Efficient Dynamic Resource Allocation Using Nephele in a Cloud Environment

V.Praveenkumar, Dr.S.Sujatha, R.Chinnasamy

Abstract— Today, Infrastructure-as-a-Service (IaaS) cloud providers have incorporated parallel data processing framework in their clouds for performing Many-task computing (MTC) applications. Parallel data processing framework reduces time and cost in processing the substantial amount of users' data. Nephele is a dynamic resource allocating parallel data processing framework, which is designed for dynamic and heterogeneous cluster environments. The existing framework does not support to monitor resource overload or under utilization, during job execution, efficiently. In this paper, we have proposed a framework based on Nephele, which aims to manage the resources automatically, while executing the job. Based on this framework, we have performed extended evaluations of Map Reduce-inspired data processing task, on an IaaS cloud system and compared the results with Nephele framework.

Index Terms— IaaS, high-throughput computing, Nephele, Map Reduce
.

1 INTRODUCTION

GROWING organizations have been processing immense amount of data in a cost effective manner using cloud computing mechanism. More of the cloud providers like Google, Yahoo, Microsoft and Amazon are available for processing these data. Instead of creating large data centers which are expensive, these providers move into architectural paradigm with commodity servers, to process these huge data [3]. Problems such as processing crawled documents or web request logs are divided into subtasks and these subtasks are distributed into the available nodes for parallel processing.

In order to process these distributed subtasks, all the cloud providers have integrated the framework in their clouds on the top of the architecture. Most popular frameworks are Google's Map Reduce [5], Microsoft's Dryad [9] and Yahoo!'s Map-Reduce-Merge [4]. These frameworks differ in design, their programming models share similar objectives, namely hiding the hassle of parallel programming, fault tolerance, and execution optimizations from the developer. The applications can be classified in terms like high-throughput computing or Many-Task computing based on the amount of data and the available number of tasks [12]. The applications can be written as sequential codes by the developer. The processing framework takes care of these tasks by distributing these into available nodes and executes each instance of the program on the appropriate fragment of data.

Growing companies processing their huge data in their own data center are obviously not an option. Instead, IaaS cloud providers such as Amazon EC2 provides same infrastructure as in the company's infrastructure in a pay-as you-go manner [1]. They provide complete control such as allocate and access the computing resources such as virtual machines (VMs), memory and processor. The charges apply for a period of time only when the virtual machines are allocated while processing. The virtual machines are categorized in different types based on their characteristics and cost.

VM abstraction of IaaS cloud fits the architectural paradigm so that the cloud providers integrate their processing framework in the cloud. The frameworks are imitating that they dynamically allocated the resources for processing. But they are allocating the resources from the beginning and won't de-allocate when the VM finished that work. As a result, rented resources may be inadequate for processing job, which may lower the overall processing performance and increase the cost.

Nephele is the first dynamic resource allocating framework [15]. This framework dynamically allocates the VM when needed for processing and de-allocates the VM when it completes its work or is not used for a long time, while job execution, automatically. But, Nephele fails to monitor the resource overload or underutilization while job execution, efficiently. In this paper, we have proposed a new parallel processing framework based on Nephele, which aims to manage the resources automatically while executing the job. Based on this framework, we perform extended evaluations of Map Reduce-inspired data processing task on an IaaS cloud system and compare the results with Nephele framework.
.

- *Praveenkumar is currently pursuing masters degree program in pervasive computing Technologies in Anna UniversityTrichy, India, PH-9976478020. E-mail: praveen.k930@gmail.com*
- *Dr.S.Sujatha is currently working as Assistant professer in master of computer application in Anna University Trichy, India, PH-01123456789. E-mail: sujatha@tau.edu.in*
- *R,Chinnasamy is currently pursuing masters degree program in pervasive computing Technologies in Anna UniversityTrichy, India, PH-9843248702. E-mail: chinnasamyrp @gmail.com*

2 RELATED WORK

Many-Task computing has been developed to accomplish many computational tasks over a short period of time, using a large number of resources, in a variety of systems. These systems share the common goals such as hiding issues of parallelism, data distribution or fault tolerance, they aim at different fields of application.

