Cost-aware Application Development and Management using CLOUD-METRIC

Alieu Jallow, Andreas Hellander and Salman Toor
Department of Information Technology, Division of Scientific Computing, Uppsala University, SE-75105, Uppsala, Sweden

Keywords: Cloud Infrastructures, Metering, Cost Estimation, Recommendations.

Abstract: Traditional application development tends to focus on two key objectives: the best possible performance and a scalable system architecture. This application development logic works well on private resources but with the growing use of public IaaS it is essential to find a balance between the cost and the performance of an application. Here we propose CLOUD-METRIC: a lightweight framework for cost-aware development of applications to be deployed in public clouds. The key functionality of CLOUD-METRIC is to allow users to develop applications on private IaaS (a dedicated cluster or an in-house cloud) while estimating the cost of running them on public IaaS. We have demonstrated the strengths of CLOUD-METRIC using two challenging use-cases orchestrated on SNIC Science Cloud, a community OpenStack cloud in Sweden, providing recommendation for a deployment strategy and associated cost estimates in Amazon EC2 and Google Compute Platform. In addition to cost estimation, the framework can also be used for basic application monitoring and as a real-time programming support tool to find bottlenecks in the distributed architecture.

1 INTRODUCTION

Consuming computational and storage resources as Infrastructure-as-a-Service (IaaS) is one of the fastest growing trends both in industry and academia. The major reasons for the adoption of this model is to reduce upfront investment cost, rapid time-to-market and availability of large capacity of production quality resources. IaaS providers such as Amazon, Google and Azure are offering service-level-agreements (SLAs) that are favorable for various business models. However, in order to realize the benefits and in particular to reduce the cost, it is essential that applications also adopt adequate design patterns to become scalable, resilient and vendor-agnostic.

Application architecture is still lagging behind the advances in modern distributed computing infrastructures. A particular challenge is that applications are traditionally designed with dedicated resources in mind. This is one of the reasons why users often fail to see benefits from adopting the cloud computing resource delivery model. This article highlights some key challenges in adoption of the cloud model and proposes a light-weight framework that will help to address those challenges and aid in the process of cloud application development. In particular:

1. [C-1.] Porting legacy applications to cloud infrastructure as virtual appliances is often trivial but to provide a cost effective execution environment is highly challenging.
2. [C-2.] Even if the application is cloud-native, elastic and fault-tolerant it is often not clear how to plan the execution environment in order to minimize the execution cost.
3. [C-3.] There is disconnect between application development and cost estimation for the deployed solution. In the scientific community, this is a major bottleneck for cloud adoption.
4. [C-4.] Without having brokering platforms that can showcase the provided services and a realistic cost comparison between different cloud providers the risk of getting vendor lock-in will always be high.

Apart from the above mentioned technical challenges another factor is the complex and varying billing models for the wide variety of cloud resources. For example, AWS offers 11 different high-level categories (T2, M4, X1 and C3 etc) and when considering versions of those, there are in total 45 different options to select one single computational resource (AWS, 2016). However the cost estimation capabilities are limited as they do not provide the users the ability to relate the estimated cost to application-

Jallow, A., Hellander, A. and Toor, S.
Cost-aware Application Development and Management using CLOUD-METRIC.
DOI: 10.5220/0006307505150522
In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 487-494
ISBN: 978-989-758-243-1

specific characteristics that may affect the actual cost of running an application. For large-scale distributed applications, and in particular for scientific applications, this issue is more prominent.

With these challenges in mind, we present CLOUD-METRIC: A lightweight framework that adds cost-awareness to the application development life cycle. Our aim is to provide a framework that helps in the design of cloud native applications by making cost considerations an integrated part of the development life cycle rather than an operational afterthought. Importantly, CLOUD-METRIC provides these cost-estimates both based on the provisioned resources and on an analysis of the *actual utilization* of the deployed resources.

The rest of the article is organized as follows: section 2 present related work. The functionality of the framework together with features are presented indetail in section 3. Section 4 describes the framework architecture. An evaluation of the framework is provided in section 5 where the utility of the framework is highlighted based on two challenging use cases. Section 6 illustrates the framework performance and section 7 presents concluding remarks.

