MultiStack: Multi-Cloud Big Data Research Framework/Platform

by

Vishrut Mehta, kumar rishabh, Reddy Raja, Vasudeva Varma

in

*The Fifth IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*

Bangalore, India

Report No: IIIT/TR/2016/-1

Centre for Search and Information Extraction Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
October 2016
Abstract—With improving efficiency and cost effectiveness of public cloud systems, there has been a growing trend to complement in-house cloud environment with them. We propose an open solution to the problem of distributing jobs in a hybrid environment, as a substitute to hadoop systems which are cloud specific.

The proposed system, MultiStack, is a big data orchestration platform for deploying big data jobs across multiple cloud providers. The specific architecture elaborated in this paper uses Amazon Web Services as the public cloud provider and Openstack as the private cloud framework. Our solution supports complete Hadoop ecosystem tools - hive, pig, Hbase, oozie etc. and on-demand scaling of Hadoop clusters. The proposed framework aims at reducing the Job completion time on workloads along with decrease in cost using Spot Instance provisioning compared to on-demand provisioning. This is achieved by providing two modes of operation: Proactive scheduling and Reactive scheduling, which takes into account user providing job characteristics(eg memory,cpu,etc), quota limitation and business objectives.

From our experiments, we conclude that Multistack is able to reduce average job completion time by 30-37% with minimal increase in cost.

I. INTRODUCTION

Virtualization has lead to higher consolidation and cost savings and has been widely adopted by enterprise and scientific community. Although, the adoption among data processing frameworks on virtualized infrastructure has been traditionally limited, there are clear advantages. Also, there is an increasing shift towards cloud marketplace due to instant availability of cheap resources through public cloud providers. Organizations want to leverage private and public cloud resources seamlessly and this hybrid scenario is becoming very common. Big data processing was one of the popular use case for adopting cloud computing and we believe big data clusters are good candidates to demonstrate the benefits of cloud bursting scenarios.

What is hybrid cloud?

A hybrid cloud is the combination of a public cloud provider with a private cloud platform which is designed for use by a single organization. The public and private cloud infrastructures, which operate independently of each other, communicate over an encrypted connection, using technology that allows for the portability of data and applications. The popular public cloud providers are Amazon Web Services[1], Google Compute Engine[2], Microsoft Azure[3], etc and private clouds such as Openstack[4], CloudStack[5], etc.

Why hybrid cloud?

- Architectural Flexibility and Technical control
- Better Security
- Utility Billing

While focusing on hybrid scenarios, the scheduler will be benefited greatly by:

- Deadline awareness: We need to know deadlines to make effective bidding and scaling for big data clusters.
- Cost awareness: Data processing is an ideal use case for hybrid clouds, as large scale data processing frameworks have built in fault tolerance capabilities which makes it very suitable.

II. MultiStack

MultiStack is a platform for multi-cloud research. It simplifies multi-cloud research by providing common functionalities, like multi cloud OS. Big data is the best use case for cloud computing.

A. MultiCloud

There are no accepted definition of multi-cloud in the current state of cloud computing. We here refer to the use of multiple cloud providers.

We also believe that MultiStack provides a good research platform for multi-cloud research beyond big data applications. The identified modules provide base for a multi-cloud platform.

III. SERVICES

In this section we identify and discuss services that a multi-cloud platform should provide.
A. Resource Management

Resource management is one of the core problems for computer science. It affects the three basic criteria for system evaluation: performance, functionality and cost. Inefficient resource management has a direct negative effect on performance and cost. In a multi-cloud environment, the platform manages not only for traditional resources like CPU, memory and storage, but also responsible for resources like cost, IP address allocation etc.

B. Monitoring

Having accurate information is essential for taking any decision in large scale distributed systems and thus a monitoring service becomes inevitable. In a multi-cloud environment the challenges of monitoring include:

- Overhead has to be very limited so as to scale to much larger number of systems.
- Dealing with much larger latency than faced by traditional monitoring systems.
- Complexity of handling variety of monitoring data formats.
- Application specific monitoring.

C. Identity and Authentication

This is one of the earliest identified roadblocks in the federated cloud environment and despite of significant efforts from both academia and industry, identity and authentication management across clouds has remained a difficult issue.

