Verifiable Resource Accounting for Cloud Computing Services

Vyas Sekar
Intel Labs

Petros Maniatis
Intel Labs

ABSTRACT

Cloud computing offers users the potential to reduce operating and capital expenses by leveraging the amortization benefits offered by large, managed infrastructures. However, the black-box and dynamic nature of the cloud infrastructure makes it difficult for them to reason about the expenses that their applications incur. At the same time, the profitability of cloud providers depends on their ability to multiplex several customer applications to maintain high utilization levels. However, this multiplexing may cause providers to incorrectly attribute resource consumption to customers or implicitly bear additional costs thereby reducing their cost-effectiveness. Our position in this paper is that for cloud computing as a paradigm to be sustainable in the long term, we need a systematic approach for *verifiable resource accounting*. Verifiability here means that cloud customers can be assured that (a) their applications indeed physically consumed the resources they were charged for and (b) that this consumption was justified based on an agreed policy. As a first step toward this vision, in this paper we articulate the challenges and opportunities for realizing such a framework.

Categories and Subject Descriptors

B.8 [**Hardware**]: Performance and Reliability; C.4 [**Performance of Systems**]: [measurement techniques]

General Terms

Measurement, Reliability, Security, Verification

Keywords

Cloud computing, Accounting, Metering, Resource auditing

1. INTRODUCTION

Computing as a service is seeing a phenomenal growth in recent years. The primary motivation for this shift is the promise of reduced operating and capital expenses, and the ease of dynamically deploying and scaling new services without maintaining a dedicated compute infrastructure. With increased popularity and adoption, however, new and unforeseen challenges emerge.

A common problem that cloud customers face today is the inability to make sense of the cost footprint of their outsourced computation (e.g., [3,4,8,14,23,28,30]). Because customers have little or no visibility into the infrastructure, these charges may have no obvious direct connection to their application tasks. For example, when execution can be elastic in reaction to dynamic workloads, customers may face inexplicable charges in a pay-as-you-go billing model. This problem is further exacerbated in shared or "public" cloud infrastructures. Providers reduce capital and management costs by multiplexing several customers on the same infrastructure using hardware virtualization. While virtualization provides some degree of isolation between customers, there are many shared resources that cannot be perfectly isolated. This can result in unforeseen *externalities* that may inflate an application's resource footprint. For example, a misbehaving application may cause cache misses or network congestion and increase the computation footprint for other applications sharing the same physical platform.

At the same time, although providers do generate income with this business model, their profitability is not clear. (Many providers are reluctant to release accurate revenue-expense estimates [2].) We speculate that this arises at least in part because providers are still struggling to figure out how to precisely monitor and bill their customers' and their own resource consumption. Fine-grained monitoring of virtualized computations is a difficult proposition; it becomes only harder when it must also be defensible enough to put on an invoice. As a result, some resources that are difficult to monitor and attribute to client computations are not accounted for. For instance, I/O time [39] and internal network bandwidth [37] are not metered, even though these have a non-trivial impact on the provider's operating costs and the performance of other applications [32]. Similarly, sharing effects (e.g., memory pressure due to contention among co-scheduled jobs) cause costs that are both difficult to measure and challenging to causally attribute to their source. Consequently, providers either incorrectly account for these costs, passing the inaccuracy on to their customers, or implicitly bear the costs, thereby increasing their own operating expenses.

As the heady honeymoon phase of cloud computing wanes, cloud providers and customers need to fine-tune their business strategies to remain cost-effective [29]; "good enough" resource accounting and billing will no longer be good enough. Even as early as 2008, 61% of IT executives and CIOs rated the "pay only for what you use" as a very important perceived benefit of the cloud model [7] and more than 80% of respondents rated competitive pricing and offering performance assurances/SLAs as very important provider attributes [6].

Despite this early confirmation that resource usage and billing are top concerns for IT managers, these have received little or no attention from the industry or the research community. Our informal discussions with industry personnel and other researchers indicate that popular perception on this topic is often quite extreme. One view believes that this is a "non-problem" in that the technical means already exist and it is only a matter of time before market forces resolve it. The opposite view believes that this is *obviously* a critical problem, but we do not have the technical means to do so! Given that popular perception is polarized, and yet there is litle

CCSW'11, October 21, 2011, Chicago, Illinois, USA.

work in this context, our goal in this position paper is to stimulate an active discussion in this topic.

