CPET- Cloud Price Estimation Tool

Akshat Kumar
University of Waterloo
a77kumar@uwaterloo.ca

Manish Chauhan
University of Waterloo
m5chauha@uwaterloo.ca

Lalit Agarwal
University of Waterloo
lagarwal@uwaterloo.ca

ABSTRACT

Cloud computing is emerging as one of the most popular technology in today's world. Due to the flexibility and reliability provided by these services, many business application have either already migrated their IT infrastructure to the cloud or are planning to do it in the near future. Cloud service providers generally allow users to pay for these services through either a reservation plan or an on-demand plan. With the reservation plan, the consumer can reduce the total resource provisioning cost. However, this resource provisioning is challenging due to the uncertainty which may either lead to under-provisioning, over-provisioning or improper provisioning of resources. The users who are planning to transition to the cloud infrastructure are undecided about the optimal pricing plans they should pay and hence become hesitant to migrate their application. It has been also been found through surveys that even some companies have reported spending more on cloud resources than they should have. In this paper, we present a cloud service price estimation tool called CPET which guides users in subscribing to an optimal Amazon EC2 server instance based on the resource requirements computed by the tool. We evaluated our tool on University of Waterloo's SYN clusters using three different kinds of workloads.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organisations**]: Network Architecture and Design

General Terms

Performance, Measurement

Keywords

Cloud computing, resource provisioning

1. INTRODUCTION

Cloud Computing has gained tremendous popularity in recent years as it allows flexibility, convenience and low cost. It provides a pool of computing resources to users. Major service providers such as Microsoft, Amazon, Google provide a variety of services and applications to cloud users by provisioning and monitoring resources. More users are getting motivated to move their application from the private servers to the cloud. Companies around the globe are realising the true potential which cloud computing services have to offer and are willing to transition to the cloud infrastructure. However, at the same time they are also discovering the practical hurdles involved with migrating to the cloud infrastructure including cost and data privacy.

Most service providers provide two provisioning plans- a long-term reservation plan and a short term on-demand plan. Cloud services such as Amazon EC2 [1] and GoGrid [5] provide services with both types of plans. The reservation plan allows users to reserve resources according to their needs through a one-time fee payment. In such type of scenarios, the resources are reserved for the users before they are utilised. For example, in Amazon EC2 [1], users can reserve a different combination of small, medium and large instances according to their requirement, where an instance is characterised and priced on the basis of parameters like CPU family/cores, memory and disk capacity. On the other hand, pricing in on-demand plans is charged by pay-per-use basis. This plan allows users to dynamically provision resources based on the application demand at that moment. Since consumer pay to the cloud service provides in advance for the resources, the cost of utilizing computing resources provisioned by reservation plans is cheaper than that provisioned by on-demand plans. It has been found that [2] Amazon EC2 reduces the total resource provisioning cost by 49% when the resources are fully utilised. Hence, users prefer to reserve resources in advance as it reduces the total resource provisioning cost.

However, inefficiency of resource provisioning or subscribing to the incorrect plan by the users may lead to three problems- under-provisioning, over-provisioning and improper provisioning. Under-provisioning problem refers to the scenario when the amount of resources provisioned as part of the reservation plan may not be enough to meet the application demand. In case of shortage of resources, the application may not be able to perform as expected. One of the ways to address this issue would be by reserving more amount of resources than the actual demand. However, over-provisioning might occur in case one of the resource pool allocated is underutilised. The users end up paying more

