CloudKon: DTS

Distributed Task Scheduling with Amazon STACK

Contents

[0. Table of Figures 1](#_Toc372790150)

[1. Abstract 2](#_Toc372790151)

[2. Introduction 2](#_Toc372790152)

[3. Background 3](#_Toc372790153)

[3.1. Amazon Elastic Compute Cloud (EC2) 3](#_Toc372790154)

[4. Motivation 4](#_Toc372790155)

[5. Proposed Solution 4](#_Toc372790156)

[6. Evaluation 4](#_Toc372790157)

[7. Related Work 4](#_Toc372790158)

[8. Conclusion and Future Work 4](#_Toc372790159)

[9. References 4](#_Toc372790160)

# Table of Figures

[Figure 1 : Architecture Diagram 4](#_Toc370105379)

# Abstract

Predictions are that by the end of this decade, we will have exascale system with millions of nodes and billions of threads of execution[1]. Task Scheduling and execution of tasks over these large scale, distributed systems plays an important role in achieving good performance and high system utilization. Many Task computing paradigm[2] aims to bridge the gap between High Performane Computing and High Throughput Computing . Tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or [data-intensive](http://en.wikipedia.org/wiki/Data-intensive_computing) but MTC tasks include loosely coupled communication intensive tasks. Today’s jobschedulers have centralized Master/Slaves architecture (e.g. Slurm, Condor, PBS,SGE), where a centralized server is in charge of the resource provisioning and job execution. This architecture has worked well in modest scales and coarse granular workloads, but it has poor scalability at the extreme scales of petascale systems with fine granular MTC workloads. The goal of this project is to provide a efficient light weight and scalable distributed execution framework built on built on open source stack[HazleCast,Cassandra] to address MTC workloads deployed over Amazon Ec2 instance in cloud environment

# Introduction

The goal of an execution fabric is to effectively utilize the execution system aiming towards high throughput and also provide efficient results for executed tasks. Today’s workload involves a lot of fine granular workloads with execution times in seconds. Centralized schedulers are optimized towards high computational massive tasks where the complex decision policy and architecture of the schedulers play a major role. But they tend to add considerable overhead while scheduling these lots of small tasks. Moreover the centralized architecture tends to be a bottleneck in scheduling and execution. The solution to this problem is to have a decentralized and simple architecture. A decentralized architecture avoids the single point of failure, while a simple architecture reduces the considerable overhead involved in decision making for scheduling.

An execution fabric requires lot of computing resources to address the ever growing workload of today’s world. Clouds seem to be a viable solution to this problem. Our solution is to build a loosely coupled compact and distributed execution fabric over public cloud (Amazon Ec2 instance) with distributed building blocks such as Cassandra [3] and HazleCast [4]. The motivation behind using open source stack in favour of extensive use of AWS [Amazon Web Services] is to decouple the fabric from AWS and provide easier transition to private cloud environment.

Recent studies suggest that clouds were not suitable candidates for scientific HPC computing [5]. The problems listed were largely because of following the same approach involved in traditional clusters and grids. Clouds differ a lot from HPC applications as they are based on virtualization and shared resources.Our work involves running applications optimized for cloud environment. Traditional workloads can also be run on our execution fabric but with suitable decomposition of the workload at the client side.

In this project we implement a scalable distributed task execution framework. We have made extensive use of HazleCast , a highly scalable data distribution platform which acts as the reliable storage for the executable tasks . Our next building block is Cassandra, a distributed NoSQL store offering scalability and high availability for monitoring of the entire system. We also leverage the Amazon Elastic Compute Cloud (EC2) to manage virtual resources.

Today’s data analytics are moving towards shorter jobs with higher throughput and shorter latency. More applications are moving towards running higher number of jobs in order to improve the application throughput and performance. The focus is shifting towards Many Task Computing paradigm. Many task computing includes loosely coupled applications that are generally communication intensive.

We propose CloudKon as a job management system that achieves good load balancing and high system utilization. The heart of the CloudKon is the distributed queuing service. We have used HazleCast to facilitate this purpose. HazelCast performs the role of a highly available and reliable distributed pool of tasks to perfection. Worker Nodes are not adminstered by a centralized dispatcher or scheduler. Our work proposes an efficient pull architecture i.e worker nodes pull the tasks from the pool if they are idle. The system is loosely coupled and each component can be scaled based on the needs.

# Background

This section covers the necessary background information on Amazon EC2, HazleCast, Cassandra and Many-Task Computing (MTC).

# Amazon Elastic Compute Cloud (EC2)

Cloud computing services are broadly categorized into three layers

* Infrastructure-as-a-Service (IaaS)
* Platform-as-a-Service (PaaS)
* Software-as-a-Service (SaaS).

The focus of this project is on IaaS since the scientific computing community mostly focuses on IaaS because of the need for compatibility with legacy applications and systems. Amazon Elastic Compute Cloud (Amazon EC2) [6] is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2 presents a true cloud hosting service, allowing you to use web service interfaces to launch instances with a variety of operating systems, load them with your custom application environment, manage your network’s access permissions, and run your image using as many or few systems as you desire.

Amazon uses XEN hypervisor [7] as a middleware to run multiple Virtual Machines on their physical infrastructure. EC2 provides a web service that allows anyone to run their own applications on Amazon’s computing infrastructure, by letting customers “rent” computing resources by the hour. Clients are given access to an “unlimited” source of compute capacity, which is delivered through what is known as EC2 instance. Basically, an instance is a running virtual machine on Amazon’s cloud platform. Each of these instances is deployed with an Amazon Machine Image (AMI), which is just a pre-configured operating system and some bundled application software.