Our own position is that for cloud computing services to become successful and sustainable, we need a systematic framework for *verifiable resource accounting*. Such a framework will benefit both cloud customers and providers. It eases any concerns that customers may have with providers' pricing and performance guarantees. It also provides customers with a basis for accurately comparing different cloud providers. At the same time, having such a framework enables providers to faithfully capture their operating expenses by billing for resource expenditures that they do not currently account for and preventing customers from trying to game their billing procedures [43]. Furthermore, easing the pricing-performance concerns will encourage more cloud adoption and improve profitability by increasing the overall utilization of a provider's infrastructure.

Having taken this position, in the rest of this paper, we explore the technical challenges and opportunities for realizing such a framework in practice. Before doing so, in Section 2, we define the problem in the context of an abstract operating model with three-high level components: the cloud customer, the cloud provider, and a service called the *verifier*. With this setup, in Sections 3 and 4, we describe the challenges involved in two different aspects of verifiability. In each case, we sketch candidate solutions to address these challenges. We discuss some outstanding issues in Section 5 and related work in Section 6 before concluding in Section 7.

2. PROBLEM STATEMENT

Our goal is to provide verifiable resource accounting for cloud environments, where computation is being leased from others. To help formally define the problem, we introduce an abstract operating model shown in Figure 1. There are three logical participants: the customer C, the provider P, and the verifier V. At a high-level, C asks P to run the computation task T. Subsequently, P gives the C a *consumption report* R describing what resources it thinks C consumed. For example, a provider may report a time series of *consumption vectors*, whose elements correspond to CPU usage, memory bandwidth, memory size, I/O bandwidth, network bandwidth, and energy, aggregated over pre-determined time quanta, for the duration of the task. C takes this report R together with the task T and additional data (the roles of which will become clearer later in this section) to the verifier V and checks if R is a valid resource report for the T.

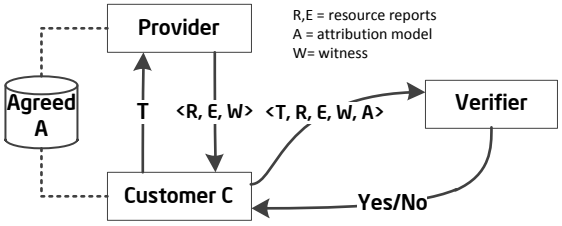

Figure 1: Conceptual Architecture.

Granularity: This determines the level of detail at which resources are tracked. On the coarse end, tracking might be done in units of core-hours for the duration of a task; on the fine end, tracking might be done in units of cycles used per second-long time period. In a sense, granularity defines the number and units of the components of a consumption vector. In the model, we capture granularity as a definition or *schema* of what consumption reports look like: over what quantum of time resource vectors are provided to the customer, and what elements each resource vector contains. For example, EC2 charges for CPU instance hours, storage size and requests, Internet bandwidth usage, and specialized services such as load balancing. Similarly, Google's AppEngine bills users for CPU Time, in/out network bandwidth, storage, and email requests.

Attribution: The *attribution* model determines how resource consumption is attributed to the owner of a task. This model represents the charging policy of a provider, and covers issues such as whether a task owner should be charged for resources used for task migration, how the cost of tasks thrashing is borne by task owners and the service provider, etc. To this end, we optionally extend the consumption report R with a separate report of indirect, *external* consumption E, similar to R, but attributed to task T due to external factors such as contention. Together, R and E make up what the provider is going to invoice customer C for task T.

In the model, we represent the attribution model A as a function that computes the values of R and E given a computation T under given conditions. Intuitively, the attribution model can be regarded as a "golden simulator" of the leased architecture (including its management software). This allows a customer to simulate ahead of time how a computation will consume given some inputs and environment conditions (including assumptions about other cotenant customers' workloads).

Verifiability: Verifiability aims to give the customer assurances about two questions:

1. *Did I* consume what I was charged?
2. *Should I* have consumed what I was charged?

The first question concerns itself with the veracity of the consumption vector. A provider should not be able to charge a customer with cycles it did not expend on her behalf. The second question concerns itself with the efficiency of the provider's infrastructure, with respect to scheduling and provisioning. If the provider conservatively—or erroneously—used 1 GByte of main memory for a task that only required 1 MByte, it should arguably not be able to pass on the cost of its inefficiency to its customer.

