Volume 2, Issue 10, October 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Dynamic Resource Allocation and Data Processing Framework for Cloud Architecture

Anusha Reddy *
PG Student, CSE Department, CMRIT, Hyderabad, India

Dr.M.Janga Reddy
Professor, Department, CMRIT, Hyderabad, India

Abstract—Cloud Computing has became an emerging area in the recent areas. It extends services like SaaS, PaaS and IaaS. One of the complicated applications in Iaas is parallel data processing. Parallel data processing is needed to integrate and deploy their programs in customer premises. However, the processing frameworks which are currently used have been designed for static, homogeneous cluster setups and disregard the particular nature of a cloud. Results into the allocated resources may not be sufficient for the large scale jobs and unnecessarily increase processing time and cost. In this paper we discuss the opportunities and challenges for efficient parallel data processing in clouds and a novel framework is designed for the parallel data processing.

Keywords— Cloud Computing, Data Processing, Iaas, Parallel Processing

I. INTRODUCTION

Emerging techniques makes processing of huge amount of data so simple with the help of cloud computing. Cloud computing ensures the availability of the data to its customers on all time with less infrastructure itself. This point makes very interesting for many companies like Google, Yahoo, or Microsoft to conduct research in cloud. The vast amount of data they have to deal with every day has made traditional database solutions prohibitively expensive. 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 Map Reduce. 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.

IaaS Infrastructure as a Service enables the companies to run with a minimal resources and hire the resources needed on demand and pay only when there is an use. This is becomes feasible by the Virtual Machines(VM) which runs at the data center to calculate the usage and do the necessary billing[1]. Since the VM abstraction of IaaS clouds fits the architectural paradigm assumed by the data processing frameworks described above, projects like Hadoop. This paper talks about the new framework for data processing called as "Nephele". This framework enables the possibility of allocating and reallocating resources dynamically in a cloud during the execution of a job.

II. LITERATURE SURVEY

Recent 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[1][2]. This section describes how these assumptions gives path for new framework. Today's processing frameworks typically assume the resources 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 computer resources available in a cloud are highly dynamic and possibly heterogeneous. With respect to parallel data processing, this flexibility leads to a variety of new possibilities, particularly for scheduling data processing jobs. The question a scheduler

has to answer is no longer "Given a set of compute resources, how to distribute the particular tasks of a job among them?", but rather "Given a job, what compute resources match the tasks the job consists of best?". This new paradigm allows allocating compute resources dynamically and just for the time they are required in the processing workflow. 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.

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.

III. SYSTEM DESIGN

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 simple master Slave Configuration.

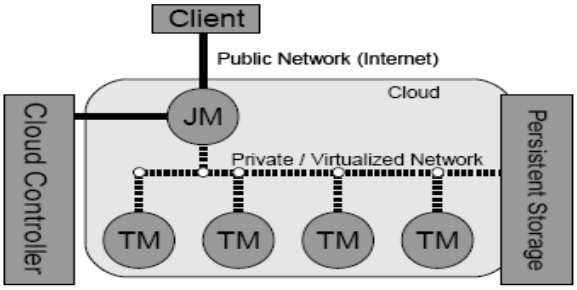

Fig 1. Structural Overview of Nephele Running in an IAAS

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 deallocate 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.xlarge" could refer to machines with 8 CPU cores, 18 GB RAM, and a 512 GB disk.

B. Job Description

No jobs in Nephele are expressed as a directed acyclic graph (DAG). 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 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. Programmers can define arbitrary types of records. From a programmer's perspective records enter and leave the task program through input or output gates. Those input and output gates can be considered endpoints of the DAG's edges which are defined in the following step. Regular tasks (i.e. tasks which are later assigned to inner vertices of the DAG) must have at least one or more input and output gates. Contrary to that, tasks which either represent the source or the sink of the data flow must not have input or output gates, respectively

C. Job Scheduling and Execution

After having received a valid Job Graph from the user, Nephele's[3] Job Manager transforms it into a so-called Execution Graph. An Execution Graph is Nephele's primary data structure for scheduling and monitoring the execution of a Nephele job. Unlike the abstract Job Graph, the Execution Graph contains all the concrete information required to schedule and execute the received job on the cloud. It explicitly models task parallelization and the mapping of tasks to instances. Depending on the level of annotations the user has provided with his Job Graph, Nephele may have different degrees of freedom in constructing the Execution Graph. Figure 3 shows one possible Execution Graph constructed from the previously depicted Job Graph (Figure 2). Task 1 is e.g. split into two parallel subtasks which are both connected to the task Output 1 via file channels and are all scheduled to run on the same instance. The exact structure of the execution graph is shown in the fig 2

