Big Data with High Performance Computing (HPC)

Tarek El-Ghazawi, Krunal Puri, and Armin Mehrabian

George Washington University

Abstract Abstract goes here

Keywords: ...

1 Introduction

With the recent emergence of Big Data over the past few years, one of the questions that arises is, how Big Data is different from High Performance Computing (HPC) at various levels such as application, technology, investment, etc. Another critical question to be asked is if Big Data is widely different from HPC, which of the technologies may we want to adopt or, do we really have to choose one from the two to apply to our problems? Another scenario is that the gap between Big Data and HPC may form a spectrum of choices rather than two independent worlds. To answer these questions in this chapter we will first bring an introduction and look at Big Data and HPC from a high level abstraction view. Then, we explain the state of the art efforts to take advantage of the best of two worlds. Finally, we will address future challenges and future directions.

In order to answer the above questions we first may need to identify the workloads and computational requirements of workflows for our applications. It is also important to not focus only current trends, and take into account the future data and computation demands. Figure 1 shows the matrix of data size versus computation size for some applications. It also shows that the current big data technologies, formed around Apache Big Data Stack (ABDS), is data-intensive but focused less on computational performance. On the other hand the High Performance Computing (HPC), which mostly applied to solving scientific problems, has been mostly concentrated on computation-savvy, data limited application such and iterative simulations.

Over the past decade, the rate of data generation has dramatically increased. As a result, we saw the emergence of data-intensive applications to solve a range of problems from data distribution and management to data processing. The open-source Apache Big Data Stack with it's Hadoop Data File System (HDFS) kernel and many higher abstraction level frameworks for data analytics and machine learning has provided a coherent ecosystem, which has dominated the

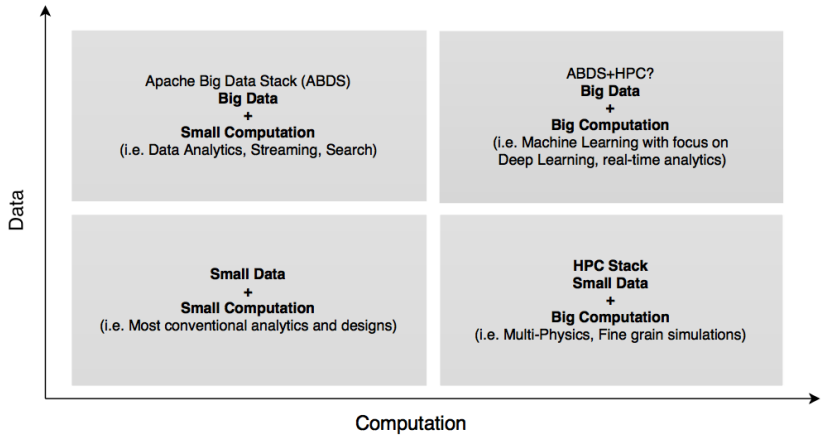

Figure 1. Matrix of data and computation applications.

industry.

In contrast, HPC with roots in computational intensive scientific computing, has mainly been applied and fine-tuned for particular problems, which resulted in significantly more limited implementations compared to ABDS. Figure 2 depicts the HPC and Big Data ecosystems from many abstraction level view. The lowest layers are the resources, resource management, and communication layers. The higher layers comprise of processing and analytical components. In remaining parts of this section we compare current typical HPC and Big Data architectures and their respective ecosystems.

1.1 HPC Architecture

A typical HPC architecture consists of processing units such a multi-core structures, possibly accelerated by GPUs, FPGAs, or even CPUs such as intel Xeon Phi. These processing units are supported by local storage units with Lustre [4] or General Parallel File System (GPFS) [15] file system or a remote storage connected through a fast network such as Infiniband.

Lustre and GPFS are parallel file systems designed mainly for clusters. Lustre, which combines the words Linux and Cluster is generally accepted as the file system with scalability for large scale cluster computing. Due to its high performance it has been continuously used by top supercomputers. On the other hand GPFS introduced by IBM and adopted by many of the biggest companies all around the word. GPFS have it roots in academic grounds and its scalability is a matter of question. Storage resources in HPC are managed by storage management unit such as integrated Rule-Oriented Data-management System

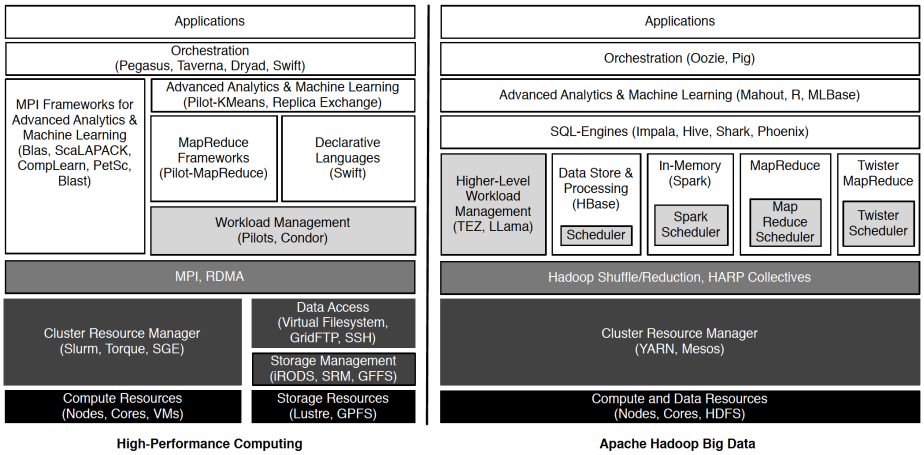

Figure 2. Apache Big Data Stack (ABDS) ecosystem vs. High Performance Computing (HPC) ecosystem [21].

(iRODS), Storage Resource Manager interface (SRM) [32] [30], Global Federated File System (GFFS) [19].

While storage resources are usually shared in applications, computational resources are local. Such local compute resources are managed by their own management units such as Slurm [38], Torque [35], and Sun Grid Engine (SGE) [16].

In HPC applications, data need to fly around across network using a low latency communication. These fast interconnect networks along with features such as non-blocking and one-sided communications allow for more data-intensive HPC within HPC environment. As a matter of fact, data-intensive applications are not totally unfamiliar to HPC community. There has been many approaches to implement MapReduce compatible with HPC environment such as MapReduce-MPI [26], Pilot-MapReduce [25], and Twister for machine learning applications [11].In a parallel effort, there has been many implementations of data intensive loosely coupled tasks at run-time level [24][29][9].

1.2 ABDS Architectures

The Apache Big Data Stack is formed around the Hadoop Distributed File System (HDFS), which originated at Yahoo but is an open source implementation of Google's file system [33] [17]. As we know now, the idea behind ABDS was to bring the computation to data, which is usually stored at cheap commodity hardware. Hadoop 1.0 also took advantage of Google's MapReduce programming model for data processing. MapReduce imposes it's own limitations since all processes is forced to be implemented in the form of a mapper followed by a

reducer. MapReduce's inherent limitations along with tight coupling of Hadoop and MapReduce proved to be inflexible when used in applications such as iterative computations, which is an indispensable piece of all Machine Learning algorithms.

As a result of such deficits in Hadoop 1.0, Apache YARN was introduced with Hadoop 2.0 as a much more lenient resource manager, enabling many applications and frameworks with higher level abstractions [37]. YARN's key feature was introduction multi-level scheduling, which would allow higher level applications to schedule their own processes. As a results we saw the emerge of many high level applications such as, HBase, a column based distributed database, Spark, Giraph, which are iterative and graph processing applications[3][39].

