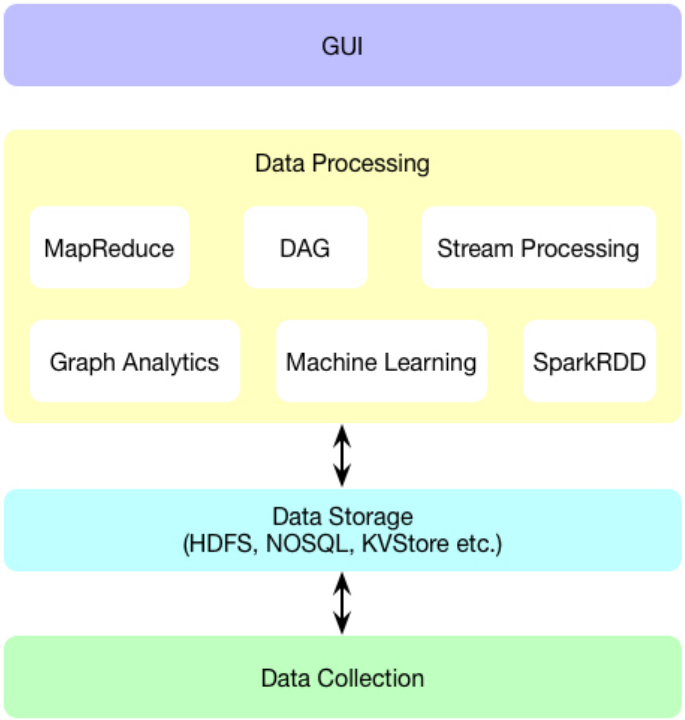
Introduction

Big data is a popular term in the last decades, with 3V characteristic: Volume, Velocity and Variety, the traditional stand-alone machine environment may cost months or years to finish a simple big data job. The research and industry have post and develop a complete solution for this, from data collection, storage and processing, the developing and evolution are happening with the development of the real-world requirements.



Especially, in the big data analytics area, with the Google MapReduce purposed, which has had a far-ranging impact on the distributed computing industry, it is built on the simple concept of mapping and reducing big data, the developer and easily overcome the massive data processing difficult. The true value of MapReduce lies with its ability to run these processes in parallel on commodity computers while balancing disk, CPU and I/O. However, the MapReduce have several fatal defects, which makes the system cannot run efficiently on particular jobs.

The first shortcoming is the intermediate data between different MapReduce job was saved into the HDFS persistent storage system, every job has to reload the data from storage again, which will cause significantly latency on data processing. The HaLoop have purposed a caching and indexing between the reducer and next mapper, which benefit a lot in the iterative processing task. The SparkRDD have purposed an In-Memory solution for data processing, with lineage mechanism, it can built lineage to rebuild the lost data after fault occurs. The second shortcoming of MapReduce is the limitation of its design, every processing task will have to been defined as MapReduce jobs, the relationships between different MapReduce job will be extremely complicated, which will directly increase the latency. The Dryad and Flume have developed a more flexible processing system based on DAG, which can significantly optimize

multi-stage tasks processing efficiency. The MapReduce is also unable to process the real-time dataflow, the Spark Streaming and Storm have made improvements and enable fast and reliable processing on the massive of streaming data. For the typically graph problem such as the shortest path, PageRank and Minimum cutting, the graph processing system like GraphLab and Pregel are more suitable.

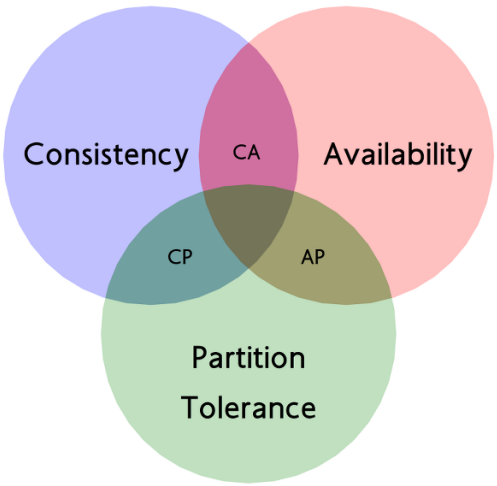
In this report, I will conclude the state-of-the-art works of distributed analytics systems, from Hadoop MapReduce, HaLoop, Spark to Naiad and Husky. I will also conclude the reason why those systems were developed, and the main idea behind them. A comparative analysis will be purposed, illustrate the advantages and disadvantages of these works. As the branch of the distributed processing systems, the GraphLab and Strom will also be discussed. At last, I will have a guess about the future development of the distributed data analytics systems.

Basics

CAP

The CAP theorem is a tool used to makes system designers aware of the trade-offs while designing networked shared-data systems. CAP has influenced the design of many distributed data systems, it states that networked shraed-data systems can only guarantee support two of the following three properties:

- `Consistency`: Every node in the distributed cluster returns the same, most recent and successful write, every client having the same view of the data.
- `Availability`: Every non-failing node return a response for all read and write requests in a reasonable amount of time.
- `Partition Tolerant`: The system will continue work in spite of network partitions.



The CAP theorem categorizes systems into three categories:

- AP: Make sure the system are available and fault tolerance, but no guarantee of the consistency. For example, the user profile in the chat application.
- CP: The availability was sacrificed in case of network partition, application like bank ATM will directly stop the service when error is occur.
- CA: System will be consistent and available in the absence of network partition, in the real world, most of the system will not use this as fault tolerance is the basic of the distributed

system.

Scalability & Availability

`Scalability` is one of the most important design goals for developers of distributed systems. A system is scalable if it remains effective as the number of users and resources increase, the main challenges include controlling resources costs, performance loss and preventing resources from running out.