Map Reduce [5] (or the open source version Hadoop [25]) runs on a large static cluster of commodity machines and process large amounts of data. Map Reduce is simple and can process huge data only when the job fits into the map and reduce pattern. In order to process an immense amount of data, programmer has to write the code in distinct map and reduce the program. The processing framework takes care of scheduling the tasks and executing them. Similarly many frameworks were introduced to coordinate the execution of a sequence of Map Reduce jobs [10], [11].

Deelman [6] introduced the Pegasus framework designed for mapping complex scientific workflows onto grid systems. In Pegasus, the users have to describe their jobs as a DAG with vertices representing the tasks and edges representing the dependencies between them. The created workflows remain abstract until Pegasus creates the mapping between the given tasks and the concrete computing resources available at runtime. The authors incorporate the aspects like the scheduling horizon which determines at which point of time a task of the overall processing job should apply for a computing resource. In contrast, Pegasus' scheduling horizon is used to deal with unexpected changes in the execution environment. Pegasus uses DAGMan and Condor-G [8] as its execution engine for processing the workflows.

Daniel Warneke [15] introduced the Nephele framework designed for dynamically allocating the resource in the IaaS cloud for task scheduling and execution. In this framework processing job split into a number of subtasks, which are assigned to different types of virtual machines and are executed automatically. Scheduling horizon in the Pegasus framework [6] is related to the stage concept in Nephele, which is designed to minimize the number of allocated instances in the cloud and clearly focuses on reducing costs.

Nephele and Dryad [9] have some similar approaches like runs DAG-based jobs and offers to connect the involved tasks through file, network or in-memory channels. However, Dryad assumes an execution environment which consists of a fixed set of homogeneous worker nodes. Its scheduler is designed to distribute tasks across the available computing nodes in a way that optimizes the throughput of the overall cluster. It does not include the notion of processing cost for particular job.

Some of the on-demand resource providing projects arose recently. Dornemann [7] presented an approach to handle peak-load situations in BPEL workflows using Amazon EC2. Ramakrishnan [13] discussed how to provide a uniform resource abstraction over grid and cloud resources for scientific workflows. Both projects rather aim at batch-driven workflows than the data intensive, pipelined workflows which Nephele focuses on.

3 Problem Statement

The IaaS cloud providers integrate the processing framework to reduce the processing time and provide simplicity to the users. Reducing process time leads to reduction in the cost for attracting the users to use their cloud services. Several frameworks have been developed with some specific features (e.g. To reduce cost or increase performance) for cloud which reduce the complexities for the user. However, the existing well known frameworks like Google's MapReduce, Yahoo's MapReduceMerge need the job to be written in a distinct map and reduce program by the developer. MapReduce is very rigid, forcing every computation to be structured as a sequence of map-reduce pairs. Nephele framework introduces some basic issues for Dynamic allocation of instances.
The existing Nephele framework has some difficulties with the resource overload and underutilization problems during job execution. And also Nephele needs more user annotations to execute the tasks.

3 Overview of Extended Nephele Framework

Proposed framework follows most of the existing Framework Nepheles' functionality. This framework differs by including the resource monitor and resource manager in the Nephele Framework to perform load balancing automatically during job execution.

3.1 Architecture

Extended framework follows a classic master-worker pattern to process the given sequential code in the IaaS cloud. The extended framework describes the master node (i.e. VM) as Job Manager (JM) which is started before submitting the job to execute. The Job Manager, receives the clients' jobs, is responsible for scheduling them and coordinates their execution. It is capable of communicating with the Cloud Controller interface which the cloud operator provides to control the instantiation of VMs. By means of the Cloud Controller the Job Manager can allocate or deallocate VMs according to the current job execution phase. VMs are referred by instances as per the common cloud computing terminology. The term instance type will be used to differentiate between VMs with different hardware characteristics. For example, the instance type "small instance" could denote VMs with one CPU core, 1.7 GB of RAM, and a 160 GB disk while the instance type "Extra large" could refer to machines with 8 CPU cores, 16 GB RAM and a 1690 GB disk.

