Synergetic Resource Stealing: We Promise It Will Not Hurt Much

Vincent W. Freeh* Xiaosong Ma*† Jonathan W. Strickland* Sudharshan S. Vazhkudai†

Abstract

A great many machines, from personal workstations to large clusters, are under utilized. Meanwhile, for the fear of slowing down the native tasks, resource scavenging systems hesitate to aggressively harness idle resources. We have developed a quantitative approach for fine-grained scavenging that can effectively utilizes very small slack periods without adversely impacting the native workload, and automatically adapts to changes in the native workload's resource consumption.

This paper envisions a generic framework, built upon the above approach, that facilitates the sharing of machines between a primary *and a* secondary *workloads, providing a unified view of the diverse systems where idle resources are available. In such a framework the primary workload performance is bounded by a configurable slowdown factor (*i.e.*, 5%) and the secondary workload aggressively utilizes the slack left by the primary workload. Thus our proposed framework creates a novel resource-sharing paradigm that results in greater resource utilization without sacrificing the performance of the primary workload.*

1 Introduction

Many have noted that resources on desktop workstations are largely unused. Well-known resource scavenging systems (*e.g.*, Condor [7]) and applications (*e.g.*, SETI@home [1]) are designed to take advantage of unused resources. Such resource scavenging is a boon because it is essentially free. Meanwhile, such idle cycles from distributed workstations combine to become significant compute power on par with state-of-the-art supercomputers. From the workstation owners' perspective, however, one important pre-condition of donating resources on their personal computers is that the scavenging application will not significantly degrade the performance of their native workloads.

Such resource under-utilization problem is not limited to personal workstations, but is pervasive in today's computing environment. For example, consider a finance corporation that owns a server that executes stock trades while the stock markets are open. It has to be built for handling the peak transaction capacity, which rarely occurs due to the bursty nature of its real-time workload. It is very desirable to concurrently execute a scavenger application that could make use of the excessive capacity of the server during non-peak periods. As another example, consider a pharmaceutical company that acquires a medium-size cluster for its R&D staff that wants to get the most out of this machine: people running quick biological database searches and interactive visualization programs would like to have online processing and fast responses, while people running large batch simulations do not mind their jobs taking longer to finish but would not want to see such large jobs wait in the queue forever.

All the above scenarios boil down to the same question: *when the primary workload leaves a significant amount of "slack" in a system, can we sneak in a secondary workload without hurting the performance of the primary workload, yet make progress on the secondary workload?* Under the desktop scavenging scenario, the workstation owners' native workload is the primary, while

*Department of Computer Science, North Carolina State University, Raleigh, NC, 27695-7534 {vwfreeh,jwstric2,xma}@ncsu.edu

†Computer Science and Mathematics Division, Oak Ridge National Laboratory vazhkudaiss@ornl.gov

the guest scavenging application is the secondary workload. Under the server and cluster scenarios, the online/interactive tasks are primary, while the offline/batch tasks are secondary.

There are no existing systems that aggressively utilize these slack periods. First, priority scheduling is not designed for such scavenging, so the impact on the primary workload from a secondary workload can be large. Next, while systems that provide quality-of-service guarantees can limit the impact of a scavenger, they are over-kill—including requiring kernel modifications. Finally, the current state of the art for impact control of a scavenger application on a primary workload tends to be over conservative. Foremost, people have chosen to err on the safe side and prevent the secondary from competing with the primary. For desktop resource stealing, the typical approach is the *stop* method, which only activates the secondary when the screen saver is on, and stops it when any user activity is detected, which signals the resource owner's return. For server and cluster environments, people use similar methods, such as having a secondary task running only during nights and weekends (when the primary workload is not active), or partitioning their systems to have separate nodes allocated for interactive and batch jobs. Though this assures no adverse impact on primary workloads, it under utilizes the system, especially when the primary is bursty, or when the primary and secondary workloads are using resources very differently and could have accommodated each other well.