2 State of the art projects

In this section we will review some of the state of the art researches and projects within HPC-Big Data domains. We categorized these project under four class of research topics, namely,

1. Integration of HPC and ABDS stack
2. Machine Learning (ML) using HPC
3. Parallel databases
4. Parallel I/O

2.1 Integration of HPC and Big Data stack

Big Data and HPC used to be used interchangeably for many years. But, with HPC targeting scientific problems with huge number of iterations, and Big Data aiming for relatively simple model on huge datasets, it seems they diverged for a while. As the rate of data generation increase, deployment of HPC tools and techniques seem more and more inevitable. Following projects are some of the state of the arts in combining HPC and Big Data stack.

High Performance Big Data System (HPBDS) project [28] : The HPBDS project aims for integrating some of the components of the HPC such as scientific libraries, fast communication and resource management features with commercial Big Data ecosystem namely, ABDS.

The primary proposed execution of HPBDS is built around Apache Hadoop and therefore named High Performance Computing Big Data Stack (HPC-ABDS). The HPC-ABDS comprise two sub-project,

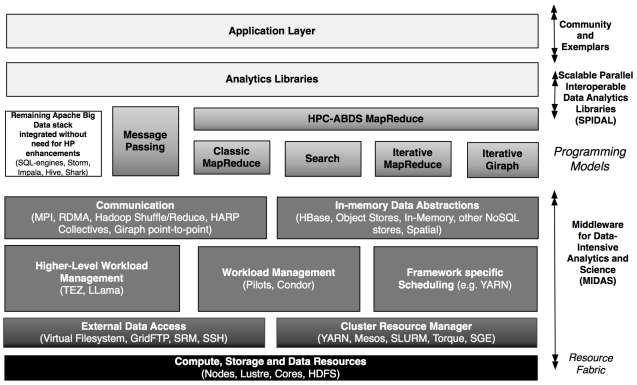

Figure 3. MIDAS and SPIDAL within HPC-ABDS stack[28]

1. Middleware for Data Intensive Analytics and Science (MIDAS)
2. Scalable Parallel Interoperable Data Analytics Library (SPIDAL)

Figure 3 depicts the structure of the MIDAS-SPIDAL with trespect to the overall HPC-ABDS stack. MIDAS is proposed to provide a lower level infrastructure based on which higher level components such as libraries operates. On the other hand SPIDAL goal is to take advantage of useful tool within HPC such as MPI, PETSc, and SCALAPACK and modify them for data intensive applications.

The HPC-ABDS project is inspired by the NIST Big Data initiative in Fall 2013, when NIST provided a collection of use cases. These use cases were analyzed against the identifiers proposed as big data Ogres [14]. The result is tabulated in Table 1. Based on the the tabulated results od the table 1, Qiu et al. [27]

Table 1. Categories of the NIST use cases.

Abbrevation	Count	Description
PP	26	Pleasingly Parallel or Map Only
MR	18	Classic MapReduce MR (add MRStat below for full count)
MRStats	7	Simple version of MR where key computations are simple reduction as found in statistical averages such as histograms and averages
MRIter	23	Iterative MapReduce or MPI
Graph	9	Complex graph data structure needed in analysis
Fusion	11	Integrate diverse data to aid discovery/decision making; could involve sophisticated algorithms or could just be a portal
Streaming	41	Some data comes in incrementally and is processed this way
Classify	30	Classification: divide data into categories
S/Q	12	Index, Search and Query
CF	4	Collaborative Filtering for recommender engines
LML	36	Local Machine Learning (Independent for each parallel entity)
GML	26	Global Machine Learning: Deep Learning, Clustering, LDA, PLSI, MDS, Large Scale Optimizations as in Variational Bayes, MCMC, Lifted Belief Propagation, Stochastic Gradient Descent, L-BFGS, Levenberg-Marquardt. Can call EGO or Exascale Global Optimization with scalable parallel algorithm
	51	Workflow: Universal so no label
GIS	16	Geotagged data and often displayed in ESRI, Microsoft Virtual Earth, Google Earth, GeoServer etc.
HPC	5	Classic large-scale simulation of cosmos, materials, etc. generating (visualization) data
Agent	2	Simulations of models of data-defined macroscopic entities represented as agents

identified five programming models to be used programs addressed in big data HPC problems. These five programming models are shown in table 2.

Table 2. Five programming models identified for big data HPC problems based on NIST big data use cases.

Programming Model	**Description**
Pleasingly Parallel	Applications such as local machine learning where processing is applied over many items in a local machine. Either Hadoop or other HPC tools could be used.
Classic MapReduce	Including MRStat, some search applications, collaborative filtering and motif finding. Traditional Hadoop and MapReduce could be used.
Iterative Map Collective	Iterative applications of MapReduce using collective communication. Hadoop 2.0 along with tools such as Spark could be used.
Iterative Map Communication	Iterative MapReduce such as Giraph with pointtopoint communication, includes most graph algorithms such as maximum clique, connected component, finding diameter, community detection).
Shared Large Memory	Thread-based (event driven) graph algorithms such as shortest path and Betweenness centrality. Large memory applications.

SPIDAL project will take advantage of the characteristics identified by analyzing NIST uses cases and programming models in Table 2 to build higher level abstractions based on the middleware analytic tools proposed by MIDAS. In the following parts of this section we introduce the algorithms introduced by SPIDAL and MIDAS.

Scalable Parallel Interoperable Data-Analytics Library (SPIDAL)

1. Graph and Network Algorithms
2. Spacial Queries and Spacial Analytics
3. Image Processing
4. Machine learning

Middleware for Data Intensive Analytics and Science (MIDAS)

MIDAS is proposed to enable a lower abstraction level, which include run time and resource management systems for SPIDAL. We now know from analysis of big data ogres and NIST use cases that big data application vary widely in requirements and characteristics. Therefore, different programming models and parallel approaches should be considered dealing with each type of problems. With this in mind, MIDAS has targeted two major objectives. First, a high

abstraction level environment that supports querying and analysis via HPC and big data tools such as SLURM and YARN as schedulers. Secondly, provide a middleware that support the four of the identified programming models in table 2 namely, Pleasingly Parallel, Classic MapReduce, Iterative Map with Collectives, and Iterative Map Communication. Figure 4 depicts the layered architecture of MIDAS.

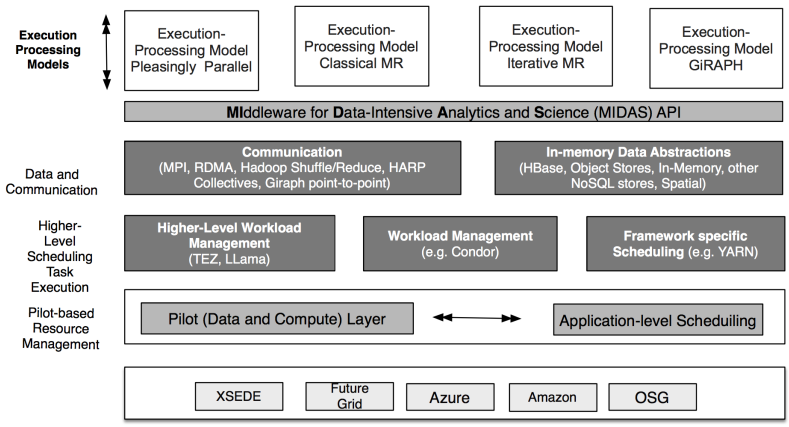

Figure 4. MIDAS architecture digram, where lower layers provide support for the top level tools [28].