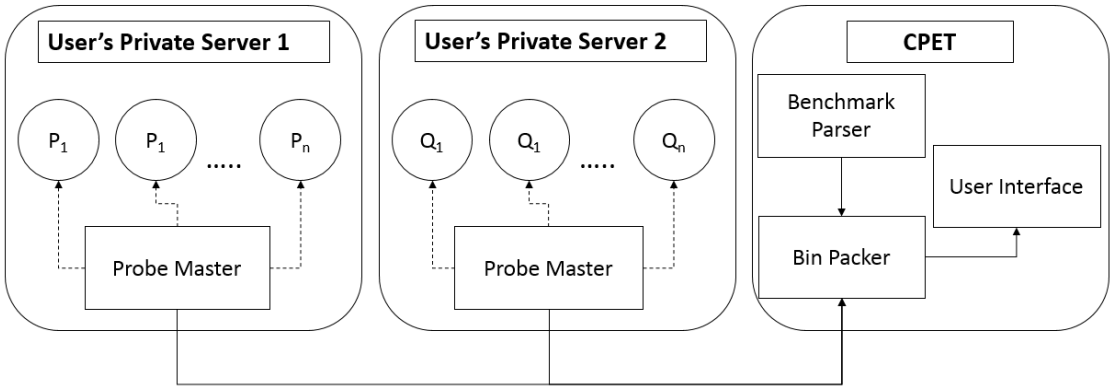

Figure 1: Operational Layout of CPET

for resources which are sometimes never used. Apart from the under-provisioning and over-provisioning problem, there is another problem of improper provisioning which might affect an application's performance. Due to demand uncertainty, users might subscribe to adequate amount of resources for one of the requirements but might feel short for other requirements. Every application has different CPU, memory requirements and sometimes these demands vary depending on the application usage. Improper provisioning generally occurs when the application is provisioned with sufficient processing resources for example but the memory resources provisioned doesn't meet the application demand. Therefore, while under-provisioning of the resources compromises the service quality, over-provisioning on the other hand wastes investment of the users. Sometimes, users are uncertain about all the resource requirements and often subscribe to a plan which fail to meet all the resource requirements.

Due to uncertainty about the computing resources required for a particular application, users are generally uncertain of the optimal cost they should pay to use the cloud services. Therefore they lack confidence to transition to the cloud. Also sometimes when users discover that they are paying more for resources, it builds distrust among the users towards the cloud providers. Hence, the user's uncertainty perception is a key driver for cloud computing continuance as the success of any service depends on a customer's continued use after initial adoption. A recent study [6] conducted by KPMG and Forbes on cloud adoption trends revealed that many companies find the transition to cloud infrastructure costly. The report finds that there is a growing consensus among the companies that the cloud computation costs have been higher than they expected. Around 33% of the users surveyed during the study mention that not only they found the resource costs to be expensive but also found it difficult to integrate their existing IT infrastructure with the cloud.

Planning the necessary amount of resources required for migrating the application from a private server to a cloud server is complicated. Load prediction plays a crucial in reducing the oversubscribed cost of under-provisioning or over-provisioning a system. Our goal is to provide an optimal estimate of the computing resources that would be required for migrating a particular application to a cloud server. In this paper, we present a tool called CPET (Cloud Price Estimation Tool) which predicts the estimated resource requirement of an application and provides optimal pricing suggestions to the users based on the resource consumption computed by the tool. The tool aims to provide assistance to users who are uncertain about the pricing plans they should subscribe to. We also believe that this tool will motivate novice users to migrate their applications to the cloud and take advantages of the computing services provided by popular cloud providers such as Amazon, Google etc.

2. CPET

In this paper, we present a utility tool called CPET which allows users to estimate the best pricing plan available for running their application on the cloud subject to the same performance and traffic as running in the private cloud. The tool collects information of the various resource requirements of the application which is to be migrated to the cloud like the number of CPU cycles required, amount of memory required, number of disk reads/writes taking place etc. It then aggregates the logs of the information collected by the tool and estimates an optimal pricing plan based on the collected logs. The current version of the tool refers the updated benchmarks for server instances provided by Amazon EC2 and suggests the best plan that the user should pay for. The tool outputs the best pricing plans for the 50th, 75th, 90th, 95th percentile and the peak value to allow users to choose a plan depending on the application sensitivity and needs.

