Programming Models for Clouds

Arjun Jayaraj, Jeffrey John Geevarghese, Karthik Rajan, Umesh Kartha and Vineeth Samuel Varghese
Department of Computer Science and Engineering
National Institute of Technology (NIT), Calicut

***Abstract*—Cloud Platforms allow programmers to write applications that run in the cloud, or use services from the Cloud, or both while abstracting the essence of scalability and distributed processing. With the emergence of Clouds as a nascent architecture, we need abstractions that support emerging programming models. In recent years, Cloud Computing has led to the design and developement of diverse programming models for Massive Data Processing and Compute Intensive Applications. We survey different programming models for the Cloud, including the popular MapReduce Model and others which improve upon its shortcomings. Further, we look at models which are promising alternatives to these parallel programming models.**

***Index Terms*—Cloud Programming, Map-Reduce, SAGA, GridBatch, Transformer Model, Cloud Interoperability.**

I. Introduction

Over the years, many organizations have built large scale systems to meet the increasing demands of high storage and processing requirements of compute and data intensive applications[6]. With the popularity and demands on data centers, it is a challenge to provide a proper programming model which is able to support convenient access to large scale data for performing computations while hiding all low-level details of physical environments.

Cloud Programming is about knowing what and how to program on cloud platforms. Cloud platforms provide the basic local functions that an application program requires. These can include an underlying operating system and local support such as depolyment, management and monitoring. There are currently three main approaches to providing application support on Cloud Platforms.

Infrastructure as a Service(IaaS): In this model, the cloud provider simply provides a scalable platform for hosting client VMs. The most well known example is Amazon's Elastic Compute Cloud (EC2), and associated services such as Simple Storage Service (S3). Clients can make use of a basic set of web services to deploy a VM, to create an instance and to secure it. It is more apt to think of these "Clouds" as a platform for VMs rather than a virtualized OS or virtualized infrastructure.

Platform as a Service(PaaS): This refers to a class of cloud platform application support tools that can be used to directly support highly parallel data analysis. These platforms make use of a software design pattern called inversion of control - the application programmer provides the kernel of the computation and the framework invokes many copies of the kernels in parallel. Perhaps the most well known example is Google's MapReduce framework which we cover in detail in Section II.

Software as a Service(SaaS): In this approach, a remote language execution environment, such as Java VM, the Microsoft CLR or a language interpreter is provided by the platform. Using these remote execution environments, an application programmer can build a data access or analysis service on a local host and then push it to the data center for deployment and remote execution. The reader is referred to [4] for a detailed discussion of the above three architectures.

MapReduce has emerged as an important data-parallel programming model for Data-Intensive computing. However, most of the implementations of MapReduce are tightly coupled with the infrastructure. There have been programming models proposed which provide a high-level programming interface thereby providing the ability to create distributed applications in an infrastructure-independent way. SAGA [1] and Transformer [5] are examples of such models which try to implement parallel models like MapReduce [3] and AllPairs [4] taking considerable burden off the application developer.

The paper is organized as follows : Section II explores the popular Map-Reduce framework. In Section III, we explore two programming paradigms, namely SAGA(Section III-A) and Transformer(Section III-B) that try to improve upon the Map-Reduce framework. In Section IV, we briefly look at a new model which does not rely on parallel programming called GridBatch.

II. Map-Reduce Framework

MapReduce [3] is a popular programming model for processing and performing data intensive tasks on large datasets. It is a Google initiative for handling large scale web content and offers an excellent framework for developing data mining and machine learning applications in data centers.This is actually an implementation of an old idea from parallel computing and programming languages. It allows programmers to think in a data-centric fashion and focusses on applying transformations to sets of data records.

The MapReduce model is comprised of 2 operations : map and reduce. Any problem formulated this way can be

