International Journal Of Advanced Research and Innovations Vol.1, Issue .1

ISSN Online: *2319 – 9253*

Print: *2319 – 9245*

A Framework for Parallel data processing in Cloud systems

Shripad Subhash Lokhande [1] **Prof. Manoj Limchand Bangar** [2] **Sunil Narayan Shirsat** [3]

1. Assistant Professor, Department of I.T. Smt. Kashibai Navale College of Engineering
2. Professor, Department of I.T. Smt. Kashibai Navale College of Engineering
3. Assistant Professor, Department of I.T. Smt. Kashibai Navale College of Engineering

ABSTRACT

Cloud computing refers to the use and access of multiple server-based computational resources via a digital network , In cloud computing, applications are provided and managed by the cloud server and data is also stored remotely in the cloud configuration. Cloud Computing is gaining acceptance in many IT organizations, as an elastic, flexible and variable-cost way to deploy their service platforms using outsourced resources. Major Cloud computing companies have started to integrate frameworks for parallel data processing in their product portfolio, making it easy for customers to access these services and to deploy their programs. However, the processing frameworks which are currently used have been designed for static, homogeneous cluster setups and disregard the particular nature of a cloud. In this paper, we discuss the opportunities and challenges for efficient parallel data processing in clouds and present our research project Nephele. Nephele is the first data processing framework to explicitly exploit the dynamic resource allocation offered by today's IaaS clouds for both, task scheduling and execution. Particular tasks of a processing job can be assigned to different types of virtual machines which are automatically instantiated and terminated during the job execution. Based on this new framework

Index Terms: Cloud Computing, loosely coupled applications, many-task computing, Many-Task Computing, High-Throughput Computing, Loosely Coupled Applications, Cloud Computing

I. INTRODUCTION

The vast amount of data they have to deal with every day has made traditional database solutions prohibitively expensive [5]. Instead, these companies have popularized an architectural paradigm that is based on a large number of shared-nothing commodity servers. Problems like processing crawled documents, regenerating a web index are split into several independent subtasks, distributed among the available nodes and computed in parallel. Many-Task Computing (MTC) paradigm [1] embraces different types of high-performance applications involving many different tasks, and requiring large number of computational resources over short periods of time. These tasks can be of very different nature, with sizes from small to large, loosely coupled or tightly coupled, or tightly coupled, or compute intensive or data-intensive. Cloud computing technologies can offer important benefits for IT organizations and data centers running MTC applications: elasticity and rapid provisioning, enabling the organization to increase or decrease its infrastructure capacity within minutes, according to the computing necessities. Today a growing number of companies have to process huge amounts of data in a cost-efficient manner. Classic representatives for these companies are operators of Internet search engines, like Google, Yahoo, or Microsoft. The vast amount of data they have to deal with every day has made traditional database solutions prohibitively expensive [5].

Instead, these companies have popularized an architectural paradigm based on a large number of commodity servers. Problems like processing crawled documents or regenerating a web index are split into several independent subtasks, distributed among the available nodes, and computed in parallel.

In order to simplify the development of distributed applications on top of such architectures, many of these companies have also built customized data processing frameworks. Examples are Google's MapReduce [9], Microsoft's Dryad's or Yahoo!'s Map-Reduce-Merge [5]. They can be classified by terms like high-throughput computing (HTC) or many-task computing (MTC), depending on the amount of data and the number of tasks involved in the computation. Although these systems differ in design, their programming models share similar objectives, namely hiding the hassle of parallel programming, fault tolerance, hiding the hassle of parallel programming, fault tolerance, can typically continue to write sequential programs.

The processing framework then takes care of distributing the program among the available nodes and executes each instance of the program on the appropriate fragment of data. For companies that only have to process large amounts of data occasionally running their own data center is obviously not an option.

Instead, Cloud computing has emerged as a promising approach to rent a large IT infrastructure on a short-term pay-per-usage basis. Operators of so called IaaS clouds, like Amazon EC2 [1], let their customers allocate, access, and control a set of virtual machines (VMs) which run inside their data centers and only charge them for the period of time the machines are allocated. The VMs are typically offered in different types, each type with its own characteristics (number of CPU cores, amount of main memory, etc.) and cost. Amazon has integrated Hadoop as one of its core infrastructure services [2]. However, instead of embracing its dynamic resource allocation, current data processing frameworks rather expect the cloud to imitate the static nature of the cluster environments they were originally designed. In this paper, we want to discuss the particular challenges and opportunities for efficient parallel data processing in clouds and present Nephele, a new processing framework explicitly designed for cloud environments. Most notably, Nephele is the first data processing framework to include the possibility of dynamically allocating/deallocating different compute resources from a cloud in its scheduling and during job execution. This paper is an extended version of [12].

The paper is structured as follows: Section 2 starts with analyzing the above mentioned opportunities and challenges and derives some important design principles for our new framework. In Section 3, we present Nephele's basic architecture and outline how jobs can be described and executed in the cloud.