`Availability` means the distributed should be continuously available, eery request received by non-failing node in the system must result in a response. High availability can be guaranteed with data replication.

Throughtput & Performance

The key questions between high throughtput and high performance are the granularity and degree of parallelism.

A `fine-grained` system running with independent small bits, the information was exchanged and synchronised often. Which need a smaller number of more expensive processors expensively interconnected, that enables rapid synchronisation between the bits processed in parallel.

A `corarse-grained` system running with large chunks that can be processed independently, the system use a large number of inexpensive processsors, inexpensively interconnected, which can maximizes the number of parts processed per minute.

The evolution of distributed analytics systems

Hadoop MapReduce

MapReduce is the heart of `Apache Hadoop`, it is a programming paradigm that enables massive scalability across hundreds or thousands of servers in a Hadoop cluster. The term `MapReduce` actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into key/value pairs.

Although Hadoop MapReduce is a powerful tool of big data, there are various limitations will be discussed:

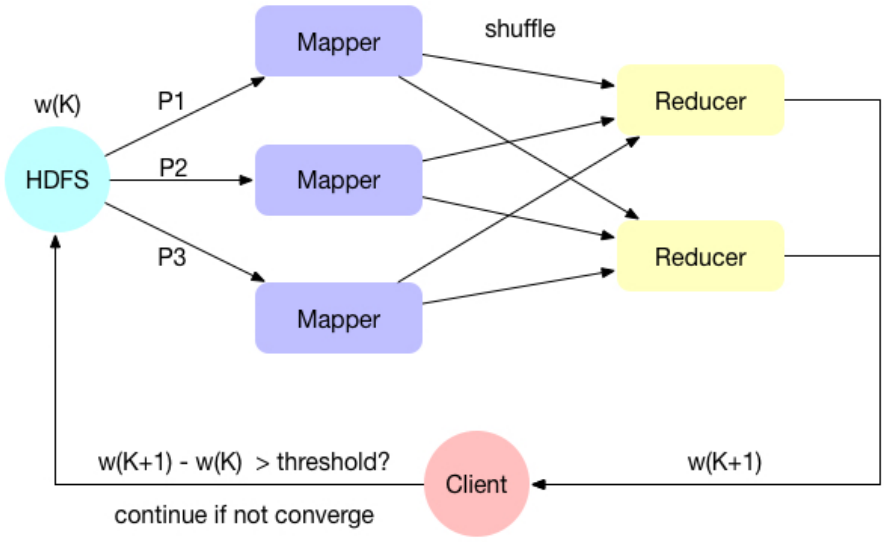
1. `No Caching`: MapReduce cannot cache the intermediate data in memory for a further requirement, which will diminishes the performance such as iterative tasks (These tasks need each output of the previous task be the input of the next stage). HaLoop and SparkRDD have make improvements on this, as them will accesses data from RAM instead of disk, which dramatically improves the performance of iterative algorithms that access the same dataset repeatedly.
2. `Slow Processing Speed`: When MapReduce process large datasets with different tasks, it will requires a lot of time to perform map and reduce functions, thereby increasing latency. This can be sloved by Dryad and FlumeJava based on the Directed Acyclic Graph

(DAG), which use a graph holds the track of operations. DAG will converts logical execution plan to a physical execution plan, which helps in minimize the data shuffling all around and reduce the duration of computations with less data volume, eventuallly increase the efficiency of the process with time.

3. `No Real-time Data Processing`: Hadoop MapReduce is designed for batch processing, which means it take a huge amount of data in input, process it and produce the output. Altough batch processing is very efficient for processing a high volume of data, but the output can be delayed significantly. Which will cause the MapReduce is not suitable for Real-time data processing. Naiad purposed a timely dataflow computational model, which suppport continuous input and output data. It emphasizes on the velocity of the data and it can be processed within a samll period of time.
4. `Spupport for Batch Processing Only`: Hadoop MapReduce only supports batch processing, it can't handle streaming, graphics, and machine learning data, so overall performance is slow. Husky proposes a unified framework that supports different types of tasks with multiple purposes. This enables high performance and supports the user-firend API in C++, Python and Scala.

In-Memory Processing

Hadoop MapReduce is not designed for iterative task like K-Means shown below. Every intermediate data will have not saved into persistent storage, which great increase the latency.



`HaLoop` is a great extension for Hadoop as it provides support for iterative application. In order to meet these requirement, several main changes that are made in Hadoop to efficiently support iterative data analysis:

1. Providing a new application programming interface to simplify the iterative expressions.
2. An automatic generation of MapReduce program by the master node using loop control module until the loop condition is met.

3. The new task scheduler supports data locality in these application in order to efficiently perform iterative operations.
4. The task scheduler and task tracker are modified not only to manage execution but also manage cache indices on slave module.

The HaLoop performance results demonstrate that pushing support for iterative programs into the MapReduce engine greatly improves the overall performance of itertive data analysis applications.

`RDD` stands for "Resilient Distributed Dataset", it is the fundamental data structure of Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster. As Hadoop MapReduce makes the iterative computing such as Logistic Regression, K-Means and PageRank slower. Although HaLoop guarantee faster computing with caching extension, the fault tolerance and other questions still exist. RDDs try to solve these problems by enabling fault tolerant distributed In-Memory computations.

Compare with HaLoop, when the worker node in Spark goes down, the system will use Lineage, a track of all the transformations that has to be applied on that RDD including from where it has to read the data, to re-compute the lost partition of RDD from the original one.

DAG Processing

Hadoop MapReduce restricts all computations to take a single input set and generate a single output set, which will cause extra overhead in solving tasks with multiply stages.