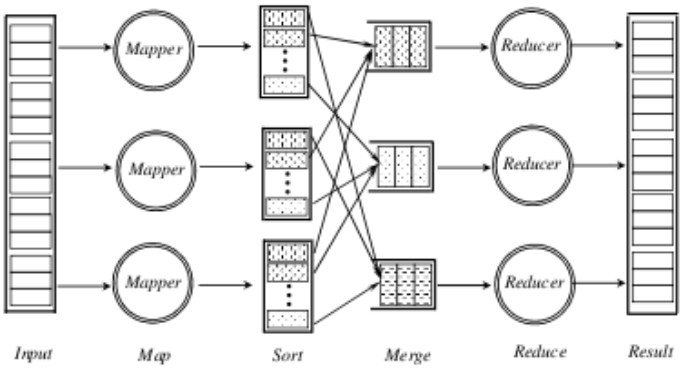

Fig. 1. Computation of MapReduce

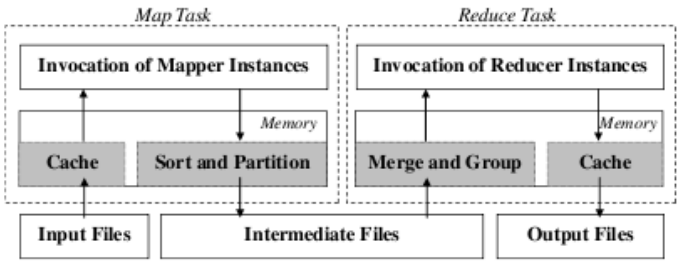

Fig. 2. DataFlow in MapReduce

parallelised automatically. It uses a user defined Map function that takes a key/value pair and computes a collection of key/value pairs.

$$map::(key1, value1) \rightarrow arraylist(key2, value2)$$

A reduce function maps a key/value-list to a list of values using a similar technique.

$$reduce::(key2, list(value2)) \rightarrow list(value3)$$

The task is carried out in 4 stages: Map, Sort, Merge and Reduce. The Map phase is fed with a set of key/value pairs. For each pair, the Mapper module generates a result. The Sort and Merge phases groups the data to produce an array, each element of which is a group of values for each key. The Reduce phase works on this data and applies the reduce function on it. The hash functions used in the map and reduce functions are user defined and varies with the application of the model. The overall computation is depicted in Figure 1.

The entire architecture (Figure 2) is grouped to form 2 independent modules.

Phase 1 (Map and Sort) **:** Each key-value pair is fed to the map function and results are sent to a global buffer. The buffer comprises of a definite number of buckets, each one for a different position. A threshhold is chosen and once the output exceeds the threshold, the buffer data is flushed on to the secondary storage. Before the buffered results are written into the disk, each bucket is sorted in the memory. Quick Sort is generally used. The data written on to the disk is thus sorted by key.

Phase 2 (Merge and Reduce) **:** Merge operation starts as and when it gets inputs. The results are grouped on the basis of key and the values are clubbed together on a list. In case of collision in buckets, the new value is appened towards the end of the value list for that key. Heap Sort is commonly used. The Reduce stage iterates over all keys and applies the user defined reduce function on them. The results obtained are written back to the disk.

MapReduce uses a Schedulling mechanism to optimise node utilization. Nodes are constantly monitored by the Scheduler to make sure that no tasks are stalled or show erroneous behaviour. Nodes tend to be slow when jobs are heterogenous or dominating users interfere with each other. Free Resources in clouds are shared by the resource owner as well as by other users who have idle resources. A static scheduling mechanism in this situation might lead to *soft failure*, where a MapReduce execution has to abort computation due to domination by its owner. Static Scheduling, hence, is not preferred in Enterprise Clouds. Realtime scheduling is higly efficient as it constantly picks out nodes that are idle and assigns new tasks to them. The Scheduler thus starts off by dispatching the map tasks to all nodes simulataneously. Whenever a reduce task is ready, it will be scheduled on the basis of the current status of resources.

MapReduce, being a parallel computing model, is both time consuming and error prone. It is not possible to incorporate a parallel computaion model for all applications. Communication, Coordination and Synchronoisation between the nodes are prime requirements. Most of the parallel programs are asynchronous and it is pretty hard to analyse the interaction between the machines.

We now look at two programming models which were actually proposed for Grid-based applications, but can adapt seamlessly to Cloud Infrastructures.

III. Parallel Programming Models

A. *SAGA - Simple API for Grid Applications*

Although Map-Reduce has emerged as an important data-parallel programming model for data-intensive computing, most, if not all, implementations of Map-Reduce are tightly coupled to a specific infrastructure. *SAGA* is a high level programming interface which provides the ability to create distributed applications in an infrastructure independent way.