There exist several types of instances, each of them with different compute capacities, memory size, I/O performance and storage. Users launch one or more instances by specifying the instance type. Then the instances will be deployed on the server and user can connect to them via SSH using their public IP address. Amazon guarantees the availability rate of 99.95% in its Service Level Agreement. That means the instances are guaranteed to be available 99.95% of the time.

Considering the ways we can have access to these instances, we can categorize them in three different types:

* **Reserved instances:** Amazon allows us to pay upfront per each instance that we want to use during a given period, and in exchange, they give us a lower hourly cost for each of them. Along with the savings, with these instances we make sure that we will have availability through all the period that we paid for.
* **On demand instances:** these are the most common type of instances. You only pay for what you use, allowing easy allocation and deallocation of resources, depending on your capacity requirements. Customers are billed at the end of each month.
* **Spot instances:** This is a very interesting concept. In order to achieve a better utilization of their infrastructure, Amazon allows us to bid on unused EC2 capacity and run instances until the current spot instance price exceeds our bid. Amazon based on the available capacity sets the spot price and load of their systems and it is updated in a 5-minute period. The prices of these instances are much lower than what you pay for On-demand instances. As a drawback, the availability of you instances is only assured while the spot price is under bid. As previously stated, Amazon automatically terminates those instances whose bid is exceeded by the spot price. Besides, one cannot stop a spot instance and use it later as it happens with on-demand or reserved instances. Spot instances can only be terminated or rebooted.

Among these types, the spot instances seem to be the most appropriate for running short-term applications under certain conditions, since they provide the same capacity and features as the other instances at a lower rate. These include scientific applications, which usually run for a predictable amount of time, lowering the costs per experiment.

# HazleCast

# Cassandra

# Many Task Computing

Many-Task Computing (MTC) was introduced by Raicu et al. [14][15] in 2008 to describe a class of applications that did not fit easily into the categories of traditional high-performance computing (HPC) or high-throughput computing (HTC). Many MTC applications are structured as graphs of discrete tasks, with explicit input and output dependencies forming the graph edges. In many cases, the data dependencies will be files that are written to and read from a file system shared between the compute resources; however, MTC does not exclude applications in which tasks communicate in other manners.

MTC applications often demand a short time to solution, may be communication intensive or data intensive, and may comprise of a large number of short tasks. Tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. The set of tasks may be static or dynamic, homogeneous or heterogeneous, loosely coupled or tightly coupled. The aggregate number of tasks, quantity of computing, and volumes of data may be extremely large. For many applications, a graph of distinct tasks is a natural way to conceptualize the computation. Structuring an application in this way also gives increased flexibility. For example, it allows tasks to be run on multiple different supercomputers simultaneously; it simplifies failure recovery and allows the application to continue when nodes fail, if tasks write their results to persistent storage as they finish; and it permits the application to be tested and run on varying numbers of nodes without any rewriting or modification.

The hardware of current and future large-scale HPC systems, with their high degree of parallelism and support for intensive communication, is well suited for achieving low turnaround times with large, intensive MTC applications. The MTC paradigm has been defined and built with the scalability of tomorrow’s systems as a priority and can address many of the HPC shortcomings at extreme scales.

# Motivation

The main factors leading to our implementation of CloudKon are as follows

* Design and architect a light-weight task execution framework for MTC workloads
* Design a simple execution framework with a no frills user interface
* Design a robust framework which can easily switch between public and private cloud environments
* Design an extremely scalable execution framework
* Design a loosely coupled framework to support future enhancements
* Evaluate CloudKon with other state-of-the-art task execution systems
* Deliver excellent throughput with <5% codebase of the job management systems

# Proposed Solution

# Evaluation

# Related Work

# Conclusion and Future Work

# References

1. P. Kogge, et. al., “Exascale computing study: Technology challenges in achieving exascale systems,” 2008
2. IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08)
3. HazleCast <http://www.hazelcast.com/>
4. Cassandra <http://cassandra.apache.org/>
5. Q. He, S. Zhou, B. Kobler, D. Duffy, and T.McGlynn. “Case study for running HPC applications in public clouds,” In Proc. of ACM Symposium on High Performance Distributed

Computing, 2010.

1. Amazon Elastic Compute Cloud (Amazon EC2), Amazon Web Services, [online] 2013,

(<http://aws.amazon.com/ec2/>)

1. Xen Hypervisor, xen.org, [online] 2013, (http://www.xen.org/products/xenhyp.html)
2. a
3. CloudKon <http://www.cs.iit.edu/~iraicu/teaching/CS554-F13/proj/CloudKon-DTS.pdf>
4. D. Thain, T. Tannenbaum, M. Livny, “Distributed Computing in Practice: The Condor Experience” Concurrency and Computation: Practice and Experience 17 (2-4), pp. 323-356, 2005.
5. M. A. Jette et. al, Slurm: Simple linux utility for resource management. In Lecture Notes in Computer Sicence: Proceedings of Job Scheduling Strategies for Prarallel Procesing (JSSPP) 2003 (2002), Springer-Verlag, pp. 44-60
6. B. Bode et. al. “The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters,” Usenix, 4th Annual Linux Showcase & Conference, 2000.
7. I. Raicu, et. al. “Falkon: A Fast and Light-weight tasK executiON Framework,” IEEE/ACM SC 2007
8. ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table, IPDPS 2013