2 RELATED WORK

Several efforts have been made to address the challenges of metering and cost estimation in cloud environments. Liew at al. in (Liew and Su, 2012) used a queuing model to predict the cloud computing resources used by a targeted application and based on that it estimates the deployment cost. The cost is estimated considering the applications' resource costs and services costs. The performance requirements of the application is predicted using defined policies. In contrast, CLOUD-METRIC relies on performance monitoring data for an application running in private resources.

A related solution to our work is proposed by Huihong He et al. in (He et al., 2012) in which they propose an approach to estimate the cost of running an application on AWS during the design phase. They model the application execution service with an UML activity diagram. The UML activity is extracted automatically with a proposed extraction algorithm. In addition, they propose a cost model to estimate the operational cost and the performance need of an algorithm during the design phase in order to produce a suitable deployment.

Recently Cloudorado (Clo, 2016) deployed a web application that provides cost comparisons for cloud servers across different cloud service providers including Google Cloud, AWS, Microsoft Azure and Cloudware. They provide an interface that allows users to specify server capacity requirements and the application provides matching servers on several cloud providers together with cost estimates. While helpful to users to compare cost for virtual private servers, it does not provide dynamic pricing based on users' application resource utilization.

Microsoft, being one of the major cloud services providers, have developed a tool that helps IT managers to quickly assess the running cost of an existing on-premise workload on the Azure cloud environment (Mic, 2016). The tool performs a scan on the existing on-premise workload and recommends matching instances on Azure. It also provides a monthly cost of the matching instances on the Azure environment. However, the tool is limited to Microsoft and VMware technologies such as Hyper-V, SCVMM, vCenter and ESXi. In our proposed framework, we perform an instance matching routine similar to Azures' matching routine but we match instances provided by different providers such at AWS and GCP.

None of above mentioned works address all the challenges we have highlighted in section 1. The framework presented in (Liew and Su, 2012) can addresses C-1, whereas the UML-based framework (He et al., 2012) provides a limited solution to C-2 and C-3. Cloudorado is a commercial project, offering advanced features and covers a range of IaaS providers. It addresses most of the highlighted challenges but the requirement of providing manual information seems unrealistic for applications with dynamic workloads. The Azure tool (Mic, 2016) addresses most of the challenges except C-4, since the solution is designed exclusively for the Azure platform, leaving the risk for vendor lock-in and a sub-optimal cost for running the applications.

3 CLOUD-METRIC

CLOUD-METRIC is a light-weight framework with the capabilities of metering computational and storage resources, cost estimation for clusters, both micro-(node based) and macro-(application-based, including all resources) level views of the execution platform, as well as recommendations for optimized resource type based on actual usage data.

CLOUD-METRIC currently provides cost-estimates for Amazon Web Services (AWS) and Google Cloud Platform (GCP) but it can easily be extended to different cloud service providers. The software and deployment guide is available via a Github repository (CLO, 2016). In order to enhance

the flexibility and ease of framework deployment, CLOUD-METRIC components are packaged using Docker containers (Merkel, 2014). The thesis report (Jallow, 2016) explains the complete technical details.

The cost estimation model is one of the core components of the framework. The model is subdivided into static and dynamic estimations. A static estimate is a one-to-one mapping: it provides the cost of acquiring resources on IaaS that are as similar as possible to the *allocated* local execution environment, including the number of CPUs, memory and disk size. Dynamic estimation on the other hand performs this matching based on the actual utilization level of the private resources. In the latter case, the framework uses utilization information together with the static parameters to recommend a potentially lower execution cost. For example, while the static estimation based on allocated resources might suggest an *xLarge* flavor of an instance, the actual usage pattern might show that the instance is under-utilized and the same performance could be attained by using a *Large* instance type, leading to a lower cost. Subsection 3.3 explains the process in detail. Here it is important to note that CLOUD-METRIC cannot be viewed as a comprehensive monitoring system as the focus is only on the parameters that have direct influence on the execution cost.

