A Runtime Cloud Efficiency Software Quality Metric

Mark Shtern
York University
Toronto, Canada
mark@cse.yorku.ca

Michael Smit
Dalhousie University
Halifax, Canada
mike.smit@dal.ca

Bradley Simmons
York University
Toronto, Canada
bsimmons@yorku.ca

Marin Litoiu
York University
Toronto, Canada
mlitoiu@yorku.ca

ABSTRACT

This paper introduces the Cloud Efficiency (CE) metric, a novel runtime metric which assesses how effectively an application uses software-defined infrastructure. The CE metric is computed as the ratio of two functions: i) a benefit function which captures the current set of benefits derived from the application, and ii) a cost function which describes the current charges incurred by the application's resources. We motivate the need for the CE metric, describe in further detail how to compute it, and present experimental results demonstrating its calculation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement, Management

Keywords

software metrics, cost-benefit analysis, resource-based attacks, adaptive systems, cloud computing

1. INTRODUCTION

Runtime software quality metrics assess the quality of a software application on the basis of measured attributes during execution. Non-functional qualities are commonly measured, including profiling and memory management analysis [3], performance (e.g. throughput, response time), security, or power consumption [5]. Runtime quality metrics are used, among others, by adaptive systems to inform decisions on when or how to increase quality [2].

This paper describes a novel runtime metric, the *cloud efficiency* (CE) metric, representing a cost-benefit analysis that compares the current benefit derived from an application to the current cost of running that application on software-defined infrastructure (SDI). SDI, typified by cloud computing in the form of infrastructure-as-a-service (IaaS), responds flexibly and dynamically to changing workload, adding and removing resources, typically in a pay-as-you-go pricing model. Cloud providers have stressed the importance of building software using "cost-aware architectures" that take ownership of managing the cost of this dynamically changing infrastructure [15]. However, current information about the cost of IaaS resources is not typically available from the provider, and this dynamism has changed the cost of delivering software from a mostly-constant or linearly-increasing value to a highly dynamic value that is difficult to accurately measure and assess. The benefit of an application is typically something a mature organization is capable of measuring, for example direct revenue (e.g. e-commerce site), ad-based revenue (e.g. news site), or business process enhancement (e.g. internal sales management site). This metric is described in detail in §3.

The CE metric is a business-driven metric that can be leveraged to detect various resource-consumption attacks on applications, including a new form of attack we identify, the *auto-immune resource attack*, where self-inflicted unnecessary cloud costs are incurred (§3.1). It can also be used to manage cloud resources effectively: e.g. informing adaptive systems, addressing the variability of cloud resources, and comparing competing cloud service providers (§3.2).

To demonstrate the feasibility and applicability of this metric, we present an experiment in Section 4. We show the CE metric is capable of providing valuable information that other metrics do not capture in a realistic scenario. We discuss the future directions for this project in Section 5.

2. BACKGROUND AND RELATED WORK

Cost-benefit analysis is well established in marketing literature. For example, Zeithaml [16] defines *perceived value* as "the consumer's overall assessment of the utility of a product based on perceptions of what is received and what is given." This examination of trade-offs is precisely the cost-benefit analysis captured by the CE metric. Decisions about value are often guided by single attributes (e.g. judging speakers by size, or computer power by processor speed) even when additional information is available [16]. The CE metric captures complex information in an expressive metric.

Software metrics have long been used to assess software, with the goal of assessing progress or outcomes (or predicting progress or outcomes), based on analysis of processes, artifacts produced during development, and resources (e.g.,

[1][4]). The idea of cost-effectiveness metrics and cost-benefit analysis has been applied in a number of domains. For example, Jogalekar and Woodside [8] used cost-effective-ness to assess the scalability of distributed systems. Kondo et al. [9] assessed the costs and benefits of the cloud for several sample application, measured only by relative performance (a common definition). Our approach is novel in that cost-benefit is defined per-application, it is applied to a domain where it has not been used in this fashion, it uses a broader definition of cost and benefit, and it produces a single high-level metric suitable for non-technical managers.