It can be seen from figure 4 that many abstraction layers has been proposed to support mentioned programming models. For instance, in applications that better suit the iterative MapReduce, most operations are collective and MPI would perform better compared to MapReduce techniques. In contrast, graph processing application require more point to point communications.

2.2 Machine Learning and HPC

Traditionally the words Machine Learning and HPC would imply two separate worlds with distinct border. But, over the past decade areas of AI such as machine learning and training neural networks of deep learning have found their ways through HPC world.

Iterative AI tools such as machine learning and deep learning initially introduced to HPC by adopting GPU accelerators and FPGAs. Such tools adopted to perform a wide range of AI task from image classification, audio recognition, data analytics. Machine learning is not a novel concept and has been around for a relatively long time. But, along with more computational power introduced by accelerators, two major factors played an undeniable role in re-emergence of machine learning and deep learning. First, data generation at the very fast rates fuels up machine learning training algorithms. In a series of reports by Domo, [1] the amount of data created at the some of the biggest Internet giants, is presented. Table show some of these numbers in only one minute time of Internet. Secondly, introduction of new training algorithms such as those adopted to train deep belief nets and resulted to the birth of the term Deep Learning [20]. Only even twenty years ago it would take very long time, if possible, to train a neural network with only ten hidden layers.

Table 3. One minute of Internet usage [1].

Company	Count
Facebook	Users like **4,166,667** posts.
Twitter	Users send **347,222** tweets.
Youtube	Users upload **300** hours of video.
Instagram	Users like **1,736,111** photos.
Pinterest	Users pin **9,722** images.
Apple	Users download **51,000** apps.
Netflix	Subscribers stream **77.160** hours of video.
Reddit	Users cast **18,327** votes.
Amazon	Receives **4,310** unique visitors.
Vine	Users play **1,041,666** videos.
Tinder	Users swipe **590,276** times.
Snapchat	Users share **284,722** snaps.
Buzzfeed	Users view **34,150** videos.
Skype	Users make **110,040** calls.
Uber	Passengers take **694** rides.

Deep Learning, which itself is a sub-category of machine learning. The major distinguishing difference between most of machine learning techniques and deep learning is in feature selection stage. In conventional machine learning techniques labeled/unlabeled data is fed into a set of hand crafted feature detectors to extract

features. A classifier takes the features as inputs to perform some statistical analysis and decide on the class of each input. Since the engineering of a good set of such hand crafted features is extremely costly and require a lot of expertise and domain knowledge, deep learning is introduced to find and tune such features automatically using machine and data to be used for a particular task.

There is a general consensus that scaling up each of the the training data size and model size will increase the accuracy of prediction in deep learning. Therefore, as the size of training data or the model increases, we require to parallelize the process on many machines and processing cores to enable the scaling. Consequently, two parallelization schemes is offered namely, data parallelization and model parallelization.

In data parallelization scheme, the training set is sharded into smaller data fragments. Each of these fragments is assigned to a different processing node. Each node replicates the global model so that each model will have a fragment of data with a copy of full model. During the training process the model is updated. Since different copies of the model are updated separately using different data, we need to synchronize all copies of the model into a global model, which is representative of all the data. The issue with such synchronizations is that the model are usually relatively big (up to Gigabytes) that we cannot easily synchronize them frequently. In addition, it is also so hard to fit large models in GPUs' memory.
In a similar manner, in model parallelization scheme, models are cut into smaller

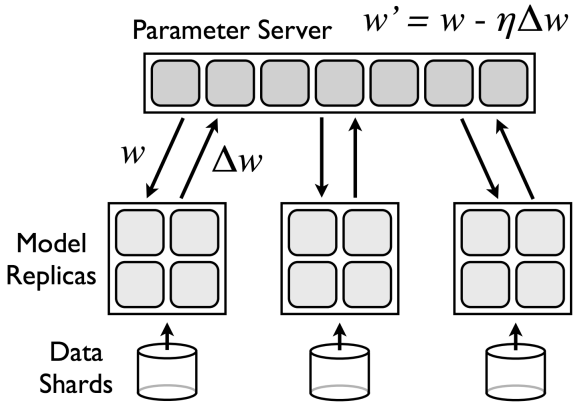

Figure 5. Data parallelism architecture performing Stochastic Gradient Descent (SGC). Model replicas at each node calculate gradients Δw on a piece of data. Gradients are later transmitted asynchronously to Parameter Server to calculate updated weights.

pieces, which allows for scaling of the model. A drawback of model parallelization is that you have to synchronize all your copies of the model even more frequently

than data parallelization. Synchronization should take place for each model at every layer of it, which for nowadays deep networks can be tens of layers. Thus, networking plays an very important role when it comes to model parallelization. For instance, a typical neural network architecture with 1 billion nodes may require tens of seconds is using a typical Ethernet networking for synchronization. As a results, HPC known tools such a Infiniband and PCI Express would greatly enhance the networking bottleneck for Inter-node and Intra-node communications respectively.

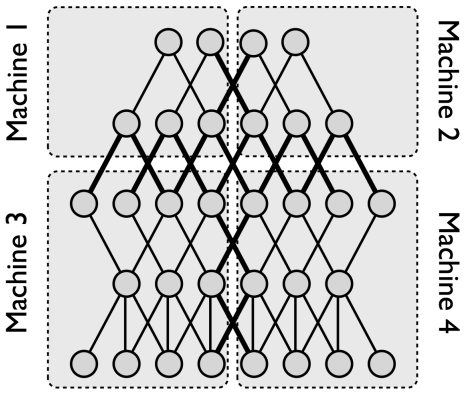

Figure 6. An example of model parallelism scheme in DistBelief framework for deep distributed neural networks by Google [8]. Model is distributed accross four machines. Only nodes, which have edges crossing from one machine to another need to report their state.

There are many primary questions to be addressed when thinking about integration of machine learning and HPC. Questions such as, aside from accelerating machine learning applications with GPUs and FPGAs, is traditional HPC ready to be the drive force behind data savvy machine learning applications? Are current top 500 supercomputers capable of powering new AI era of machine learning and deep learning? From software point of view, does conventional HPC require extensive modification to deal with mostly correlated data variables in machine learning application? In the remaining of this section we will briefly introduce some of the ongoing researches and projects concentrated empowering machine learning with HPC medium.

Project Adam: Building an Efficient and Scalable Deep Learning Training System [7]

In 2012 Google announced that they have trained an unsupervised image recognition neural network with 1 billion connections using 10 million training images,

which can recognize cats from Youtube videos. Their training system consisted of a cluster of 1,000 computers with an aggregate of 16,000 cores [23].

Project Adam by Microsoft could be thought as the successor of Google's project. The goal of the project Adam is to train a photograph classifier using 14 million images of Imagenet [10] into 22,000 image categories. The trained system by project Adam is claimed to be 50 times faster than the state of the art system. Speed is not the only concern of Microsoft researchers, it is also claimed that the system will be as twice accurate with 30 times less number of machines.

High level architecture

Adam is similar in architecture to the Multi-Spert system introduced by Farber at Berkeley [13]. A set of machine have the role of providing the inputs of the neural network with the input data, while another set train the model. Adam takes advantage of both model and data parallelism. Models are partitioned into many worker machines. Data is also fragmented to provide data parallelism. In order for each piece of the model to be trained only by a small chunk of data, many replicas of the same piece of model in parallel are trained on different data chunks.