The tool is primarily designed for users who are willing to migrate their applications to the cloud infrastructure but are uncertain about the number and the type of server instances they should reserve in advance. This tool will also help in guiding first-time users who want to transition to the cloud from their private server. Since the success of any service depends on the number of users who continue to use their service, our tool will promote trust among the users towards these cloud service providers and help them to retain existing users as well add new users.

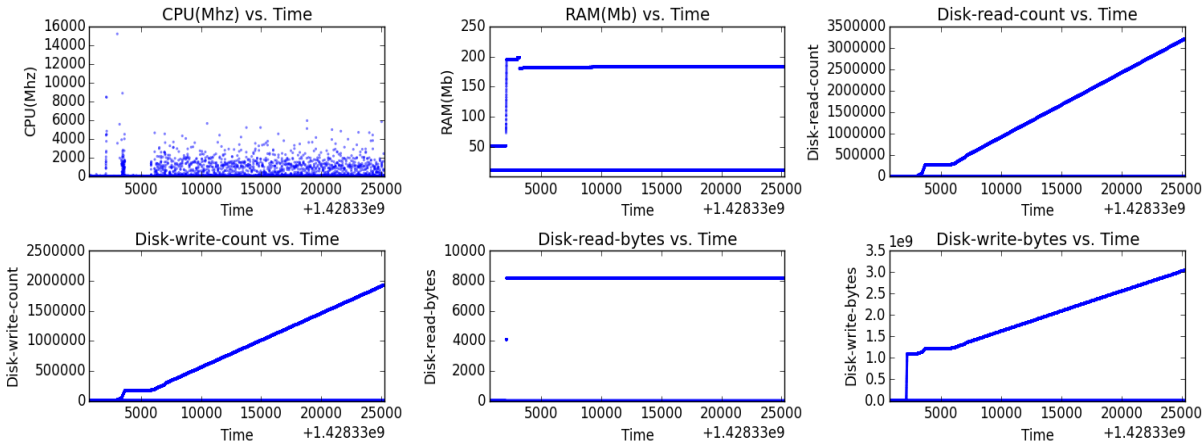

Figure 2: Scatter graph showing consumption of individual computing resources for a loaded MySQL process. The x-axis in each graph represents the epoch timestamp value.

3. CPET- ARCHITECTURE

The architecture of the system is divided into four modules- Probe master, Benchmark parser, Bin packer, and User interface which performs various functionalities of the tool. Only the Probe Master is required to be executed on the user's private machine where the application which is to be migrated is running. The rest of the three modules can run on a remote server thus reducing the computation load on the user's private machine. Figure 1 gives an overview of the operational layout of the tool. In case the user has multiple private servers, the tool can run independently on each server and the logs from all the private servers of the users can be aggregated and sent to the remote server for computing the best pricing plan. The detailed information about each module is presented in the sub-sections below.

3.1 Probe Master

The probe master module is tasked with collecting the resource consumption of processes. Different instances of the probe master can run independently and monitor processes and generate log files recording the consumption of different types of computing resources. Internally the instances of probe master can simultaneously monitor multiple processes, this achieved by ensuring that there is no locking between the monitoring of the different processes. Each process is monitored on a separate thread and the records are written to a separate file.

Probe master is programmed in the Python 3.4 for ensuring platform independence and allow speedy development. Probe master uses psutil (python system and process utilities) a cross-platform library for retrieving information on running processes and system utilization in Python. We have also made use of the sub-process module for detecting the actual CPU clock speed of the host machine, we use platform specific commands after detecting the host operating system.

The module runs on the private server of the user and takes as input the list of process names which needs to be migrated to the cloud. The input for the probe master module is a configuration file that contains information containing the process names or process IDs (PIDs) which needs to be monitored and the path to the directory where the output of the tool needs to be stored. The output of the probe master instances are a collection of csv files that contain quantitative descriptions about the consumption of different computing resources for each of the computing resources. Each csv files are named after the process that they contain information of. As shown in figure 2, the tool generates consumption logs for the CPU, RAM, Hard-Disk and other activities of the processes and sends these logs in the form of a text file to the bin-packing module for further processing. Some of the resource parameters collected includes cpu consumption (MHz), RAM consumption (MB), disk read/write count etc.