In `Dryad`, each job will be represented with a DAG, the intermediate vertices were writen to channels, and more operation than map and reduce will be used, such as join and distributed. With dataflow, the developer do not need to worry about the global state of the computing system, just need to write simple vertices that maintain local state and communicate with other vertices throught edges. Compare with DAG, MapReduce is just a simple form of dataflow, with two types vertices: the mapper and the reducer. Compare with MapReduce, Dryad offers more advantages:

1. Big jobs will be more efficient with Dryad
2. Dryad can provides explicit join, comnbines inputs of different types
3. Dryad "Split" proceduces outputs of different types

`FlumeJava` is a higher level interfaces to control data-parallel pipeline of MapReduce jobs. This allowed developers to write code which would be used to build an execution plan for a series of MapReduce jobs. With similar DAG idea, there are some difference between FlumeJava and Dryad:

	Dryad	FlumeJava
Publish Year	2007	2010
Purpose	General purpose distributed execution engine based on DAG Model	A higher level interface to control data-parallel pipeline
Implementation	C++	Java
Worker Model	Job Manager and Daemons	Pipeline Executor
Computation Model	Jobs were specified by arbitrary DAGs with vertexs as a program and each edges as data channel	Primitives: parallelDo, groupByKey, combineValues and flatten
Optimization	Graph composition, vertices are grouped into stages, pipelined execution, runtime dynamic graph refinement	Deferred evalution and execution plan
Scheduling	Based on network locality	Batch execution and MapReduce scheduling
Storage	Use local disks and distributed FS similar to GFS	GFS, local disk cache
Fault Tolerance	Task re-execution in context of pipeline	Similar to MapReduce

Stream Processing

Hadoop MapReduce was designed to support batch processing, it is not suitable for streaming data processing. Stream processing should enable users to query continuous data stream and detect conditions fast within a small time period fro mthe time of receiving the data, the detection time period may vary from few milliseconds to minutes.

The `Naiad` project is an investigation of data-parallel dataflow computation like Dryad, but with a focus on the low-latency streaming and cyclic computations. It introduces a new computational model called `Timely Dataflow`, which combines low-latency asynchronous message flow with lightweight coordination when required. Naiad's most notable performance property, when compared with other data-parallel dataflow systems, it its ability to quickly coordinate among the workers and establish that stages have completed. Naiad support efficient implemntations of a variety of progremming patterns, including nested iterative algorithms and incremental updates to iterative computations.

Popular stream processing framework includes Spark Streaming, Storm and Flink.

Machine Learning Processing

Many machine learning problems reply on large amounts of data for training, companies nowadays training algorithms with terabytes or petabytes of data, and create models out of it. Such models consist of weights that will optimize for error in inference for most cases. The number of weights/parameters run into orders billions to trillions. In such big model, learning on a single machine is not possible. It is useful to have a framework that can be used for distributed learning as well as inference.

A system called `Parameter Server` have been purposed for solving LDA algorithm efficiently, this framework have developed as a more general platform called `ps-lite` for now, the development history is below:

1. In 2010, Alex Smola purposed a parallel-LDA computing framework, which is the first generation parameter server, which use memcached as the parameter storage system. It can successfuly training LDA model in parallel, but still lack of efficiency and flexibility.
2. Jeff Dean from Google purposed the second generation parameter server called DistBelief, which stores massive deep learning model parameters into the global parameter server nodes. It efficiently solve the SGD and L-BFGS algorithm training problem in parallel.
3. Mu Li purposed the third generation parameter server called ps-lite, which is a more general platform support flexible consistency models, elastic scalability, and continuous fault tolerance.

General Purpose Platform

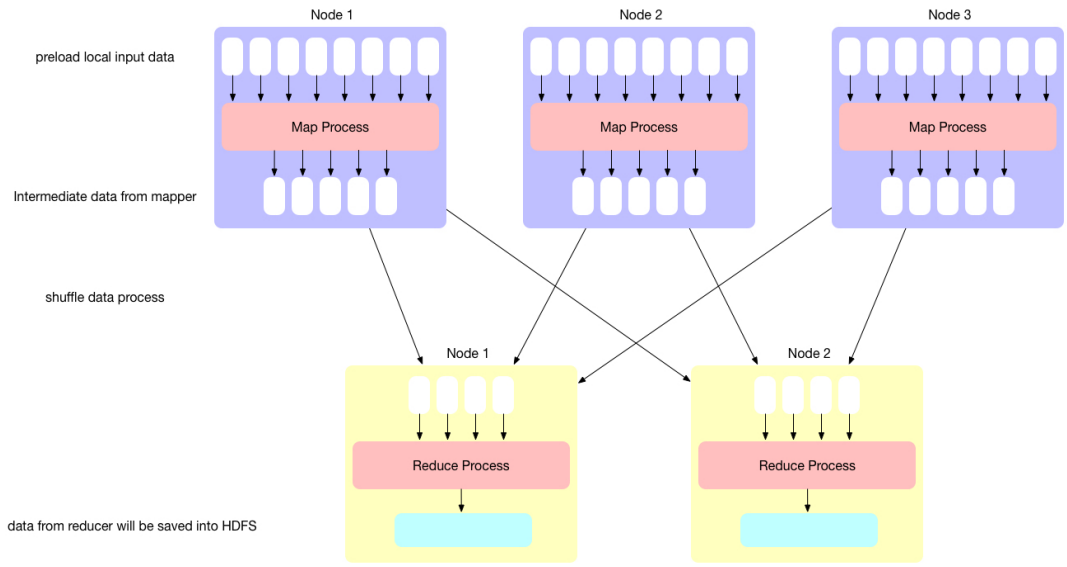
System like Hadoop and Spark have been widely adopted for big data processing, however, sometimes over-simplified API stop developers from more find-grained control and designing more effifient algorithms, but using sophisticated DSLs may result in development cost and bugprong programming.