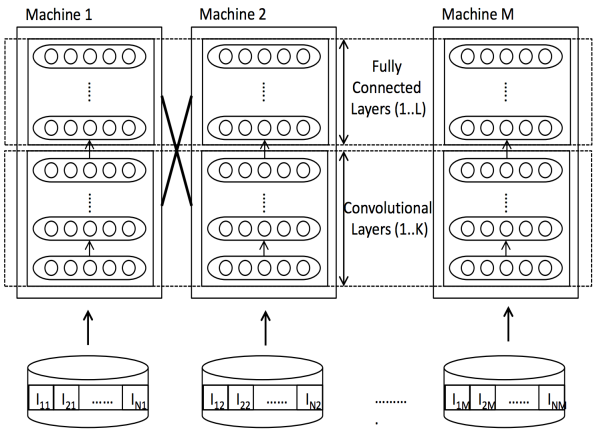

Figure 7. Model and data partitioning in Adam architecture[7].

A global Parameter Server stores the global values for the updated weights of all model pieces and replicas. The communication between the models and the Partition Server is asynchronous. Although the asynchrony between models and Parameter Servers may seem to introduce inconsistencies, but due to the resilient

nature of the neural networks, they usually get trained successfully. As a matter of fact Microsoft researchers found that when they switched from asynchronous to synchronous communication between the models and the Parameter Server, the accuracy of the model dropped. The drop in the accuracy is justified as most of cost functions is datasets such as Imagenet and MNist are not convex functions. As a result, during the training process, the trainer may gets stuck at local optima rather than the global optima. The asynchronous updating of the weights cause the trainer in local optima get some momentum to escape local optima towards a global optima.

Convolutional neural networks are a special type of neural networks that has one or more convolution layers. Such networks are primary choice for image classification and vision since convolutional layers are capable of detecting features. It should be noted that Adam is not merely an image classification system and is capable of training any deep neural network with back-propagation. Figure 7 shows that Adam is partitioned in a way that each machine possess some of the convolution and some of the fully connected layers. This type partitioning allows for the communication of the convolution layers across machines be minimized.

Intra machine architecture and optimizations

Training process on each single machine is multi-threaded. Threads hold activation functions and weights to use during feed-forward and back-propagation through None Uniform Memory Access (NUMA) fashion to reduce the cross-memory bus traffic.

Earlier we mentioned that neural networks are resilient so that we could use asynchronous communication between machines and global Parameter Server for updating weights. Same argument could be applied here for weight updates by threads on a single machine. On each machine a shared model holds the weights from all of threads weight updates. In order to avoid the lock times, threads update weights on the shared model without locking. Resilience of the neural networks along with commutativity and associativity properties of weight updates make the model work despite possible overwriting and race conditions of weight updates. This is one of the most important optimizations that allow scaling of deep neural network training process.

Memory optimizations

During the training process weights need to be transmitted back and forth among layers. In order to minimize redundant copies, a pointer is passed along rather than rather than the values. In addition, in the model parallelism architectures, there are some communication between layers across multiple machines. These non-local communications are performed through a library built by project Adam

team. The library is custom fit to the communication design of Adam, which supports pointer addressing to blocks of neurons that their output needs communication.

The partitioning of the models is Adam is done in a way that the working sets fit inside a L3 cache. Besides, an assembly kernel is implemented and tuned to exploit the locality during feed-forward and back propagation. The kernel allows for optimal matrix multiplication whether row major or column major blocks of weights are to be multiplied.

Global Parameter Server

The Global Parameter Server holds the updated weights of the overall model from all machines. Due to high volume of weight updates, the Global Parameter Server of Adam require a more complex design rather than a conventional key value store. Figure 8 depicts the architecture of the Global Server node for Adam.

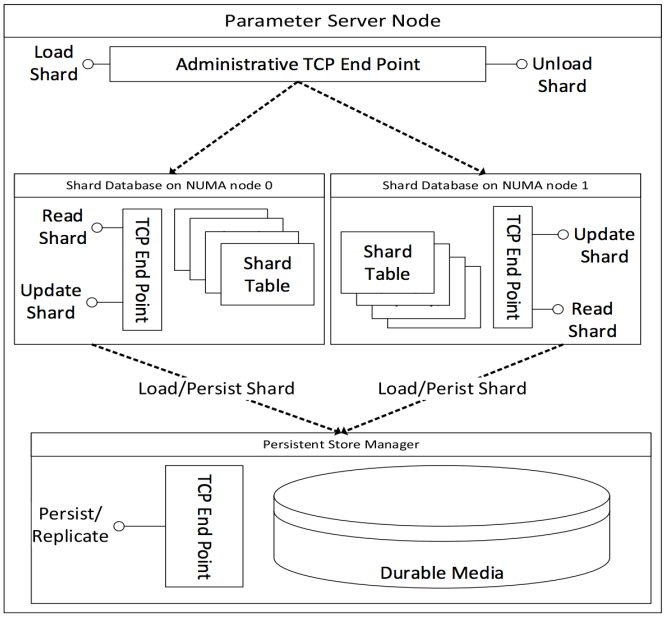

Figure 8. Global Parameter Server architecture of Adam[7].

Model parameters on the parameter server is partitioned into weight shards of size 1MB. The model partition in the Parameter Server helps with the spatial

locality of the updates. In addition, to lower the communication between L3 caches and the parameter server, weight updates are not sent over to the parameter server once they are ready. Instead, weights are batched into blocks and then transferred over to parameter server. To assure the fault tolerance of the processes on the parameter server, there are three copies of weight shards available. One of the shards is assigned as the primary and the other two as the secondary to be used in the primary is cannot be used.

Project Adam's Hardware

Adam is formed of a cluster of 120 machines in three racks connected using IBM G8264 switches. Machine is Adam cluster are ervers of HP Proliant servers consisting of dual Intel Xeon E5-2450L processors with 16 cores and 98GB main memory along with two 10Gb NICs and one 1Gb NIC. There are four 7200 rpm hard drive storages for each machine. One of the hard drives that has the Windows 2012 server as operating server is of size 1TB. The rest three hard drives each are 3TB and are connected is RAID fashion. Out of 120 machines, 90 are allocated for training the model, 20 for parameter server, and the rest 10 for image inputs.

Results

The accuracy of trained models is always one of the most important concerns. Adam's trained models are evaluated against two benchmarks. First, the MNIST benchmark that is well known to the machine learning community, which contains image of a series of handwritten digits. Second, the ImageNet, which is an image dataset with 22k object categories.

Adam architecture for testing against the MNIST benchmark consisted of two convolutional layers, two fully connected layers, and a ten class softmax output layer[34]. The accuracy of Adam compared to the top 1 model by Goodfellow et al [18] is shown in table 4.

Table 4. Adam accuracy on MNIST benchmark compared to the existing top accuracy model.

Model	**Accuracy**
Goodfellow et al.	99.55%
Adam (asynchronous)	99.63%
Adam (synchronous)	99.39%

It should be noted that the Adam accuracy is reported with asynchronous and synchronous communication of model shards in machine to the global parameter server. We mentioned that due to resiliency of the neural networks weight updates can be sent to global parameter server asynchronously. Even withing a machine

weight updates coming from multiple threads is also asynchronous without using locks. The results in table 4 show that asynchrony not only did not lower the Adam accuracy, but also improved it by 0.24%.