SAGA supports different programming models and concentrates on the interoperability on Grid and Cloud infrastructures[1]. SAGA supports job submission across different distributed platforms, file access/transfer support and support logical file, as well as Check Point Recovery and

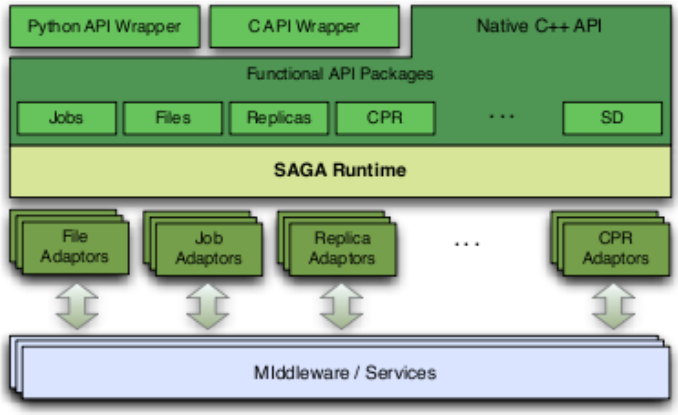

Fig. 3. Components include Programmer's interface, SAGA engine and functional adapters

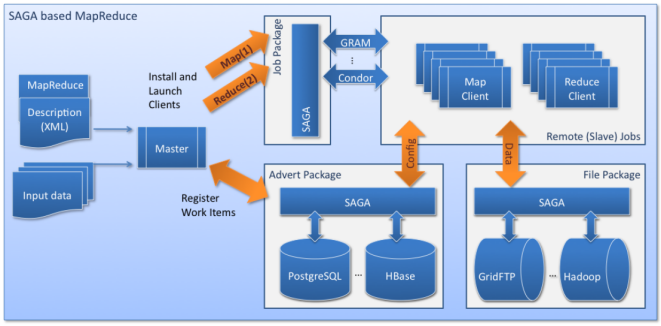

Fig. 4. Control flow diagram for SAGA-MapReduce

Service Discovery. SAGA API is written in C++ and supports other languages like Python, C and JAVA. The run-time environment decision making is given support by the engine that loads relevant adaptors (Figure 3).

SAGA Map-Reduce Implementation has instructions on location policy for distributed job scheduling that is written in between the core logic (Figure 4). The advantage of this approach is portability over a wide range of generic systems and does not have the single system dependency that the original Map-reduce implementation had. The drawback is the complexity of the implementation that requires system semantic capabilities to some extent in each of the new system added and the difficulty in achieving system specific optimizations. But these drawbacks are claimed to be restricted within in the framework.

The API provides all the functionalities required to implement Map-reduce algorithm with the complex functionalities like generation of blocks from input, classification of intermediate result, assigning map and reduce workers hidden from the user view. The master processes in the system have the responsibility of

- initiating all worker processes to achieve the map and reduce algorithm as per user definition
- Coordinating the whole of operation which includes assigning data for map workers, handle intermediate output files,etc.

The master process is the same for all Map Reduce algorithms. The worker process may get assigned for map or reduce function as per master processs discretion.A similair implementation of the All-Pairs model using SAGA is mentioned in [1].

SAGA adapter exists for different functionalities. The process of creating the required adapter in a light weight fashion is the feature that allows extending applications to different systems without much overhead making it an effective programming abstraction layer. This also allows the interoperability between different Cloud like infrastructure.

SAGA map reduce is not yet optimized, does not support multi threading. But the low overhead of creating adapters makes deploying SAGA implementation on different clouds light weight. When changed from one cloud to other, the application was just required to instantiate a different set of security credentials for each cloud.

B. *Transformer*

Even though there are existing programming models based on C++ and Java in the industrial market, they suffer from certain shortcomings. First, the programmers have to master the bulky and complex APIs in order to use the model. Secondly, most programming models are designed for specific programming abstractions and created to address one particular kind of problem. There is an absence of a universal distributed software framework for processing massive datasets. To address the above mentioned shortcomings, a new framework called *Transformer* is proposed which supports diverse programming models and is not problem specific.

Transformer is based on two concise operations: *send* and *receive*. Using transformer model we can build Map-Reduce[3], Dryad and All-Pairs[4] models. The architecture of the Transformer model is divided into two layers : Common Runtime and Model Specific System. This is done to reduce coupling. Runtime system handles tasks like flow of data between machines and executing tasks on different systems making use of send and receive functions from runtime API. Model- specific layer deals with particular model tasks like mapping, data partitioning, data dependencies etc.