D. Data Management

To build consistent, available and scalable data management system across clouds has multiple challenges involved like data retention/redundancy/On-demand movement of data between clouds with proper access settings and many more.

E. Billing and Quota Enforcement

Quota refers to the resource limits enforced on processess and users. Unified billing over multiple cloud providers and spending available budget wisely on resources from multiple clouds. Also, the unit of resources across clouds is variable.

IV. BIG DATA FRAMEWORKS

There are many big data processing frameworks into existence with different goals in mind. We will give a brief introduction of few popular ones.

A. Hadoop

Apache Hadoop[6] is the first and most popular data processing framework based on MapReduce paradigm[8]. Hadoop ecosystem is maturing and contains tools for most aspect of data processing. Apache Hadoop contains two core components - Hadoop Distributed File System (HDFS) and Hadoop MapReduce framework.

B. Spark

Apache Spark [9] is part of Berkeley Data Analysis Stack - BDAS and aims to combine batch and iterative computing on a single platform. It is based on the Resilient Distributed Datasets [10] abstraction and makes aggressive use of memory. BDAS also has a diverse toolchain for data processing wide range of processing, comparable to Hadoop’s entire stack of tools.

V. COST-AWARE SCHEDULING

A. Spot Instances

Spot instances in AWS(Amazon Web Services) marketplace allow users to bid for spare Amazon EC2 compute capacity. Since Spot instances are often available at a discount compared to On-Demand pricing, you can significantly reduce the cost of running your applications, grow your application’s compute capacity and throughput for the same budget, and enable new types of cloud computing applications. The key differences between Spot instances and On-Demand instances are that Spot instances might not start immediately, the hourly price for Spot instances varies based on demand, and Amazon EC2 can terminate an individual Spot instance as the hourly price for or availability of Spot instances changes.

$$OptimalSpotinstanceCount = \arg \max ((d_m - d_i) - \beta Cost(i) \times d_i - \gamma i) \quad (1)$$

where, $d_m$ is estimated deadline with running current set of instances $d_i$ is estimated deadline after scaling the cluster with spot instances $Cost(i)$ is estimated cost of running i spot instances. This is simple regression over past pricing data. $\beta$ and $\gamma$ represent the weights that controls penalty.

In this equation we are trying to maximize the gain achieved by adding $i$ spot instances. The first term captures time saving by adding extra instances. Second term represents the cost of running $i$ spot instances. We penalize spawning many more instances by using third term.

Algorithm 1 Cost-aware autoscaling of clusters

1: Monitoring data about each Job.
2: Find optimal type of instance for scaling.
3: Predict the expected finish time with current set of instances.
4: if Current spot instance cost < Upper limit by user then
5: Find optimal number of spot instances to add using spot instance price history and deadline prediction using equation 1.
6: end if

In Algorithm 1 we describe scaling algorithm with spot instance. In [11], the author proposes the use of spot instances to accelerate map reduce jobs. Authors also note the challenges that arise due to transit nature of spot instances.
VI. DEADLINE-AWARE SCHEDULING

In many enterprises, data processing jobs are quite repetitive. James Horey et al. [12] presents challenges and approaches for providing big data as a service. A. Ganapathi et al. [14] presents statistical machine learning based methods for predicting job completion time as function of provided resources.

VII. POLICY BASED SCHEDULING

The large scale enterprises are guided by policy decision reflecting business needs. Traditionally the policy dictated a variety of decisions mainly controlling resource allocation to different divisions (allocating 70% resources to online services and 30% resources to the batch processing). We try to categorize these policies to incorporate the user and the admin needs. They are: 1) Pro-active approach 2) Reactive approach

A. Pro-active Approach

Pro-active is based on metrics defined by the respective users/admins.

The metrics defined by admin that are currently supported include:
- Storage/Bandwidth Quota per tenant
- Number of job instances per tenant

The metrics defined by users that are currently supported include:
- Job Characteristics (Memory/IO/Storage)
- Cost Limitations per Job

The pro-active approach is supplied in job file in json format. MultiStack will take these parameters and convert them to policies. Before execution of the user-defined, it will do a sanity check of all the policies defined to see if they are not violated. If they violated, MultiStack will ask the user to modify the respective parameters till the policy is honored, else the job would be terminated. For instance, if the cost per job is getting violated, the platform will ask the user to update the job parameters submit the job again.