A general research platform called `Husky` is able to help developers implement applications of different characteristics, for example, coarse-grained and fine-grained, iterative and non-iterative, sychronous and asynchronous workloads, and achieves performance close to or better than specialized systems and programs.

Hadoop MapReduce

`MapReduce` is mainly used for parallel processing of large sets of data stored in Hadoop cluster. In the very begining, it is a hypothesis designed by Google to provide parallism, data distribution and fault-tolerance. MapReduce process data in the form of key-value pairs. A KV pair is the mapping element between two linked data items.

For processing large sets of data MR comes into the picture. The developers are able to write MapReduce applications that could be suitable for their bussiness scenarios. The MR work flow undergoes different phases and the end result will be stored in HDFS with replications. JobTracker plays the vital role in scheduling jobs and will keep track of the entire map and reduce jobs. The detail source code can be found in my Github:
[hadoop_mapreduce_process_source_code]



The mapreduce process can be illustrated with core map and reduce process, but some details were hidden, the whole work flow should be:

Split->Mapper->Partioner->Sort->Combiner->Shuffle->Sort->Reducer->Output

Mapper Phase

In mapper phase, the input data is going to split into 2 components, key and value. The key is writable and comparable in the processing stage. Value is writable only during the processing stage. In this stage, one block is processed by one mapper at a time, in the mapper, a developer can specify his own bussiness logic as the requirement.

Partition Phase

The partition module in Hadoop MapReduce also play a very important role to partition the data received from either different mappers or combiners. Partitioner reduce the pressure that builds on reducer and gives more performance. There is a customized partition which can be performed on any relevant data on different basis or conditions.

Combine Phase

Combiner is also called mini reducer, usually the code is similar to the reducer. When the mapper output is huge amount of data, it will require high network bandwidth. To solve this issuce, the combiner can be placed after mapper to improve performance.

Shuffle & Sort

After the mapping process, there are shuffle and sorting process on the intermediate data. The data will be stored in the local file system without having any fault tolerance like replications or checkpoint in Hadoop nodes. In detail, Hadoop uses a Round-Robin algorithm to write the intermediate data to the local storage.

Reducer Phase

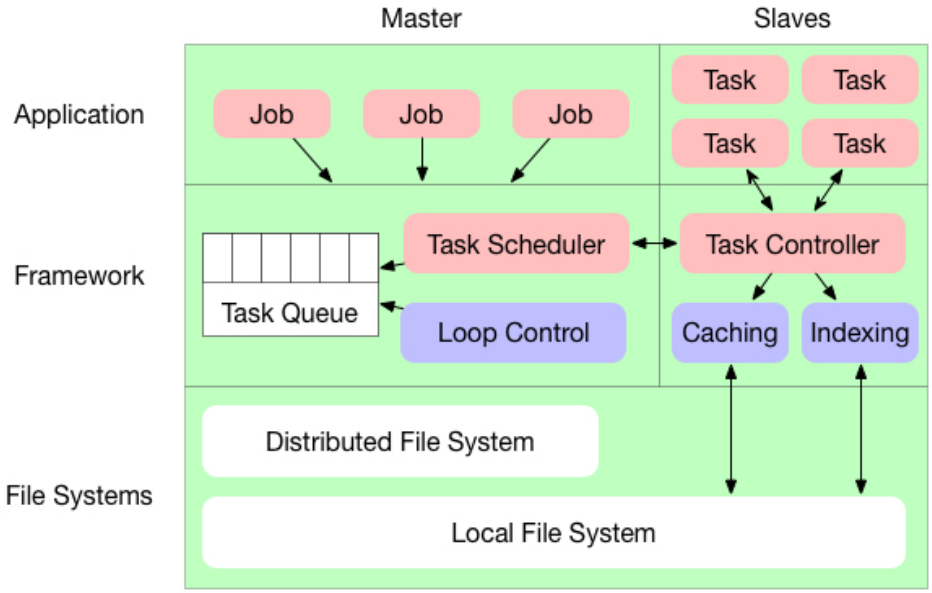
When the data was shuffled and sorted, it will be pass as the input to the reducers. In this phase, the developer can define cutomized bussiness code and use writer to writes data from reducer to storage like HDFS. Reducer mainly do operations on data from mapper, and finally will output data named like part-r-0001 etc.

HaLoop

HaLoop was developed as an extension of Hadoop and, along with the processing of data providers, is another way to perform iterative calculations on Hadoop MapReduce.

Architecture

To support extra features, there are some changes based on Hadoop MapReduce:



- Loop Control, Caching, Indexing: New module in HaLoop to support caching.
- Task Scheduler/Taracker, Job, Task: Based on Hadoop, made some chagnes to communicate with new HaLoop module.

Features

Important features of HaLoop after feasible changes have been made on Hadoop:

1. `Mapper Input Cache`: HaLoop's mapper input cache can avoid non-local data reads in the mapper during non-initial iterations. In the last iteration, if the mapper performs a non-local read on the input split, the split is cached on the local disk of the mapper's physical node. Through loop-aware task scheduling, the mappings in subsequent iterations can read this data from the local disk without having to read them from systems

such as HDFS.
2. `Reducer Input Cache`: HaLoop is able to cache the reducer input and create local cache data in all Reducers. In addition, the reducer input is cached before each reduce call, so the tuples in the reducer input cache are sorted and grouped by the reducer input key.
3. `Reducer Output Cache`: The reducer output cache stores and indexes the latest local output on each reducer node. This cahce is used to reduce the cost of evaluating fixed point termination conditions. If the application tests the convergence condition by comparing the current iteration output to the previous output, the cache will enable the framework to perform the comparison in a distributed manner.