To aid in the verification process, the provider may optionally give C a *witness* W. We leave the specification of the W open depending on the type and level of verifiability desired by C. For example, the witness may simply be hardware attestations to guarantee the integrity of the reports. As discussed earlier, the customer has access to a logical *verifier*, an oracle that returns Yes/No answers given a consumption report R and E, the witness W, the task T, and the attribution model A. The simplest verifier might be one that uses W as a sanity check to identify gross tampering with R and E. A more involved verifier may use A together with the W to emulate the task T and compute local versions of R' and E'. Then it can check if these emulated values are close to the actual R and E received. One concern is that having to additionally run a verifier reduces the appeal of outsourcing to the cloud. Note that the verifier need not be run by the customer (i.e., it could be a third-party software service) and that its compute cost will be significantly smaller than the original T.

3. "DID I"–VERIFIABILITY

In this section, we focus the first question: Did I consume what I was charged? Addressing this question requires infrastructure support to ensure that (a) the provider does not create spurious charges of cycles that were never consumed by any principal (i.e., some conservation of work), and (b) the provider correctly assigns the consumption of a resource to the principal responsible for using that resource.

To address these sub-challenges, we start with a hypothetical clean-slate solution and then proceed to discuss how we can efficiently approximate these with existing technologies.

3.1 A clean slate solution

In order to obtain very fine-grained resource footprints of a wide spectrum of resources, the reporting mechanism should ideally be implemented at a trusted hardware layer. This is because of two reasons. First, the hardware is in the best position to observe the physical resource usage of a number of resources (e.g., cache misses, I/O requests) that may be abstracted away from the software stack. Second, a hardware root-of-trust is more reliable than OS- or VMM-level trust in a virtualized environment [5].

Suppose we had a trusted hardware layer that provides the following primitive. For each time epoch, the hardware generates an *attested* report specifying the active *atomic principals* and the amount of each resource consumed by each such principal during this epoch. Specifically, a trusted monitoring component on the hardware generates for each timestep t an attested log entry of the form $\{U, \langle S_1, \ldots, S_N \rangle\}$, where U refers to an atomic principal associated with the hardware context and S_is are the various resources that need to be accounted for. To provide the hardware layer with visibility into application-layer principals, we can extend ideas from prior work on resource containers [13] on practically exposing higher-layer principals to lower-layers.

Having a trusted consumption monitor satisfies two key requirements. First, because the reports are generated using in-hardware monitoring that is trusted to report *actual* consumption, a provider cannot convincingly charge customers for resources that it *never* consumed. Second, a provider cannot double-charge the same resources to multiple customers, since only a single customer is associated with every reported consumed resource, and the monitor reports only what was consumed.

3.2 Practical Approximations

While the above clean-slate solution is conceptually complete, there are three practical challenges: reporting bandwidth for transmitting fine-grained per-epoch measurements, performance overhead for getting such attested measurements, and dependence on a trusted hardware primitive that does not exist today. Next, we discuss potential solutions to address each challenge.

Reporting bandwidth: Reporting the entire trusted resource consumption log described above may be prohibitive. For example, assuming reports of 10 32-bit resource features per second, a large provider like Amazon (say 100K instances) would have at least an overhead $10 * 32 * 100K \approx 32$ Mbps of outbound traffic to send these reports (ignoring metadata and cryptography). To reduce this bandwidth cost, we envision an *aggregation* step that processes the log entries before generating R. That is, given the attested reports across multiple time epochs, a *trusted aggregator* generates a statistical summary of the resource consumption vector across time. Different classes of resources may define domain-specific aggregation functions. For example, we may want the *sum* as an aggregator for CPU cycles, for memory we may want to find the *max* and the *sum* of the utilization, and for I/O resources we may want to count the *total bandwidth-time footprint*. Note that the aggregation can even take place *inline*, meaning that individual log entries need not be physically generated or stored anywhere.

