Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud

Dr. A. Kannagi[1] , Syed Rasheed Uddin[2], P. Amarnath[3], Chinnaiah[4]
[1]Professor, [2, 3, 4] Assistant Professor
Department of Computer Science and Engineering,
Malla Reddy College of Engineering, Hyderabad.

ABSTRACT

In recent years ad-hoc parallel data processing has emerged to be one of the killer applications for Infrastructure-as-a-Service (IaaS) clouds. 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. Consequently, the allocated compute resources may be inadequate for big parts of the submitted job and unnecessarily increase processing time and cost. In this paper we discuss the opportunities and challenges for efficient parallel data processing in clouds and present our research project. It 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.

Keywords: Task computing, query processing, dynamic resource allocation, Task Computing

I. INTRODUCTION

For organizations that exclusive need to process vast sums Today a developing number of organizations need to prepare gigantic measures of information in a cost-effective way. Great agents for these organizations are administrators of Internet web indexes, similar to Google, Yahoo, or Microsoft. The boundless measure of information they need to manage each day has made customary database arrangements restrictively costly [5]. Rather, these organizations have advanced a design worldview in light of countless servers. Issues like preparing slithered archives or recovering a web file are part into a few autonomous subtasks, appropriated among with lease a huge IT infrastructure on a fleeting pay- per-use premise. Administrators of alleged Infrastructure-as-a-Service (IaaS) mists, similar to Amazon EC2 [1], let their clients apportion, get to, and control an arrangement of Virtual Machines (VMs) which keep running inside their server farms and just charge them for the timeframe the machines are designated. The VMs are regularly offered in various sorts, every sort with its own qualities (number of CPU centers, measure of primary memory, and so forth.) and cost.

This paper is an extended It includes further details on scheduling strategies and extended experimental results. The paper is structured as follows: Section II, starts with analyzing the above mentioned opportunities and challenges and derives some important design principles for our new framework. In Section III, we present Nephele's basic architecture and outline how jobs can be described and executed in the cloud. Section IV, provides some first figures on Nephele's performance and the impact of the optimizations we propose. Finally, our work is concluded by related work (Section V) and ideas for future work

II. CHALLENGES AND OPPORTUNITIE

Current data processing frameworks like Google's Map Reduce or Microsoft's Dryad engine have been designed for cluster environments. This is reflected in a number of assumptions they make which are not necessarily valid in cloud environments. In this section we discuss how abandoning these assumptions raises new opportunities but also challenges for efficient parallel data processing in clouds.

A. OPPORTUNITIES

Today's processing frameworks typically assume the re-sources they manage consist of a static set of homogeneous compute nodes. Although designed to deal with individual nodes failures, they consider the number of available machines to be constant, especially when scheduling the processing job's execution. While IaaS clouds can certainly be used to create such cluster-like setups, much of their flexibility remains unused. 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.

B. CHALLENGES

The cloud’s virtualized nature helps to enable promising new use cases for efficient parallel data processing. However, it also imposes new challenges compared to classic cluster setups. The major challenge we see is the cloud’s opaqueness with prospect to exploiting data locality: In a cluster the compute nodes are typically interconnected through a physical high performance network. The topology of the network, i.e. the way the compute nodes are physically wired to each other, is usually well known and, what is more important, does not change over time. Current data processing frameworks offer to leverage this knowledge about the network hierarchy and attempt to schedule tasks on compute nodes so that data sent from one node to the other has to traverse as few network switches as possible [9]. That way network bottlenecks can be avoided and the overall throughput of the cluster can be improved. In a cloud this topology information is typically not exposed to the customer [29]. Since the nodes involved in processing a data intensive job often have to transfer tremendous amounts of data through the network, this drawback is particularly severe; parts of the network may become congested while others are essentially unutilized. Although there has been research on inferring likely network topologies solely from end- to-end measurements (e.g. [7]), it is unclear if these techniques are applicable to IaaS clouds. For security reasons clouds often incorporate network virtualization techniques (e.g. [8]) which can hamper the inference process, in particular when based on latency measurements.