The platform boots the following EC2 instances based on the job characteristics:

<table>
<thead>
<tr>
<th>Job characteristic</th>
<th>EC2 instance type</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU intensive</td>
<td>c3, c4, x1</td>
<td>c3, c4 and x1 instances are compute and memory optimized</td>
</tr>
<tr>
<td>Storage/disk IO intensive</td>
<td>i2, r3</td>
<td>i2 and r3 instances are optimized for storage performance and high IO tasks</td>
</tr>
<tr>
<td>GPU intensive</td>
<td>g2</td>
<td>g2 instances are optimized for gpu compute applications</td>
</tr>
<tr>
<td>General Purpose</td>
<td>t2, m4, m3</td>
<td>t2, m4 and m3 are low cost general purpose instance type</td>
</tr>
</tbody>
</table>

TABLE I

EC2 INSTANCE TYPE SELECTION ON BASIS OF JOB CHARACTERISTICS.

B. Reactive Approach

Currently, the Multistack platform tries to take reactive approach to spot instances. To enable the platform to respond reactively, the user must submit cost bid for the spot instances to be used to run the job. Suppose the current cost of spot instance exceeds the user given value, the platform will try rerun the job later when the price bid falls below the user-defined price.

VIII. RELATED WORK

Apache Hadoop [6] has been an enabler for big data processing on commodity hardware. The adoption of cloud has made it possible to run hadoop cluster of very large size within minutes. Although hadoop has been a phenomenal success, configuring and managing hadoop clusters has been non-trivial. As organizations are moving to virtualized environment, the need to run hadoop seamlessly and effectively over virtual machine clusters are emerging. The possibility of using both public and private cloud for running jobs respecting deadlines and minimizing the cost is exciting. As early adopters of both hadoop and cloud, we believe resource optimization across multiple cloud is one of the most important issue. Over years, we have tried to improve our infrastructure for better management and utilization of resources. We shifted a majority of our systems to private cloud using OpenStack. We are now experimenting with the possibility of optimizing and automating hadoop deployments on cloud. We propose MultiStack, a system for running on-demand hadoop across multiple cloud.

We propose to build a middleware for Hadoop family of project for on demand provisioning and smart scheduling of hadoop jobs, MultiStack [16].

MultiStack will provide users the ability to run different Hadoop jobs on different cloud providers. It provides both command line and web-based clients to submit jobs. It employs smart scheduling to estimate job completion time and costs involved, and schedule them accordingly to minimize both.

We have designed a system bringing previously discussed concepts to real world application.

MultiStack brings together the ability to process big data using various frameworks onto various public and private clouds. In the current implementation, MultiStack uses Amazon EC2 API [17].

Fig. 1. HadoopStack Overview
IX. EXISTING SOLUTIONS/TOOLS

In this section we would like to review existing tools with similar aims.

A. VMWare Serengeti

The open source project Serengeti [18] is an initiative by VMWare to bring virtualization benefits to hadoop.
- Its limitation is, the full support is available mainly to VMware vSphere based virtualization. Although, other platform are supported but with limited features.

B. OpenStack Savanna

This is an open source effort mainly led by Mirantis inc. 1
- Mainly limited to OpenStack and Hadoop.

C. Amazon Elastic MapReduce

Amazon provides a Elastic MapReduce Service [19] which has overlapping goals. Allows to use spot instances to scale.
- Limited only to Amazon.

D. Hadoop on Azure

To summarize, current state of art addresses subproblems of our goal. Combining them is not possible due to many issues like licensing, different architectures, etc.

X. ARCHITECTURE

Figure 2 shows various components and their interactions. The client interactions with the system is through REST calls.