Measurement overhead: Running fine-grained measurements on a per-instruction or per-CPU cycle granularity can introduce a non-trivial performance penalty for the application processes being monitored [17, 34]. We discuss three potential opportunities to reduce this performance impact:

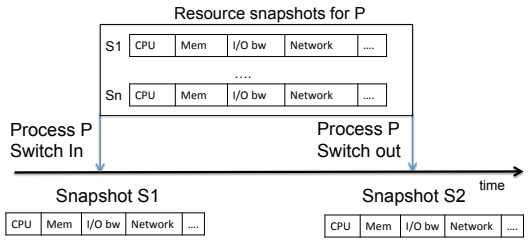

Figure 2: Taking snapshots of the resource consumption vector before/after a context switch

- The first, is *offloading* monitoring to a dedicated co-resident processor on a multi-core platform, similar to previous work on Log-Based Architectures [34]. The main idea is to generate events (e.g., explicit memory operations) that are efficiently routed to a secondary core, which then performs on-line analysis or packaging for off-line perusal, leaving the main core to continue performing its main functionality.
- The second is *sampling*. Instead of maintaining very fine-grained per-cycle or per-second counters, we can use a subset or sample of the resource utilizations. Sampling meshes well with the idea of aggregation to reduce reporting bandwidth. If we are only interested in a coarse-grained or aggregate statistic, we can choose a suitable sampling strategy that can give tight bounds on the bias and variance of the estimate (e.g., [10]). However, a sampling approach must be constrained to forestall adversarial activity such as cycle stealing [43], where malicious VMs opportunistically swap themselves out before a sample is taken, to charge their own activities to another VM's customer. Randomization seems to be a cheap but effective solution to counter such malice.
- Further along the spectrum, we can take *snapshots* of the resource consumption vector every time a new principal (e.g., a process) acquires or releases a resource; Figure 2 demonstrates this for the CPU, where process context switches are the acquisition/release boundaries. As long as the trusted monitor can guarantee that no principal other than the allocated one can use the resource between consecutive acquire/release events, then such measurements can be used to generate high-accuracy reports with minimal overhead.

Relaxing hardware dependence: Realizing such trusted hardware capabilities typically involves fairly long development cycles on the order of tens of months. A natural question, then, is if we can approximate the trusted reporting and aggregation primitives using existing hardware and software capabilities.

The ideas behind Log-based Architectures [34] can help here. Rather than performing a full per-principal attested measurement in an *online* fashion, we can use simpler, but dedicated hardware that just records the instruction stream. Then, we can have a post-processing step that reconstructs the sequence of actions and associates the resource consumptions to the active principals in each epoch. The challenge here is to provide this post-mortem processing with sufficient context to do the attribution accurately.

The capabilities we desire, strictly speaking, can be implemented at the VMM layer. Unfortunately, modern VMMs are sufficiently complex and involve several millions of lines of code. Thus, adding the VMM to our trusted computing base may not be a feasible alternative. Fortunately, what we need is a very small subset of the functionality that VMMs provide and this can be implemented as a small statically verifiable module. Here we can build on the promise of several recent efforts for building a small, but trusted,

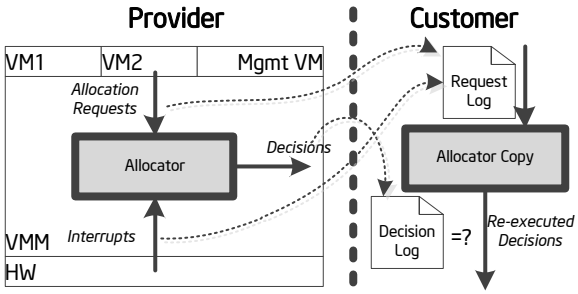

Figure 3: Prescriptive "Should I"-verifiability.

shim layer that performs a more restricted set of monitoring tasks. For example, recent work has shown that it is possible to implement a shim layer to intercept network packets [16] and to isolate trusted functionality from a legacy OS [26]. The challenge here is to evaluate whether the software-based solutions have enough visibility into monitoring low-level effects (e.g., cache stress), if they need to be reported as well.

4. "SHOULD I"-VERIFIABILITY

"Should I"-verifiability concerns itself not with whether specific usage attributable to a customer took place—the subject of the previous section–but with the justification for that usage. Given the workload at the infrastructure provider and the resource allocation policy agreed upon by the customer, should the customer's job have consumed the charged resources?

We assume here that the customer and the provider have an agreed-upon specification of the "right" resource-allocation policy. We consider two types of specifications: (1) a *prescriptive* specification that describes in some executable pseudocode the resource allocation decisions that should be made, given current workloads; or (2) a *quantitative* specification that describes an upper bound for the resources the allocator should devote, as a function of the given task and other workload. We study these two kinds of "Should I"-verifiability below, first by considering ideal solutions and then relaxing those solutions to make them practical.