In this paper, we argue for *quantitative study* of workloads' *performance compatibility*, and present a *generic performance impact control framework*. This framework characterizes a secondary workload's impact an a variety of key resources on the target system, and manipulates its execution dynamically based on the real-time monitoring of the primary workload's consumption of these resources. It aims to maximize the secondary's throughput while restricting its performance impact on the primary *under a pre-specified and user-configurable impact threshold*.

The rest of the paper is organized as follows. Section 2 introduces, as background, our design of the *Governor* impact control system, which performs aggressive resource scavenging on desktop systems and has spurred the ideas in this paper. Section 3 discusses design issues as well as open problems related to a generic impact control framework. Section 4 describes other related work.

2 Background: Governor

This section gives background information regarding the *Governor*[1] impact control framework we designed for desktop resource scavenging systems [10]. Concepts and methodologies developed here will be extended for the generic workload impact control framework proposed in this paper.

The goal of the Governor is to control the impact (*i.e.*, slowdown) the scavenger has on the native workload under a desired level, yet to maximize the throughput of the scavenger. Instead of assigning scavenging processes low priority, and relying on the operating system to schedule these processes unfavorably, Governor throttles the intensiveness of resource scavenging by inserting sleeping time between time intervals that a scavenging process can execute. This reduces the demand on resources and hence reduces the impact on the native workload. Governor performs fine-grained impact control by choosing and adjusting the ratio between "run time" and "sleep time". We define the *throttle level*, β, to be $t_{run}/(t_{run} + t_{sleep})$. β varies from 0 to 1. $\beta = 0$ means the scavenger is not running at all, while $\beta = 1$ means the scavenger is running at full speed, without being slowed down.

How do we find the appropriate β for a given impact level? First, we employ impact benchmarking to characterize the effect of a scavenger on major resource types. Since native workload can be viewed as a combination of resource consumption components, we establish a *resource vector*, $R = (r_1, r_2, ..., r_n)$, where each r_i is a system resource, such as CPU, memory, disk bandwidth, network bandwidth, etc. We design a set

[1] Governor: A feedback device on a machine or engine that is used to provide automatic control, as of speed, pressure, or temperature. – dictionary.com

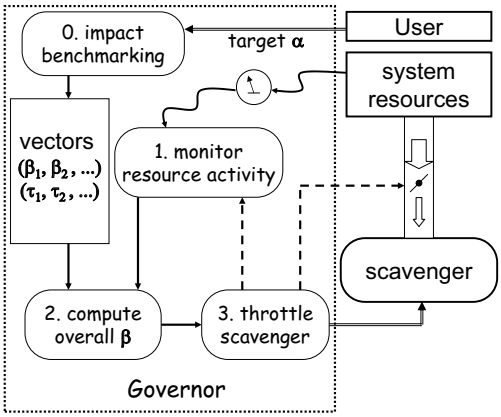

Figure 1. Governor architecture

of micro-benchmarks that stress each individual resource in R. A scavenger is executed at various throttle levels and the performance impact on the micro-benchmark is recorded. Given enough data points, we can estimate the function $\mathrm{impact}_i(\beta)$. Suppose Governor wants to restrict the maximum impact on any resource to a target impact level, α. The corresponding β to use in throttling the scavenger is determined as $\beta_i = \mathrm{impact}_i^{-1}(\alpha)$.

Then, knowing how to restrict the scavenger for a single-minded micro-benchmark, we need to decide the appropriate throttle level for a complex, and ever-changing native workload. Our solution is to periodically monitor the native workload: for resources that have native consumption above their individual trigger level τ_i detected, we activate the corresponding β_i. When multiple resources have their βs turned "on", Governor chooses the most restrictive one among them to be the overall throttle level.

Figure 1 depicts the Governor framework in working. Note that Step 0 is likely to be performed when a scavenger is first installed in a donated workstation, while Steps 1-3 are periodically repeated whenever the scavenger is running. The dotted arrow from inside the Governor box to the "resource valve" shows that the Governor is able to control resource consumption implicitly.

Our experiments have shown that Governor is able to closely approximate a preset target impact level, for both scavenging applications we tested (one CPU- and one I/O-network-intensive). Compared with the process priority based method, it has wider range as well as much finer granularity in impact control, and allows higher scavenging application throughput when delivering the same amount of impact on a mixed native workload, especially for the non-CPU-intensive scavenger.