3. THE CLOUD EFFICIENCY METRIC

The CE metric is the ratio of a benefit function**:**cost function, where both functions update as frequently as possible. For practical use, the ratio may be replaced by other relationships such as difference or statistical correlation. Each function captures a value for either a single application, an application component, or a set of applications. There is therefore no universal threshold of good and bad efficiency, although CE can be used for comparing different applications if consistent cost and benefit functions are used. The complexity of this metric lies in accurately defining the cost function and the benefit function; the calculation of the metric will be illustrated by a series of illustrative examples.

The **cost function** must capture the real cost of offering a given application on the cloud for the given period (e.g., per hour). This function excludes other costs related to the application, e.g. the cost of development, the cost of goods sold, and customer support. It is not a measure of overall profitability; rather, it captures current infrastructure costs.

Capturing this cost information is challenging, as regular cloud users can typically relate anecdotes about how they thought they were closely tracking costs, yet somehow amassed several hundred dollars in bandwidth or storage charges without noticing. Per-service runtime tracking of SDI costs is typically not supported via provider APIs. Many providers bill for the allocation of compute resources (VMs), but also charge based on consumption for bandwidth (with varying rates per network), storage (in various mechanisms), storage retrieval, monitoring, IP addresses, and then a suite of value-added services. Prices are reported per cloud account, not per application, and the cause of increased costs are not readily *traceable* back to the cause. A variety of pricing models and billing periods are used, particularly across providers in multi-cloud environments.

The result is a cost function that expresses a sum of the products of the consumption of each available cloud resource and the price for that resource, where neither the consumption nor the price are available directly from web APIs. The information must be calculated based on metrics acquired from the provider and by instrumenting the infrastructure.

The **benefit function** must capture the benefit the application provides to the organization. Typically organizations have mechanisms for assessing this benefit at least at the macro-level. The benefits may come from many sources: revenue, advertising, brand awareness, customer satisfaction, number of repeat customers, or any number of business-specific metrics. In the case of an e-commerce web application, this could capture information about earnings, click-through-rates, and costs-per-click from the Google Adsense API; transaction information from PayPal; or visitor information from Google Analytics.

The **collection of lower-level metrics** to inform the two functions requires mechanisms for distributed, scalable collection of heterogeneous metrics that can then be aggregated and manipulated at runtime, pairing consumption metrics with prices. Complex event processing is one approach to managing these metrics; previous work in [14] describes a distributed, scalable, complex-event-processing-based monitoring infrastructure for acquiring and aggregating metrics from multiple cloud providers.

The cloud efficiency metric can be augmented with the ability to acquire metadata about instances and price from a cloud metadata service [13], information about running instances from a variety of providers, and resource consumption information from monitoring sources like Ganglia and CloudWatch, and aggregate these metrics into a stream of cost-related metrics (cost per resources, total per provider, total per resource type, etc.). It is this the novel ability to track cost so precisely at runtime that enables the CE metric.

3.1 Example: Resource-consumption Attacks

Although there are benefits to dynamic software-defined infrastructure with pay-as-you-go billing, one disadvantage is the possibility of attacks designed to increase the costs of an application's SDI without a corresponding increase in the benefit of the application. When phrased this way, it is apparent the CE metric can serve as an able indicator of various attacks that can elude common security measures. The CE metric focuses on high-level business concepts (e.g., cost, benefit) that are orthogonal to the standard security measures that are focused on technical approaches (e.g., deep packet inspection, behavioural and statistical analyses). It enables threat assessment on the basis of cost-benefit analysis: rather then suggesting the presence of malicious traffic, it identifies the presence of unprofitable traffic. Defining a range of tolerable CE values is tantamount to expressing risk tolerance, defining an acceptable level of reduced efficiency before mitigation will begin. In the short term, a system might temporarily halt the automated acquisition of additional cloud resources while notifying an administrator to investigate; the investigation is made possible by the traceability of the cost function.