In theory, a prescriptive specification can be verified via re-execution (Figure 3). Assuming that all resource allocation decisions are made by a self-contained component—the *allocator* handling memory, CPU, I/O, and other resource allocation decisions—a verifier with a trustworthy log of all inputs and outputs to the allocator can re-execute it and compare allocation decisions to those logged, to ensure the remote allocator was run correctly. This is similar to the notion of remote verification of distributed computations, as in PeerReview [20]. In PeerReview, a computation (the allocator in our case) is checked by a remote verifier by being re-executed according to a tamper-evident record of its inputs, comparing the outputs to a tamper-evident record of the outputs at the provider, and looking for deviations between the two streams of outputs.

In practice, this idealized solution to prescriptive "should I"-verifiability is hampered by several challenges. First, the raw log of inputs to the allocator is voluminous (possibly multiple entries per second from each machine in the infrastructure); collecting and sharing that log with the customer could require significant bandwidth and storage. Second, making that log tamper-evident requires computational resources for cryptographic or other "conditioning" of the log, possibly involving a TPM in the process. Third, the logic of a resource allocator (its code) or the policy may be proprietary to the provider, so sharing its specifics to enable re-execution may be unacceptable for privacy reasons.

Quantitative "should I"-verifiability, ideally, requires no information about the logic of an allocator, but more information (than the prescriptive approach) about the state of the whole system. Given the (tamper-evident) time series of resource utilizations and their allocations to different customer principals, the quantitative approach applies mathematical functions to the inputs to estimate an upper bound on utilization for the customer's task. If that upper bound is significantly lower than the charged resources, the customer suspects the provider for liberal scheduling. This concept appears related to SLA verification, with a twist. While SLA verification establishes that a service provided a minimum guaranteed level of resources to a customer, quantitative verifiability establishes that the service allocated to a customer (and subsequently charged for) no more resources than required by the submitted task.

In practice, the challenges with quantitative verifiability are somewhat steeper in terms of the volume and integrity-protection of raw data, since now the verifier requires not only decisions made by the allocator but also instantaneous measurements of state properties, such as utilization. On the other hand, quantitative verifiability does not require information about the allocator's code or policy, since the verifier is independent of the logic of the allocator and requires no re-execution or emulation.

In what follows, we study approximations of these ideal verifiability approaches to address the corresponding challenges.

4.1 Practical Approximations

We first consider prescriptive verifiability. The log-volume challenge is one of location. One way to address it is by moving the verifier (or the verifier's trusted agent) closer to the source of the log, on the provider's platform. For instance, if the provider's platform includes an execution environment trusted by the verifier (e.g., a verifier-trusted minimal OS and application, booted on the provider's platform using a hardware root of trust), then the verification could run there, obviating the need for transporting or storing large logs. This is complementary to doing local aggregation for "Did I"-verifiability.

The second challenge, that of the tamper-evidence of logs, remains difficult regardless of where the verifier operates. On one hand, collecting the logs must be trustworthy and, on the other hand, the logs must maintain their integrity before analysis by the verifier. Similar to "Did I"-verifiability, an approach to mitigate this challenge might be to use a trusted hypervisor to collect and authenticate the log (e.g., by trapping on logged events so that the VM cannot evade monitoring) before forwarding it to the verifier; e.g., via techniques like Secure In-VM Monitoring [36]. Combining this trusted log collector with a trusted execution environment for the verifier's code would compound the benefits of addressing both log-related challenges.

The final question is what the verifier code should be, whether it runs on the provider's platform or at the customer. Certainly running the exact same allocator would simplify re-execution, but at the cost of the provider's privacy. One approach might be to encode an abstraction of the allocator that hides the details of how decisions are made, protecting the provider's policy. To do this, some choices that are deterministic in the provider's code could become non-deterministic in the abstraction; any choice that would be allowed by possible provider policies would then be a plausible, legal but non-deterministic choice. This inherent tension between privacy and accountability is not unique to our framework. However, we can actually leverage the inherent fuzziness of verification in our favor. That is, any practical verification step will have to tolerate some noise in measurement. Consequently, we do not need

exact or deterministic replay and an approximate model of the proprietary code might suffice.

Some of the proposed approaches above apply just as well to the challenges of quantitative verifiability. The privacy of fine-grained measurements of the performance of a VMM on which the customer's VM executes (as well as co-tenant VMs competing for local resources) is better preserved by running the verifier locally; in this way, performance logs are never directly disclosed outside the provider's platform. However, formal privacy-preserving techniques, such as differential privacy, might be required to ensure that the aggregates collected and used to ensure compliance by the verifier do not, themselves, disclose sensitive information about cotenant VMs [38].