Example

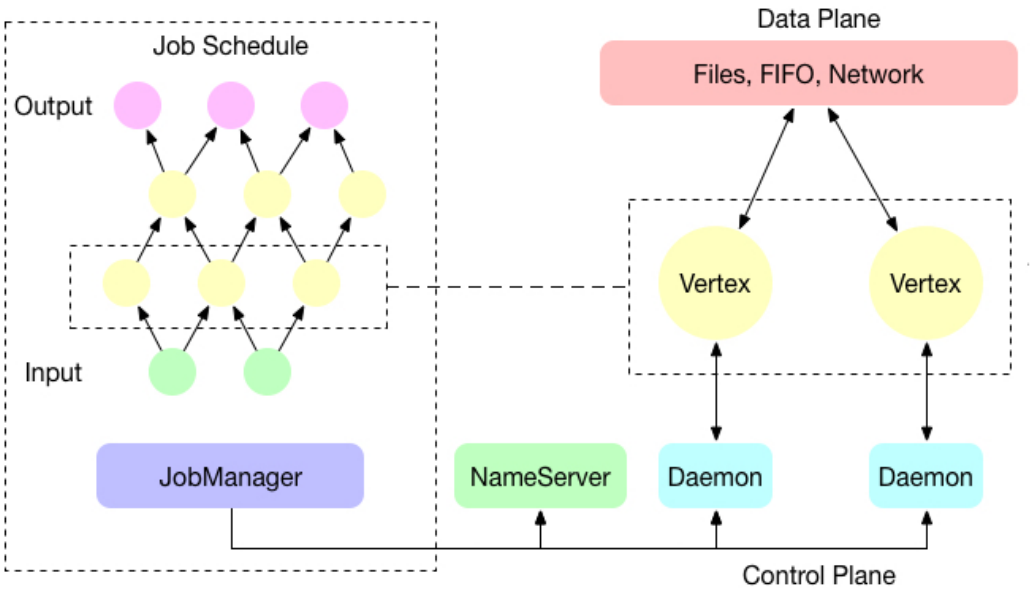
To enable these features, the new API call is simple and esay:

```
Job job = new Job();

// Ingore the same process in Hadoop MapReduce
...

// Trun on the intput/output caching
job.SetReducerInputCache(true);
job.SetReducerOutputCache(true);

job.Submit();
```

Dryad

Dryad is a versatile, high-performance distributed exectuion framework that builds an execution engine that can handle many of the challenges of creating large distributed and concurrent applications. Dryad supports many different data transfer mechanisms between computing vertex and explicit data stream graph construction and refinement.

Architecture



- `Job Manager`: Application-specific code that contains communication diagrams for building jobs and library code to schedule work across available resources. All data is sent directly between the vertices, and the administrator is only responsible for controlling the decision and is not responsible for any data transfer.
- `Name Server`: The location of each machine node in the network topology is exposed, and decisions are made based on location.
- `Daemon`: Runs on each compute node in the cluster and is responsible for creating processes on behalf of the job manager.
- `Vertex`: Executed on the node, data is sent from the job manager to the daemon and then from the cache. The daemon acts as an agent, so the job manager can communicate with remote vertices and get computational state.
- `Data Storage`: Dryad uses a distributed storage system to store output data, and large files can be divided into small chunks that are replicated and distributed on the cluster's local disk.

Dryad Graph

Dryad have defined a simple language that make it easy to specify commonly-occurring communication rules. It was "embedded" in C++ as a library using a mixture of method calls and operator overloading.

- `Vertices`: Created by calling the appropriate static program factory, the parameters can be set at creation time and then form closure and sent to remote process for execution.
- `Edges`: Created by applying a composition operation to two existing graph.
- `Job Stages`: When the graph was constructed with vertices and edges, every vertices was placed in a "stage" to simplify job management. The stage topology is the "skeleton" or summary of the overall job.

Run-time Graph Refinement

Dryad used the stage manager callback mechanism to implement the run-time optimization rules, which allow it scale to very large input sets when observing scarce network bandwidth.

If a computation is associative and communtative, Dryad can perform a data reduction, which can benefit the aggregated tree. A typical application could be the histogramming operation, which takes as input a set of partial histograms and output their union. The Dryad's implementation is to attaches a custom stage manager to the input layer, then the manager can receive callback notification when upstream vertices have completed and rewrites the graph with appropriate refinements.

FlumeJava

Flumejava was a library used in Google, based on expressive and convenient small set of composable primitives. By using deferred evaluation and optimization, the API could automatically transformed into an efficient execution plan. It is also a run-time system that executing optimized plans, which can transform logical computations info efficient programs.

Abstractions

Core Collections:

Name	Description
PCollection	The central class of the FlumeJava library, a immutable bag of elements with type T. It can be created from an in-memory java Collection type, or by reading a file in one of several possible formats
PTable	The second core class, represents a huge immuntable multi-map with keys of type. It is the subclass of PCollection<Pair<K, V>>

Core Operations:

Name	Description
parallelDo	Support elementwise computation over the input, can be used to express both the map and reduce parts of MapReduce
groupByKey	Converts a multi-map of type PTable into a uni-map of type PTable, it capture the essence of the shuffle step of MapReduce, there is also a variant that allows specifying a sorting order for the collection of values for each key
combineValues	Takes input and an associative combining function, return the output where each input collection of values has been combined into a single output value.
flatten	Take a list of input and return the single output that contains all the elements of the input
count	Take a PCollection and returns a PTable mapping each distinct element ofthe input PCollection to the number of times it occurs
join	Implement a kind of join over two or more PTables sharing the common key type
top	Take a comparison function and count N and return the greatest N elements of its receiver PCollection according to the comparison function

Deferred Evaluation

FlumeJava's parallel operations are executed lazily using deferred avaluation, in order to enable optimization, each PCollection object is represened internally either in deferred or materialized state. When a operation like parallelDo() is called, it just create a parallelDo deferred operation object and return a new deferred PCollection points to it. The result of executing a series operations is thus a DAG (Directed Asyclic Graph) of deferred PCollections and operations, it is also called the execution plan.