A. Client tools

- Web Interface - The web component of MultiStack will allow a user to submit jobs, mention the input and output location and the deadline. MultiStack accepts EC2 credentials and hence works with most of the cloud providers - AWS, OpenStack, CloudStack, Eucalyptus, HPCloud etc. The resource allocation will be completely transparent to the user and optimized to minimize cost and job completion times.
- hstack-cli - This component is built keeping devops in mind. Using command line tool, a variety of jobs can be automated.
- hFlow - Its a web interface for composing map-reduce workflows. It allows users to chain different map-reduce functions. These functions can be submitted by a user or may belong to an existing map-reduce repository. We plan to use Apache Oozie [7] as our Workflow Scheduler.

The end user is exposed with only two APIs in this framework to conduct a workflow or independent multiple jobs. For Cluster API, the user will have to submit his requirements through a POST request. The parameters needed includes a json text or file as:

```
{
  jobs: {
    id:1
    RAM:4096(in MB)
    vcpu:4
    storage:10240(in MB)
  },
  {
    id:2
  }
}
```

It would create a cluster in Openstack, if there are resources left, else it would wait until a job is specified for that cluster and then spot instances are created by the provisioning unit. We could also add or delete machines in a cluster whenever needed. For Jobs API, the user will have to specify the information for a job along with the job characteristics and the scheduling unit will then analyze the need of spot instances to be created if the user has demanded for lower job completion time which will be described below. We could also add or delete a machine for a current running hadoop job. The parameter needed includes a json text or file as:

```
{
  Link: <jar url>
  Command: <hadoop command with input/output>
  Cluster: <cluster id>
  Cost limit: <in dollars($)>
  Job characteristics: < refer from Table I >
}
```

The following table shows APIs.

<table>
<thead>
<tr>
<th>API Group</th>
<th>HTTP verb</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>POST</td>
<td>Create a cluster</td>
</tr>
<tr>
<td>Cluster</td>
<td>PUT</td>
<td>Add machines to existing cluster</td>
</tr>
<tr>
<td>Cluster</td>
<td>GET</td>
<td>List cluster(s)</td>
</tr>
<tr>
<td>Cluster</td>
<td>DELETE</td>
<td>Destroy a cluster</td>
</tr>
<tr>
<td>Job</td>
<td>POST</td>
<td>Run a job</td>
</tr>
<tr>
<td>Job</td>
<td>PUT</td>
<td>Add machines to existing job</td>
</tr>
<tr>
<td>Job</td>
<td>GET</td>
<td>List job(s)</td>
</tr>
<tr>
<td>Job</td>
<td>DELETE</td>
<td>Cancel a job</td>
</tr>
</tbody>
</table>

TABLE II

API DESCRIPTION

1http://www.mirantis.com/
B. MultiStack Server

In this section we describe:
- api-server - It provides a REST API for communicating with the MultiStack.
- job manager - Responsible for managing and tracking user jobs.
- database - For job management.
- scheduling unit - Its a pluggable scheduler interface for scheduling decision of jobs.
- provisioning unit - Unit responsible for spawning and configuring instances.
- monitoring unit - Aggregates system info from multiple instances.

When the user submits a job request, the Job Manager daemon starts and stores the information in the Job DB with the status of scheduling. Simultaneously, the Scheduling unit is initiated which is the most integral part of the system. It decides the allocation of the resources on the multi-cloud (Openstack and AWS) depending on factors such as resource usage, job completion time, cost limit and the job characteristics specified by the user. As per the given job, MultiStack predicts the job completion time on our private network and then finds an Optimal Spot Instance count i.e. equation 1, depending on the spot instance history pricing and deadline prediction after scaling the cluster by i spot instance for a given cost limit specified by the user in the API request. The resource usage statistics are obtained by nova API in Openstack and ec2 API in Amazon Web Services. As the count is decided by the scheduling unit, the job db stores the final resource allocation on the hybrid cloud for this particular job and changes the job status to provisioning. The Provisioning Unit initializes the cluster and adds the instances from Openstack and AWS after the decision of the scheduling unit. After the cluster is initialized, the hadoop jobs are are initialized in the hybrid cloud. This is how Multistack Server works.

