Towards An Effective and General Resource Accounting and Control Framework in Consolidated IT Platforms

Byung Chul Tak, IBM TJ Watson Research Center
Youngjin Kwon, Univ. of Texas at Austin
Bhuvan Urgaonkar, Penn State University

Abstract

In today's consolidated IT platforms, the capability to accurately *account* and proactively *control* the overall hardware resource usage among hosted applications can be necessary for a variety of resource management actions, such as capacity planning, dynamic resource reallocation and/or load balancing, and possibly also for auditing and explicit or implicit billing. We find that the increasing use of shared services in IT platforms, as exemplified by software-as-a-service offered by many clouds, renders existing solutions for accounting inadequate. We argue that current state-of-the-art is insufficient to meet the demand of today's large scale system management. In this paper we discuss the fundamental problems and propose new directions for better resource accounting and control.

1 Introduction

Achieving operational excellence on modern information technology (IT) platforms continues to become ever more challenging. The complexity of IT platforms keeps growing with emergent paradigms such as software defined network (SDN). Furthermore, recent trend in Big Data is necessitating increase of platform scales to unprecedented degrees. Today's management capabilities struggle to keep up with the increase of such complexity and scale [6, 7]. A large portion of complexity stems from the sharing and consolidation of computing resources. Increasingly, IT platforms consolidate multiple S/W applications on a shared set of hardware equipment for reasons of cost-efficacy or organizational necessity. Such consolidation and sharing occur in a wide variety of platforms. Considering a broader context, sharing also occurs for software, non-IT infrastructure resources such as cooling and power, and even personnel.

A clear understanding of how sharing happens for various resources and the ability to control the sharing can provide significant advantages in numerous management scenarios. For one thing, it can be crucial for performance management. Suboptimal or untimely reaction to the current resource sharing state may adversely impact business operations. For instance, a load imbalance towards one of the replicas in a shared database VM may impact negatively the end-to-end delay of all the services that rely on it. Many industry data support that revenue is highly sensitive to even sub-second increase of delays [11]. Such understanding also helps admins get answers to questions that are difficult to address otherwise. E.g. Which app's request is triggering sudden burst of CPU saturations in one of the key-value storage servers deep down in the service pipeline? To which replica should I redirect such workloads so that overall resource utilization stays within a safe range? Is it possible to apply resource capping so that fairness is maintained for the end users? Would we be needing accounting techniques for power and, even, administrative costs? Imagine that an admin detects a fast heat build-up at some part of the data center. Who is responsible for the heat, and who should be charged for the corresponding cooling cost?

However, ascertainment of accurate resource usage information (the activity we refer to as *resource accounting* in this work) is a challenging objective. First, in many cases there are no clear indicators that tell what portion of the overall usage of the platform's resources stem from (and should be attributed to) which component or entity that may run elsewhere. For instance, at the database node of a multi-tier e-commerce application, it is not clear who the originators for given database queries are unless the software stacks at each tier is modified to carry request identifiers. Further, if we are to control resource usage, this information has to be available real time. Another difficulty is that, even with such knowledge, resource attribution is non-trivial due to multiplexing, request aggregation and/or buffering. One example is the write activity of disk I/O in which OS kernel combines multiple writes and issue them much later in time non-deterministically. These challenges are intensified especially in today's complex service architectures in which many services build on top of other heterogeneous services via interface such as REST.

Based on our investigation, we believe that current state-of-the-art is insufficient to deliver aforementioned resource accounting and controlling capabilities to achieve operational excellence. Some of the existing techniques rely on modifying (all or part of) the OS or middleware stacks to enrich the collected data [1, 10]. Often, they focus on one specific component and implement the accounting and controlling functions by modifying source code [5, 9]. However, we cannot hope to have the source code ready or expect all of them to be instrumented. And, even with some instrumentations, gathered data may not contain enough information for our purpose. In response to such limitations, we set this vision of building resource accounting and control framework that is (i) generally applicable to existing environments, (ii) accurate enough for any type of resources, including both IT and non-IT resources, and (iii) flexible enough to accommodate a wide-range of resource management policies. This work represents our initial effort towards realizing this vision. We focus on IT resource such as CPU, I/O and network in a virtualized environment.

