Exploiting Heterogeneity in Amazon EC2 for Saving Cost

M. Jalal*, V. Varadarjan†, M. Swift†
*ECE Department, University of Wisconsin-Madison
jalal@wisc.edu
†CS Department, University of Wisconsin-Madison
{venkatv, swift }@cs.wisc.edu

I. Abstract

Performance heterogeneity observed during running instances on Amazon EC2 even amongst the nodes with the same specifications, makes us revise the pricing policies. In EC2, pricing is per hour and if the client is idle he wouldnt be charged but even when hes being charged he might not be charged fairly enough. In this project we aim to provide a client-side strategy management for selecting the nodes from EC2 and migrating the instances depending on various metric including node capabilities and needed performance. The aim of this project is gaining more efficiency improvement through use of better policies and assigning the best node type of each zone as a duplication in the bundle of nodes which we select for a client. In case of failure in EC2 nodes or overload in any of the primary nodes selected by client, load will be migrated to another sub-type node in the client node bundle within the same instance type.

II. Introduction

Having multiple cloud provider in cloud computing era, makes metrics like pricing to be more pronounced. In this study we focus on Amazon EC2 pricing which is flat rate pricing. EC2 claims that Pay only for what you use . EC2 even has a bill calculator using which you can estimate your monthly cost. Amazon instances are divided into standard, micro, high-memory, high-CPU, cluster compute, cluster GPU, high I/O and high storage instances each of which are tuned for specific workloads[7] . All the aforementioned instance types can be bought in reserved or on-demand fashion. Also recently Amazon has announced its spot instances which can be bid on the price the customer is willing to pay and these instances use the unused EC2 capacity. Amazon EC2 tries to maintain the same CPU computing capability for each instance type even though there might be some slight CPU frequency heterogeneity in the underlying infrastructure and calls it ECU which stands for Elastic Computing Unit and the higher the ECU is, the more computing power that instance type has. In Table 1 all the instance types are shown categorized in their instance family and their specifications like ECU, memory, disk and I/O capabilities are shown accordingly. Focus of this study is performance variation due to hardware heterogeneity. Upgrading the hardware is one main reason for hardware heterogeneity even within the same instance type. In this study m1.small instance has been selected from first generation standard instances. Within this instance type, there are four various hardware type named AMD, E5430, E5507, and E5645 which differ slightly in CPU frequency while differing almost significantly in cache size shown in Table 2. In this research, we focus on customer-level placement strategies in which multiple workloads can run at the same type. Within the same instance, there is significant hardware differences which creates performance gaps. In these strategies we decide to migrate and tasks on the instance sub-type which has better performance to save cost. These strategies are not dependent on Amazon EC2 provider and run on client-side [8].

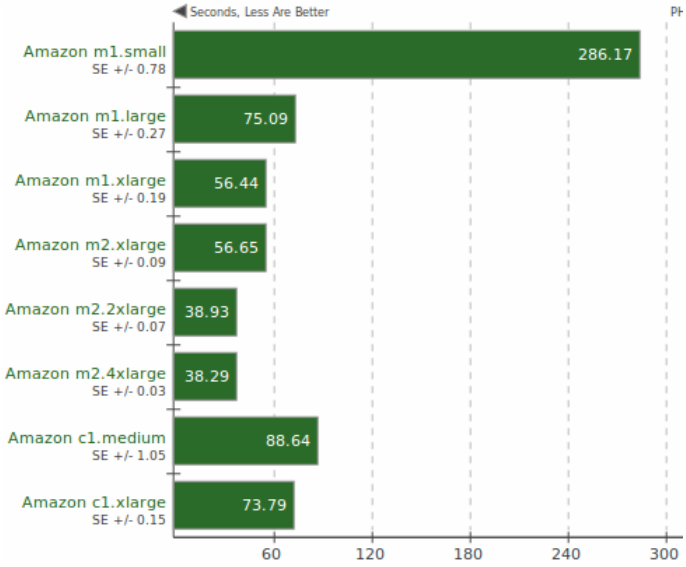

Fig. 1. Time to compile Apache on various instance types

In Figure.1 the huge performance heterogeneity of running jobs on various instance types is illustrated. Though the focus of this study is sub-type performance heterogeneity, this figure is a good illustration of heterogeneity in clouds. This figure also depicts that for running a task with optimum performance you should be aware of each instance type specification as each instance type is tuned for running a specific kind of workload.