3 Discussion

Generalizing Governor First, we need to generalize the Governor to work for arbitrary primary and secondary workloads. For all of our target environments, including desktop resource stealing/aggregation, high-end servers, and clusters, we propose to have our impact control framework work hand in glove with the job scheduling agent.

Because Governor does not control the primary workload and considers all non-secondary activity to be primary, the Governor implicitly handles arbitrary primary workloads of any number of processes. The straightforward method for controlling multiple secondary workloads is to run them sequentially, not concurrently. This way the individual β vectors of each secondary are used instead of a composite vector determined by the set of secondaries concurrently executing. Therefore, we simply need to repeat the impact benchmarking process for each secondary application to obtain their β vector. After such per-application key parameters are obtained, Governor can apply the automatic and adaptive process throttling individually for these applications.

A more challenging task here is to characterize *ad hoc* secondary tasks. Static, *a priori* impact benchmarking is adequate for resource scavenging systems with uniform and predictable resource usage pattern, such as SETI@home and a few distributed storage aggregation systems. This benchmarking can be performed naturally as a part of the system's installation process on a donated computer. On the other hand, systems like Condor or a general server/cluster environment will have unknown secondary jobs come and go. An attractive solution is to have a short *diagnostic benchmarking stage* before scheduling a secondary job. For a long-running or irregular job, this stage is entered *periodically* to update its impact characteristics (*i.e.*, the β vector). The job migration facilities

supported by systems such as Condor works perfectly together with this dynamic and adaptive impact benchmarking scheme.

Finally, although Governor can characterize a given secondary workload's performance impact on a given primary workload, and throttle the secondary to bound this impact by a target threshold, we need to take one step further to better coordinate resource sharing between the primary and the secondary workloads. For example, to maximize overall system resource utilization on a cluster, we want to avoid scheduling two CPU-intensive workloads head on where they compete for the resource. Instead, a CPU-intensive secondary workload prefers to be placed on the nodes where the workloads resource usage is complementary, *i.e.*, not CPU-intensive. Similarly, in a storage space scavenging system, the storage manager prefers to place the "hottest" data on personal workstations where the average I/O and network upload traffic is low.

Here we introduce the concept of "performance compatibility". One primary and one secondary workloads are considered *performance compatible* if the secondary does not have a high impact on the primary. In general, a pair of workloads will not be performance compatible, if the primary happens to stress one or more resources on which the secondary shows heavy impact. This compatibility can be quantified by the throttle level curve for the secondary to restrict its impact on the primary at various target levels. With workloads' pairwise compatibility quantified, the job/data scheduler can perform intelligent "matchmaking": it sends secondary tasks to the places where they will be less annoying and have more space to safely consume more resources. This global optimization can be reduced to a vertex matching problem on a weighted bipartite graph.

Dynamic Impact Measurement In evaluating Governor's impact control performance, we performed external experiments to measure the impact that the primary workload actually suffered by comparing the execution time or throughput of the primary workload executing solo versus that when sharing resources with a secondary workload at various throttling levels. Although the Governor has successfully bounded the overall performance impact in several cases, our straightforward techniques may not work for any arbitrary environments or secondary workload. Therefore, we look for an on-line, dynamic feedback mechanism that measures the performance degradation of the primary workload.

The major challenge in measuring actual impact is how to determine the slowdown of an *ad hoc* workload as it executes. A possible solution is to extend our workload monitoring facility to work together with the secondary workload's throttling in collecting and calculating statistics to infer the native workload's throughput change. Metrics such as the instruction count per time unit will be more suitable for this purpose. For instance, if we find a primary workload as an average instruction throughput of t_0 during a large number of monitoring intervals without an concurrent secondary, and an average instruction throughput of t_1 during intervals sharing resources with a secondary, we can roughly derive the actual performance impact. This information can provide crucial feedback to the impact control framework for its self-adjusting.