3.1 Cost Estimation

The cost estimation module is the principal component of the framework. It estimates the cost of running an application on AWS and GCP platform. The cost estimation consist of two algorithms: The matching algorithm which maps the static information of nodes to the closest matching instances on AWS EC2 and GCP CE, and the cost estimator algorithm which implements the following formulas to calculate a monthly cost estimate for AWS and GCP respectively:

$$Cost_{AWS-monthly} = Cost_{hourly} * T_{uptime} + Storage_{size} * Storage_{unitcost}$$

$$Cost_{GCP-monthly} = Cost_{hourly} * T_{uptime} * Discount_{su} + Storage_{unitcost} + OS_{unitcost} * T_{uptime}$$

$Cost_{*-monthly}$ is instance monthly cost,$Cost_{hourly}$ is hourly unit cost, T_{uptime} is uptime in hours per month, $Storage_{size}$ is the disk size, and $Storage_{unit}$ cost is unit cost of disk type per *GB* per month. In AWS, the cost includes the operating system cost as well.

$Discount_{su}$ is the Sustained Usage Discount (currently amounts to 0.70 for maximum monthly usage), $OS_{unitcost}$ is premium Operating System (OS) unit cost. In GCP, OS cost is not included in the machine type hourly unit cost. Thus it is separately added in the GCP cost estimate.

For high availability and economic reasons almost every IaaS provider offers services in different regions. CLOUD-METRIC uses the default settings *US* region for the GCP and *US-East-1* region for AWS resources. However the cost of running applications on other regions are also available. The framework computes only monthly estimates which conforms to the On-Demand (Pay-as-you-go) subscriptions on AWS and GCP. We used the *Regular VM* class[1] for monthly estimation on GCP CE instances and On-Demand instances on AWS EC2.

Cost estimates on Individual Nodes and on Clusters: CLOUD-METRIC estimates the cost of individual nodes in the cluster as well as the overall cluster cost. The overall cluster cost estimation is the sum of the individual instances' cost together with any offered discounts by the service providers. Our estimation component calculate cost for all regions on both AWS and GCP. The framework presents all this information in a user-friendly web interface. Figure 1 illustrates the various estimated costs of running the master node of the Hadoop cluster in different regions on the AWS platform.

Region	Cost Estimate per Month
US-EAST-1	$39.14
US-WEST-1	$50.94
US-WEST-2	$39.14
EU-WEST-1	$42.06
AP-SOUTHEAST-1	$59.58
AP-SOUTHEAST-2	$59.70
AP-NORTHEAST-1	$59.70
SA-EAST-1	$80.44
EU-CENTRAL-1	$45.08
US-GOV-WEST-1	$45.26

Figure 1: Master node on AWS regions.

AWS and GCP On-Demand Cost Estimates: we have used the closest possible matching instances from AWS and GCP. The comparison uses `c3.xlarge` from AWS and `N1-HighCPU-4` from GCP over several percentage of usage in a month.

The results of the cost estimation showed a large difference between the monthly cost. Google cloud offers automatic discount on the hourly charges of virtual machines for every additional minute of machine usage on top of the initial 25% usage in a month. This discount scheme is called Sustained Usage Discount (SUD) on GCP. AWS, on the other hand, charges a

[1]GCP provides two virtual machines (VM classes: Regular and Preemptible). Regular VM runs until terminated by user whereas Preemptible are short living VMs.

fixed hourly cost for each virtual machine for every hour of usage in a month. The framework incorporated these on-demand subscription discounts in its cost computation algorithm. Users can in the WebUI see how the hourly unit cost of instances changes over a monthly period of utilization.

3.2 Resource Monitoring and Metering

The monitoring aspect of the framework provides the ability to visualize the performance of each node in the cluster and the overall cluster. The parameters we monitor in each node are the ones that have direct effect on the cost. The monitoring data is used in the implementation of the recommendations, discussed in subsection 3.3.

Single Node Monitoring: For each node in the test environment based on Hadoop cluster, there is a resource monitoring process that sends resource usage data to the database in regular intervals.