Various attacks can be effectively detected using a CE metric with appropriate definitions of cost and benefit functions; four examples are named below. Each attack is characterized by a disproportionate increase in cost as resources are consumed without providing benefit to the organization, resulting in a decline in the CE metric.

We define here a **autoimmune resource (AIR) attack**, a novel "attack" on cloud resources, where the user through carelessness or error incurs unnecessary charges on their own resources. There are reported cases of users increasing their Amazon EC2 bill (in at least one case, by an order of magnitude [7]) due to oversights, misconfigurations and/or misunderstanding of the cloud provider's pricing model. Specific examples of such oversights include configuring high-traffic internal application connections across regions or data centers; abandoning object store storage buckets; stopping instances without terminating them; downloading data at high rate from long-term cloud storage services; and manually terminating instances that are part of an auto-scale deployment. At present, the AIR attack is not detected by any security mitigation tools (i.e, firewalls, intrusion detection systems, etc.) as the attack comes within one's own infrastructure.

A **denial of service** (DoS) attack increases resource utilization without a corresponding increase in revenue. Using only throughput, utilization, response time, or page impressions to monitor an application will result in the addition of resources until the attack ends or resources sufficient to respond to the spurious requests are acquired (at potentially significant expense).

In **cost-of-service** (CoS) attacks, the goal is to increase the cost of a cloud deployment without necessarily denying service (as described by Idziorek and Tannian [6]). Even without auto-scaling, the outgoing cost of bandwidth can accrue rapidly at the transfer rate available. Standard protection mechanisms do not mitigate against this attack.

A **low-and-slow DoS** attack [12], also referred to as a low-bandwidth application-layer DoS attack, is predicated on detailed knowledge of the target application. The attacker is able to slowly degrade the performance of the application by exploiting weaknesses they have identified. Traditional defence mechanisms are often unable to protect against these messages due to the absence of malicious content. In many cases, the only way to detect the occurrence of this type of attack is to have intimate knowledge (i.e., a comprehensive model) of the application and content of the messages so as to be able to detect the malicious intent at the level of application usage pattern. Developing these models is a complex and challenging process.

3.2 Example: Cloud Resource Management

Despite the potential advantages of software-defined infrastructures, there are challenges in managing resources in a cost-effective manner. The ways in which the acquisition of VMs differs from the commonly-used utility analogy (such as the lack of standardization, variability of supply, and the need to manage capacity at a fine-grained level) poses challenges that can be addressed using the CE metric.

A key feature of SDI is the ability for an adaptive manager to autonomously self-configure, self-heal, and self-manage; to do so, they typically optimize a utility function, and require information about the application and its environment [2]. The efficiency metric can be used to assess the environment, and in combination with other metrics to identify healing actions required. Configuration decisions can be made to proactively maximize CE.

Resources a cloud provider deems identical may have performance variations, by as much as 40% [11]. When allocated resources do not meet expectations, an adaptive system's response is to acquire more resources. This increases cost to the user and revenue for the provider, even when the shortfall is the responsibility of the provider. Service levels are difficult to detect and enforce, and may not even be guaranteed. The CE metric is a more robust measure in such variable environments, as it is not tied to performance.

There is no cross-provider measure of resource capability (e.g., Google measures CPU allocations using GCEUs, while Amazon uses ECUs). In contrast, electricity consumption is measured using SI standard units (watts, amperage). CE provides a standard, cloud-agnostic measure that can compare application deployments on two different providers. For example, although Provider A's instances may cost more, they may perform better over time and allow greater revenue with fewer resources for the given application. This decision can be made at run-time in a multi-cloud environment using a cloud broker [10].

4. CASE STUDY

To demonstrate the use of this metric, we defined the CE metric for a web application, deployed it to Amazon EC2, and observed the CE metric (in contrast to other runtime metrics) during a denial-of-service attack. The web application emulates an e-commerce bookstore built using Java EE, and has several use case scenarios: adding items to a shopping cart, checking out a shopping cart, user registration, and browsing products.