III. DESIGN

Based on the challenges and opportunities outlined in the previous section we have designed Nephele, a new data processing framework for cloud environments. Nephele takes up many ideas of previous processing frameworks but refines them to better match the dynamic and opaque nature of a cloud.

A. ARCHITECTURE

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

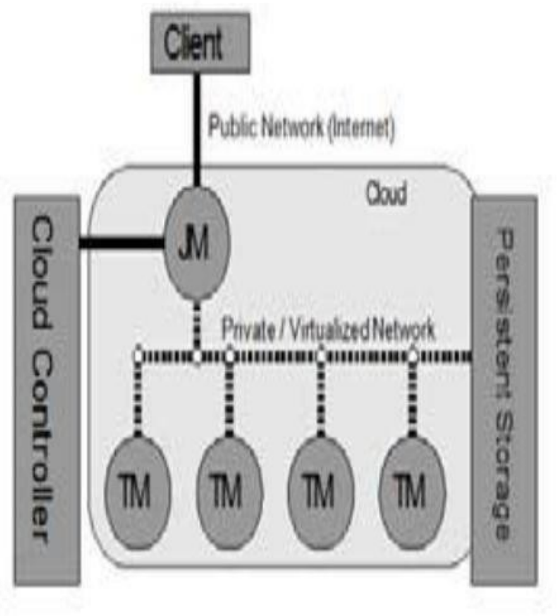


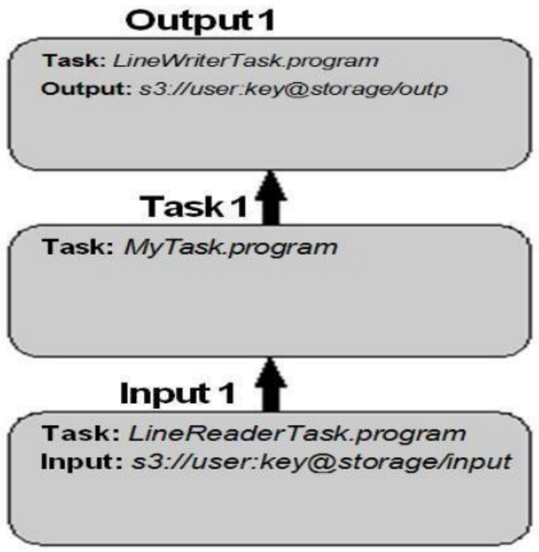
Fig. 1: Structural Overview of Nephele Running in an Infrastructure-as-a-Service (IaaS) Cloud

Before submitting a Nephele compute job, a user must start a VM in 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 interface the cloud operator provides to control the instantiation of VMs. We call this interface the Cloud Controller. By means of the Cloud Controller the Job Manager can allocate or de allocate VMs according to the current job execution phase. We will comply with common Cloud computing terminology and refer to these VMs as instances for the remainder of this paper. The term instance type will be used to differentiate between VMs with different hardware characteristics. E.g., the instance type “m1.small” could denote VMs with one CPU core, one GB of RAM, and a 128 GB disk while the instance type “c1. X large” could refer to machines with 8 CPU cores, 18 GB RAM ,and a 512 GB disk.

The actual execution of tasks which a Nephele job consists of is carried out by a set of instances. Each instance runs a so-called Task Manager (TM). A Task Manager receives one or more tasks from the JobMan- ager 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 job's particular tasks, how many and what type of instances the job should be executed on, and when the respective instances must be allocated/de allocated to ensure a continuous but cost efficient processing. Our current strategies for these decisions are highlighted at the end of this section. 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.

