International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 6, August - 2012

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

G.SIREESHA*, L.BHARATHI**

*(Department of Computer Science, AME,JNTU, PALWANCHA,INDIA

** (Asst prof in Department of Computer Science, AME,JNTU,PALWANCHA,INDIA

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. 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, we perform extended evaluations of MapReduce-inspired processing jobs on an IaaS cloud system and compare the results to the popular data processing framework Hadoop.

Keywords – **Many-Task Computing, High-Throughput Computing, Loosely Coupled Applications, Cloud Computing**

I. INTRODUCTION

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. Instead, these companies have popularized an architectural paradigm based on a large number of commodity servers. Problems like processing rawled 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, Microsoft's Dryad , or Yahoo!'s Map-Reduce-Merge .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 [20]. Although these systems differ in design, their programming models share similar objectives, namely hiding the hassle of parallel programming, fault tolerance, and execution optimizations from the developer. Developers 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.

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 It includes further details on scheduling strategies and extended experimental results.

DESIGNING & IMPLEMENTATION

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.

Architecture

Nephele's architecture follows a classic master-worker pattern as illustrated in

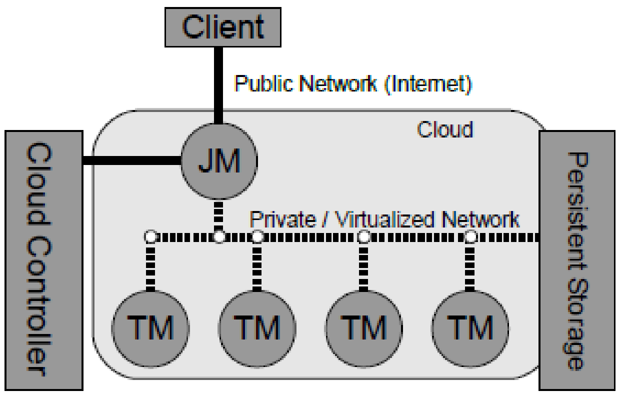


Fig. Structural overview of Nephele running in an Infrastructure-as-a-Service (IaaS) cloud

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 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. 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 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 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/deallocated 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.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 . 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.

.

III **Problem Definition**

Similar to Microsoft's Dryad , 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. 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 simplifies the implementation of classic data combining functions like, e.g., join operations .
Second and more importantly, though, the DAG's edges explicitly model the communication paths of the processing job. As long as the particular 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. asks 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.

IV INDENTATIONS

<u>Modules:</u>

<u>NETWORK MODULE</u>

Server - Client computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients. Often clients and servers operate over a computer network on separate hardware. A server machine is a high-performance host that is running one or more server programs which share its resources with clients. A client also shares any of its resources; Clients therefore initiate communication sessions with servers which await <u>(listen to) incoming requests.</u>

<u>LBS SERVICES</u>

In particular, users are reluctant to use LBSs, since revealing their position may link to their identity. Even though a user may create a fake ID to access the service, her location alone may disclose her actual identity. Linking a position to an individual is possible by various means, such as publicly available information city maps. When a user u wishes to pose a query, she sends her location to a trusted server, the anonymizer through a secure connection (SSL). The latter obfuscates her location, replacing it with an anonymizing spatial region (ASR) that encloses u. The ASR is then forwarded to the LS. Ignoring where exactly u is, the LS retrieves (and reports to the AZ) a candidate set (CS) that is guaranteed to contain the query results for any possible user location inside the ASR. The AZ receives the CS and reports to u the subset of candidates that corresponds to her original query.

SYSTEM MODEL:

The ASR construction at the anonymization process abides by the user's privacy requirements. Particularly, specified an anonymity degree K by u, the ASR satisfies two properties: (i) it contains u and at least another K * 1 users, and (ii) even if the LS knew the exact locations of all users in the system.

- We propose an edge ordering anonymization approach for users in road networks, which guarantees K-anonymity under the strict reciprocity requirement (described later).
- We identify the crucial concept of border nodes, an important indicator of the CS size and of the query processing cost at the LS.
- We consider various edge orderings, and qualitatively assess their query performance based on border nodes.
- We design efficient query processing mechanisms that exploit existing network database infrastructure, and guarantee CS inclusiveness and minimality. Furthermore, they apply to various network storage schemes.
- We devise batch execution techniques for anonymous queries that significantly reduce the overhead of the LS by computation sharing.