For the ImageNet dataset, Adam was trained with a common architecture as reported in earlier works in literature. The architecture consist of 5 convolutional layers, 3 fully connected layers, and a 22k-way softmax layer. The accuracy of Adam compared to the top 1 system by Le et al [23] is tabulated table 5.

Table 5. Adam accuracy on ImageNet benchmark compared to the existing top accuracy model.

Model	**Accuracy**
Le et al. (without pre-training)	13.6%
Le et al. (with pre-training)	15.8%
Adam	29.8%

The results in table 5 show a much more significant improvement by a factor of almost 2× compared to the top accuracy model by Le et al [23]. The training process took 10 days to complete on a total of 62 machines, which is 32 times less than 2000 machines used by the top 1 model.

In conclusion, Adam succeeded to outperform the top 1 model in accuracy, number of machines used, and XXXX through a set of optimizations. It also seems that asynchrony plays a major role in enabling Adam to scale well training very large models.

Dadiannao: A machine-learning supercomputer [6]

With the re-emergence of neural networks in 2006 and introduction of deep learning algorithms, one stream of research has been focused on specialized hardware tailored to deep learning applications rather than using multi-purpose GPUs [36][12].

One of such specialized deep learning accelerators is the one introduced by Chen et al, namely DianNao [5]. DianNao comprises two major group of components. First, buffers for retrieving and holding inputs and outputs of neurons and synapses. Second, a Neural Functional Unit (NFU), which performs typical computations required to calculate outputs of neurons. These calculations are mainly matrix multiplication of weights and input in the first stage, summation of the results of first stage in the second stage, and applying an activation function in the third stage. Depending on the architecture of network and the layer types some other types of operation is also embedded such as convolution, averaging, etc. Figure 9 depicts the block diagram of the DianNao accelerator. In order to evaluate the performance of DianNao, Chen et al. structure two neural networks

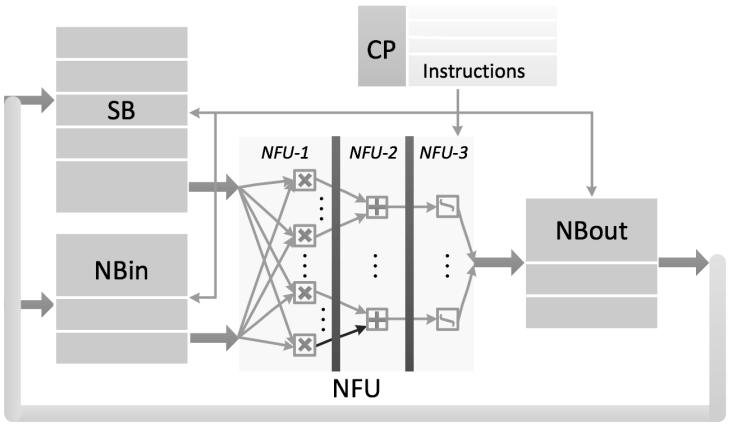

Figure 9. Block diagram of the DianNao accelerator[6].

with similar architecture, but one with GPU and the other one with DianNao acceleration. They also implemented the same network using CPU only without any acceleration. Figure 10 compares the speedup of GPU over both DianNao and CPU.

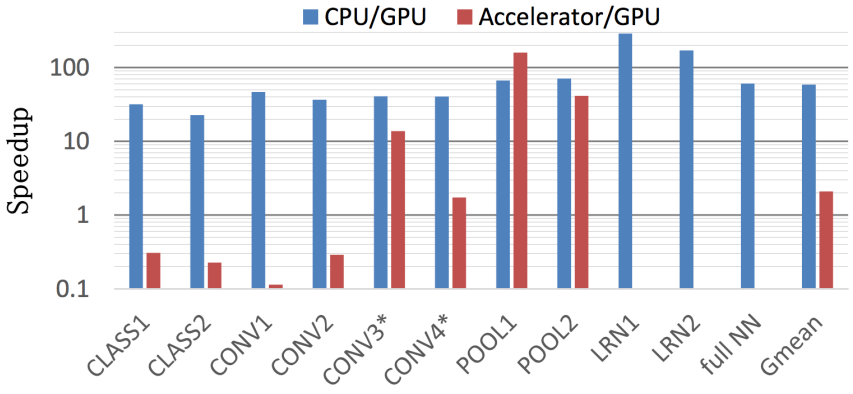

Figure 10. Speedup of CPU/GPU and CPU/DianNao for various layer types of network[6].

We can see from figure 10 that DianNao give speedups up to 100 for some of the layers. However, in some other layers, mostly convolutional layers with private kernels and classification layers, which are common in both Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN), GPUs may be

preferred over DianNao. Authors believe that memory bandwidth is the major reason that DianNao performs worse than GPU in such layers. It should be noted that both convolutional and classification layers share the property that they have high number of synaptic connections even in range of millions. As DianNao takes up only about 53% of the GPU (K20M), which is used for the sake of this comparison, it has the potential to compete GPUs.

Dadiannao, the machine learning supercomputer

While the authors name Dadiannao the machine learning supercomputer, it seems the seems the design of Dadiannao were more focused on DNNs and CNNs. With that in mind, Dadiannao is capable of holding large models of even up to tens of GB in size in a multi node structure.

As we mentioned in the earlier part of this section, memory is thought to be the bottleneck of DianNao in convolutional layers with private kernel and classifier layers. The design of Dadiannao as a supercomputer aims to address these problems. First, in order to minimize data movement cost, synapses are placed near to the neurons consuming their data in a fully distributed architecture with no main memory unit. Secondly, nodes are biased towards storing data rather than computation. Thirdly, since the number of synaptic connections is orders of magnitude bigger than neurons, instead of sending synaptic weights to neurons, neuron values is sent to synapses. This saves a lot in inter node communication bandwidth. Fourthly, intra-node, the storage units are broken into many storage units to increase the bandwidth.

The high level architecture of Dadiannao consist of many nodes arranged in a mesh configuration. Each node contains the storage units to store the synaptic and neuron values. In addition a, NFU unit is embedded similar to what introduced in DianNao [5].The NFU in Dadiannao is capable of carrying out more complex types of computation depending on the layers and mode of operation (training or inferencing). Figure 11 shows the block diagram of a node for Dadiannao architecture.

It should be noted that although SRAMs are generally preferred over other types of memory such as DRAM and eDRAM due to the higher speed, their size is limiting factor for them. A typical eDRAM can accommodate more than 2.5× data compared to a SRAM. It should be noted that more storage density is required if parameters of big models are to be stored within chip to avoid external memory access. As a result, eDRAM is selected over SRAM in Dadiannao's proposed design. Refuting the memory bandwidth limitations for for computational units allows for scaling since more inputs and outputs to each neuron can be calculated simultaneously. However, eDRAMs suffer from some drawbacks compared to SRAMs namely, slower response, periodical refreshing requirement, and destructive read that may result in inaccurate values.