TABLE II
Hardware heterogeneity in M1.Small instance type

CPU	Frequency	Cache	Introduced
AMD 2218 HE	2.6 GHz	1 MB	Q1 2007
Intel E5430	2.66 GHz	6 MB	Q4 2007
Intel E5507	2.26 GHz	4 MB	Q1 2010
Intel E5645	2.4 GHz	12 MB	Q1 2010

Table 2 shows the heterogeneity in m1.small instance type and shows that the cause for hardware heterogeneity is different years these instances have been introduced to EC2 cloud.

III. Related Work

It is shown in [2] that variation within the same sub-type of an instance is slightly small while the variation between different sub-type of an instance might be about 60 percent. It is mentioned if EC2 tenants select better sub-type machines for running their jobs, they can gain 30 percent saving in the cost. They have tried to reduce the cost by finding the best performing node in an instance

Instance Family	Instance Type	ECU	Memory (GB)	Disk (GB)	I/O
Standard	m1.small	1	1.7	160	moderate
	m1.medium	2	3.75	410	moderate
	m1.large	4	7.5	850	high
	m1.xlarge	8	15	690	high
Micro	t1.micro	Up to 2	631 MB	EBS only	low
High-Memory	m2.xlarge	6.5	17.1	420	moderate
	m2.2xlarge	13	34.2	850	high
	m2.4xlarge	26	68.4	1690	high
High-CPU	c1.medium	5	1.7	350	moderate
	c1.xlarge	20	7	1690	high
Cluster	cc1.4xlarge	33.5	23	1690	very high
	cc1.8xlarge	88	60.5	3370	very high
GPU	cg1.4xlarge	33.5	22	1690	very high

TABLE I
AMAZON EC2 INSTANCES [5].

type and run the job on it. Also they have tried to model the cost as well as cost saving using elaborate formulas. They have used Httperf for web server throughput measurement as an application-level benchmark. Their dynamic network testing is rather CPU hungry than network hungry. CloudMeter[1], a paper written by Farley et al. , studies nature and range of variation across various workload and has realized the significant performance heterogeneity in the cloud computing environment. One main source for this heterogeneity is different node architectures across the cloud. In our study, currently we have 4 different node architectures among which instances are being migrated. In [1] tenants affects the placement policy only by starting and stopping the instances. Depending on minimizing the cost or latency, tenants select different strategie among up-front exploration, and opportunistic replacement.

In terms of heterogeneity in clouds, several studies has been conducted. In [4] they have used GPUs for VMM cloud management task acceleration. For showing benefits of hypervisor acceleration, they have used GPU for speeding up the MD5 hashing. They have shown that memory cleanup, batch page table updates, memory hashing, memory compression and virus signature scanning in VMM will benefit significantly by use of GPU acceleration. Authors in [3] have used Google trace in a heterogeneous cloud environment and have studied various scheduling schemes. They have shown that the heterogeneity in the Google Trace public workload will decrease the efficiency of traditional scheduling algorithm. Not only heterogeneous, the workload is highly dynamic over time including both short times and very long-term jobs.They have provided new resource management strategies which will fit the dynamicity of the workload as well as heterogeneity of the machines as well as tasks.

IV. PROPOSED IDEA

Given a mix of workloads with different performance on different node types and a mix of node types, we desire to study efficiency of migration on customer-side if they are running their jobs on bad nodes. Amazon EC2 currently provides no API for moving your job from one node to another within the same instance type hence we believe existence of such an API is necessary for better pricing and gaining better performance within the same paid price. Customer decides to select one of the placement strategies depending on his needs and tries to gain the most possible performance within the price he pays by appropriate timing of his job injection into the cloud as well as duration of his job after selecting a suitable strategy.