SCHEDULED TASK:

Recently, considerable research interest has focused on preventing identity inference in location-based services. Proposing spatial cloaking techniques. In the following, we describe existing techniques for ASR computation (at the AZ) and query processing (at the LS). At the end, we cover alternative location privacy approaches and discuss why they are inappropriate to our problem setting. This offers privacy protection in the sense that the actual user position u cannot be distinguished from others in the ASR, even when malicious LS is equipped/advanced enough to possess all user locations. This spatial K-anonymity model is most widely used in location privacy research/applications, even though alternative models are emerging.

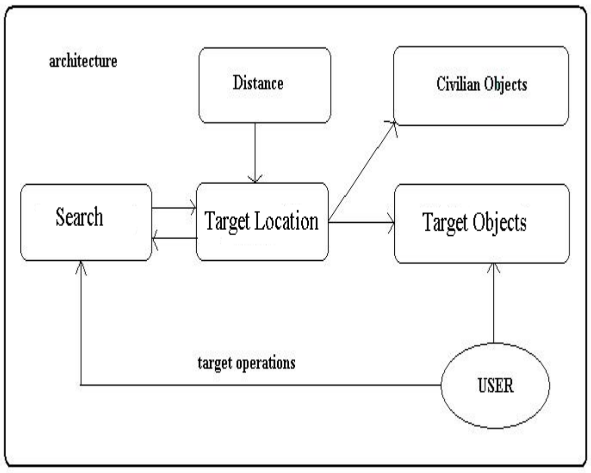
QUERY PROCESSING:

Processing is based on implementation of the theorem uses (network-based) search operations as off the shelf building blocks. Thus, the NAP query evaluation methodology is readily deployable on existing systems, and can be easily adapted to different network storage schemes. In this case, the queries are evaluated in a batch. we propose the network-based anonymization and processing (NAP) framework, the first system for K- anonymous query processing in road networks. NAP relies on a global user ordering and bucketization that satisfies reciprocity and guarantees K-anonymity. We identify the ordering characteristics that affect subsequent processing, and qualitatively compare alternatives. Then, we propose query evaluation techniques that exploit these characteristics. In addition to user privacy, NAP achieves low computational and communication costs, and quick responses overall. It is readily deployable, requiring only basic network operations.

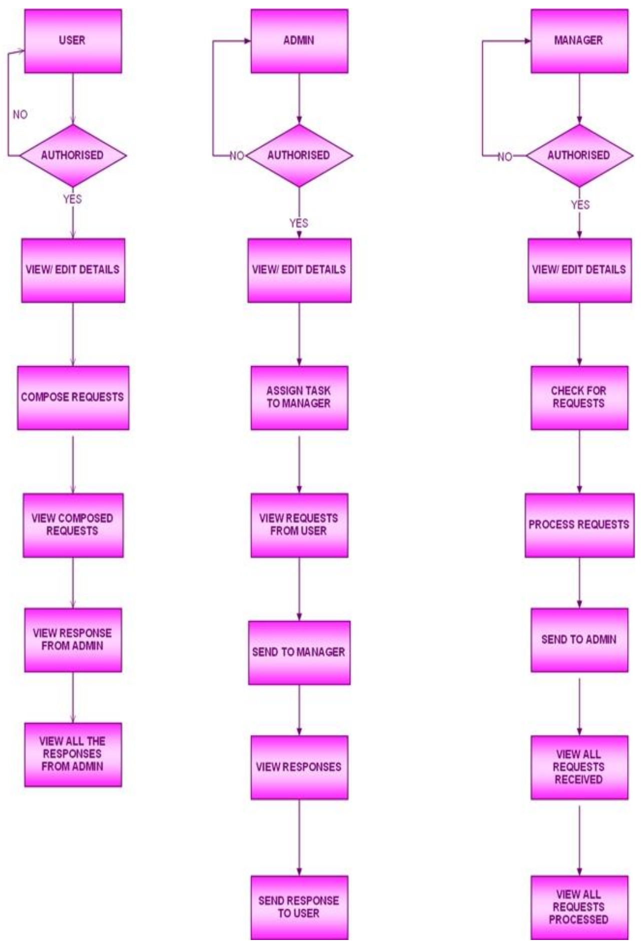
.