Transformer has a master-slave architecture. Run time has components like manager, supervisor, worker, file and message sender/receiver. Model specific layer has only one component called controller. Every node has two communication components: a message sender and one message receiver. The master node issues commands for task execution on the the slave nodes. The slave nodes return the

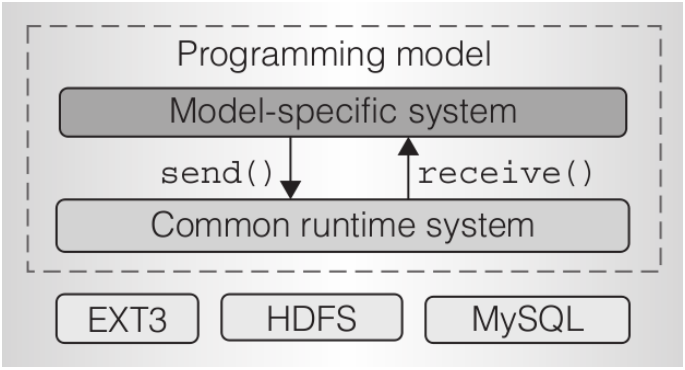

Fig. 5. Transformer's layered architecture

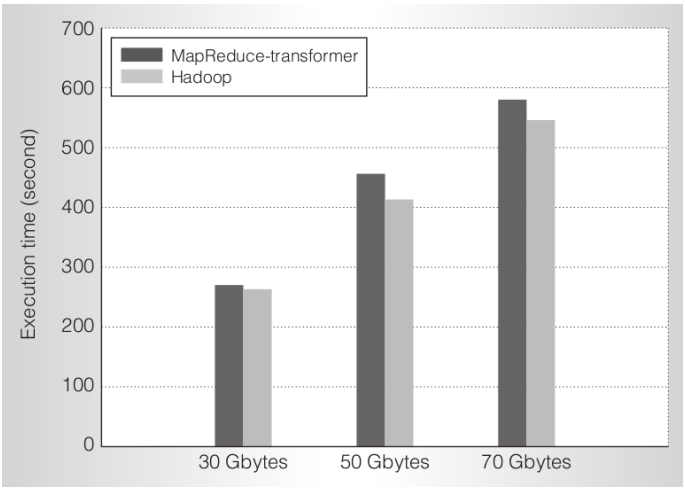

Fig. 6. Statistics of Word Count Application

status of execution when its over.The fault tolerance strategy is agile in nature. Failure is detected by the run-time system whereas fault-recovery is handled by model-specific layer. This involves re-running the tasks or resending data.

Transformer system is coded in Python using Actor Model as its design pattern. This approach have been found efficient than conventional multi-threaded applications. Communication between nodes is done using massage passing mechanism opposed to semaphores and conditions in threaded approach. Since the frequency of communication is high, asynchronous network programming is adopted and moreover the message is serialized before sending it.

Using the Transformer Model, all three known parallel programming models,namely Map-Reduce, Dryad and All-Pairs are implemented. It is interesting to note the implementation of Map-Reduce using the Transformer model. A comparison between Hadoop and Transformer's Map-Reduce using a word-count application is mentioned in [5]. The current implementation mentioned in [5] is a bit slower than Hadoop, mainly because of the coarse-grained data-partition strategy for the map stage. See Figure 6 for the

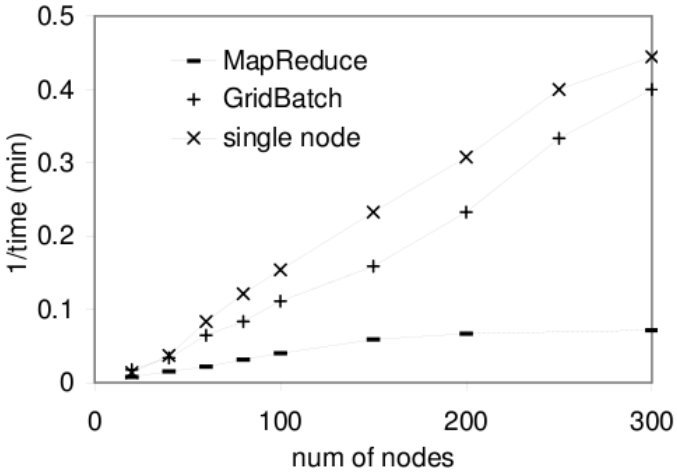

Fig. 7. Duplicate detection. Y axis is inverse of time in minutes. X axis is the number of nodes

Experimental results from the word-count application. For a detailed analysis of all the three models mentioned above, see [5].