Cluster Monitoring: The framework provides the functionality to visualize the overall performance of an entire cluster. In this case, the stored data of the individual nodes (mentioned in Single Node monitoring) are grouped and presented as an overall usage of resources. The monitoring charts display the aggregated percentage usage of the cluster.

The metering activity in the framework is carried out by the *resource miner* component. This component should run as a deamon on every single compute node in the application execution environment. The *resource miner* component acquires both static and dynamic information. Static information includes the hostname, operating system, number of CPUs, memory size, and disk size. For dynamic information, the *resource miner* component uses a Python module called psutil, a cross-platform library for retrieving system information such a CPU, network and memory usage.

3.3 Instance Recommendation

Apart from one-to-one mapping where the aim is to find the closest possible instances, the framework can also recommend instances with reduced cost without compromising on the overall application performance.

The algorithm for recommendations first query for average utilization of individual nodes and construct a hypothetical instance from the data retrieved. This constructed instance is then matched with similar instances on AWS and GCP as outlined in subsection 3.2. The details of the recommendation shown in Algorithm 1.

Algorithm 1: Recommendation algorithm.

Data: Resource Utilization Data, Pricing Data, Instances Types on AWS and GCP, Cluster
Output: Recommended Instances, EstimatedMonthlyCost of Recommended Instances

foreach *Node in Cluster* **do**
  get CPUSize, MemorySize, DiskSize, OS, AverageCPUUtilized and AverageMemoryUtlized
  estimate *CPUCount* and *Memory*
  $ExpVal = \log_2(CPUCount)$;
  if *ExpVal is not an integer* **then**
    $OptimizedCPU = 2^{\lceil ExpVal \rceil}$;
    $OptimizedMem = Memory$
  end
end
foreach *OptimizedCPU, OptimizedMemory* **do**
  get *MatchingInstances* on AWS and GCP
end
foreach *MatchingInstances* **do**
  Compute cost of instance in AWS and GCP
end

4 ARCHITECTURE

The architecture of the framework adopts a modular approach which leads to flexible deployment options. Each component of the system is independent and communicate with other components using well defined interfaces. The deployment model varies from a single-node setup for a small-scale application metering to multi-node deployment where multiple applications can use the framework simultaneously.

The framework is designed to be portable and efficient. The architecture consist of two main components: foreign and native components. The architecture employs a push-based model for external node registration. This process is carried out by the foreign components *resource miner* and *resource monitor* as shown in figure 2.

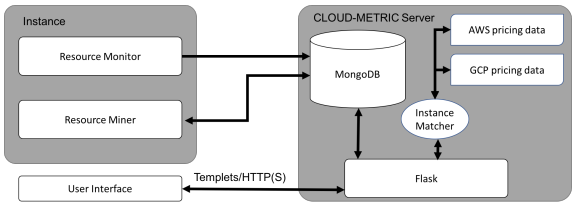

Figure 2: CLOUD-METRIC framework architecture.

Foreign Components are regarded as external because they are designed to execute on each node of the application execution environment. The components are lightweight python modules, carefully designed not to add significant load on the execution environment. The *resource miner* performs the metering activity and sends the static information of the appli-

cation setup to the framework. The *resource monitor* is responsible for the resource monitoring activity. It reports resource usage data in regular intervals, configurable according to the requirements.

Native Components of the framework consists of a database, the backend implementation, and the frontend web-interface written with Flask (fla, 2016). All native components are implemented using Python, and external dependencies are limited to Flask, MongoDB and Pymongo[2]. We chose to use a NoSQL solution to ensure scalability for potentially very large deployments. MongoDB is a lightweight document store, does not require a strict database schema, works well for horizontal scaling and is available as a production quality software.

