Advanced Framework for Dynamic Allocation of Resources for Optimal Data Processing in the Cloud

M. Sudhakar Reddy, B. R. Sarath Kumar
Dept. of CSE, Lenora College of Engineering, India

Abstract
Infrastructure-as-a-Service (IaaS) is one of the emerging services provided by the Cloud Companies. Today there are so many frameworks for parallel data processing to provide Infrastructure-as-a-Service. Some of these frameworks use static, homogeneous resource allocation technique, which can’t satisfy dynamic nature of cloud computing. Some of the new frameworks like Nephele overcome these drawbacks by using Dynamic resource allocation but still contain some setbacks like Resource under utilization. Nephele doesn’t de allocate the Virtual Machine-(VM) which contains some intermediate results even it completed its execution to provide those results to other Virtual machines. Our new framework introduces Alternative Virtual Machine (AVM) to store intermediate results so wastage of resources can be reduced.

Keywords

I. Introduction
“Cloud Computing” is a computing model, not a technology. In this model "Customers" plug into the "cloud" to access IT resources which are priced and provided “on-demand”. Essentially, IT resources are rented and shared among multiple clients such as office space, apartments, or storage spaces are used by clients. Delivered over an Internet connection, the “cloud” replaces the company data center or server providing the same service. Thus, Cloud Computing is simply IT services sold and delivered over the Internet. Infrastructure as a Service (IaaS) is a service model that provides computer infrastructure on pay per usage basis to support enterprise operations. Typically, IaaS provides hardware, storage, processor time, servers, and Data center space or network components.

Today so many companies have to process the huge amounts 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 Infrastructure-as-a-Service (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.

Current data processing frameworks like Hadoop rather expect the cloud to imitate the static nature of the cluster environments they were originally designed for. E.g., at the moment the types and number of VMs allocated at the beginning of a compute job cannot be changed in the course of processing, although the tasks the job consists of might have completely different demands on the environment. As a result, rented resources may be inadequate for big parts of the processing job, which may lower the overall processing performance and increase the cost. Dynamic resource allocation technique is used in the new framework Nephele [8] to overcome the drawbacks of static data processing frameworks. But Nephele also contains some drawbacks like resource underutilization. Nephele keeps the Virtual Machine which contains the intermediate results to provide to other Virtual Machines as an input in the part of data processing. So this leads to wastage of high potential resources. And Nephele doesn’t use the cache concept. Caching is used to cache the frequently used data which is queried from datacenters to reduce the burden on data centers and save the computational resources.

In this paper, presented a new processing framework explicitly designed for cloud environments. This data processing framework includes the possibility of dynamically allocating/deallocating different compute resources from a cloud in its scheduling and during job execution as well as it uses the Alternative Virtual Machine (AVM) to store the intermediate results of main Virtual Machine (VM). Here AVM is the low configured Virtual Machine which contains fewer amounts of main memory and processing speed because these are not intended to process the data just for storing intermediate results and providing those results to other Virtual Machines. And also this new framework uses the caching of frequently accessing data to save the computational power.

II. Requirements
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. Facilitating such use cases imposes some requirements on the design of a processing framework and the way its jobs are described.
First, the scheduler of such a framework must become aware of the cloud environment a job should be executed in. It must know about the different types of available VMs as well as their cost and be able to allocate or destroy them on behalf of the cloud customer. Second, the paradigm used to describe jobs must be powerful enough to express dependencies between the different tasks the job consists of. The system must be aware of which task’s output is required as another task’s input. Otherwise the scheduler of the processing framework cannot decide at what point in time a particular VM is no longer needed and deallocate it. The MapReduce pattern is a good example of an unsuitable paradigm here: Although at the end of a job only few reducer tasks may still be running, it is not possible to shut down the idle VMs, since it is unclear if they contain intermediate results which are still required. Finally, the scheduler of such a processing framework must be able to determine which task of a job should be executed on which type of VM and, possibly, how many of those. This information could be either provided externally, e.g. as an annotation to the job description, or deduced internally, e.g. from collected statistics, similarly to the way database systems try to optimize their execution schedule over time [2].

Virtual Machine has to transfer the intermediate results to Alternative Virtual Machine after completing its execution. The other Virtual Machines can identify the particular Alternative Virtual Machine (A VM) and give notice to the JM. Then JM deallocates completed VM’s. Here AVM has fewer configuration compared to the VM’s this because to save the resource cost. 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 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.

### III. Design

Based on the requirements outlined in the previous section a new data processing framework is designed for cloud environments. This framework takes up many ideas of previous processing frameworks but refines them to better match the dynamic nature of a cloud.

#### A. Architecture

Framework architecture is as follows in fig. 1

![Architecture of New Framework for Dynamic Resource Allocation](image)

Before submitting a 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 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. After completing execution Virtual Machine transfers its results to respective Alternative Virtual Machine (AVM) and gives notice to the JM. Then JM deallocates completed VM’s. Here AVM has fewer configuration compared to the VM’s this because to save the resource cost.

#### B. Job Structure

Similar to Microsoft’s Dryad [4], jobs in this framework 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 [5]. 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 which instance might still require data from what other instances and which instance can potentially be shut down and deallocated. Defining a 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. 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. Figure 2 illustrates the simplest possible Job Graph. It only consists of one input, one task, and one output vertex total three vertices in the Job Graph.

C. Job Scheduling and Execution

Once we submit the valid job graph Job Manager allocates the instances for executing the each sub task. Based on the job graph and user directions Job manager decides number of instances and type of the instances to start the execution. Once any instance completed its execution, intermediate results will be transferred to alternative Virtual Machine (AVM) and that instance will be deallocated immediately. If any instance requires input from other instances which was already completed and deallocated then the Job Manager gets those results from the particular AVM’s. This framework also uses the caching concept. The Cached data is stored in the persistence storage of the cloud. Caching is useful when user is requesting the same data frequently, instead getting that data from the data centers every time that data is stored in the persistence storage. Then that cached copy is given to the user for sub sequent request. This caching reduces the cost of requesting the data center every time. And it also reduces the burden on the data centers. Whenever the data is modified new copy of data is cached to avoid the unreliability in the data being used by the user.

IV. Conclusion

In this paper we have discussed the technique of parallel data processing in the cloud environments and present the new advanced framework for dynamic resource allocation for parallel data processing. This framework avoids some draw backs of already existed dynamic resource allocation frameworks. This framework leads to some new research ideas in the cloud technology. Here we propose some future advancement for this paper: this framework avoids several drawbacks but it still requires some user annotation or user direction for dividing the job into sub task and identifying the dependencies between the sub tasks. So we are interested in developing this framework without the user annotations in the future enhancements.

References


M. Sudhakarreddy is an M.Tech (CSE) Student in Lenora College of Engineering, Ramchodavaram. He received his B.Tech (IT) from JNTUK. His areas of interest are Cloud Computing, Web Technologies and Web Services.

Prof. B.R. Sarath Kumar is a Ph.D holder and Principal of Lenora College Engineering. He has authored several technical papers in National and International conferences and journals. He is a member of several professional societies like IEEE, ACM, Indian Science Congress, and Computer Society of India. His areas of interest include Artificial Intelligent, Web Technologies, Nanotechnologies and Network Security.