In order to acutally trigger the evaluation of a series of parallel operations, the developer need to call FlumeJava.run(), this will first optimizes the execution plan and visits each of the deferred operations in the optimized plan. When a deferred operation is evaluated, it will convert its result PCollection into a materialized state, FlumeJava will automatically deletes any temporary intermediate files.

Optimizer

1. `ParallelDo Fusion`: ParallelDo producer-consumer fusion is essentially a combination of functions or loops. For example, if a parallelDo operation executes the function f, the result is consumed by another parallelDo operation execution function g, and the two parallelDo operations are replaced with a single multiple output that computes the two functions.

2. `MSCR Operation`: The core optimizer in FlumeJava, which converts a combination of ParallelDo, GroupByKey, CombineValues, and Flatten operations into a single operation. To bridge the gap between these levels of abstraction, the optimizer includes an intermediate-level operation called MSCR opertiaon. This operation summarizes MapReduce by allowing multiple reductions and combinations.
3. `Overall Stragegy`: The optimizer is capable of executing a series of execution plans with overall goals, including: receiving flattening, boosting combined values, and merging MSCR.

Spark

`Spark` is a general-purpose computing platform. Essentially, it is a "computing engine" that is responsible for arranging, distributing, and monitoring applications that consist of many computing tasks in many machines or clusters. Spark extends the popular MapReduce model by using DAG processing similar to FlumeJava and Dryad to ensure fast and efficient data processing. The main feature of Spark is the in-memory RDD calculation model, which ensures fast operation and fault tolerance. Spark, on the other hand, is designed to cover a wide range of workloads, including batch applications, iterative algorithms, interactive queries, and stream processing. A highly accessible product available in Spark that provides simple APIs in Python, Java, Scala and SQL.

RDDs

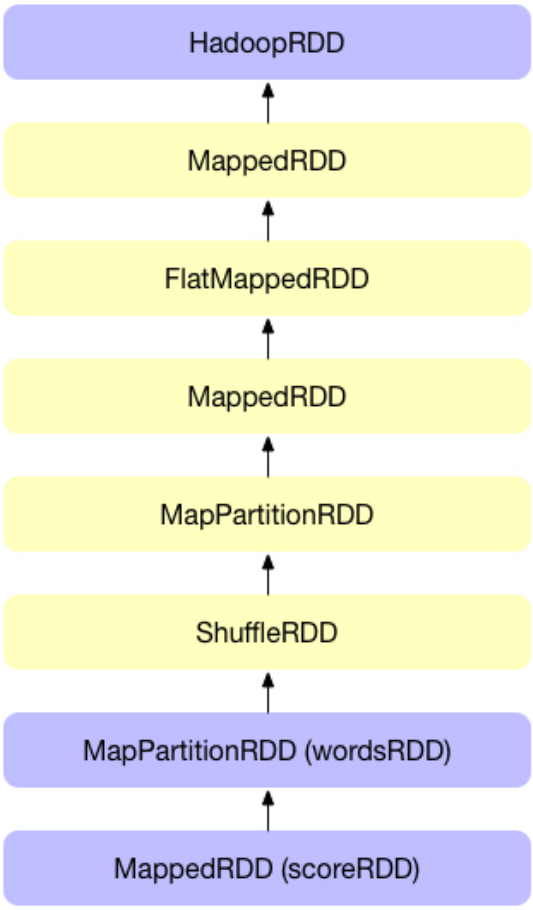
RDD is a collection of read-only partition records that can only be created by deterministic operations on stable storage or data in other RDDs.

RDDs have the following properties:

1. `Immuntability and partitioning`: An RDD consisting of a collection of partition records. Partitioning is the basic unit of parallelism in RDD. Each partition is a logical partition of data that is immutable and created by some transformation on the existing partition. Immutability helps to achieve consistency in calculations.
2. `Coarse Granular Operations`: A coarse-grained operation is applied to all elements in a dataset. For example, a map, filter, or groupBy operation will be performed on all elements in the RDD partition.
3. `Fault Tolerance`: Since RDD is created by a set of transformations, it records these transformations using the Lineage Graph, which recovers the data by re-executing the transformation instead of directly losing the data.
4. `Lazy Evaluation`: As with delayed execution in FlumeJava, RDD does not execute directly. Spark will use them in the final operation to calculate the RDD lazily so that it can manage the transformation.
5. `Persistence`: Developers can choose RDD storage, with options such as memory storage or distributed storage systems.

RDDs Lineage Example

In RDDs Lineage, each RDDs keeps track of its parent, as the example illustrate below.




```
var file = spark.textFile("hdfs://...")
var wordsRDD = file.flatMap(line => line.split("")).map(word => (word,
1)).reduceByKey(_ + _)
var scoreRDD = words.map{case (k, v) => (v, k)}
```