3.2 Benchmark Parser

Cloudlook [4] is a popular website which provides constantly updated benchmarks for server instances at popular public cloud providers. The benchmark parser module fetches and parses the current benchmarks for Amazon EC2 server instances so that the benchmarks can be referred while estimating the optimal cost. The benchmark parser fetches the complete html of the couldlook website and then uses the python library lxml [7] for initial pruning after that the python library called BeautifulSoup [3] is used for fine grained HTML parsing. The information of benchmarks collected include parameters like available CPU, RAM, disk rate/seeks etc. The parsed information about the different benchmarks is then sent to the bin-packing module for further processing. The output of the module is a csv file containing the benchmarks of all the tiers offered by amazon.

3.3 Bin Packer

Bin Packer performs the final operation: packing processes for their migration to cloud servers. It receives process resource consumption report from Probe Master, and server's resource statistics from Benchmark Parser and uses Pandas library [10] for analysis. This is a multi-dimensional Bin Packing problem where choice exists even for the bins. During our experiments, we found that finding optimal solution with intelligent heuristics or other algorithms (e.g. Greedy) is not guaranteed. To make sure user always gets the best price, exhaustive search was the only option which is very

time expensive. We checked the available pricing models and deduced that the optimal way to minimize the price is to use minimum servers. Bin Packer takes advantage of this fact. It starts with trying to host all processes on one server. If it turns out to be infeasible then Bin Packer gradually increments the number of server by one and tries again. For any given number of servers, Bin Packer tries all the possible permutations of processes. Bin Packer generates the three best prices with the details of process-server sets and their mappings as shown in figures 6 and 7. This process is repeated for each of 50, 75, 90, 95 and peak values of the all user processes.

3.4 User Interface

The tool comes along with a graphical user interface which allows the user to enter all the process names for which the resource requirements needs to be analysed. This module then calls the probe master module along with the list of process names to analyse the computing resources required. Once the processing is done, the bin-packing module sends back the best cloud service plan suggestion so that it can be displayed to the user. Figure 3 shows a screenshot of the user interface developed in Java using NetBeans [9].

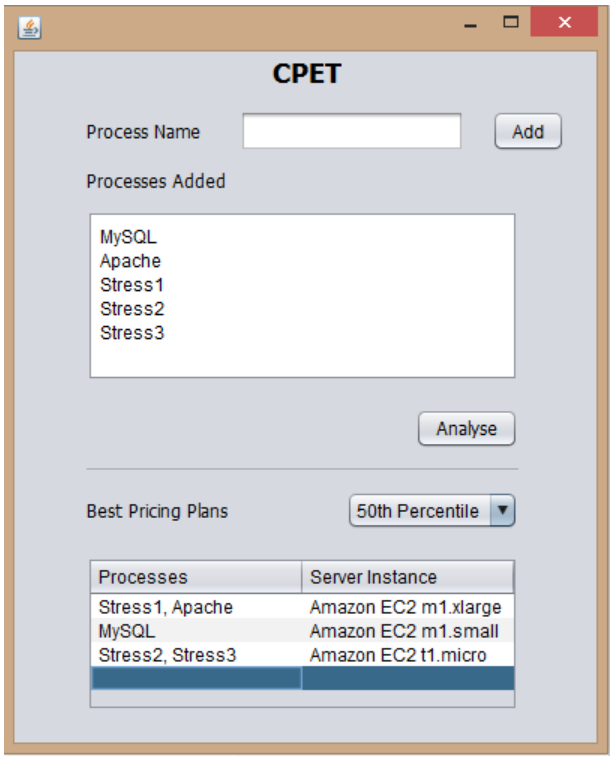

Figure 3: A screenshot of the user interface

Figure 4 gives an overview of the system layout of CPET. The user interface module calls the other three modules along with the list of processes to be monitored. Once the analysis is done by the bin-packer module based on the information provided by probe master and benchmark parser modules, it returns back the list of best pricing plans available to the user for each percentile.