After having specified the code for the particular tasks of the job, the user must define the DAG to connect these tasks. We call this DAG the Job Graph. The Job Graph maps each task to a vertex and determines the communication paths between them. The number of a vertex's incoming and outgoing edges must thereby comply with the number of input and output gates defined inside the tasks. In addition to the task to execute, input and output vertices (i.e. vertices with either no incoming or outgoing edge) can be associated with a URL pointing to external storage facilities to read or write input or output data, respectively. Fig. 2, illustrates the simplest possible Job Graph. It only consists of one input, one task, and one output vertex. Fig


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

1. 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.

2. Number of Subtasks Per Instance

By default each subtask is assigned to a separate instance. In case several subtasks are supposed to share the same instance, the user can provide a corresponding an- notation with the respective task.

3. 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.

4. Channel Types

For each edge connecting two vertices the user can determine a channel type. Before executing a job, Nephele requires all edges of the original Job Graph to be replaced by at least one channel of a specific type. The channel type dictates how records are transported from one subtask to another at runtime. Currently, Nephele supports network, file, and in- memory channels. The choice of the channel type can have several implications on the entire job schedule.

A more detailed discussion on this is provided in the next subsection.

IV. EVALUATION

In this section we want to present first performance results of Nephele and compare them to the data processing framework Hadoop. We have chosen Hadoop as our competitor, because it is an open source software and currently enjoys high popularity in the data processing community. We are aware that Hadoop has been designed to run on a very large number of nodes (i.e. several thousand nodes). However, according to our observations, the software is typically used with significantly fewer instances in current IaaS clouds. In fact, Amazon itself limits the number of available instances for their MapReduce service to 20 unless the respective customer passes an extended registration process [2]. The challenge for both frameworks consists of two abstract tasks: Given a set of random integer numbers, the first task is to determine the k smallest of those numbers. The second task subsequently is to calculate the average of these k smallest numbers. The job is a classic representative for a variety of data.

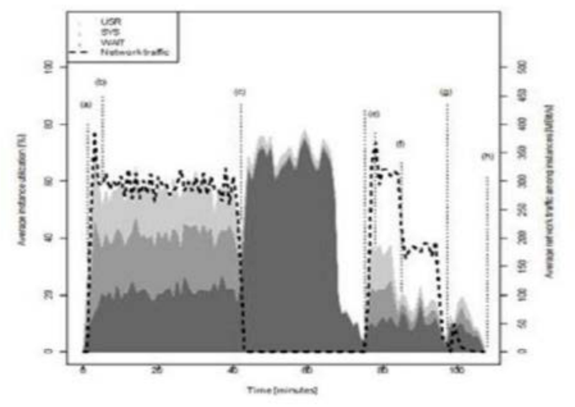
V. RELATED WORK

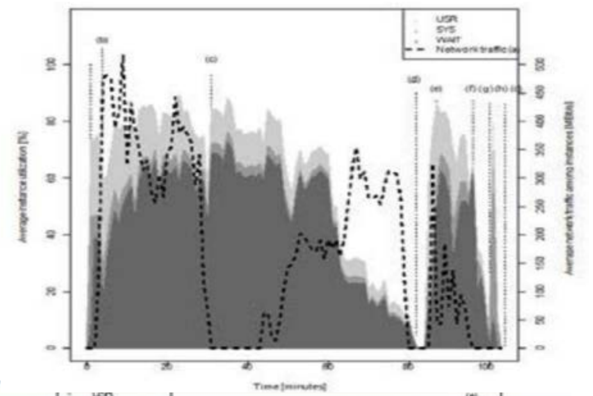
In recent years a variety of systems to facilitate MTC has been developed. Although these systems typically share common goals (e.g. to hide issues of parallelism or fault tolerance), they aim at different fields of application. Map Reduce [9] (or the open source version Hadoop [25]) is designed to run data analysis jobs on a large amount of data, which is expected to be stored across a large set of share- nothing commodity servers. Map Reduce is highlighted by its simplicity: Once a user has fit his program into the required map and reduce pattern, the execution framework takes care of splitting the job into subtasks, distributing and executing them. A single Map Reduce job always consists of a distinct map and reduce program. However, several systems have been introduced to coordinate the execution of a sequence of Map Reduce jobs [17,19]. Map Reduce has been clearly designed for large static clusters. Although it can deal with sporadic node failures, the available compute resources are essentially considered to be a fixed set of homogeneous machines. The Pegasus framework by Deelmanetal.