A. Simulation

Multi-tenant CloudMeter simulator developed in Dr. Swifts research group was configured in way to run multiple workload simultaneously. This simulator was configured to have have three later of administration named customer level, application level and data-center level. A set of simulations has been done to study efficiency of migration on tenant-side from one bad node to another node with better performance for the mentioned task in the same instance type for reducing the price. Simulator output is all of the instances of each tenant in the beginning of each time quanta as well as all the instance killings and migrations during that quanta. Also performance has been calculated for all the active instances of a tenant in the end of each time quanta.The final result of simulator includes number of migrations in the whole run for each tenant as well as their effective rate and total done work. In [] they have used historical performance of a tenants job for realizing it future performance needs.

B. Configuration File

This simulator accepts a configuration file consisting information regarding the workloads. The configuration file includes number of tenants, number of instances each tenant has, migration penalty in second, job duration and type of strategy. In table 4 you can see all the aforementioned information regarding tenant configuration. Also various instance types and within each various instance sub-type are provided in the configuration file including information about fraction of machines covered by this instance sub-type and its mean as well as its standard deviation.

TABLE III
PERFORMANCE STATISTICS FOR NER IN M1.SMALL INSTANCE TYPE

CPU	Fraction	Mean	StDev
AMD 2218 HE	0.18	9.07	0.29
Intel E5430	0.25	10.47	0.44
Intel E5507	0.35	10.38	0.62
Intel E5645	0.22	11.94	0.47

In table 3 part of our configuration file for m1.small subtype related to NER workload is shown.

C. Workload

Different aspects of the system behaviour must be stressed so we need to have at least two extreme workloads one concentrated on CPU-intensive computations like NER(a natural language recognizer) and the one concentrated on bandwidth-hungry applications like

Strategy	Start Quanta	T	A	B	Migration Penalty	mu	AlphaAgg	AlphaServ	Workload Type
CPU-MAX	0	13	20	10	180	2	0	1	NER
CPU-MAX	0	13	20	10	180	2	0	1	Apache
CPU-MAX	10	14	20	10	180	2	0	1	NER
CPU-MAX	10	14	20	10	180	2	0	1	Apache

TABLE IV
TENANT-RELATED INFORMATION IN CONFIGURATION FILE

Apache Web Server. In future studies, it is advisable to use more workload for different kind of computing including HPC.

D. Placement Strategies

There are various placement strategies in CloudMeter simulator which are basically categorized into upfront exploration and opportunistic replacement. In the former, tenants launch more instances (A+B) than what they really need and kill the B worst instances, say in terms of performance while in the latter, depending on future performance needs for the instances, tenants decide to migrate an instance or not. CPU-MAX and PERF-MAX strategies both belong to upfront exploration and they kill the B worst instances based on maximum CPU and maximum performance respectively. Also CPU-OPREP and Upfront-OPREP are two opportunistic replacement strategies studied in this research.

I have selected CPU-OPREP, Upfront-OPREP, and CPU-MAX among the various placement gaming available in CloudMeter due to lack of time.

E. Simulation Results

We have done various studies in our workload regarding duration of jobs and their start time on number of migrations. Also number of migrations per hour has been investigated in some of the selected strategies. We also decided to exploit Firstly I ran three different durations (very short, medium, and long) on the unmodified CloudMeter first with non-overlapping jobs as shown in Table 5 as well as injecting the jobs when 70 percent of previous jobs has finished in overlapping fashion depicted in table 6. I did the same kind of simulations in my modified simulator shown in Table 7 and Table 8 respectively.

TABLE V
NUMBER OF MIGRATIONS AND EFFECTIVE RATE IN SINGLE-WORKLOAD SIMULATOR RUNNING NON-OVERLAPPING JOBS

Strategy	Job Duration	Migrations	Effective rate
CPU-MAX	2	0	1.05
CPU-OPREP	2	0	1.04
CPU-MAX	20	72	9.52
CPU-OPREP	20	18	6.88
CPU-MAX	100	63	5.93
CPU-OPREP	100	24	5.56

TABLE VI
NUMBER OF MIGRATIONS AND EFFECTIVE RATE IN SINGLE-WORKLOAD SIMULATOR RUNNING OVERLAPPING JOBS

Strategy	Job Duration	Migrations	Effective rate
CPU-MAX	20	28	9.2
CPU-OPREP	20	47	7.16
CPU-MAX	100	72	6.82
CPU-OPREP	100	23	6.38