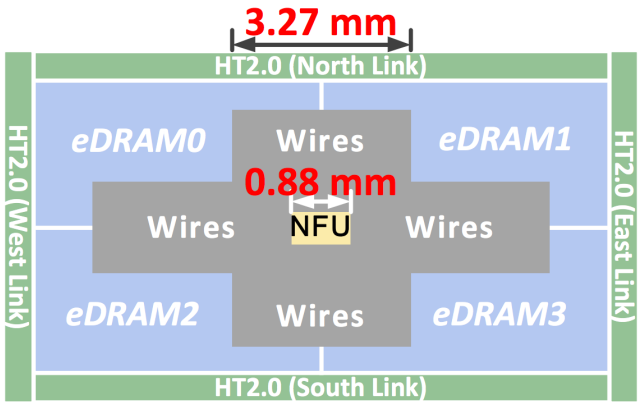

Figure 11. Block diagram of a node within Dadiannao architecture [6].

In order to exploit high bandwidth within nodes, a tile fashion layout is implemented as shown in figure 12. In this design output neurons are assigned to various tile units. This enables NFUs to process data from multiple neurons in parallel.

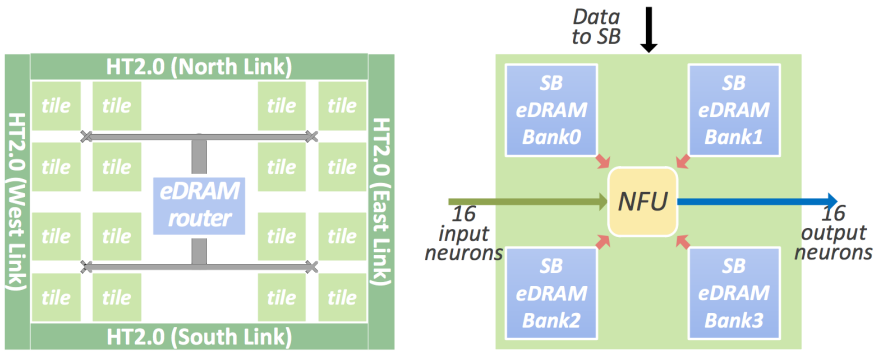

Figure 12. Left: Tile implementation of a node. Right: Layout of unit inside a tile. [6].

Tiles in figure 12 are connected to two eDRAM reservoirs in the middle, one for input neurons from tiles and one for output neurons. It should be noted that even though there are many tiles and NFUs in a node, the number of neurons in a typical deep learning architecture is much more to be a one to one relationship between NFUs and neurons. As a result, each NFU has the job of calculating many neuron outputs.

Experimental Results

In this part we present the performance and energy consumption Dadiannao. In figure 10, performance of Dadiannao in the training mode is compared against its NVIDIA K20M GPU implementation counterpart.

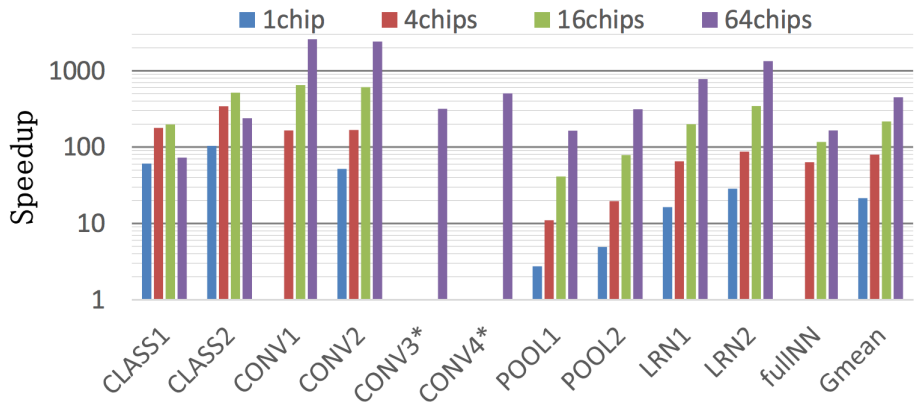

Figure 13. Speedup of Dadiannao/GPU for various types of layers. CONV1 and fullNN layers require at least 4 nodes. CONV3* and CONV4* need at least 36 nodes to operate[6].

Figure 13 shows that Dadiannao could reach a speedup of more than 100 over the GPU system in almost all types of layers. The speedup is even higher up to more than 2000 for the CONV1 and CONV2 layers. It should be noted that the size of CONV3* and CONV4* layers (which have private kernels) is relatively large in the range of GB. As a result, the performance of these layers can only be measured of on a more than 36-node arrangement. The fullNN layer, which is a fully connected layer also accounts for more than 50M synaptic connections and hence cannot be implemented on a system with only 1 node. Generally speaking, average speedup of 21.38×, 79.81×, 216.72×,and 450.65× is reported for Dadiannao over GPU for 1-node, 4-node, 16-node, and 64-node configurations, respectively. However, the speedup and scaling of various types of layers are wildly different due to the inherent characteristics of each type of layer.

Similarly, the performance of Dadiannao against GPU baseline in evaluated from energy point of view. Figures 14 and 15 depict these performance analysis for both training and inference modes. In the training mode the energy reduction rates against the GPU baseline are 172.39×, 180.42×, 142.59×, and 66.94× for the 1-node, 4-node, 16-node, and 64-node respectively. The energy reduction number reported in the inference mode is even higher. In the inference mode, the energy reduction for the 1-node, 4-node, 16-node, and 64-node configuration is 330.56×, 323.74×, 270.04×, and 150.31×, respectively.

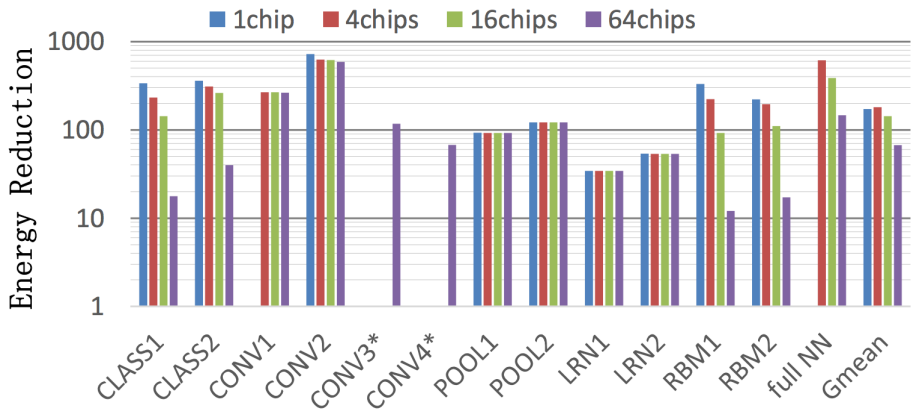

Figure 14. Comparison of energy reduction of Dadiannao against GPU in the training mode[6].

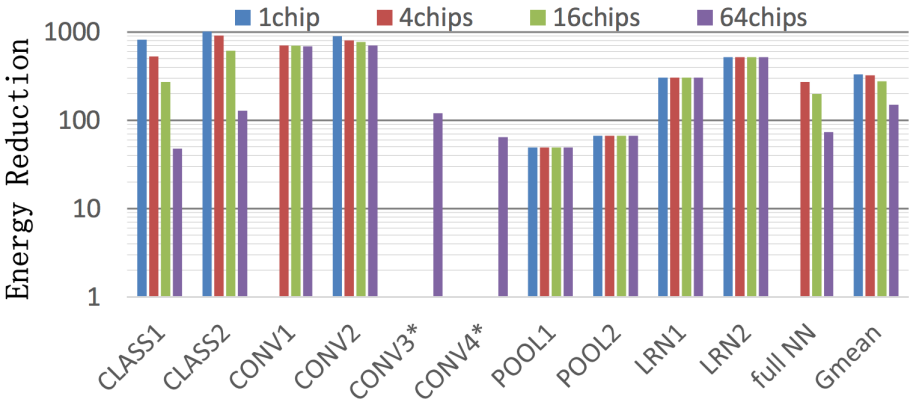

Figure 15. Comparison of energy reduction of Dadiannao against GPU in the inference mode[6].