Requirements and Resources

Current data processing frameworks like Google's Map Reduce or Microsoft's Dryad engine have been designed for cluster environments. In this section, we discuss how abandoning these assumptions raises new opportunities but also challenges for efficient parallel data processing in clouds.

An Opportunities One of an IaaS cloud's key features is the provisioning of compute resources on demand. New VMs can be allocated at any time through a well-defined interface and become available in a matter of seconds. Machines which are no longer used can be terminated instantly and the cloud customer will be charged for them no more. Moreover, cloud operators like Amazon let their customers rent VMs of different types, i.e., with different computational power, different sizes of main memory, and storage. Hence, the compute resources available in a cloud are highly dynamic and possibly heterogeneous.

This new paradigm allows allocating compute resources dynamically and just for the time they are required in the processing workflow. For, e.g., a framework exploiting the possibilities of a cloud could start with a single VM which analyzes an incoming job and then advises the cloud to directly start the required VMs according to the job's processing phases. After each phase, the machines could be released and no longer contribute to the overall

cost for the processing job. First, the scheduler of such a framework must become aware of the cloud environment a job should be executed in. Second, the paradigm used to describe jobs must be powerful enough to express dependencies between the different tasks the jobs consists of. The system must be aware of which task's output is required as another task's input.
Finally, the scheduler of such a processing framework must be able to determine which task

Nephele's Architecture

Nephele's architecture follows a classical master-worker pattern as illustrated in Fig. 1.

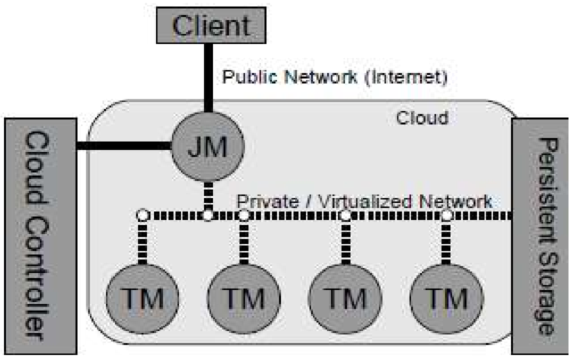


Figure 1: Structural overview of Nephele

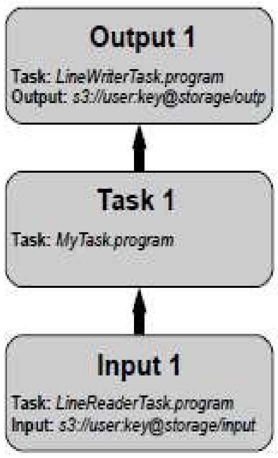
running inside a compute cloud Before submitting a Nephele compute job, a user must start an instance inside the cloud which runs the so called Job Manager (JM). The Job Manager receives the client's jobs, is responsible for scheduling them and coordinates their execution. It is capable of communicating with the cloud controller through a web service interface and can allocate or de allocate virtual machines according to the current job execution phase. We will comply with common cloud computing terminology and refer to these virtual machines as instances for the remainder of this paper. The term instance type will be used to differentiate between virtual machines with different hardware characteristics. The actual execution of tasks which a Nephele job consists of is carried out by a set of instances. Each instance runs a local component of the Nephele framework we call a Task Manager (TM). A Task Manager receives one or more tasks from the Job Manager at a time, executes them and after that informs the Job Manager about their completion or possible errors. Unless a job is submitted to the Job Manager, we expect the set of instances (and hence the set of Task Managers) to be empty. Upon job reception the Job Manager then decides, depending on the particular tasks in-side the job, how many and what type of instances the job should be executed on, and when the respective instances must be allocated/deallocated in order to ensure a continuous but cost-efficient processing. The concrete strategies for these scheduling decisions are explained later in this section. The newly allocated instances boot up with a previously compiled virtual machine 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. Initially, the virtual machines 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 [3]).

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.

We also decided to use DAGs to describe processing jobs for two major reasons: The First reason is that DAGs allow tasks to have multiple input and multiple output edges. This tremendously simples the implementation of classical data combining functions like, e.g., join operations [6].

Second and more importantly, though, the DAG's edges explicitly model the communication paths which exist inside the processing job. As long as the particular processing tasks only exchange data through these designated communication edges, Nephele can always keep track of what instance might still require data from what other instances and which instance can potentially be shut down and deallocated. Defining a 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. Tasks are expected to contain sequential code and process so-called records, the primary data unit in Nephele. Users may define arbitrary types of records, all implementing a common interface. From a programmer's perspective records enter and leave the task program through input or output gates. A task may have an arbitrary number of these input and output gates, which are at runtime connected by the Nephele framework to transport records from one task to the other.