5. DISCUSSION

We have already highlighted many of the key technical challenges in the previous section. Here, we discuss other economic and policy concerns.

Incentives for providers: The incentives for customers of cloud services to desire and demand verifiable resource accounting is quite obvious. The three main incentives for providers, however, are more subtle and worth highlighting. First, providing more accountability in cloud billing will encourage more customers to move services to the cloud. A major concern for cloud providers is that their profit margins are hurt when their infrastructure is *underutilized*. Increasing the customer base will alleviate this concern. Second, deploying the mechanisms needed to provide verifiability will also ensure that providers can be less conservative in how they charge customers for specific resources or induced externalities. Third, anecdotal evidence suggests that cloud customers may try to "game" the system [28, 43]. The mechanisms we envision can enable providers to better detect such evasion.

Won't the market solve this problem? A common objection is that market effects will obviate the need for verifiable accounting. In other words, providers who consistently overcharge customers or are unable to correctly attribute resource consumption to customers and pay the cost themselves will eventually be weeded out. There is some truth to this statement; economic realities will remove obvious inefficiencies. But three key factors suggest that market effects alone will not be sufficient. First, the economics for most infrastructure services (e.g., telephony, data networks, public utilities) invariably have a natural economies-of-scale property. Economies of scale implies that these markets converge to a handful (2-3) of active and profitable providers. Consequently, there may be insufficient competition to deal with the problem. Second, while the choice of migrating across cloud platforms is nice in theory, it is difficult in practice. These include both difficulty in performing an apples-to-apples comparison across diverse service cloud offerings (platform services vs. software services) and the non-trivial costs in migrating applications across providers [23]. Finally, even for the market effects to eventually manifest, the technical means for providers to eliminate the inefficiencies must exist! As we discused, there are unresolved technical challenges in attribution and accounting in the face of external effects in a cloud environment.

Relaxing provider assistance: The previous sections implicitly assumed that the provider generates an attested report of resource consumption. It would also be interesting to explore to what extent we can relax this provider support. There are two possible options for customers.

The first option is to use *resource prediction*. While such tools are motivated by provisioning or scheduling applications, they can also provide basic verification checks. For example, the client can extract key features of the workload from its own tamper-resistant execution log and check if the predicted resource consumption roughly matches the reports from the provider. Resource prediction is challenging because workloads are hard to characterize and there are many hidden factors. Two recent approaches show promise to counter these challenges. Mesnier et al. use a notion of relative fitness to accurately extrapolate results across different configurations [27]. Similarly, Huang et al. show intelligent selection of program features and new modeling algorithms can improve the accuracy [22].

The second opportunity is for different customers to *collaboratively* detect violations [40]. Suppose each client C_i receives a time vector per resource j: $R_{i,j} = [M_{i,j,1}, M_{i,j,2}, \cdots, M_{i,j,T}]$. As a simple check, the clients can check for *conservation of resources* in the $R_{i,j}$, or other similar consistency invariants across distinct reports [11]. That is, if the provider has a cap $\hat{M}_j$ on how much of resource j it can allocate per time quantum, the clients can check if $\forall t, \sum_i M_{i,j} \leq \hat{M}_j$. If there is missing data (e.g., the provider's capacity is unknown or some customers do not participate), then they could use tomographic techniques [42] to guess these unknown values. The challenge here is to enable collaborative verification without compromising the participants' private information (e.g., via secure multi-party computation).

Other cloud charging models: So far, our focus was on pay-per-use billing models where customers are charged per quantum of resource consumed. There are other "time"-based billing models in use today; the most popular example being machine instances on Amazon's EC2. In this time-based billing model, the problem turns from one of verifiable accounting to one of *SLA verification*—Did my task get the expected compute throughput over the time it should have been active? While SLA verification seems like a different problem, we can reuse the underlying mechanisms presented earlier. For example, the Did-I step from Section 3 can now reflect a *throughput* vector and we can check if the throughput is what the customer paid for. However, the throughput could be lower than what was paid for either because of provider inefficiencies (e.g., the provider did not schedule it correctly) or because the task was itself idle (e.g., stalling for remote requests). Then, the Should-I strategies from Section 4 can help distinguish these two cases.