The executions of tasks are carried out by a set of instances called Task Managers (TM). These worker nodes receive one or more tasks from the Job Manager at a time, execute them, and after that, inform the Job Manager about their completion or possible errors. JM allocates the subtasks to the TM according to the type and size of the job. The newly allocated instances boot up with a previously compiled VM image. The image is configured to automatically start a Task Manager and register it with the Job Manager. Once all the necessary Task Managers have successfully contacted the Job Manager, it triggers the execution of the scheduled job.

Resource monitor is used to calibrate the amount of subtasks being distributed to each instance (VMs) by using the Resource Manager. Resource Manager is the responsible for reallocating the subtasks to the instances according to the execution phase.

The following figure shows the Extended Nephele structure.

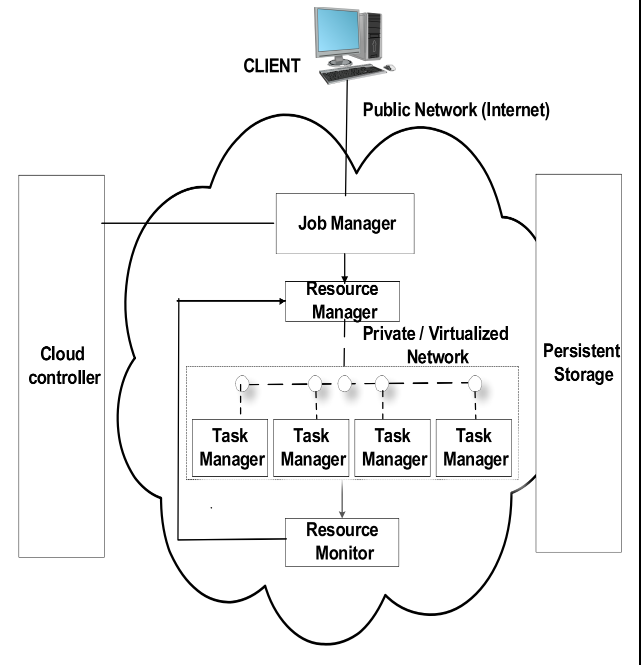

Fig. 1. Structural overview of the Extended Nephele Framework running in an Infrastructure-as-a-Service (IaaS) cloud.
.

Fig. 1. illustrates the extraneous work done by the proposed parallel data processing framework in performing the parallel execution of tasks on the supplied job and monitoring the resource utilization of a task that makes of the parallel computing capabilities of the cloud.

Initially, the VM images used to boot up the Task Managers are blank and do not contain any of the data the Nephele job is supposed to operate on. As a result, we expect the cloud to offer persistent storage (like, e.g., Amazon S3 [2]). This persistent storage is supposed to store the job's input data and eventually receive its output data. It must be accessible for both the Job Manager as well as for the set of Task Managers, even if they are connected by a private or virtual network.

3.2 Job Description

The job of the Extended Nephele Framework is expressed as a directed acyclic graph (DAG) which allows tasks to have multiple inputs and output gates. Each vertex in the graph represents a task of the overall processing job; the graph's edges define the communication flow between these tasks.

Defining an Extended Nephele job comprises three mandatory steps:

First, the user must write the program code for each task of his processing job or select it from an external library. Second, the task program must be assigned to a vertex. Finally, the vertices must be connected by edges to define the communication paths of the job.

The user has to submit the job as the DAG graph which specifies the tasks as the vertices and communication flow as the edges. This DAG graph is called as the Job Graph in the Extended Nephele framework. Users should be able to describe the tasks and the relationships on the abstract level

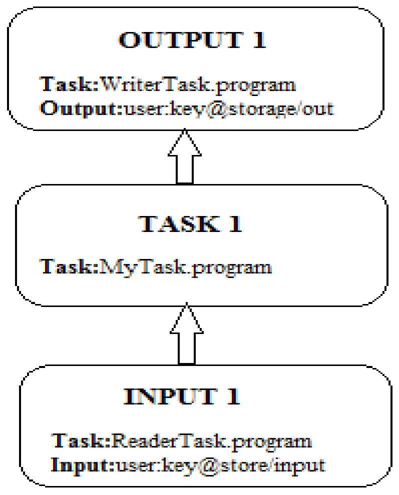

Fig. 2. An example of Job Graph.