The database consist of three collections, the first stores clusters' information, the second stores the metered data and the third stores the resource usage data. By default the monitored data collection is capped to store a maximum of 2 day of monitored data for 20 nodes in an environment. This can be modified depending on the application's requirements. The database design supports multiple application development environments. This allows experts to compare different strategies with different versions of their applications. Another key module in the system is the IaaS provided price lists. The pricing data of both GCP and AWS is extracted from (gcp, 2016) and (aws, 2016) respectively. The framework uses a *JSON* format for the pricing data list. GCP pricing list is available for developers in *JSON* format. However, AWS pricing list is not readily available. It is first extracted as *CSV* format and than converted to *JSON*. In order to validate the framework's estimates, we have also developed a testing module called *CM-estimator*. Results in section 5 will show the accuracy of our estimates in comparison with the GCP Price Calculator (PC) and AWS Simple Monthly Calculator (SMC).

5 FRAMEWORK EVALUATION

5.1 Cost Estimation

The viability of CLOUD-METRIC framework depends on the accuracy of the cost estimates. *CM-Estimator* is the component responsible for cost estimation in the framework. It currently supports AWS and GCP IaaS and exposes well-defined interfaces to add support for other IaaS providers. Figure 3 presents the accuracy of the provided cost estimates by comparing the cost calculated by *CM-Estimator* with the IaaS providers' native calculators.

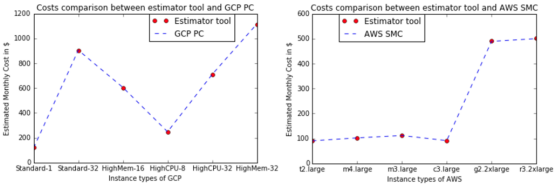

Figure 3: Cost comparison between CM-Estimator, AWS-SMC and GCP-PC Calculators.

Here it is important to note that AWS and GCP uses different pricing strategies. GCP has a Sustained Usage Discount (SUD) both for the compute and storage whereas AWS offers flat rates with fixed hourly price. Cloud-METRIC incorporates all these details to provide the closest possible estimations for a variety of resources.

5.2 Use cases

To evaluate the strengths of CLOUD-METRIC, we have presented two use cases based on the horizontally scalable execution environments. The use cases cover both execution platform for an already established application (use case-1) and the support of CLOUD-METRIC framework in the process of application development.

For both presented use-cases we have used SNIC Science Cloud (SSC) (ssc, 2016), an OpenStack based community cloud solution for Swedish academia. SSC is a national-scale cloud with the focus on providing IaaS.

Use case-1: A small Hadoop cluster. For this use case we setup a small Hadoop cluster with default settings. Once CLOUD-METRIC components (Foreign Components) are deployed on the machines in the cluster, it first performs a static one-to-one mapping of the resources to the public IaaS, i.e. it finds the closest matching resource flavors available on AWS and GCP. In this case, each node of the local Hadoop cluster matched the `c4.2xlarge` instance on AWS EC2 and `N1-HighCPU-8` on GCP CE. Table1 (first part) presents the cost estimation of the static one-to-one mapping. This gives the user of the private IaaS information on what the cost would be to move the setup to public IaaS. The calculated cost for GCP includes the offered discounts.

Further we consider two execution scenarios to highlight the framework's strength to recommend cost effective execution environments. In the first scenario the resources are under-utilized whereas in the second scenario, resources are occupied with a very high workload. In the first scenario, after monitoring the

[2]Pymongo is a Python distribution containing tools for working with MongoDB

local environment for a certain duration, based on the actual usage the CLOUD-METRIC framework recommended `c3.xlarge` and `N1-HighCPU-4` for each node on AWS and GCP respectively. By matching resources in the public IaaS based on the observed actual usage the framework helps reducing the cost over a naive static mapping without compromising the performance of the application. In this case, the recommended resource flavors are almost half of the price compare to the resource flavors recommended based on the static initial mapping. Table 1 (second part) presents the individual and the total execution cost of running the Hadoop cluster. With this test we would only like to compare the cost of an IaaS before and after adding the resource utilization metrics. The calculated price of 629.22$ for AWS resources is based on flat pricing model whereas GCP calculated cost of 324.18$ is based on Sustained Usage Discount pricing model. In case of GCP the prices fluctuate depending on the resource usage but for AWS the prices are fixed.

Table 1: Price variation for Hadoop cluster based on the workload. Cost presents the monthly charges on AWS and GCP in US Dollars. The reduced and increased costs are based on both static and dynamic information.