2 Problem and Challenges

2.1 Problem Model and Definition

We use Figure 1 that shows a representative platform that we use to drive our discussion. It illustrates various key entities and the relationships between them. We refer to a platform user or their application whose resource usage must be separately tracked and accounted as a *chargeable entity* (CE). The concept of CE is not limited to subscribers of software services. We define it in a broader sense to recognize groups of computing equipment as CEs as well. Figure 1 shows several types of such CEs - Group 1, 2, User 1, 2, and 3. However, if needed, any other entities such as "Business Logic VM" can be treated as separate CEs.

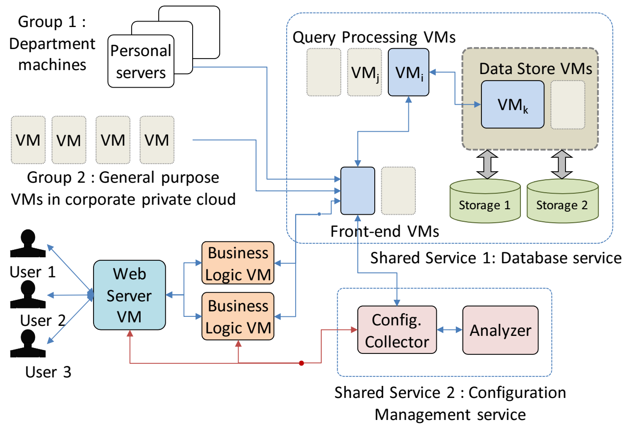


Figure 1: Model of resource accounting problem context. We assume the virtualized environment.

Also shown in the figure are example shared services that the platform offers to its CEs - a database service and a configuration management service. These shared services themselves have multiple components that span several VMs or servers. Internally the database service maintains functionally separate group of VMs for load balancing. The arrow from the configuration management service to the database service represents the fact that services may build on top of other services. Note that each CE does not exclusively own its database process. One database instance (in our example, two front-ends and three query processing VMs for one instance) may be shared by an unrelated group of tenants. One good example of such a multi-tenant database service is the Microsoft SQL Azure [8]. In SQL Azure, tenants only see their own database spaces, but it may physically share the underlying database process with other tenants. Our problem setup opens up many interesting questions. What is the CPU usage break-down of VM$_i$ by CEs? If VM$_i$'s CPU approaches saturation, which CE's workload should we redirect to another front-end so that they will be handled by VM$_j$? For each CE, is I/O traffic to Storage 1 and 2 evenly distributed from VM$_k$? Whose workload is the cause of the bursty CPU and network bandwidth usage at VM$_i$?

Problem Definition: Given a set of CEs and an accounting granularity Δ, the goal of the *resource accounting* is to infer, for each CE c at server s, the time-series $\mathcal{U}_c^i(t, s)$ for each resource type i, where

$0 \leq \mathcal{U}_c^i(t,s) \leq 1$ to represent proportions.

As a corollary, if we let $\mathcal{V}^i$ be the proportion of unused resource at time t at server s, it would follow that $\sum_c \mathcal{U}_c^i(t,s) + \mathcal{V}^i(t,s) = 1$. Similarly, $\sum_{t_0 \leq t \leq t_n} \sum_s \mathcal{U}_c^i(t,s)$ would represent the usage of resource type i by CE c in the entire platform during the time period of $t_0 \leq t \leq t_n$.

2.2 Challenges

Lack of direct indicators: A key difficulty in resource accounting arises due to lack of direct indicators of CEs responsible for the currently in-progress resource activities at the servers, especially in shared services. Unlike application-owned software, a shared service *may only be exercised by a CE indirectly*, making it more difficult to ascertain this relationship. In Figure 1, the query processing VMs of the DB service is merely invoked indirectly by the CEs through the front-ends, oblivious of who it is working for. Solving this problem, in general, requires some form of statistical inference based on probabilistic models to capture this causation, and closely related examples can be seen in some work [1, 2, 4].

Mismatch of resource principals and CEs: The shared service's software design and configuration *may not be amenable to easy adaption of existing solutions for local accounting*. For example, the data store component in Figure 1 multiplexes the resources assigned to its internal schedulable entities (e.g., threads) in highly application-specific (and possibly unknown) ways among the activities it carries out on behalf of CEs rendering a solution such as resource containers difficult to adapt.

3 Overview of Design Issues

Any accounting solution must have two elements: (i) *local monitoring* and (ii) *collective inference*. We use the phrase "local monitoring" to refer to facilities within each server that record events and statistics pertaining to the resource usage of (or on behalf of) each CE. The phrase "collective inference" refers to the functionality needed to combine the pieces of information offered by local monitoring to create a correct overall picture of accounting.