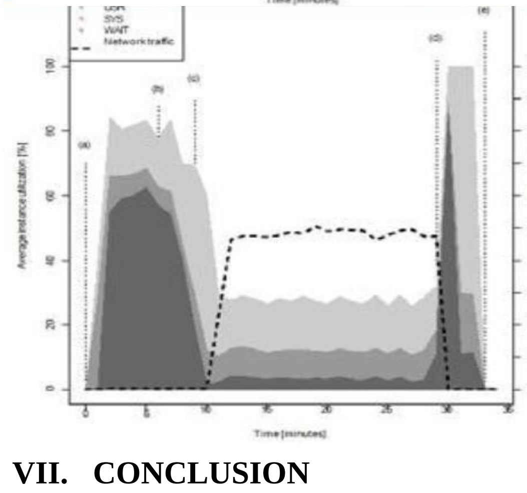
VI. RESULTS

show the performance results of our three experiment, respectively. All three plots illustrate the average instance utilization over time, i.e. the average utilization of all CPU cores in all instances allocated for the job at the given point in time. The utilization of each instance has been monitored with the Unix command "top" and is broken down into the amount of time the CPU cores spent running the respective data processing framework (USR), the kernel and its processes (SYS), and the time waiting for I/O to complete (WAIT). In order to illustrate the impact of network communication, the plots additionally show the average amount of IP traffic flowing between the instances over time. We begin with discussing Experiment 1 (Map

Reduce and Hadoop): For the first Map Reduce job, Tera Sort, fig. 7, shows a fair resource utilization. During the map (point (a) to (c)) and reduce phase (point (b) to (d)) the overall system utilization ranges from 60 to 80%. This is reasonable since we configured Hadoop's Map Reduce engine to perform best for this kind of task. For the following two Map Reduce jobs, however, the allocated instances are oversized: The second job, whose map and reduce phases range from point (d) to (f) and point (e) to (g), respectively, can only utilize about one third of the available CPU capacity. The third job (running between point (g) and (h)) can only consume about 10 % of the overall resources







VII. CONCLUSION

In this paper we have discussed the challenges and opportunities for efficient parallel data processing incloud environments and presented Nephele, the first data processing framework to exploit the dynamic resource provisioning offered by today's IaaS clouds. We have described Nephele's basic architecture and presented a performance comparison to th well- 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/de allocate virtual machines in the course.

VIII. REFERENCES:

[1] Amazon Web Services LLC,"Amazon Elastic Compute Cloud", (Amazon EC2). [Online] Available: http://www. aws.amazon.com/ec2/,2009.

[2] Amazon Web Services LLC,"Amazon Elastic Map Reduce", [Online] Available: http://www.aws.amazon.com/ elastic mapreduce/,2009.

[3] Amazon Web Services LLC,"Amazon Simple Storage Service", [Online]Available: http: //www.aws.amazon.com/ s3/, 2009.

[4] D. Battr´ S. Ewen, F. Hueske, O. Kao, V.Markl, D. Warneke. e,"Nephele/PACTs: A Programming Model and Execution Frame- work for Web-Scale Analytical Processing", In SoCC '10: Proceed- ings of the ACM Efficient Parallel Processing of Massive Data Sets", Proc. VLDB Endow., 1(2):1265- 1276,2008.

[5]H.chihYang,A.Dasdan,R.-L.Hsiao, D. S. Parker. "Map Reduce-Merge: Simplified Relational Data Processing on Large Clusters", In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 1029-1040, New York, NY, USA, 2007. ACM.