AWS	Cost $	GCP	Cost $
One-to-One mapping (Static information)			
C4.2XLARGE	314.61	N1-HighCPU-8	162.09
C4.2XLARGE	314.61	N1-HighCPU-8	162.09
Total Estimate	629.22		324.18
Reduced cost with under utilized resources			
C3.XLARGE	161.62	N1-HIGHCPU-4	84.20
C3.XLARGE	161.62	N1-HIGHCPU-4	84.20
Total Estimate	323.24		168.41
Increased cost with full resource utilization			
C4.2XLARGE	314.61	N1-HIGHCPU-8	162.09
C4.2XLARGE	314.61	N1-HIGHCPU-8	162.09
Total Estimate	629.22		324.18

In the second scenario we created an extensive workload on the local execution environment. Since the resources now become completely occupied, CLOUD-METRIC updated the recommendations. Table 1 (third part) illustrates the increase in the cost (compare to the second scenario) as the resources are fully utilized. Here, we would expect the framework to recommend the same deployment strategy as for the static mapping (when it assumes 100% resource utilization) since the aim is not to compromise on the performance of the application. This behavior is confirmed in the table.

While use case-1 highlighted the utility of CLOUD-METRIC in planning the migration of an already established application to public resources, the second use case shows how the service can be used to support development of interactive parallel computing applications.

Use case-2: Guiding interactive parallel computing with IPython Parallel MOLNs (mol, 2016) is an orchestration software that creates dynamically scalable clusters configured to run parallel computational experiments with the systems biology simulation software PyURDME (pyu, 2016). MOLNs can configure and provision virtual clusters in OpenStack IaaS or Amazon EC2.

To enable a better understanding of the performance trade-offs in different parts of commonly executed parallel computations, to potentially suggest improvement of the implementation and to highlight cost-considerations we enabled the MOLNs orchestration tools with CLOUD-METRIC. We considered two computationally interesting and different scenarios for a MOLNs cluster comprising of 6 nodes and a total of 94 vCPUs deployed in AWS: (a) We conducted a Monte Carlo experiment where the PyURDME application is invoked to generate simulation output in the form of spatio-temporal data. A large number of independent realizations are generated (10^4) and written to a SSHFS file system on the virtual cluster. In the next phase, (b) we read all this data in parallel and compute a summary statistic. In both cases, we use CLOUD-METRIC to study the aggregated resource usage of the entire cluster and the cost aspect of our computations.

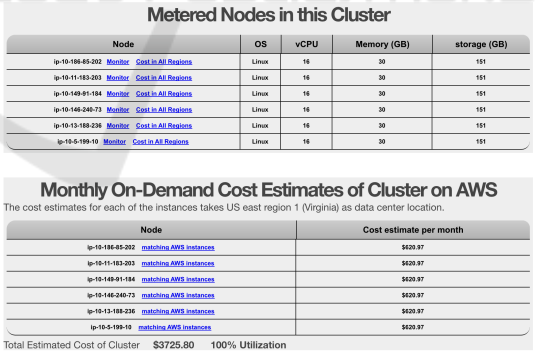

Figure 4: Static cost for MOLNs cluster in Amazon EC2.

Fig. 4 shows the projected cost of the cluster for one month of sustained allocation in AWS. As can be seen, the cost is $3725. For a typical research group this is substantial and it highlights the importance of adopting a dynamic resource usage model when moving scientific applications to the cloud. Since in this case the entire computation completed in about 2 minutes, by automating the provisioning and tear down of the cluster MOLNs succeeds in helping scientists leverages the pay-as-you go model of public IaaS.

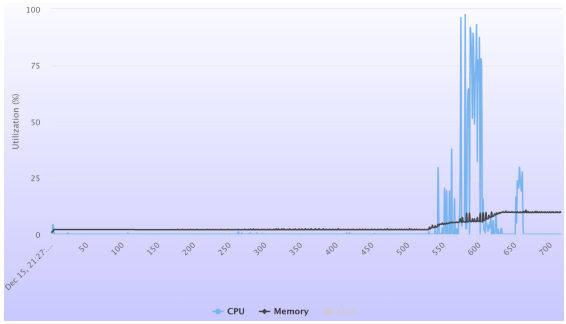

Figure 5: Aggregated CPU and memory usage over the entire cluster during a computational experiment.