One thing to note that there are no fixed SLAs for spot instances so there is a termination notice, which give us a two-minute warning before any EC2 Spot instance is terminated. When the on spot instance is terminated, the rest of the map reduce jobs left are run in the Openstack cluster until the spot instances are spawned again. When the Spot market price falls back below our bid price, our Spot Auto Scaling group could automatically launch new Spot instances, if needed.

XI. Features

- Auto Scaling - Based on the deadline, infrastructure and cost requirements, the scheduler will smartly scale the resources allocated to a job. The user is also provided an option to manually scale up/down.
- Ability to run across multiple cloud providers - If you have multiple jobs and access to multiple cloud providers, MultiStack provides you the ability to run different jobs on different clouds.
- Priority based Job scheduling for minimizing cost and completion time - MultiStack uses machine learning to smartly allocate jobs across multiple cloud providers aiming to reduce your cost and job completion time.
- Performance optimization with storage integration - With storage administration privileges, MultiStack auto-replicates the resources that being utilized heavily, thus improving performance.
- Client Tools - A web interface and a simple command line interface for interacting with MultiStack.

XII. Evaluation

For Multistack evaluation, we show a tradeoff by reducing the execution time of the job with minimal increase in cost in a hybrid scenario. We will be using Openstack as the private cloud platform and AWS as the public cloud platform.

<table>
<thead>
<tr>
<th>Flavour</th>
<th>On Demand Price</th>
<th>Avg/Spot Instance Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>$0.044</td>
<td>$0.0071</td>
</tr>
<tr>
<td>m1.medium</td>
<td>$0.087</td>
<td>$0.0082</td>
</tr>
<tr>
<td>m1.large</td>
<td>$0.175</td>
<td>$0.0166</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>$0.35</td>
<td>$0.0357</td>
</tr>
</tbody>
</table>

TABLE III

EC2 Pricing

For testing, we will be using different map reduce programs and would run them on Openstack and AWS individually. We will run a simple wordcount program for different data sizes like 100MB, 500MB, 1GB and 5GB. This example is very standard and its source codes is easily available online. We will be creating a cluster of 4 nodes of m1.medium flavour in which we will have hadoop installed. The specification of m1.medium instance constitutes of 2 vcpus, 4GB RAM and 400GB hard disk. For running this testbed/cluster of 4 nodes in AWS, the on demand instance would cost a total of $0.348 per hour as shown in Table III.

From the above Figure 3, we could see that compared to Openstack, the average job completion time has reduced by 30-37% and compared to AWS it is still less, but the cost has been reduced from $0.348 to $0.0328. So the average cost has been significantly reduced to 10-30% compared to AWS On-Demand pricing.
MultiStack is under development and there are lot of interesting features being added. In this section we brief about the work currently in progress as part of the MultiStack project.

A. Dynamic Replication in Storage Systems

With the Algorithm 1, we can scale processing of data clusters at low cost, but storage system can become new bottleneck once the processing is scaled. We are exploring techniques to identify access patterns of storage objects and use them to optimize the replica count of objects and their placement. Frequency based load balancing is traditionally applied in such scenarios [20] [21], but we are working on techniques which go beyond simple frequency based approaches and uses other high level features like user modeling as well. We are in stage of combining learned user models with learned data models into a single unified model.

B. Fault Prediction

Use of device data and past faults to predict future faults. This work is in early phase and we have not collected enough fault data to conduct useful experiments.

C. Interplay of various learning sub-system

With more than one learning based models in a system, we want to characterize the interplay among them. The decisions of different learning systems are interlinked. How to make coherency across in system decision.

D. Auto-selection of data processing frameworks

It is well known that not all frameworks are made equal. End-user of the data processing system often does not understand the design decisions behind data processing framework designs. They might not know the right tool for the job. Choosing right framework is not a straight forward procedure given that it depends on complex data points such as data size, computation nature etc.

We propose to combine learning and heuristics based methods to identify most suitable platform for running a given processing task for the given data.

A graph showing performance of a algorithm on different framework and different data size.

XIV. Conclusion

In this paper, we detailed MultiStack, a cost and deadline aware middleware for running big data processing jobs across multiple clouds. We discussed architecture and various design choices in implementation. We also provided primary evaluation of our system and cost savings.