There exist a large number of techniques and tools for local monitoring that one could choose from. These existing techniques span a wide spectrum of the "level of detail" they offer at the cost of generality, application intrusiveness, and overheads posed. At one end of this spectrum are techniques that can instrument user-space and OS/VMM code to create a very detailed record of a shared service's resource usage that contains sufficient information for collective inference [3]. At the other end of the spectrum are CE-oblivious resource usage reporting tools that rely on information available within the server's OS and VMM. E.g., top, and iostat.

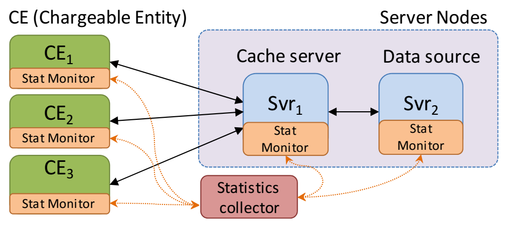
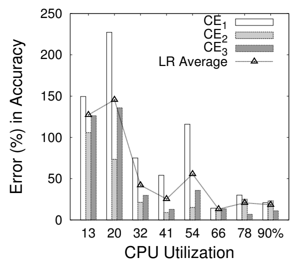
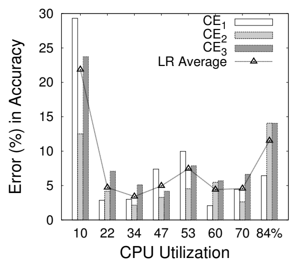


Figure 2: Synthetic shared service setting.



(a) Steady workload.



(b) Bursty workload.

Figure 3: Impact of burstiness and CPU util to LR result.

Then, is it good enough a solution if we combine easy-to-use local monitoring tools with suitable inference technique such as linear regression (LR)? In order to answer this, we have conducted experiments using synthetic shared service of Figure 2. For the local monitoring, we employed `top`, `iostat` and `iptables` for CPU, disk and network monitoring, respectively. For the inference, we chose the popular LR technique, relating inbound network traffic volume $x_i(t)$ from each CE$_i$ to the CPU utilization $y(t)$ of Svr$_1$. That is, we have formed following equation for each interval t.

$$a_0 + a_1 \cdot x_i(t) + a_2 \cdot x_2(t) + ... + a_n \cdot x_n(t) = y(t)$$

Then, we solved it using the least squares method to obtain the coefficients $(a_0, a_1 ... a_n)$ for each interval t. Since each coefficient represents the contribution of each $x_i(t)$ to the resource usage, we treated them as proportions to divide the CPU utilization $y(t)$, giving us the break-down of CPU by each CE at Svr_1 at time t. Since we wrote server codes, we were able to measure the exact amount of CPU resources consumed by each CE, which enabled us to calculate the accuracy of LR-based technique.

We show one selected result in Figure 3 from the set of experiments conducted. It shows the effect of workload burstiness on the accuracy over full range of CPU utilizations. We find that the efficacy of LR depends upon the extent of variations within the imposed workload. Intuitively, better accuracy is achieved with bursty workloads because the higher variety/dynamism in the input data supplies more information to LR; we expect this basic insight to apply to any statistical inference technique for accounting. To summarize, we find that the efficacy of LR relies upon both the quality of data it gathers as well as the presence/extent of correlation between its inputs and outputs.

Generally speaking, collective inference is a statistical learning problem that must derive models that can meaningfully tie together the data provided by local monitors, possibly filling in any gaps or discrepancies within these data. The efficacy of such inference crucially depends upon the resource usage phenomena collected by local monitoring elements. Existing monitoring tools that are not application-intrusive have been designed for information collection at the granularity of OS/VMM-relevant abstractions (e.g., threads, TCP connections) that may not coincide with the needs of our accounting. Consequently we identify the following design principle that underlies our accounting solution: *our local monitoring must explicitly capture information pertaining to resource usage on behalf of CEs to allow accurate accounting by our collective inference.*

4 Design of Our Prototype

The design principle learned in Section 3 guides us to try different approach. Instead of seeking to leverage existing tools with LR, we decide to put more effort into collecting higher quality monitoring data in an attempt to reduce the complications from the inference algorithms and to gain better accuracy. We have implemented our prototype techniques within the hypervisor of Xen to accomplish transparency to the VMs as well as applications.