However, CLOUD-METRIC lets us draw some more conclusions about our setup. Fig 5 shows the total aggregated CPU and RAM usage over the entire cluster during the computational phase (first peak) and the data processing phase (second, smaller peak). As can be seen, although MOLNs manages to make use of a large fraction of the cluster resources for phase (a) it is not optimal and the CPU utilization fluctuates quite heavily, suggesting idle phases. The reason for this could be attributed to e.g. bottlenecks in writing result to disk or delays in the start of new task by the broker. More detailed application level investigations would be needed to determine the root cause, but CLOUD-METRIC would immediately provide a view on whether any code changes would improve the utilization level. We also see that in the second phase (b) we do not come close to leveraging the total compute capacity of the cluster. This suggests that we should consider either refining the code to use fewer workers during this phase (allowing them to do other work) or seek to improve the data read performance.

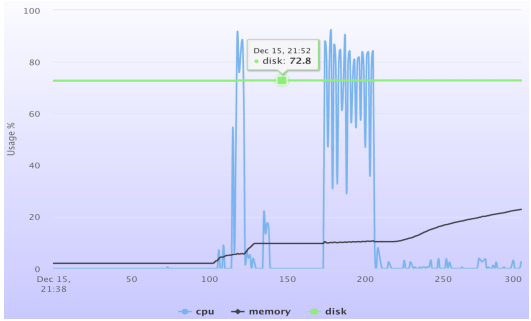

Figure 6: Resource usage for one of the six worker nodes in the MOLNs cluster.

Finally, zooming in one the detailed usage of an individual worker node in Fig. 6 reveals some interesting information. Here we executed a second pass of phase (a) for an even larger number of simulations (5×10^4). This increases the chunk size (amount of work given to each worker). As can be seen, it does not improve the fluctuations in CPU utilization even on a single node, so this makes it more plausible that it is caused by delays in staging results to storage.

6 PERFORMANCE EVALUATION

In this section the focus is on the resource usage and the overhead imposed on the analyzed environment by CLOUD-METRIC itself. For the results presented in this section, we have used application settings described in use case-1 in subsection 5.2.

Foreign Components: *resource miner* and *resources monitor* are the components running on each node in the application's execution environment. The frequency of the reports sent by the *resources monitor* can be tuned depending on the application's requirements but to illustrate a realistic scenario the presented results are based on regular reporting with 60 seconds interval. The cumulative overheads created

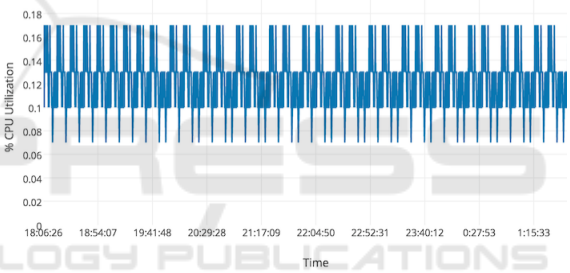

Figure 7: Resources *miner* and *monitor* usage on Hadoop cluster master node.

by these components are presented in figure 7. The results shows the percentage usage of less than 0.5% over a period of 8 hours. The memory usage was consistently under 2MB. The usage pattern affirms that these components are lightweight and the required tasks can be achieved with minimal resource usage.

Native Components: The native framework consist of a database component, web front-end, cost estimation and recommendation components. The preferable deployment model is to run the native components of the CLOD-METRIC on a dedicated node. In our use cases, it is deployed on a separate VM in the same OpenStack tenant as the analyzed application environment. We have evaluated the CPU, memory, network and disk utilization on the node hosting the framework. For database we have used MongoDB with standard settings. MongoDB has already been well tested for different deployments models (Nyati et al., 2013),(Parker et al., 2013).

