since it depends on numerous factors such as algorithm implementation, dataset

size, hardware specifications, user policies, etc., HRS is capable of automatically deciding, deploying and managing complex data analytics task workflows over a cloud-enabled infrastructure that consists of different hardware accelerators and clusters of general purpose CPUs. We showcased its initial architecture that is based on a modular design and we have depicted its interactions with external components such as the Big Data Framework, the cloud infrastructure and the user.

## ACKNOWLEDGMENT

This work has been supported by the European Commission in terms of the H2020 E2Data Project (780245).

## REFERENCES

[1] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, "The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things," *IDC Analyze the Future*, vol. 16, 2014.

[2] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[3] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: a Warehousing Solution over a Map-Reduce Framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[4] "Apache Presto," https://prestodb.io/.

[5] M. Bittorf *et al.*, "Impala: A modern, open-source sql engine for hadoop," 2015.

[6] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache flink: Stream and batch processing in a single engine," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 36, no. 4, 2015.

[7] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[8] X. Meng *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[9] "Forecast: Discrete GPUs, Worldwide, 2016-2022, 2q18 Update." [Online]. Available: https://www.gartner.com/doc/3871682/forecast-discrete-gpus-worldwide-

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] W. Tan and V. K. Vavilapalli, "First Class GPUs support in Apache Hadoop 3.1, YARN & HDP 3.0," Aug. 2018. [Online]. Available: https://hortonworks.com/blog/gpus-support-in-apache-hadoop-3-1-yarn-hdp-3/

[12] "Enabling GPUs in the Container Runtime Ecosystem," Jun. 2018. [Online]. Available: https://devblogs.nvidia.com/gpu-containers-runtime/

[13] C. Chen, K. Li, A. Ouyang, Z. Zeng, and K. Li, "GFlink: An In-Memory Computing Architecture on Heterogeneous CPU-GPU Clusters for Big Data," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1275–1288, Jun. 2018.

[14] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. Ieee, 2009, pp. 44–54.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[16] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS workshop*, no. EPFL-CONF-192376, 2011.

[17] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proc. 9th Python in Science Conf*, vol. 1, 2010.

[18] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *arXiv preprint arXiv:1512.01274*, 2015.

[19] C. J. Rossbach, Y. Yu, J. Currey, J.-P. Martin, and D. Fetterly, "Dandelion: A compiler and runtime for heterogeneous systems," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 49–68. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522715

[20] E. S. Chung, J. D. Davis, and J. Lee, "Linqits: Big data on little clients," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 261–272. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485945

[21] P. Li, Y. Luo, N. Zhang, and Y. Cao, "Heterospark: A heterogeneous cpu/gpu spark platform for machine learning algorithms," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2015, pp. 347–348.

[22] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[23] Y. Yuan, M. F. Salmi, Y. Huai, K. Wang, R. Lee, and X. Zhang, "Spark-gpu: An accelerated in-memory data processing engine on clusters," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016, pp. 273–283.

[24] M. Kaufmann and K. Kourtis, "The hcl scheduler: Going all-in on heterogeneity," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. Santa Clara, CA: USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/hotcloud17/program/presentation/kaufmann

[25] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters," in *EuroSys*, 2016.

[26] A. Hayashi, K. Ishizaki, G. Koblents, and V. Sarkar, "Machine-learning-based performance heuristics for runtime cpu/gpu selection," in *PPPJ*, 2015.

[27] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998.

[28] C.-K. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 45–55, 2009.

[29] D. Grewe and M. F. P. O'Boyle, "A static task partitioning approach for heterogeneous systems using opencl," in *Compiler Construction*, J. Knoop, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 286–305.