Output 1
Task: LineWriterTask.program
Output: s3://user:key@storage/outp
Task 1
Task: MyTask.program
Input 1
Task: LineReaderTask.program
Input: s3://user:key@storage/input

After having specified the code for the particular tasks of the job, the user must define a so-called Job Graph. The Job Graph maps each task to a vertex of a directed acyclic graph (DAG) and determines the communication paths between these. Vertices with either no incoming or outgoing edges are treated specially in Nephele: The tasks assigned to these vertices are considered to be either data sources (input vertices) or sinks (output vertices) in the processing workflow. They can be associated with a URL pointing to where to read or write the data. Figure 2 illustrates the simplest possible Job Graph, consisting only of one input, one task and one output vertex. One major design goal of Job Graphs has been simplicity: Users should be able to describe tasks and their relation- ships on a very abstract level, leaving aspects like task parrallelization and the mapping to instances to Nephele. However, users who wish to specify these aspects explicitly can provide further annotations to their job description.

These annotations include:

Number of subtasks: A developer can declare his task to be suitable for parallelization. Users that include such tasks in their Job Graph can specify how many parallel subtasks Nephele should split the respective task into at runtime. Subtasks execute the same task code, however, they typically process different fragments of the data.

- Number of subtasks per instance: By default each (sub)task is assigned to a separate instance. In case several (sub)tasks are supposed to share the same in- stance, the user can provide a corresponding annotation with the respective task.

Sharing instances between tasks: Subtasks of different tasks are usually assigned to different (sets of) instances unless prevented by another scheduling restriction. If a set of instances should be shared between different tasks the user can attach a corresponding annotation to the Job Graph.

- **Channel types**: For each edge connecting two vertices the user can determine a so-called channel type. Before executing a job, Nephele requires all edges of the original Job Graph to be replaced by a channel type. The channel type specifies how records are transported from one (sub)task to another at runtime. The choice of the channel type can have several implications on the entire job schedule including when and on what instance a (sub)task is executed.

- Instance type: A (sub)task can be executed on different instance types which may be more or less suit able for the considered program. Therefore we have developed special annotations task developers can use to characterize the hardware requirements of their code. However, a user who simply utilizes these annotated tasks can also overwrite the developer's suggestion and explicitly specify the instance type for a task in the Job Graph. If the user omits to augment the Job Graph with these specifications, Kneehole’s scheduler will apply default strategies which are discussed in the following subsection. Once the Job Graph is specified, the user can submit it to the Job Manager, together with the credentials he obtained from his cloud operator. The credentials are required since the Job Manager must allocate/deal locate instances from the cloud during the job execution on behalf of the user.

VI. CONCLUSION

In this paper, we have discussed the challenges and opportunities for efficient parallel data Processing in cloud environments and presented Nehalem, the first data processing framework to exploit the dynamic resource provisioning offered by today’s Iasi clouds. We have described Kneehole’s basic architecture and presented a performance comparison to the established data processing framework Hadoop. The performance evaluation gives a first impression on how the ability to assign specific virtual machine types to specific tasks of a processing job, as well as the possibility to automatically allocate/deal locate virtual machines in the course of a job execution, can help to improve the overall resource utilization and, consequently, reduce the processing cost. In Particular, we are interested in improving Nephele’s ability to adapt to resource overload or Underutilization during the job execution automatically. Our current profiling approach builds a valuable basis for this; however, at the moment the system still requires a reasonable amount of user annotations. In general, we think our work represents an important contribution to the growing field of Cloud computing services and points out exciting new opportunities in the field of parallel data processing.

REFERENCES

1. D. Batter´, S. Ewan, F. Hueske, O. Kao, V. Markl, and D. Warneke “Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing,” Proc.
2. ACM Symp. Cloud Computing (SoCC ’10), pp. 119-130, 2010.
3. D. 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.
4. B. Claudel, G. Huard, and O. Richard. Taktuk, adaptive deployment of remote executions. In HPDC ’09: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing, pages 91–100, New York, 2009. ACM.
5. R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE: easy and e_cient parallel processing of massive data sets. Proc. VLDB Endow.,1(2):1265{1276, 2008.
6. M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, “Maximum Likelihood Network Topology Identification from Edge-Based Unicast Measurements,” SIGMETRICS Performance Evaluation Rev., vol. 30, no. 1, pp. 11-20, 2002.
7. R. Davoli, “VDE: Virtual Distributed Ethernet,” Proc. Testbeds and Research Infrastructures for the Development of Networksand Communities, Int’l Conf., pp. 213-220,2005.
8. T. Dornemann, E. Juhnke, and B. Freisleben,“On-Demand Resource Provisioning for BPEL
9. Workflows Using Amazon’s Elastic ComputeCloud,” Proc. Ninth IEEE/ACM Int’l Symp. Cluster Computing and the Grid (CCGRID ’09),pp. 140-147, 2009.
10. Foster and C. Kesselman, “Globus: AMetacomputing Infrastructure Toolkit,” Int’l J.Supercomputer Applications, vol. 11, no. 2, pp.115-128, 1997.