Fig. 2. illustrates the simplest Job graph which consists of one input, one Task and one output vertex. For generating the job graph user must have some ideas about aspects like number of subtasks, number of subtasks per instances, sharing instances between tasks, channel types and instance types for job descriptions.

Once the Job Graph is specified, the user submits it to the Job Manager together with the credentials which the user has obtained from the cloud operator. The credentials are required since the Job Manager must allocate/deallocate instances during the job execution.

3.3 Job Scheduling and Execution

After receiving the valid Job Graph the JM converts it into the Execution Graph which is the primary data structure for scheduling and monitoring the execution of the extended Nephele job. It contains all the concrete information required to schedule and execute the tasks in the cloud. Fig. 3. Shows the Execution Graph for the given Job Graph (i.e, Fig. 2.). Here Task 1 is, e.g., Split into two parallel subtasks which are both connected to the task Output 1 using file channels and are all scheduled to run on the same instance.

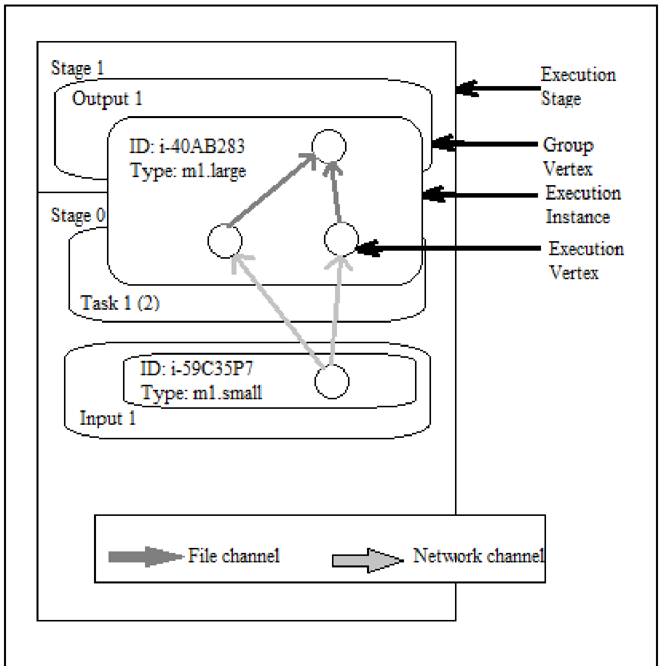

Fig. 3. An Execution Graph created from the original Job Graph.

The Basic Execution Graph structure is no longer a pure DAG. It resembles in two different levels of details, an abstract level and a concrete level. On the abstract level, the Execution Graph equals the user's Job Graph. In the concrete level more fine-grained graph defines the mapping of subtasks to instances and the communication channels between them.

Execution Graph consists of a Group Vertex for every vertex in Job Graph represent distinct tasks of the overall job. Execution stages are used to avoid the instance type availability problems in the cloud. Subtasks are represented in the Execution graph called Execution vertex which is controlled by its corresponding Group vertex. Each subtask is mapped to an Execution Instance which is defined by an ID and an instance type representing the hardware characteristics of the corresponding VM.

After submitting the job to the JM, it divides the job into subtasks and schedules them into a number of Task managers according to the number of subtasks. These subtasks are given to the TM using the any type of channel according to the type of the job. The channel may be network, file or in-memory channel.

3.4 Load Balancing

After the execution starts the TM must be monitored for providing better performance. While executing if any of the instances execute more tasks compared to others it may cause resource overload problem. Or at the same time any of the instances are process below the normal processing range it cause underutilization problem. These two problems take effect in the billing charges. To overcome this inside the IaaS cloud there must be a resource monitor check the TM and has to load balance the resource. This can be done by using user notification or by the job manager has to allocate/deallocate resource according to the memory used in the instances.