TABLE VII
NUMBER OF MIGRATIONS AND EFFECTIVE RATE IN MULTI-WORKLOAD SIMULATOR RUNNING NON-OVERLAPPING JOBS

Strategy	Job Duration	Migrations	Effective rate
CPU-MAX	2	0	0.13
CPU-OPREP	2	0	0.125
CPU-MAX	20	30	1.16
CPU-OPREP	20	193	1.07
CPU-MAX	100	31	5.73
CPU-OPREP	100	991	5.28

TABLE VIII
NUMBER OF MIGRATIONS AND EFFECTIVE RATE IN MULTI-WORKLOAD SIMULATOR RUNNING OVERLAPPING JOBS

Strategy	Job Duration	Migrations	Effective rate
CPU-MAX	20	13	1.72
CPU-OPREP	20	333	1.59
CPU-MAX	100	18	6.62
CPU-OPREP	100	1456	6.07

As shown in the above tables, when we have very short term jobs theres no migration involved no matter what strategy we use. Also it is shown that when we have overlapping jobs injected in the 70 percent of completion of half of the previously injected time at start time, depending on used strategies various benefits are shown. As an example in CPU-MAX injecting jobs in overlapping method reduces number of migrations significantly while in CPU-OPREP strategy using the overlapping method will increase number of migrations significantly.

V. CONCLUSION

Heterogeneity in cloud environment due to various hardware upgrades will end in significant performance heterogeneity even among the same instance sub-types. EC2 flat hourly rate doesnt address this issue and no matter how much performance you gain, you will have the same bill while using the same instance type. In this research we have modified the simulator for being able to run multiple workload and multiple available instance type of the approximately the same computing capability. Results show that in the configured simulator theres slightly less delay and depending on the type of placement strategy used we have either increased or decreased number of migrations comparing their single-workload counterpart. This work is yet in progress and results are not very finalized. I also studies effect of job injection time on number of migration and realized that if all the tenants inject their jobs simultaneously, we will have tremendous number of migrations no matter what the selected strategy is. Also If we inject the tasks when about 70 percent of half of the other tenant tasks have been accomplished, we will face less migrations due to lightly loaded nodes.

VI. Future Work

I believe not only we should study the performance fluctuations in m1.small but also we should consider other instance types. Performance fluctuation is almost doubled in m1.xlarge instances which will then show how beneficial our study truly is. In a wider study I would extract the information for all the EC2 instance types (though Im agree that selecting m1.small will lead to the approximately same result). Amazon EC2 has this idea of spot pricing [9] which should be investigated further as they have tried to improve the utilization using this idea. Using the spot price history will help us have better accuracy in our simulation results. Also we can study effect of single-thread and multi-thread on performance heterogeneity as well. Eventually we can use CPU benchmarking information in [6] for extending simulator (which currently supports m1.small instance type of Amazon EC2) to cover other cloud providers architecture. For this purpose, migration penalty modeling for various cloud provider should be investigated as well as preparing configuration files accordingly.

VII. Acknowledgement

I am grateful for Dr. Swifts guidance throughout the project and for providing me with the CloudMeter simulator. Also I would like to thank Venkatanathan Varadarjan for helping me in configuring the simulator. Thanks are due to Dr. Shan Lu for encouraging me to pursue this project and her comments in better accomplishing it.

References

[1] B. Farley et al., *More for Your Money: Exploiting Performance Heterogeneity in Public Clouds*, SOCC 2012.
[2] Z. Ou et al., *Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2* , HotCloud 2012.
[3] C. Reiss et al, *Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis*, SOCC 2012.
[4] S. Suneja et al., *Accelerating The Cloud with Heterogeneous Computing* , HotCloud 2012.
[5] H.Zhuang, *Performance Evaluation of Virtualization in Cloud Data Center* , M.Sc. Thesis, Aolto University, 2012.
[6] *CPU Benchmarking in the Cloud*, http://blog.cloudharmony.com/2010/05/what-is-ecu-cpu-benchmarking-in-cloud.html.
[7] *Amazon Instance Types* http://aws.amazon.com/ec2/instance-types/.
[8] *Amazon EC2 Pricing* http://aws.amazon.com/ec2/pricing/.
[9] *Spot Pricing* http://aws.amazon.com/ec2/spot-instances/.