6. RELATED WORK

The initial success of cloud computing and the development of new software paradigms to support this emerging class of applications has motivated a lot of related work in these areas. We discuss these next and put these in the context of our vision for enabling verifiable resource accounting.

Benchmarking: Cloud customers today have a number of cloud providers that vary in both their service offerings and infrastructure capabilities. Recent work from Li et al. compares the costs of running a particular application under different popular providers and suggests that this choice is not easy [23]. Along a similar vein, because the workloads for cloud applications are themselves not well understood, recent work makes the case for a unified set of benchmarks to evaluate common cloud computing frameworks [35, 41]. The emergence of these tools for comparing and benchmarking providers reflects the need for cloud customers to objectively evaluate providers' cost-performance tradeoffs. Such tools are intended to help customers in choosing a suitable service provider. The mechanisms used in benchmarking can also be extended to provide initial Should-I verifiability; e.g., comparing the resource footprints of running the same workload across different providers and checking for inconsistencies.

Optimizing cloud platforms: Configuring and provisioning cloud platforms is a non-trivial optimization problem. There are several aspects that need to be accounted for including scheduling, server placement, network locality, etc. Several such optimizations have been proposed in the literature to overcome inefficiencies in existing cloud computing platforms [15, 24, 25, 33]. A case for verifiable accounting will further inspire the development and adoption of such optimizations in today's cloud frameworks.

Resource monitoring: Several efforts in industry [1, 9, 18] and academia [17] have identified challenges in scalably monitoring resource consumption in cloud virtualized environments. Our framework can build upon and extend such monitoring tools. However, there are two key ways in which our vision differs. The first is verifiability; the existing work focuses more on efficient collection and does not provide any vidence of correctness to the cloud customers. The second is that many of these proposals are provider-centric; we want to take these a step further and also empower cloud customers. Closest to our problem definition lies recent work on verifiable network measurement [11], although the technical challenges of solving this problem for OS-level resource accounting are significantly more complex than verifiable packet- or flow-level measurements.

Cloud accountability and security: Cloud security is a growing concern both in industry and the research community. For example, cloud customers may want to ensure that the provider faithfully runs their application software [19]. While knowing that the application code ran unmodified or verifying the input-output consistency is no doubt useful, they are not sufficient for verifiable accounting because of environment effects. Other work ensures that the provider that has not tampered with or lost their data [12, 21], or that it respects certain performance or consistency guarantees [31]. These target other aspects of accountability; our work focuses specifically on accountability for resource accounting.

7. CONCLUSIONS

Our position is that for cloud-based leased computing paradigms to be sustainable and dependable in the long term, they need a mechanism for verifiable resource accounting. Such a mechanism will benefit both cloud customers and cloud providers. It gives customers assurances that they really did consume and should be paying for the computing resources they are billed. At the same time, it affords providers with the opportunity to improve their profit margins by more accurately accounting for resources and by increasing their infrastructure utilization via increased cloud adoption. As a first step toward realizing this vision, in this paper, we defined the problem of verifiable resource accounting and highlighted the challenges and potential alternatives to realize this vision in practice. As future work, we plan to implement and release a practical reference implementation of a prototype system for verifiable resource accounting.

8. REFERENCES

[1] www.jinspired.com.

[2] Cloud Revenue: The Question Vendors Keep Dodging. http://goo.gl/UVQQi.

[3] Cloud storage providers need sharper billing metrics. http://goo.gl/Drq4J.

[4] Hotcloud'11 post-session q&a. http://www.usenix.org/events/hotcloud11/stream/warfield/index.html.

[5] Infrastructure security: Getting to the bottom of compliance in the cloud. http://www.rsa.com/document.aspx?id=10745.

[6] IT Cloud Services User Survey, pt.3: What Users Want From Cloud Services Providers. http://blogs.idc.com/ie/?p=213.

[7] IT Cloud Services User Survey: Top Benefits and Challenges. http://blogs.idc.com/ie/?p=210.

[8] Service billing is hard. http://perspectives.mvdirona.com/2009/02/16/ServiceBillingIsHard.aspx.

[9] Vmware vcenter chargeback. http://www.vmware.com/products/vcenter-chargeback/overview.html.

[10] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proc. STOC*, 1996.

[11] K. Argyraki, P. Maniatis, and A. Singla. Verifiable Network-Performance Measurements. In *Proc. CoNEXT*, 2010.