Conclusion and Outlook

Machine learning and deep learning have already become a part of our lives that we even do not notice them anymore. From search engines, voice recognition systems in our phones, and self-driving cars, all the way to commercial and financial analytics, they all benefit from machine learning and deep learning. With the help of fast CPUs and GPUs, training of models within machine learning and deep learning frame encountered a huge boost in the recent past. However, it seems that the future attempts to guarantee the the scalability of such systems should be focused on designing specialized software and hardware frameworks tailored

to these applications such as project Adam and Dadiannao supercomputer.

2.3 Parallel Databases

Although the idea of parallel database system was refuted by a paper prophesying demise of database machine in 1983 but that did not stop the adoption of parallel database. Companies like Teradata, Tandem and many other startup companies pioneered a path for successful parallel database systems. The success of parallel database is accorded to wide adoption of relational model which favors the idea of parallel dataflow graph leading to pipelined parallelism and partitioned parallelism.(Explain if necessary) (Ten years later a paper from Dewitt explains how parallel database system would be the future of high performance database systems). With the advances in high speed interconnect network, the client-server operating system gave much higher performance over price than its mainframe equivalent. A parallel database system can be classified based on the terms of how data is shared amongst multiprocessor system i.e. shared-memory, shared-disk or shared-nothing.

By the early 90's, it was clear that shared-nothing architecture along with conventional processor, memories and disk would be the future of scalable parallel and distributed system. Linear speedup and linear scaleup are two characteristics of an ideal parallel database system. (Add definition for both if necessary) .But usually a sub optimal speedup and scaleup is observed due to these three issues: startup, interference and skew (definition) SQL language was introduced by Codd, in his historic paper, to support data independence and boost programmer's productivity by separating execution logic from query itself. However, it came along with added benefits of parallelism, making it possible to represent each relational operator on a data flow graph and dividing data into a parallel data streams for partitioning. There are several partitioning strategies like round-robin, hash, range partitioning etc. Although, there is a heated argument about the performance benefits of using a Map Reduce system against the parallel DBMS system, former has been widely adopted because it offers better performance with complex queries on unstructured data.

High performance computing systems have high speed network, parallel file systems and diskless nodes while Big data analytics research has evolved from an increasing demand for extracting knowledge from large data set on a big and small cluster with low speed network with each node having its own local file system and a disk. For example, Lustre file system is widely used in HPC environment whereas Hadoop's HDFS is popular for big data analytic applications. The main focus of HPC machine is optimize and exploit the resources, with a parallel file system.

During the past decade, several attempts are made to bridge the gap between HPC and Big data analytics giving rise to new terms called High Performance

Data Analysis. In next section, we would be discussing about the different research that has been propelled towards having an amalgamation of these two giants that have been on different track so far.

2.4 Project SCOPE

Structure Computation Optimization for Parallel Execution (Scope) is a distributed computing system, that incorporates best characteristics of traditional parallel database system and map reduce execution engine like scalability and robust implementation. It is being used in many of the online service from Microsoft like Bing. A physical execution plan containing a directed acyclic graph are crafted from a script by a Cascade framework based optimizer. Like MapReduce, a Job Manager provides fault tolerance and recovery and at the same time a declarative scripting language similar to SQL that provides abstraction and highly extensibility through user defined functions and relational operations.

SCOPE Platform Architecture

Storage and analysis of large scale data sets is performed on a distributed data platform called COSMOS which provides data replication and meta-data for fault tolerance. Each server has a direct attached disk storage disks that contains a distributed data. End to end on disk data for reliability by checks for corrupt and stale data periodically before it's utilized by the system. For performance, each job is divided into a sub jobs of computation and distributed among huge among of CPU's and storage devices.

Writes are always append-only and serialization is used for concurrent writes. The storage system consist of a physical partitions called extends, which are the unit of space allocation and replication, typically of few 100 MB size. Application extends, often compressed and decompressed at client side, contains application-defined records and define a block boundary. Local as well as remote read and write are enabled through high-bandwidth network. Directed acyclic graph (DAG) are used for modeling a query plan. Job submission and management, job queue monitoring, interoperability, job status tracking, error reporting and transferring data in and out of COSMOS are provided through an interface from front-end services.

Data Representations-Streams

It supports both structure and unstructured stream. Unstructured stream are defined as a logical sequence of bytes that requires an extractor for users to interpret it. An extractor defines the schema and iterator interface for transforming the data. It also provides an outputters that writes output in an unstructured streams.

Structured stream are handled by build-in function that allow an unvarying access time by storing data with into different schemas which are self-contained, that is, with rich metadata information such as schemas, structural properties and access method.

Usually these data are horizontally portioned with different schemes like hash and range partitioning. Each structural stream can consist of several physical extents which allows effective replication and fault tolerance. Data is stored based on store affinity that assigs affinity id, with each affinity group having same id. Storage priority is given first at same machine level, if the storage is full then at same rank and then at next close rank[KP1].

Another optimization to improve performance is stream references, in this technique the output stream is referenced with another corresponding partition based on affinity id based on the storage priority shown above. Some other features are Indexing for random access, column group for vertical partitioning.

Query Language for Scope

SCOPE query language resembles SQL and has many SQL-like extensions that enable users to separate application logic from underline implementation on a distributed system. For unstructured stream, it provides personalized commands for EXTRACT and OUTPUT, for reading from source and writing data to a sink. Structure stream removes the constraints of specifying columns because it is already contained in the stream metadata during compilation. Reference clause can be used to specify affinity of output to another stream. REDUCE command can be replaced by adding an optional PRESORT and PRODUCE clause.

Query compilation and optimization

Diagram: COMPILATION OF SCOPE SCRIPT

Code Generation and Runtime Engine

SCOPE runtime engine supports all the relational operators along with build in and optimized UDOs like DefaultTextExtractor and DefaultTextOutputter. Simple functions like open, next and close are performed by operator which inherently supports pipelining of these operators. The runtime engine, after optimization, generates a physical execution tree by doing these tasks:

1. The whole plan is analyzed to produce corresponding codes for operator based on the internal operator template.
2. Output tree is partitioned at exchanges and spool operators to generate a super vertex that contains a series of physical exchanges.

3. Break large super vertex into smaller ones to avoid skew delay due to longer execution time.
4. An assembly is made by compiling all the generated code into it and the entry functions for each super vertex is written into super vertex definition.
5. Each super-vertex is schedule to execute on a single machine.
6. Like traditional db system, a queue of additional super vertices are maintained through admission control technique when all resources are allocated.

Compilation script output consist of three components:

1. Algebra file: Enumeration of all super-vertex and the data flow relationship between them.
2. Super vertex definition file: Specification and entry point in the generated code for all super vertex.
3. Assembly: Contains the generated code.

Execusion Model

Similar to MapReduce, SCOPE implements a pull execution model. Super vertex, reads data either locally or remotely, are executed in a pipelined fashion with an end result written to a local disk and waiting for the next vertex to pull data. It contrast with the execution model of parallel database which transfer intermediate data directly without any disk access. However, pull execution model has an advantage that both parties involved in data exchange need not run simultaneously and the data can be recovered from disk on failure.