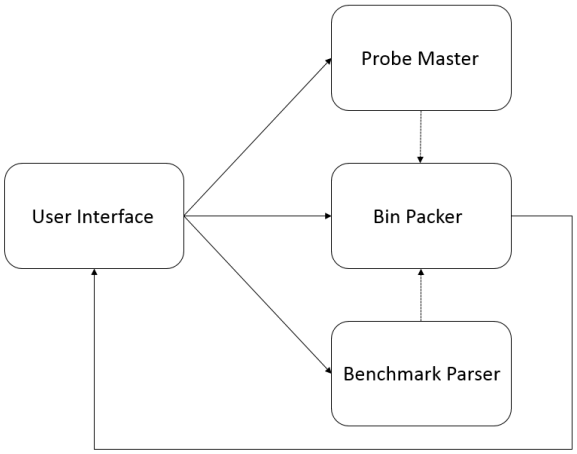

Figure 4: System Layout of CPET

4. EXPERIMENTAL SETUP

The evaluation of CPET has demanded a large amount of effort and we had to make some unforeseen additions to CPET in order to facilitate correct evaluation. For the evaluation we were granted access to University of Waterloo's SYN clusters [12]. These clusters are a multi-tenant compute facility which are generally used for network research. The SYN facility consists of 4 independent computing clusters named red, green, blue and yellow. Each cluster has 15 regular compute nodes and 1 large compute node. The compute nodes of the SYN cluster are stateless and the home directories are stored on a central admin node called as the white node. The white node is a gateway for accessing the other compute nodes in different clusters. The home directories are stored on a shared file system and can be accessed by all the compute nodes.

For evaluating CPET, we used two compute nodes: green00 and green01, of the SYN cluster which were used to host 4 different kinds of workloads. From the evaluation, we were able to verify and evaluate all the features of CPET modules. We made the experiment in three stages to ensure the correct behaviour by incrementally increasing the complexity of the experiment with every stage.

4.1 First Stage

The first stage of the experiment used one compute node to run all the CPET modules as well as the 1 instance each of 2 kinds of workloads. The workloads used for this stage were an instance of community version of MySql 5.5.41 database server serving minimal traffic and an instance Apache 2.4.7 webserver also serving minimal traffic. For this stage we collected logs for both the workloads for 17 hours. We were able to observe that even though the servers were serving minimal traffic their background tasks did impose a substantial footprint.

It was also made clear by this stage that we needed to add more workload instances to make effective use of the bin-packing module. We wanted to record a case where the bin-packing was able to give a non-trivial solution to the

packing problem. So it was decided that it was critical for successful evaluation to consider additional workloads. From this experiment stage we were able to confirm basic stability and compatibility of CPET modules.

4.2 Second Stage

For the second stage of experiment we made some changes to the experiment structure and added multiple instances of a new kind of workload. We also made some changes to CPET for allowing the use of PIDs for identifying processes that need to be monitored instead of just identifying them by their process name only. This change was required because of how we wanted to use the new workload for our experiment bringing the total to 22.

For this stage we added a new workload by installing the stress [11] command that is used stress is a deliberately simple workload generator for POSIX systems. It imposes a configurable amount of CPU, memory, I/O, and disk stress on the system. Stress command is written in C, and is free software licensed under the GPLv2. It is a single file called 'stress.c' whose internal organization is in essence a loop that forks worker processes and then waits for them to either complete normally or exit with an error. It has been used in a lot of product development labs as well as in research projects. We also used the mysqlslap [8] utility for increasing the mysql server load for the duration of the experiment.