4.1 Local Monitoring

The key aspect of local monitoring that we need to perform is identifying and recording information about resource principals and scheduling events of interest. Recall from Section 2, that the real challenge in answering this question arises when CE c uses resources on server s indirectly, i.e., when a shared service component running on s consumes resources on behalf of c.

Identifying the Causation and Interval: To recognize a CE c that is more than one hop away, we adopt the principles introduced in vPath [12]. Following its key idea, we are able to identify who the originating CE is upon the message arrival to the target server s. Then, the thread that received the message is associated with the CE c and we start to track the CPU consumption and I/O activities. If the thread spawns new threads or fork another process, we also associate them to the c. When the thread that received initial message replies back to the sender, we consider it as the signal of the end of resource consumption on behalf of c.

Thread Scheduling Events: Within the interval identified by the arrival and reply of the message, the execution of the thread that is associated with c may be arbitrarily interleaved with the thread associated with another CE d. Therefore, in order to correctly perform the resource accounting, we also gather the thread scheduling events.

4.2 Collective Inference

Given the extensive information that our local monitoring gathers, collective inference for accounting CPU and network/disk IO bandwidth essentially boils down to simple aggregation of the resource usage information collected by various local monitoring units. Due to lack of space, we omit the algorithmic and implementation details.

4.3 Controlling The Resource Usage

The goal here is to limit the resource consumption by CEs or to maintain specified ratio of consumption among CEs, in a VM-transparent way. To achieve this, whenever we detect that a thread associated with c is about to be scheduled, we manipulate the density of the timer interrupts sent to the CPU core. Generating faster timer interrupts will

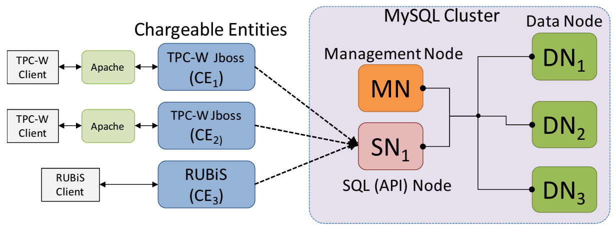


Figure 4: Shared MySQL cluster setting.

Phase	Time Window	Workload	Top User
Phase 1	0-400s	All CEs generate light loads	CE_2
Phase 2	400-600s	CE_2 starts to issue CPU-heavy requests	CE_2
Phase 3	600-1200s	CE_2's workload overwhelms CPU, load increases every 100s	CE_2

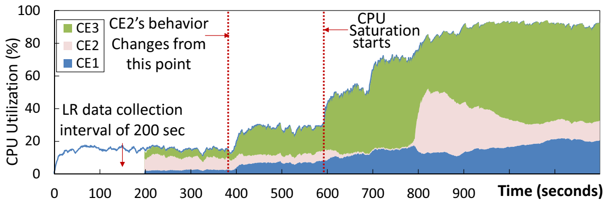
Table 1: Workload scenario.

have to effect of taking away the resource since the thread will be descheduled faster and vice versa.

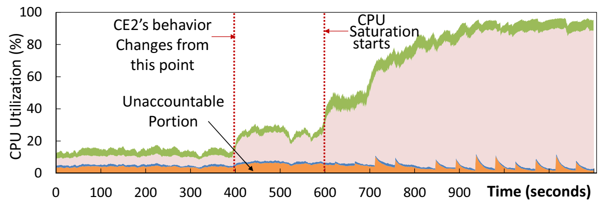
5 Analysis of Our Solution's Effectiveness

We have conducted preliminary evaluation of the solution approaches described in Section 4. In this section we compare the performance of our solution against the LR techniques Figure 4 shows the set-up of our MySQL cluster used here. We compare the efficacy of Ours and LR in the following online resource control situation: we wish to ensure that when the aggregate workload imposed upon the MySQL cluster causes its server CPUs to saturate, we identify the contribution of various CEs to this "overload," and then enforce targeted CPU throttling only to the CE causing the overload. We configure our CEs to impose a dynamically changing workload (consisting of three phases) on MySQL as described in Table 1.