We deployed the bookstore using the Amazon Elastic Beanstalk service. We configured an Amazon Elastic Load Balancer (ELB) with one application server to start, with auto-scaling up triggered if CPU Utilization exceeded 70% for one minute, and scaling down if CPU utilization remained under 50% for one minute, with a minimum delay of 6 minutes between scaling events (this configuration is based on Amazon defaults). The application servers connected to a common database running on Amazon RDS (a Micro DB instance, mySQL 5.5, 630MB RAM, bursting to 2 ECUs).

We defined the **benefit function** based on the number of transactions that represented successful checkouts. The **cost function** is substantially more complex, and involves a variety of low-level workarounds to accurately measure cost details not provided by Amazon[1]. As discussed in §3, there are many factors in calculating cost, and they are not made available by the provider. We implemented a cost aggregator using our monitoring solution [14], where we acquired consumption data by requesting the running instances from EC2, and for each calculating storage use and IO read/writes. To capture network data, each VM was configured with an agent that used `iptables` counters to measure traffic to private networks (assumed to be \$.01/GB though in some cases it is free) and to public networks (\$.12/GB) separately; although the price differs by an order of magnitude, per-instance traffic charges are not available through the Amazon API. We also calculated charges based on optional additional services, such as detailed monitoring metrics captured by Amazon, guaranteed IO levels, additional public IP addresses, and elastic load balancing. Our system can distinguish among the Amazon pricing models (on-demand pay-as-you-go instances were used in this case study, but also available are reserved instances with an upfront fee and a reduced hourly rate, and spot instances which have variable pricing as users bid on spare capacity). Services other than compute VMs were also calculated, including the DB, IO operations on the DB, the storage of the WAR file on Amazon S3, and data throughput for the ELB. The consumption data was paired with pricing data [13], to produce a total cost per hour that was updated each time new data was received (several times per second).

We monitored the CE metric, its individual cost and benefit components, and the total number of page impressions. The latter metric is commonly used as a measure of a site's popularity and a proxy for a value function. We generated a standard traffic workload using a typical mix of scenarios (browse, add to cart, checkout, register) for the first half of the experiment, then initiated an escalating denial-of-service attack on the application. The values of each of the four tracked metrics over time are shown in Figures 1 and 2.

[1]The API offers estimates of the current monthly bill, and the billing site offers the ability to download spreadsheets listing services and charges after they occur, but "live" information about current charges is not available.

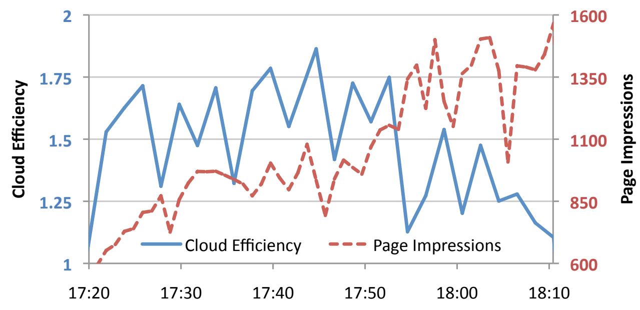

Figure 1: Comparing the CE metric to an alternate runtime metric, page impressions.

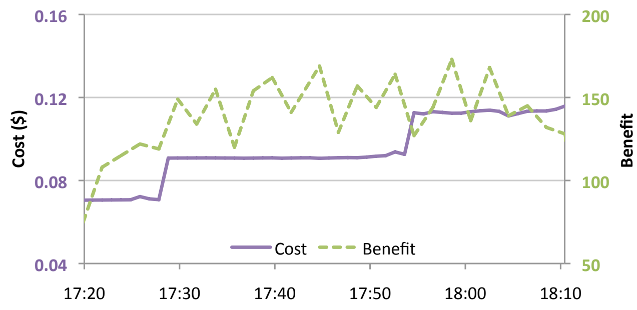

Figure 2: The cost and benefit functions that comprise the CE.