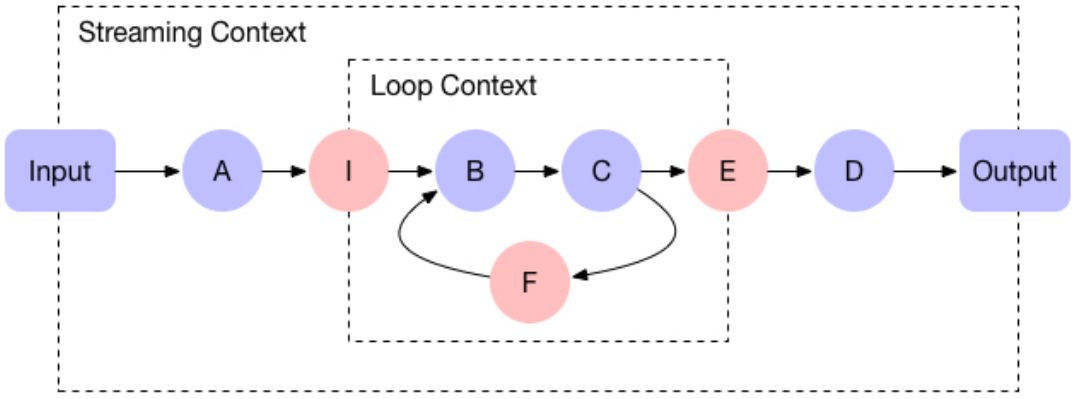
Naiad

`Naiad` is a genera purpose distributed sytem for executing data parallel, cyclic dataflow programs, it can fulfills all of these requirements and supports a wide variety of high-lvel programming models while achieveing the same performance as specialized system. A computational model called timely dataflow is adopt in Naiad, which provide the basis for an efficient, lightweight coordination mechanism.

Timely Dataflow

`Data stream` is a popular abstraction of parallel programming because it is combinable. Data stream developers can focus on writing relatively simple vertices that maintain local state, and can only communicate with other parts of the system through well-defined edges, rather than inferring the global state of the system. MapReduce is a static form of data stream with mapper and reducer, two user-defined vertices. DryadLINQ and Spark use modern language features

for functional programming to build data flow diagrams that run on a cluster.



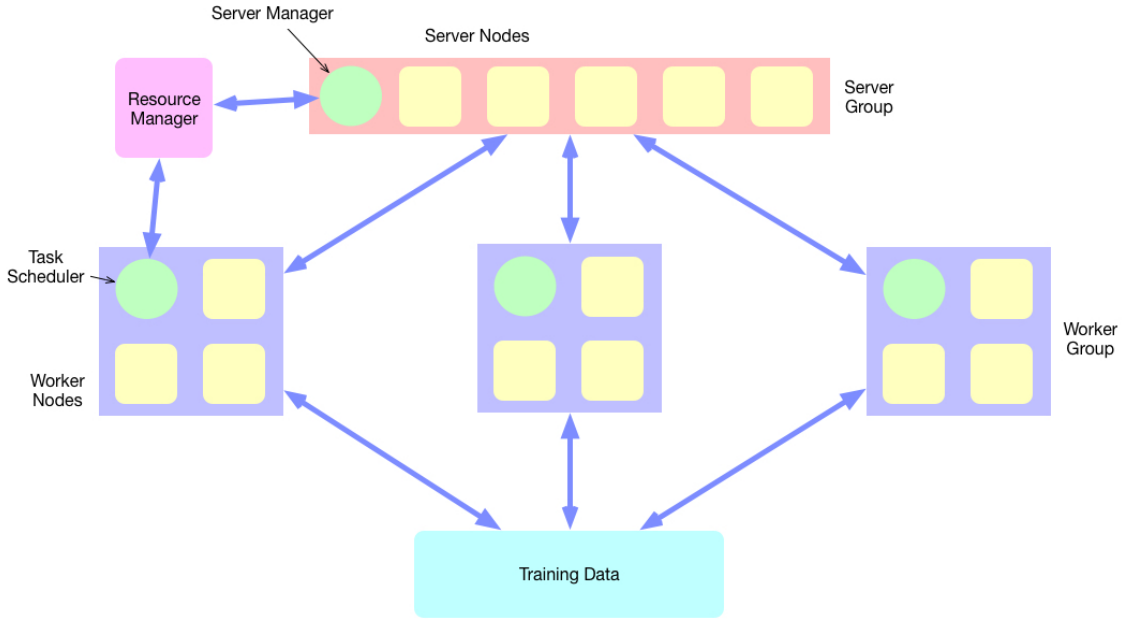
`Timely dataflow` add timestamps to messages flowing between vertices in a dataflow graph. For example, in the simple stream above, vertices A and D are not in any loop context, so the timestamps for these vertices do not have a loop counter. Vertices B, C, and F are nested within a single loop context, so their timestamps have a single loop counter. Ingrees I and Egress E verties are located at the boundary of the loop context, adding and removing loop counters, respectively, to add from the timestamp as the message passes through them.

PsLite

`Ps-lite` is the third generation, lightweight and efficient implementation of parameter server framework. It provides clean and powerful APIs for model parameter query and updates. Three core functions list below:

- Push(keys, values): Push a list of key-value pairs to the server nodes
- Pull(keys): Pull the values from servers for a list of keys
- Wait(): wait until a push or pull finished

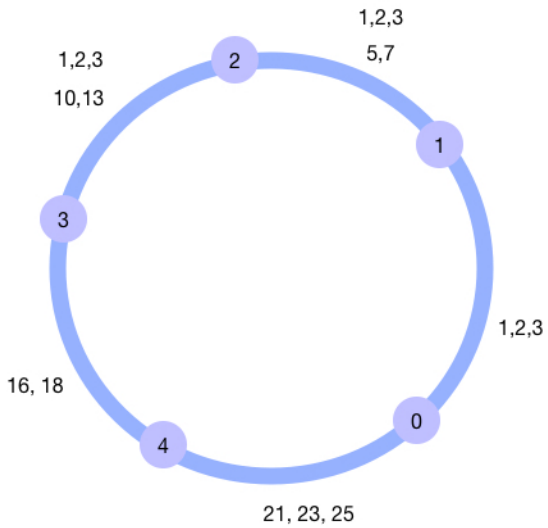
Architecture



- `Server Node`: Store part of the parameters of the model, provide query service to the worker nodes. Server nodes can communicate with each other achieve data replication
- `Server Manager`: Maintain the meta data of server nodes, for example, the running state and range of the parameters
- `Worker Node`: Store part of the training data, using the gradient descent method to calculate the parameters with iteration, it can only commuicate with server nodes
- `Task Scheduler`: Allocate jobs and monitor the process of the worker nodes

Fault Tolerance

To avoid the consequence of the fault in the server nodes, ps-lite achieve fault tolerance by using Dirtributed Hash Table (DHT), it can ensure the system is able to add and remove node dynamically, also guarantee different node maintain the replication from other nodes.