Runtime Data Exchange

Since shuffling of data is the most expensive part considering the network overhead, SCOPE runtime provided many partitioning and merge operators. Hash partitioning Range partitioning

Job Scheduling

SCOPE's job manager is an adaption from Dryad, which takes care of job graph and resource scheduling on clusters. DAG of super vertices is scheduled to execute separately on different machines. Vertices are classified into different stages by Job manager and all the vertices in a single stage perform same task. Data storage and computation can be performed by all the machine.

Diagram SCOPE job scheduling

Local system resources are maintained by processing node services and it schedules execution of local vertices along with monitoring state of a job. Based on available resources and priorities it generates process on behalf of Job managers

and maintains a local queues of vertex execution request from different Job manager. It stores the assembly of vertex in a cache and uses it if the vertex is executed again.

An initial DAF is formed, when the job begins, to oversee relationship among different vertex. According to execution results and variation in system environment, the job graph is dynamically adapts itself. Resource availability and vertex priority defines the actual vertex scheduling order and it's the task of Job Manager to notify any vertex execution failure.

Runtime optimization

Stil working on these parts

Adaptive Hybrid OLAP Architecture

OLAP (Online Analytic processing) has a fundamental structure of a data cube which consist of multi-dimensional data that enables user to explore relation between different more one than one dimensional data. An adaptive hybrid OLAP architecture is proposed in [] to exploit different memory subsystem and memory access patterns of CPU and GPU. In multidimensional OLAP cube (MOLAP) data is preprocessed during the cube creation time and due 0to its sparsity and huge size its more suitable to be processed on host (CPU) keeping in mind the NUMA(Non uniform memory access) architecture.

However, GPU's architecture is suitable to processes relational OLAP (ROLAP) cube which is created by manipulating data to appear like traditional OLAP with slicing and dicing functionality.

References

1. Data Never Sleeps, https://www.domo.com/blog/2015/08/data-never-sleeps-3-0/
2. NIST Big Data Working Group (NBD-WG), http://bigdatawg.nist.gov/usecases.php
3. Borthakur, D., Gray, J., Sarma, J.S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., et al.: Apache hadoop goes realtime at facebook. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data. pp. 1071–1080. ACM (2011)
4. Braam, P.J., et al.: The lustre storage architecture (2004)
5. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O.: Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In: ACM SIGPLAN Notices. vol. 49, pp. 269–284. ACM (2014)
6. Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al.: Dadiannao: A machine-learning supercomputer. In: Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on. pp. 609–622. IEEE (2014)
7. Chilimbi, T., Suzue, Y., Apacible, J., Kalyanaraman, K.: Project adam: Building an efficient and scalable deep learning training system. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). pp. 571–582 (2014)
8. Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q.V., et al.: Large scale distributed deep networks. In: Advances in Neural Information Processing Systems. pp. 1223–1231 (2012)
9. Deelman, E., Gannon, D., Shields, M., Taylor, I.: Workflows and e-science: An overview of workflow system features and capabilities. Future Generation Computer Systems 25(5), 528–540 (2009)
10. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. IEEE (2009)
11. Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.H., Qiu, J., Fox, G.: Twister: a runtime for iterative mapreduce. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. pp. 810–818. ACM (2010)
12. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 449–460. IEEE Computer Society (2012)
13. Farber, P., Asanovic, K.: Parallel neural network training on multi-spert. In: Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on. pp. 659–666. IEEE (1997)
14. Fox, G.C., Jha, S., Qiu, J., Luckow, A.: Towards an understanding of facets and exemplars of big data applications. In: In proceedings of Workshop: Twenty Years of Beowulf (2014)
15. Frank Schmuck, R.H.: GPFS: A Shared-Disk File System for Large Computing Clusters http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.7147
16. Gentzsch, W.: Sun grid engine: Towards creating a compute power grid. In: Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on. pp. 35–36. IEEE (2001)

17. Ghemawat, S., Gobioff, H., Leung, S.T.: The google file system. In: ACM SIGOPS operating systems review. vol. 37, pp. 29–43. ACM (2003)
18. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. arXiv preprint arXiv:1302.4389 (2013)
19. Grimshaw, A., Morgan, M., Kalyanaraman, A.: GffsâĂŤthe xsede global federated file system. Parallel Processing Letters 23(02), 1340005 (2013)
20. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation 18(7), 1527–1554 (2006)
21. Jha, S., Qiu, J., Luckow, A., Mantha, P., Fox, G.C.: A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures. In: 2014 IEEE International Congress on Big Data. pp. 645–652. IEEE (jun 2014), http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6906840
22. Kashyap, S.N., Fewings, A.J., Davies, J., Morris, I., Green, A.T.T., Guest, M.F.: Big data at hpc wales. arXiv preprint arXiv:1506.08907 (2015)
23. Le, Q.V.: Building high-level features using large scale unsupervised learning. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. pp. 8595–8598. IEEE (2013)
24. Luckow, A., Santcroos, M., Merzky, A., Weidner, O., Mantha, P., Jha, S.: PâĹŮ: a model of pilot-abstractions. In: E-Science (e-Science), 2012 IEEE 8th International Conference on. pp. 1–10. IEEE (2012)
25. Mantha, P.K., Luckow, A., Jha, S.: Pilot-mapreduce: an extensible and flexible mapreduce implementation for distributed data. In: Proceedings of third international workshop on MapReduce and its Applications Date. pp. 17–24. ACM (2012)
26. Plimpton, S.J., Devine, K.D.: Mapreduce in mpi for large-scale graph algorithms. Parallel Computing 37(9), 610–632 (2011)
27. Qiu, J.: Towards HPC-ABDS : An Initial High-Performance Big Data Stack 1(1), 1–22 (2014)
28. Qiu, J., Jha, S., Luckow, A., Fox, G.C.: Towards hpc-abds: An initial high-performance big data stack. Building Robust Big Data Ecosystem ISO/IEC JTC 1 Study Group on Big Data pp. 18–21 (2014)
29. Raicu, I., Foster, I.T., Zhao, Y.: Many-task computing for grids and supercomputers. In: Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on. pp. 1–11. IEEE (2008)
30. Rajasekar, A., Moore, R., Hou, C.y., Lee, C.A., Marciano, R., de Torcy, A., Wan, M., Schroeder, W., Chen, S.Y., Gilbert, L., et al.: irods primer: integrated rule-oriented data system. Synthesis Lectures on Information Concepts, Retrieval, and Services 2(1), 1–143 (2010)
31. Reed, D.a., Dongarra, J.: Exascale computing and big data. Communications of the ACM 58(7), 56–68 (2015), http://dl.acm.org/ft{_}gateway.cfm?id=2699414{&}type=html
32. Shaun De Witt, R.: The storage resource manager interface specification (2007)
33. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. pp. 1–10. IEEE (2010)
34. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: null. p. 958. IEEE (2003)
35. Staples, G.: Torque resource manager. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. p. 8. ACM (2006)

36. Temam, O.: A defect-tolerant accelerator for emerging high-performance applications. In: ACM SIGARCH Computer Architecture News. vol. 40, pp. 356–367. IEEE Computer Society (2012)
37. Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al.: Apache hadoop yarn: Yet another resource negotiator. In: Proceedings of the 4th annual Symposium on Cloud Computing. p. 5. ACM (2013)
38. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer (2003)
39. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. pp. 2–2. USENIX Association (2012)

All links were last followed on October 5, 2014.