IV. GridBatch Framework

Recently, an alternative to parallel computational models has been suggested that enables users to partition their data in a simplified manner while having highest possible efficiency.

The Grid Batch system has two fundamental data types: table or indexed table. A table is a set of rows that are independent of each other. An indexed table has all the properties of a table in addition to having an index associated with each record.

The two major software components of Grid Batch System are the Distributed File System (DFS) and the Job Scheduler. The DFS is responsible for storing and managing the files across all the nodes in the system. A file is broken down into many pieces and each of these pieces is stored on a separate node. The Job Scheduler constitutes of a master node and associated slave nodes. A job is broken down into many smaller tasks by the master node, and each of these tasks is distributed amongst the slave nodes.

The basic map and reduce operators in MapReduce system are extended in Grid Batch model. These are, namely, Map operator, Distribute operator, Join operator, Cartesian operator, Recurse operator and the Neighbor operator. For a detailed description, see [2]

The Grid Batch model claims a number of advantages over the popular Map-Reduce framework, namely

- *Interlace detection* :Interlacing detection is not possible to be implemented on MapReduce models as databases do not preserve sequence semantic, whereas this can be performed quite easily on a Grid Batch model with

high efficiency with its inherent neighbor and recurse operators.

- *Duplicate Detection* :From experimental results, it can be seen that MapReduce saturates at 150 nodes, owing to the limited network bandwidth. On the other hand, we learn that GridBatch scales super-linearly because little network bandwidth is consumed here. The objective mentioned in [2] would not be met in the MapReduce system even with a node capacity of 1000 nodes, whereas GridBatch system can process the entire data in just 4.3 minutes using only 250 nodes.
- *Data write throughput* : The throughput from a single server architecture as in MapReduce model will be limited by the server capabilities and the network bandwidth. In contrast, GridBatch system with its distributed architecture achieves the objective with high efficiency because the DFS servers write to different nodes at any given time, thus not saturating any network link.

V. Conclusion

With the emergence of Clouds and a general increase in the importance of data-intensive applications, programming models for data-intensive applications have gained significant attention: a prominent example being *Map-Reduce*. The usability and effectiveness of a programming model is dependent upon the desired degree of control in the application developement, deployment and execution. Programming systems such as SAGA (Section III-A) and Transformer (Section III-B) provide developers with the ability to express application decompositions and coordinations via a simple, high-level API. Models such as these provide motvation for further abstractions at multiple-levels. Recent frameworks (Section IV) understand the inherent difficulty in writing parallel programs and try to make the developers job easy. It is clear that these cloud platforms, their features and support for application programming is still in flux. A new successful platform innovation will have an enormous impact.

VI. Future Work

Although the primary paradigm underlying all the frameworks we have surveryed in this paper is parallel programming, as GridBatch in Section IV points out, parallel programming is not an easy task and makes the job of the developer inherently difficult. Future programming frameworks must allow client systems to develop robust, scalable programming models that, while relying on parallel computation, abstracts the details of parallelism. The end programmer should be exposed only with a set of APIs rather than the details of the distributed hardware, and frameworks like SAGA(Section III-A) and Transformer(Section III-B) try to acheive this at the cost of performance.

References

[1] Chris Miceli et al., *Programming Abstractions for Data Intensive Computing on Clouds and Grids*, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[2] Huan Liu, Dan Orban, *GridBatch: Cloud Computing for Large-Scale Data-Intensive Batch Applications*, 8th IEEE International Symposium on Cluster Computing and the Grid, 2008.

[3] Chao Jin and Rajkumar Buyya, *MapReduce Programming Model for .NET-based Cloud Computing*, The University of Melbourne, Australia.

[4] Dennis Gannon, *The Computational Data Center A Science Cloud*, Indiana University.

[5] Peng Wang et al., *Transformer: A New Paradigm For Building Data-Parallel Programming Models*, IEEE Computer Society, 2009.

[6] Shantenu Jha, Daniel S. Katz, Andre Luckow, Andre Merzky, Katerina Stamou, *Understanding Scientific Applications For Cloud Environments*, 2009.