For example, the DHT is able to make sure node 0 maintain the origin data of (1,2,3), while node 1 and 2 also maintain these data as replication. This can not only aviod the influence from fault node, but able to provide load-balancing service when the network bandwidth consuption is heavy on particular node.

Husky

Husky is a open-source, efficient and expressive distributed computing framework, it gives developers freedom to express more data interaction patterns, it also minimizes programming complexsity nad guarantees automatic falult tolerance and load balancing.

Husky adopt the Master-Slave architecture, a cluster consis of one master and multiple workers, the master is responsible to coordinate the workers, while workers perform actual computations.

At Husky, global objects are partitioned and distributed to different jobs based on a consistent hash that supports dynamic addition and deletion of machines without having to reset everything. It is also an important implementation of design load balancing and fault tolerance.

There are core primitives:

1. `Compressed Pull`: A pull operation, Husky improve this by using compressed Bloom Filter to reduce network traffic.
2. `Shuffle Combiner`: Reduce the outbound messages of the worker, Husky using shuffle in the combiner to address certain scalability problem.
3. `Cache-Aware Optimization`: Using a dedicated data structure to aviod efficient problem in coarse-grained opertaion due to the poor locaity.

Related Systems

There are other systems designed for solving large scale graph and streaming processing problem.

Pregel

`Pregel` is a data flow paradigm and system for large-scale graphics processing created in Google to solve problems that are difficult or difficult to solve using only the MapReduce framework. Pregel's computational paradigm is now used by many graphics processing systems, and many popular graphics algorithms have been converted to the Pregel framework.

Pregel is essentially a message interface that is constrained to the edge of the graph. The idea is to "think like a vertex." An algorithm within the Pregel framework is an algorithm in which state calculations for a given node depend only on the state of its neighbors. The Pregel calculation takes the graph and the corresponding set of vertex states as its inputs. In each iteration, called a superstep, each vertex can send a message to its neighbor and process the message. Thus, each super step includes a round of messages and an update of the global vertex state passed between neighbors.

Spark Streaming

`Spark Streaming` enables stream processing for Apache Spark's language integrated API. That is: Letting developers write streaming jobs the same way as write common batch jobs.

Spark streaming is built upon a micro-batch approach. It discretizes the input data stream into batches of data, while the Spark engine continues to process the RDDs.

Comparison of distributed analytics systems

	MapReduce	**HaLoop**	**Dryad**	**FlumeJava**	**Spark**	**Naiad**	**Ps-Lite**	**Husky**
Year	2004	2010	2007	2010	2010	2013	2014	2016
Language Support	Java	Java	C#	Java	Scala, Python etc.	C#	C++	C++, Python etc.
Type	Batch	Batch	Batch	Batch	Comprehensive	Streaming	Batch	Comprehensive
In-Memory Support	x	✓	✓	✓	✓	✓	✓	✓
Data Flow	MR	MR	DAG	DAG	DAG	Timely	x	x
Latency	High	Middle	Middle	Middle	Middle	Low	Middle	Middle
Performance	Middle	Middle	High	High	High	High	Middle	High
Fault Tolerance	Replication	Replication	Replication	Replication	Lineage	CheckPoint	Replication	Replication
Messaging	TCP	TCP	TCP	TCP	TCP	TCP	ZeroMQ	ZeroMQ
Community Adaption	Wide	Selective	Selective	Wide	Wide	Growing	Selective	Selective

Future

At the 2018 World Artificial Intelligence Summit, Xiao Feng, CEO of China Wanxiang Co., Ltd., said that the blockchain plus encryption algorithm is a perfect match. The Internet is an "information machine" and the blockchain is a "fact machine." The data privacy will be protected, data assets will be secured, data sharing will be motivated and data calculations will be open. And he predict a decentralized distributed system will appear in the next five years.

I also think it is possible, as the blockchian tecenique geting more popular and mature, the governments and customers start to emphasize on the data privacy, most of the data processing task tend to be executed on the personal devices rather than the centralized servers. As the stand-alone devices' hardware keeping update and the emergence of 5G technology, the computing power and communication bandwidth are guaranteed.

I was an android system developer before I came to CUHK, as I known in the XiaoMi system team, they already start the project which is able to allow offline AI training in the MIUI system. I highly agree that the decentralized data processing is the future technique tread, and a new generation decentralized large scale computing system may be built in the next 10 years.

Conclusion

In this course survey report, I first list the basics knowledge that will be invovled inthe paper, then I illustrate the disadvantages of the Hadoop MapReduce, I analyzed the advantages of other data processing system such as FlumeJava, Spark and Husky and compare them with Hadoop MapReduce, and how those systems solve the weakness in MapReduce. In the remaining sessions, I introduce the techniques details, question they solved and main ideas behind those distributed computing system. Also, a comprehensive analysis was give among those works. Finally, I give a comparsion table about those systems and give a opinion that the decentralized computing may the future tread of large scale data analytics systems.

References

- MapReduce: Simplified Data Processing on Large Clusters
- HaLoop: Efficient Iterative Data Processing on Large Clusters
- Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks
- FlumeJava: Easy, Efficient Data-Parallel Pipelines
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
- Naiad: A Timely Dataflow System
- Scaling Distributed Machine Learning with the Parameter Server
- Husky: Towards a More Efficient and Expressive Distributed Computing Framework
- HALOOP: Iterative MapReduce
- An introduction to timely dataflow
- Distributed Algorithms and Optimization, Spring 2015: Lecture 8