For the experiment we executed the stress command in a loop multiple times, we used a shell script that was executed by a python script inside different processes. Form this experiment stage we were able to verify the stability and correctness of the probes in cases a large number of processes were monitored by a single instance of the probe and also the fidelity of log aggregation module and the bin packing module.

Processes	# of instances running on Green00			# of instances running on Green01		
	Stage 1	Stage 2	Stage 3	Stage 1	Stage 2	Stage 3
Apache	1	1	1	0	0	0
MySQL	1	1	1	0	0	0
Stress	0	20	20	0	0	20
Probe Master	1	1	1	0	0	1
Log Aggregator	1	1	1	0	0	1

Figure 5: Experimental Setup

4.3 Third Stage

For the third stage we almost doubled the number of workloads by using an additional compute node to the experiment by considering the 42 workloads running on two separate computing nodes running one CPET probe each. From this stage we were able to definitively verify the stability of the log aggregation module for the case of multiple probes contributing logs. We used two compute node in the green cluster green00 and green01. We generated load on the mysql server by using the mysqlslap [8] utility and simulated two kinds of workloads using the stress command's command line arguments. Figure 5 provides a summary of the experiment setup in different stages.

50th Percentile		75th Percentile		90th Percentile		95th Percentile		100th Percentile	
Solution ID	Price	Solution ID	Price	Solution ID	Price	Solution ID	Price	Solution ID	Price
1	93.8	1	167.4	1	224.1	1	231.1	1	231.1
2	102.9	2	169.4	2	224.2	2	231.2	2	231.2
3	103.4	3	170.9	3	227.6	3	234.6	3	234.6

Figure 6: Estimated Prices (in cents/hour)

5. RESULTS

CPET repeats the Bin Packing of the group of the user processes for each of the five percentile. All the prices estimated by CPET during this experiment are shown in figure 8. Every dot in the graph represents a feasible solution. X-axis shows the index of a feasible solution which is incremented each time a feasible solution is found. Y-axis shows the price in cent/hour for a feasible solution. As the required resource values of the user processes increases there are less possible permutations of them which can be hosted on a specific set of servers. This behaviour can be confirmed from the figure 8 in which numbers of feasible solutions are decreasing as the percentile of resource consumption for all the processes of the group is increased, individually. This graph also confirms that prices presented in figure 6 are optimal.

Processes	Server Instance
MySQL, Apache, Stress7, Stress12, Stress14, Stress13, Stress15, Stress17	Amazon EC2 m1.xlarge
Stress 8	Amazon EC2 m1.small
Stress 9	Amazon EC2 t1.micro
Stress 3	Amazon EC2 m1.medium
Stress 11	Amazon EC2 m1.medium
Stress 0, Stress 1, Stress 2, Stress 3, Stress 4, Stress 5, Stress 6, Stress 16	Amazon EC2 m1.xlarge

Figure 7: Estimatd server instances for various processes

CPET generates the best prices estimated and hosting details for the processes entered by the user. As shown in figure 6, for a set of user processes, CPET gives 15 choices to the user: three solutions for each of the five percentiles. 50-percentile results are most economical, but since they only satisfy half of the resource requirements of a process, this provisioning is more likely to result in bad application performance. On the other hand, price estimated for the 100 percentile i.e. peak values are most expensive, but with this provisioning it is safe to assume that user application's demand will be met for almost all the time. Depending upon the application sensitivity and available budget, user can select any of these choices. Figure 7 shows the details of the process hosting for the first solution of 50 percentile. It lists the set of processes and the corresponding server on which they need to be hosted to achieve the estimated performance and price.