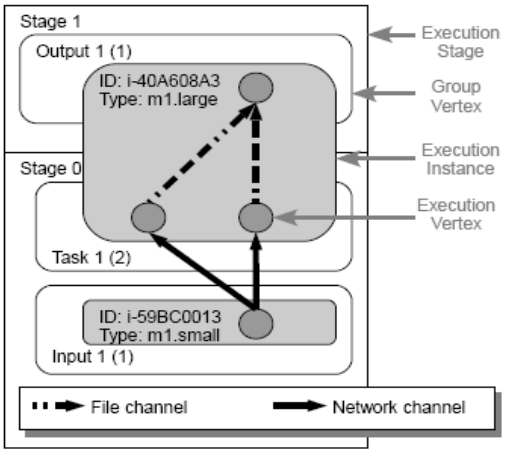

Fig 2. An Execution Graph created from the Job Graph.

Nephele requires all edges of an Execution Graph to be replaced by a channel before processing can begin. The type of the channel determines how records are transported from one subtask to the other. Currently, Nephele features three different types of channels, which all put different constrains on the Execution Graph[4][5].

1. **Network channels:** A network channel lets two subtasks exchange data via a TCP connection. Network channels allow pipelined processing, so the records emitted by the producing subtask are immediately transported to the consuming subtask[8]. As a result, two subtasks connected via a network channel may be executed on different instances. However, since they must be executed at the same time, they are required to run in the same Execution Stage.

2. In-Memory channels: Similar to a network channel, an in-memory channel also enables pipelined processing. However, instead of using a TCP connection[6], the respective subtasks exchange data using the instance's main memory. An in-memory channel typically represents the fastest way to transport records in Nephele, however, it also implies most scheduling restrictions: The two connected subtasks must be scheduled to run on the same instance and run in the same Execution Stage.

3. File channels: A file channel allows two subtasks to exchange records via the local file system. The records of the producing task are first entirely written to an intermediate file and afterwards read into the consuming subtask. Nephele requires two such subtasks to be assigned to the same instance. Moreover, the consuming Group Vertex must be scheduled to run in a higher Execution Stage than the producing Group Vertex[10]. In general, Nephele only allows subtasks to exchange records across different stages via file channels because they are the only channel types which store the intermediate records in a persistent manner.

IV. CONCLUSIONS

The confronts and opportunities for efficient parallel data processing in cloud environments is discussed in detail. A novel framework called "Nephele: , the first data processing framework to exploit the dynamic resource provisioning offered by today's IaaS clouds is analyzed in depth. The performance evaluation talks about 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 of a job execution, can help to improve the overall resource utilization and, consequently, reduce the processing cost.

REFERENCES

[1] D. Battr´e, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing. In SoCC '10: Proceedings of the ACM Symposium on Cloud Computing 2010, pages 119– 130, New York, NY, USA, 2010. ACM.

[2] 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. VLDB Endow., 1(2):1265– 1276, 2008.

[3] H. chih Yang, A. Dasdan, R.-L. Hsiao, and 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, pages 1029–1040, New York, NY, USA, 2007. ACM.

[4] Amazon Web Services LLC. Amazon elastic Compute Cloud Amazon EC2) http://aws.amazon.com/ec2/,2009

[5] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang. Maximum Likelihood Network Topology Identification from Edge-Based Unicast Measurements. SIGMETRICS Perform.

[6] I. Raicu, I. Foster, and Y. Zhao. Many-Task Computing for Grids and Supercomputers. In Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008. Workshop on, pages 1–11, Nov. 2008.

[7] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde. Falkon: a Fast and Light-weight tasK executiON framework. In SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing, pages 1–12, New York, NY, USA, 2007. ACM.

[8] L. Ramakrishnan, C. Koelbel, Y.-S. Kee, 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. In SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pages 1–12, New York, NY, USA, 2009. ACM.

[9] R. Russell. virtio: Towards a De-Facto Standard for Virtual I/O Devices. SIGOPS Oper. Syst. Rev., 42(5):95–103, 2008.

[10] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO - DB2's LEarning Optimizer. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pages 19–28, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[11] The Apache Software Foundation. Welcome to Hadoop! http: //hadoop.apache.org/, 2009.

[12] G. von Laszewski, M. Hategan, and D. Kodeboyina. Workflows for e-Science Scientific Workflows for Grids. Springer, 2007.