Figures 8 and 9 present CPU and network consumption of the framework for a fixed window of 8

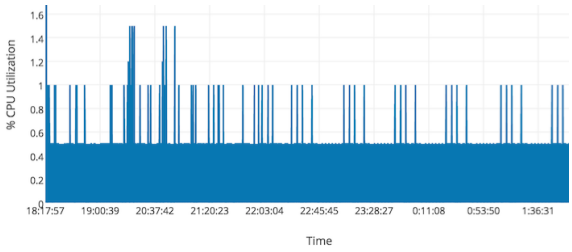

Figure 8: CPU utilization.

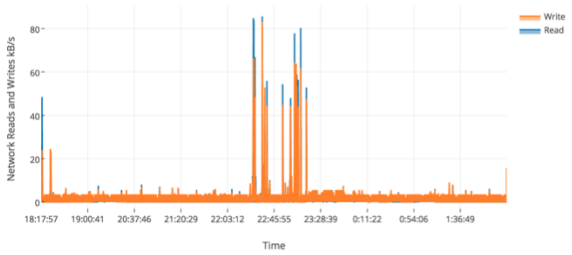

Figure 9: Network consumption while monitoring Hadoop cluster.

hours. Throughout the experiment the CPU utilization was less than 2% and the active memory utilized by the Python processes was under 300 KB/s. For these measurements we have monitored the Hadoop cluster discussed in use case-1. The low usage of resources show that the framework is stable and expected to manage and meter multiple execution environments simultaneously.

7 CONCLUSION

With CLOUD-METRIC we have presented a framework that eases the transition of applications from a dedicated or private virtual execution environment to public cloud infrastructures. CLOUD-METRIC is a lightweight and easy-to-use framework that can manage multiple execution environments simultaneously. We have also demonstrated how framework can support the development phase of complex applications.

ACKNOWLEDGMENT

This work was supported by the Swedish strategic research programme eSSENCE(ess, 2016) and Göran Gustafsson Foundation(ggf, 2016). The development and testing of CLOUD-METRIC were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) (sni, 2016) at Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX) (upp, 2016).

REFERENCES

(2016). Aws instance type. https://aws.amazon.com/ec2/instance-types/.

(2016). Aws price calculator. http://a0.awsstatic.com/pricing/1/ec2/.

(2016). Cloudorado. https://www.cloudorado.com/.

(2016). essence: The e-science collaboration. http://essenceofescience.se/.

(2016). Flask. http://flask.pocoo.org/.

(2016). Github repository for cloud-metric. https://github.com/ajallow07/Cloud-Metric.

(2016). Google price calculator. https://cloud.google.com/products/calculator/.

(2016). Goran gustafssons stiftelser. http://gustafssonsstiftelser.se/.

(2016). Microsoft azure cost calculator. https://azure.microsoft.com/en-us/pricing/calculator/.

(2016). Molns. https://github.com/MOLNs/molns.

(2016). pyurdme. www.pyurdme.org.

(2016). Snic science cloud. https://cloud.snic.se.

(2016). Swedish national infrastructure for computing (snic). http://www.snic.vr.se/.

(2016). Uppsala multidisciplinary center for advanced computational science. http://www.uppmax.uu.se/.

He, H., Ma, Z., Li, X., Chen, H., and Shao, W. (2012). An approach to estimating cost of running cloud applications based on aws. In *2012 19th Asia-Pacific Software Engineering Conference*, volume 1, pages 571–576.

Jallow, A. (2016). Cloud-metric: A cost effective application development framework for cloud infrastructures. Master's thesis, Uppsala University, Department of Information Technology.

Liew, S. H. and Su, Y. Y. (2012). Cloudguide: Helping users estimate cloud deployment cost and performance for legacy web applications. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 90–98.

Merkel, D. (2014). Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239).

Nyati, S. S., Pawar, S., and Ingle, R. (2013). Performance evaluation of unstructured nosql data over distributed framework. In *Advances in Computing, Communications and Informatics (ICACCI), 2013 International Conference on*, pages 1623–1627.

Parker, Z., Poe, S., and Vrbsky, S. V. (2013). Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 5:1–5:6, New York, NY, USA. ACM.