Discussion: In the first few minutes, an instance is added to meet demand, after which the number of resources (and the cost) stabilize under normal conditions. CE is maintained as the number of requests handled (and the value) increases. After this, the value remains roughly constant throughout the experiment. When the DoS attack begins around 17:45, we see an increase in the page impressions, followed by the addition of a third instance. The addition of the third instance to handle the DoS attack reduces the cost efficiency metric, as no additional value is added despite the increase in capacity. The Page Impressions metric shows high traffic, a positive indicator, which is misleading.

5. CONCLUSION AND FUTURE WORK

This paper introduced the CE metric, a runtime quality metric expressing a cost-benefit analysis of applications deployed on software-defined infrastructure, suitable for use to mitigate resource-based attacks and to more effectively manage SDI. Results were presented for an application running on EC2 in which the CE metric indicated the presence of a resource-based attack. The next steps of this project include exploring an adaptive manager to analyze and plan based on CE(e.g when and how to mitigate a slow DoS attack), exploring more complex cases with a focus on the design of comprehensive and expressive functions, and investigating root-cause analysis based on cloud efficiency metric.

Acknowledgment

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Ontario Centres of Excellence (OCE).

6. REFERENCES

[1] B. Boehm, C. Abts, A. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. Cost estimation with COCOMO II. *Upper Saddle River, NJ: Prentice-Hall*, 2000.

[2] Y. Brun, R. Desmarais, K. Geihs, M. Litoiu, A. Lopes, M. Shaw, and M. Smit. A design space for self-adaptive systems. In *Software Engineering for Self-adaptive Systems II*, pages 33–50. 2013.

[3] W. De Pauw, N. Mitchell, M. Robillard, G. Sevitsky, and H. Srinivasan. Drive-by analysis of running programs. In *Proc. ICSE Workshop of Software Visualization, ICSE Workshops*, 2001.

[4] N. Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions on Software Eng*, 20(3):199–206, 1994.

[5] A. Hindle. Green mining: investigating power consumption across versions. In *Proceedings of the 2012 International Conference on Software Engineering (ICSE)*, pages 1301–1304, 2012.

[6] J. Idziorek and M. Tannian. Exploiting cloud utility models for profit and ruin. In *IEEE International Conference on Cloud Computing*, pages 33–40, 2011.

[7] P. Ipeirotis. The Google attack: How I attacked myself using Google spreadsheets and ramped up a $1000 bandwidth bill. `http://www.behind-the-enemy-lines.com/2012/04/`, April 2012.

[8] P. Jogalekar and M. Woodside. Evaluating the scalability of distributed systems. *Parallel and Distributed Systems, IEEE Transactions on*, 11(6):589–603, 2000.

[9] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson. Cost-benefit analysis of cloud computing versus desktop grids. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 1–12, 2009.

[10] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski. Introducing STRATOS: A cloud broker service. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 891 –898, 2012.

[11] J. Schad, J. Dittrich, and J.-A. Quiane-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, 3(1), 2010.

[12] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, and V. Theodorou. Towards mitigation of low and slow application ddos attacks. In *IEEE International Workshop on Software Defined Systems, International Conference on Cloud Engineering*, 2014.

[13] M. Smit, P. Pawluk, B. Simmons, and M. Litoiu. A web service for cloud metadata. In *IEEE Congress on Services*, pages 24–31, Los Alamitos, CA, USA, 2012.

[14] M. Smit, B. Simmons, and M. Litoiu. Distributed, application-level monitoring of heterogeneous clouds using stream processing. *Future Generation Computer Systems*, 29(8):2103–2114, 2013.

[15] W. Vogels. Day 2 keynote. Presentation at re:Invent, Amazon Web Services, 2012.

[16] V. Zeithaml. Consumer perceptions of price, quality, and value: a means-end model and synthesis of evidence. *The Journal of Marketing*, 52(3):2–22, 1988.