Connecting Objective Impact with User-Perceived Impact Our discussion so far has been focused on *objective* performance impact, which does not directly translate into *resource or primary workload owner perceived* impact. The latter is a complex issue involving many factors besides the slowdown ratio, such as the lengths, interactive nature, frequency of the primary tasks, as well as the resource or workload owners' sensitivity and personality. Eventually, it is the primary workload owners (no matter whether they also own the computing resources) who decide whether a second workload is getting into their way.

An investigation very closely related to ours was by Gupta et al. [5], which studied real workstation owners' discomfort from impacted performance when resource scavenging applications run on their computers. In this project, testers rate the discomfort they experienced in carrying out a va-

riety of daily tasks such editing PowerPoint documents and playing video games, with different resource scavenging benchmarks running in the background. The feedback data show that a significant amount of CPU, memory, and disk I/O resources can be scavenged often without causing obvious user discomfort.

We believe that there is a positive correlation between the objective performance impact and the subjective user discomfort. Therefore, the user interfaces used in Gupta et al.'s study could be combined into our framework, as a bridge to connect objective performance impact control and subjective impact feedback. When a primary workload owner or system administrator scrolls an "impact bar" to ask for lower impact, the impact benchmark and workload monitoring facilities in our framework can adjust the secondary workload's throttle level in a case-by-case manner.

4 Other Related Work

Besides systems that stop the secondary workload altogether upon the activation of primary workload (*e.g.*, Condor [7] and SETI@Home [1]), a couple of resource scavenging system adopt priority-based approaches [6, 9]. Such approaches are best-effort, limited to cycle stealing scavenging systems, platform dependent, less powerful in impact control then the Governor. One additional approach to containing resource consumption for scavenging applications is through the use of virtual machines [4, 8]. While virtual machines are well known for sophisticated resource isolation through partitioning [2], concurrent execution, etc., they may also result in unsatisfactory performance, heavyweight implementations, and lack of performance guarantees [3]. Compared to these existing approaches, our proposed process-throttling method is lightweight, operates at user level, performs full-range and fine-grained impact control, and does not require the modification of either the scavenger programs or the operating system.

In addition, many studies have been done on job coscheduling in parallel or distributed environments (*e.g.*, [11]). Although existing work also analyzes the different workloads' resource usage patterns and optimize the system's overall resource utilization, our proposed framework is unique in the sense that it controls the interference *between* workloads in a quantitative manner, similar to what people have been doing to ensure QoS.

References

[1] Seti@home: The search for extraterrestrial intelligence. http://setiathome.ssl.berkeley.edu/.

[2] Planetlab: An open platform for developing, deploying and accessing planetary-scale services. http://www.planet-lab.org, 2005.

[3] P. Barham, B. Dragovic, K. Frase, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of SOSP'03*, October 2003.

[4] A. Chien, B. Calder, S. Elbert, and K. Bhatia. Entropia: Architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5), 2003.

[5] A. Gupta, B. Lin, and P. Dinda. Measuring and understanding user comfort with resource borrowing. In *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, 2004.

[6] P. Krueger and R. Chawla. The stealth distributed scheduler. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 336–343, 1991.

[7] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, 1988.

[8] R. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J. Jornada, and W. Cirne. Non-dedicated distributed environment: A solution for safe and continuous exploitation of idle cycles. In *Proceedings of the Workshop on Adaptive Grid Middleware*, 2003.

[9] K.D. Ryu, J.K. Hollingsworth, and P.J. Keleher. Efficient network and I/O throttling for fine-grain cycle stealing. In *Proceedings of Supercomputing'01*, 2001.

[10] J. Strickland, V. Freeh, X. Ma, and S. Vazhkudai. Governor: Autonomic throttling for aggressive idle resource scavenging. Submitted for publication, http://www.csc.ncsu.edu/faculty/freeh/-governor.pdf.

[11] Y. Zhang, A. Sivasubramaniam, J. Moreira, and H. Franke. Impact of workload and system parameters on next generation cluster scheduling mechanisms. *IEEE Trans. Parallel Distrib. Syst.*, 12(9), 2001.