The framework does a runtime based analysis and monitoring of the utilization of the allocated resources by the task that performs parallel computation and solves the discrepancies (overutilization of underutilization of the allocated resources) and does the corresponding actions like allocating or de-allocating the needed resource and thus saving the cost of computation involved in the task thereby reducing the time consumed and also the capital expenditure consumed. The idea illustrated by the above figure is explained by the following pseudo code.

```
# Initialize the job scheduler

Init()

#Allocate the resources

allocateResource()

int x=calculateMemoryNeeded();

allocateAppropriateMemory(x,num_instances);

startResourceMonitor();

if(resource_occupancy<allocated_memory){

        int x=calculateDifference(instance);

        reduceAmount(x,instance);

        }

if(resource_occupancy>allocated_memory){

        int x=calculateDifference(instance);

        addAmount(x,instance);
        }
```

4 Conclusion

In this paper we proposed a parallel processing framework extending Nephele which avoids the resource overload and underutilization during job execution and executed a word count application based on this framework presented a performance comparison to the Nephele framework and found out as a substantial methodology for using the allocated resources efficiently without wasting the resource thereby reducing the cost involved in the computation task. For future work we are working for including a Scheduling algorithm in this parallel processing framework which reduces the user notification to execute the job.

REFERENCES

[1] Amazon Web Services LLC," Amazon Elastic Compute Cloud‖(Amazon EC2),",htthttp://aws.amazon.com/ec2 /, 2012

[2] Amazon Web Services LLC, "Amazon Simple Storage Service,"http://aws.amazon.com/s3/ , 2012.

[3] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou, "SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets," Proc. Very Large Database

[4] H. Chih Yang, A. Dasdan, R.-L. Hsiao, and D.S. Parker, "Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters," Proc. ACM SIGMOD Int'l Conf. Management of Data, 2007

[5] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proc. Sixth Conf. Symp. Operating Systems Design and Implementation (OSDI '04), p. 10, 2004

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, and D.S. Katz, "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," Scientific Programming, vol. 13, no. 3, pp. 219-237, 2005.

[7] T. Dornemann, E. Juhnke, and B. Freisleben, "On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud," Proc. Ninth IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGRID '09), pp. 140-147, 2009

[8] J. Frey, T. Tannenbaum, M. Livonia, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Cluster Computing, vol. 5, no. 3, pp. 237-246, 2002

[9] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," Proc. Second ACM SIGOPS/EuroSys European Conf.Computer Systems (EuroSys '07), pp. 59-72, 2007[10]

[10] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 1099-1110, 2008.

[11] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the Data: Parallel Analysis with Sawzall," Scientific Programming, vol. 13, no. 4, pp. 277-298, 2005.

[12] I. Raicu, I. Foster, and Y. Zhao, "Many-Task Computing for Grids and Supercomputers," Proc. Workshop Many-Task Computing on Grids and Supercomputers, pp. 1-11, Nov. 2008.

[13] L. Ramakrishnan, C. Koelbel, Y.-S. Lee, R. Wolski, D. Nurmi, D. Gannon, G. Obertelli, A. YarKhan, A. Mandal, T.M. Huang, K. Thyagaraja, and D. Zagorodnov, "VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance," Proc. Conf. High Performance Computing Networking, Storage and Analysis (SC '09), pp. 1-12, 2009.

[14] The Apache Software Foundation "Welcome to Hadoop!" http://hadoop.apache.org/,2012

[15] D. Warneke and O. Kao, "Nephele: Efficient Parallel Data Processing in the Cloud," Proc. Second Workshop Many-Task Computing on Grids and Supercomputers (MTAGS '09), pp. 1-10, 2009

[16] T. White, Hadoop: The Definitive Guide. O'Reilly Media, 2009. Endowment, vol. 1, no. 2, pp. 1265-1276, 2008.