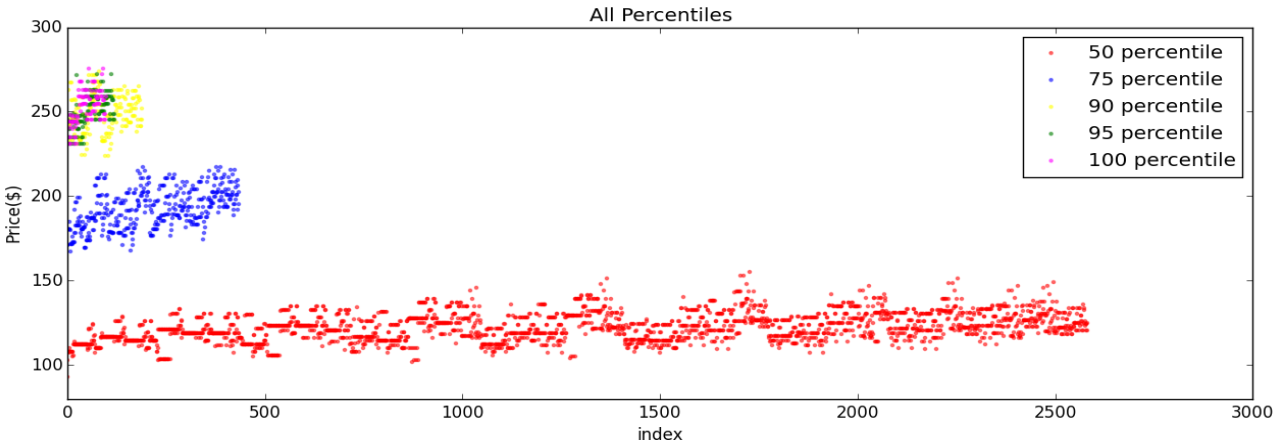

Figure 8: Scatter graph of Prices (in US $) as predicted by CPET for 50th, 75th, 90th, 95th and peak percentile

6. CHALLENGES AND FUTURE WORK

The tool currently lists the optimal pricing plan for Amazon EC2 server instances only. We decided to test our tool on Amazon EC2 server only as it is considered to be one of the most popular cloud service platform. In the future, we plan to extend our tool to include plans provided by other cloud service providers like Google, Microsoft etc as well. The users will then be able to choose server instances by paying optimal costs across all these cloud service platforms.

For the experiments, we referred a third-party website called CloudLook which contains updated benchmarks for these services. We plan to implement our own modules which computes benchmarks by running our application on different instances provided by various cloud service providers and analysing their services.

7. CONCLUSION

In this paper, we have presented CPET: a price estimation tool which saves money for the cloud subscribers. CPET is very user-friendly as it requires minimal input and no intervention from the user. CPET provides detailed resource provisioning and the flexibility of choosing a plan which fits in to the user's budget. This encourages the user to migrate his applications to the cloud and hence more revenue for the cloud service providers.

8. ACKNOWLEDGMENTS

We would like to thank Professor Bernard Wong for his continued guidance and support throughout this research and for finding time from his schedule for meeting with us to discuss about the project. Thanks are also due to Ms. Lori Paniak for assisting us in setting up the experiments on the SYN clusters for evaluating our tool.

9. REFERENCES

[1] Amazon ec2. `http://aws.amazon.com/ec2/`.
[2] Amazon ec2 reserved instances. `http://aws.amazon.com/ec2/reserved-instances/`.
[3] Beautiful soup. `http://www.crummy.com/software/BeautifulSoup/`.
[4] Cloudlook. `http://www.cloudlook.com/amazon-ec2-instance-types/`.
[5] Gogrid. `http://www.gogrid.com/`.
[6] Kpmg. `http://www.kpmg.com/global/en/issuesandinsights/articlespublications/cloud-service-providers-survey/pages/default.aspx/`.
[7] Lxml. `http://lxml.de/`.
[8] Mysqlslap. `https://dev.mysql.com/doc/refman/5.1/en/mysqlslap.html/`.
[9] Netbeans. `https://netbeans.org/`.
[10] Pandas. `http://pandas.pydata.org/`.
[11] Stress. `http://people.seas.harvard.edu/~apw/stress/`.
[12] University of waterloo's syn clusters. `http://uwlabinfo.wikidot.com/syncluster`.