[12] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable Data Possession at Untrusted Stores. In *Proc. CCS*, 2007.

[13] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proc. of OSDI*, 1999.

[14] R. Cohen. Navigating the Fog - Billing, Metering and Measuring the Cloud. http://cloudcomputing.sys-con.com/node/858723.

[15] J. Dai, J. Huang, S. Huang, B. Huang, and Y. Liu. Hitune: Dataflow-based performance analysis for big data cloud. In *Proc. USENIX ATC*, 2011.

[16] C. Dixon, H. Uppal, V. Brajkovic, and D. Brandon. ETTM: A Scalable Fault Tolerant Network Manager. In *Proc. NSDI*, 2011.

[17] J. Du, N. Sherawat, and W. Zwaenepoel. Performance Profiling in a Virtualized Environment. In *Proc. HotCloud*, 2010.

[18] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, R. Hundt. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro*, 2010.

[19] A. Haeberlen, P. Aditya, R. Rodrigues, and P. Druschel. Accountable Virtual Machines. In *Proc. of OSDI*, 2010.

[20] A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: Practical accountability for distributed systems. In *Proc. of SOSP*, 2007.

[21] A. Juels and B. S. Kaliski. PORs: Proofs of retrievability for large files. In *Proc. CCS*, 2007.

[22] L. Huang et al. Predicting execution time of computer programs using sparse polynomial regression. In *Proc. NIPS*, 2010.

[23] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *Proc. of IMC*, 2010.

[24] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz and I. Stoica. Improving mapreduce performance in heterogeneous environments. In *Proc. OSDI*, 2008.

[25] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker and I. Stoica. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *Proc. EuroSys*, 2010.

[26] J. M. Mccune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proc. Eurosys*. ACM, 2008.

[27] M. Mesnier, M. Wachs, R. R. Sambasivan, A. X. Zheng, and G. R. Ganger. Modeling the Relative Fitness of Storage. In *Proc. SIGMETRICS*, 2007.

[28] A. Mihoob, C. Molina-Jimenez, and S. Shrivastava. A Case for Consumer-centric Resource Accounting Models. In *Proc. IEEE Cloud Computing*, 2010.

[29] J. C. Mogul. Operating systems should support business change. In *HotOS*, pages 8–8, Berkeley, CA, USA, 2005. USENIX Association.

[30] Y.-J. Ngo. Metering in the cloud. http://www.bizspark.com/Blogs/yi-jian_ngo/Lists/Posts/Post.aspx?ID=85.

[31] R. A. Popa, J. R. Lorch, D. Molnar, H. J. Wang, and L. Zhuang. Enabling Security in Cloud Storage SLAs with CloudProof. In *Proc. USENIX ATC*, 2011.

[32] R. Iyer et al. Virtual Platform Architectures for Resource Metering in Datacenters. In *Proc. SIGMETRICS*, 2009.

[33] S. Babu. Towards automatic optimization of mapreduce programs. In *Proc. SOCC*, 2010.

[34] S. Chen et al. Log-Based Architectures for General-Purpose Monitoring of Deployed Code . In *Proc. Workshop on Architectural and System Support for Improving Software Dependability*, 2006.

[35] S. Huang et al. The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Proc. ICDE Workshops*, 2010.

[36] M. I. Sharif, W. Lee, W. Cui, and A. Lanzi. Secure In-VM Monitoring Using Hardware Virtualization. In *Proc. CCS*, 2009.

[37] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proc. NSDI*, 2011.

[38] T. Ristenpart et al. Hey, You, Get off of my cloud: Exploring Information Leakage in Third-Party Compute Clouds. In *ACM CCS*, 2009.

[39] M. Wachs, L. Xu, A. Kanevsky, and G. R. Ganger. Exertion-based billing for cloud storage access. In *Proc. HotCloud*, 2011.

[40] C. Wang and Y. Zhou. A collaborative monitoring mechanism for making a multitenant platform accountable. In *Proc. HotCloud*, 2010.

[41] Y. Chen, A. Ganapathi, R.Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *Proc. MASCOTS*, 2011.

[42] Y. Yardi. Network Tomography: estimating source-destination traffic intensities from link data. *J. Am. Statistics Association*, 91(433), 1996.

[43] F. Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler Vulnerabilities and Attacks in Cloud Computing. http://arxiv.org/abs/1103.0759.