V DIAGRAMS

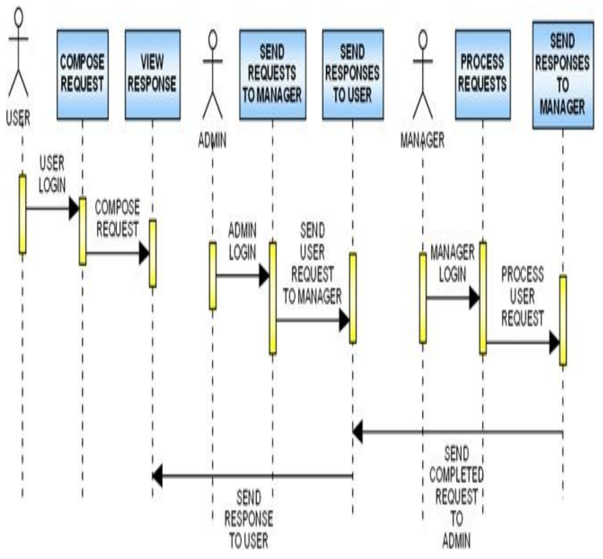
ARCHITECTURE



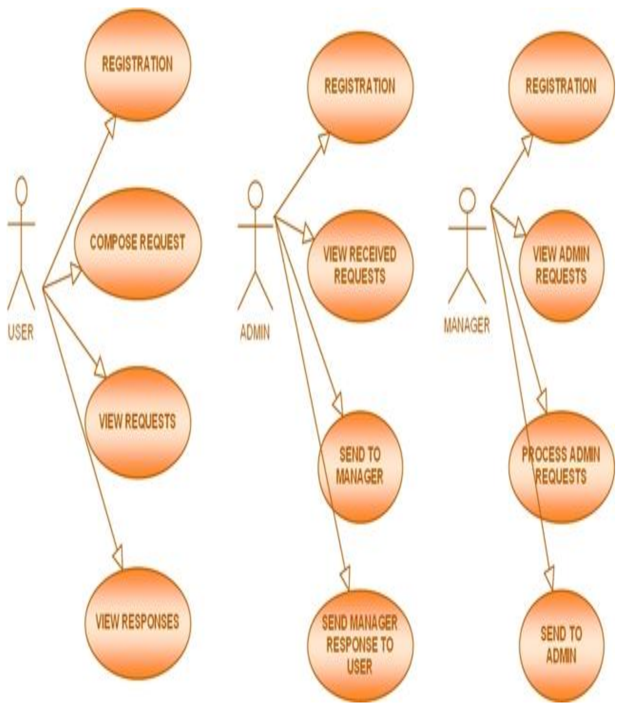
DATAFLOW DIAGRAM



SEQUENCE DIAGRAM



USE CASE DIAGRAM



CONCLUSION

In this paper we have discussed the challenges and opportunities for efficient parallel data processing in cloud 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 the 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/deallocate virtual machines in the course of a job execution, can help to improve the overall resource utilization and, consequently, reduce the processing cost. With a framework like Nephele at hand, there are a variety of open research issues, which we plan to address for future work. 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] Amazon Web Services LLC. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2009.
[2]Amazon Web Services LLC. Amazon Elastic MapReduce. http: //aws.amazon.com/elasticmapreduce/, 2009.
[3] AmazonWeb Services LLC. Amazon Simple Storage Service. http: //aws.amazon.com/s3/, 2009.

[4] 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, New York, NY, USA, 2010. AC.
[5] 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.
[6] 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.
[7] 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. Eval. Rev., 30(1):11–20, 2002.
[8] R. Davoli. VDE: Virtual Distributed Ethernet. Testbeds and Research Infrastructures for the Development of Networks & Communities, International Conference on, 0:213–220, 2005.
[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation,Berkeley, CA, USA, 2004. USENIX Association.
[10] 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. Sci. Program., 13(3):219–237, 2005.
[11] T. Dornemann, E. Juhnke, and B. Freisleben. On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. In CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pages 140–147, Washington, DC, USA, 2009. IEEE Computer Society.
[12] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. Intl. Journal of Supercomputer Applications,, 1997.
[13] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor- G: A Computation Management Agent for Multi-Institutional Grids. Cluster Computing, 5(3):237–246, 2002.
[14] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pages 59–72, New York, NY, USA, 2007. ACM.
[15] A. Kivity. kvm: the Linux Virtual Machine Monitor. In OLS '07: The 2007 Ottawa Linux Symposium, pages 225–230, July 2007.
[16] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. Eucalyptus: A Technical Report on an Elastic Utility Computing Architecture Linking Your Programs to Useful Systems. Technical report, University of California, Santa Barbara, 2008.
[17] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1099–1110, New York, NY, USA, 2008. ACM.
[18] O. O'Malley and A. C. Murthy. Winning a 60 Second Dash with a Yellow Elephant. Technical report, Yahoo!, 2009. [19] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the Data: Parallel Analysis with Sawzall. Sci. Program., 2005.
[20] 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.
[21] 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,, New York, NY, USA, 2007. ACM.
[22] 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.
[23] R. Russell. virtio: Towards a De-Facto Standard for Virtual I/O Devices. SIGOPS Oper. Syst. Rev., 42(5):95–103, 2008.
[24] 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,San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.