Results: Figure 5 shows CPU accounting for SN's server as carried out by LR and ours, respectively. We use a "stacked" representation, where the area under the curve corresponding to a CE represents the CPU usage charged to it. During phase 1, both LR and ours produce correct rank orders of CEs, although LR slightly overestimates the CPU consumption for CE_2. However, during phase 2, LR starts to report incorrect rank order: it determines CE_3 to be the cause of the increased CPU usage. Upon investigating the reason for this mistake by LR, we find the following. While CE_2 issues CPU-heavy requests and waits for MySQL's response,



(a) Result of CPU accounting using LR in stacked graph.



(b) Result of CPU accounting using our prototype.

Figure 5: Comparison of CPU accounting results. CPU usage of SQL node is being accounted. By comparing the areas of equivalent color we can see the rank order determined by each technique as well as accuracies. The result of ours includes the 'unaccountable' portion. This can be divided among chargeable entities by any reasonable policy.

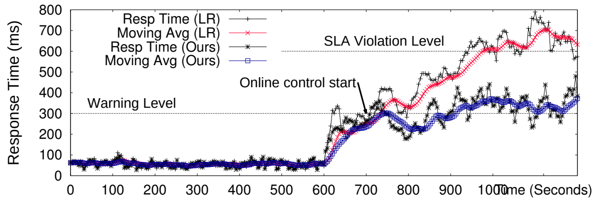


Figure 6: Response time change of RUBiS. This shows the development of RUBiS response time for two cases - throttled by LR, and controlled by ours. Since LR picks the wrong CE (CE_3) as a culprit of performance degradation, throttling the request rate of CE_3 is ineffective. However, our technique is able to contain the response time so that SLA is not violated.

CE_3 continues to issue requests at a relatively high rate that are not CPU-heavy. However, the higher rate of requests coming from CE_3 causes LR to infer spurious positive correlation between CE_3's requests and SN's CPU usage. In fact, LR is unable to correct this throughout phase 2.

During phase 3, starting at t=600s, CE_2 starts to saturate the CPU by drastically increasing the workload it imposes as described in Table 1. Since our technique identifies the true cause of the overload, we are able to initiate the CPU throttling for CE_2, preventing the SLA violation. This demonstrates one promising capability of our technique (which is the thread-level monitoring technique) in critical resource managements of such shared resources.

6 Conclusion

In this paper we have highlighted the need for a better resource accounting and control capabilities for the management of large-scale distributed systems. In order to understand the problem in more depth, We have explored two different accounting techniques built with different balance of emphasis - LR-based collective inference vs. fine-grained thread-level system monitoring. Our analysis revealed that there could be cases where more fine-grained monitoring information does not only provide better accuracy, but also impact cricital management decisions.

References

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2eprof: Automated end-to-end performance management for enterprise systems. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '07, pages 749–758, Washington, DC, USA, 2007. IEEE.

[2] A. Anandkumar, C. Bisdikian, and D. Agrawal. Tracking in a spaghetti bowl: monitoring transactions using footprints. In *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 133–144, New York, NY, USA, 2008. ACM.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, Berkeley, CA, USA, 2004.

[4] M. Y. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-based faliure and evolution management. In *Proceedings of the 1st conference on Networked Systems Design and Implementation*, Berkeley, CA, USA.

[5] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: tracking energy in networked embedded systems. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 323–338, Berkeley, CA, USA, 2008. USENIX Association.

[6] E. M. Haber, E. Kandogan, and P. Maglio. Collaboration in system administration. *Queue*, 8(12):10:10–10:20, Dec. 2010.

[7] E. Kotsovinos. Virtualization: Blessing or curse? *Queue*, 8:40:40–40:46.

[8] Microsoft SQL Azure. `http://www.microsoft.com/en-us/sqlazure/default.aspx`.

[9] V. Narasayya, S. Das, M. Syamala, S. Chaudhuri, F. Li, and H. Park. A demonstration of sqlvm: performance isolation in multi-tenant relational database-as-a-service. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, New York, NY, USA, 2013. ACM.

[10] J. Reumann and K. G. Shin. Stateful distributed interposition. *ACM Trans. Comput. Syst.*, 22(1):1–48, Feb. 2004.

[11] Latency Is Everywhere And It Costs You Sales - How To Crush It. `http://highscalability.com/blog/2009/7/25/latency-is-everywhere-and-it-costs-you\-sales-how-to-crush-it.html`.

[12] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang. vpath: precise discovery of request processing paths from black-box observations of thread and network activities. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, Berkeley, CA, USA.