Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

# Reprint of "Robust partial-load experiments with Showstopper"

Andrej Podzimek[a,c,*], Lubomír Bulej[a,c], Lydia Y. Chen[b], Walter Binder[a], Petr Tůma[c]

[a] Faculty of Informatics, Università della Svizzera italiana, Switzerland
[b] IBM Zurich Research Lab, Switzerland
[c] Department of Distributed and Dependable Systems, Charles University in Prague, Czech Republic

A R T I C L E I N F O

*Article history:*
Available online xxxx

*Keywords:*
Partial load
Resource sharing
Performance evaluation
Resource isolation
Load control
Reproducible load

A B S T R A C T

Concurrent execution of multiple applications leads to varying partial utilization of shared resources. Understanding system behavior in these conditions is essential for making concurrent execution efficient. Unfortunately, anticipating behavior of shared resources at partial utilization in complex systems is difficult, realistic experiments that reproduce and examine such behavior are therefore needed.

To facilitate experiments at partial utilization, we present a tool that accurately controls the processor utilization of arbitrary concurrent workloads, either establishing constant partial load or replaying a variable load trace. We validate the ability of the tool to enforce the configured partial utilization on multiple platforms, and use the tool to collect novel information on system behavior at partial utilization levels.

In detail, our experiments show how to examine the complex relationship between utilization and throughput, useful for tasks such as performance debugging or system dimensioning, and we show this relationship for the DaCapo benchmarks. Further, we show that CPU pinning (a technique used to improve workload isolation) can benefit from dynamic response to system utilization, improving system efficiency with partial utilization. Finally, we show that the overhead of virtualization also changes with partial utilization and CPU allocation.

## 1. Introduction

For reasons ranging from efficiency to maximizing achievable performance, concurrent execution of multiple workloads is the norm for most computer systems today. Workload colocation has been researched extensively, with a number of results spanning several decades—for example, Chase et al. [1] achieves demonstrated energy savings of 29% and projected energy savings of as much as 78% on a web workload.

To support concurrent execution efficiently, essential system features such as task scheduling and resource allocation have to deal with resource sharing, which influences multiple aspects of performance [2–4]. There are numerous possible interactions—workloads running on a contemporary server can influence each other through mechanisms such as simultaneous multithreading [5], same page merging [6], frequency boosting with thermal budgeting [7], and more.

The multitude of possible resource interactions is difficult to capture comprehensively—research advances therefore often tackle selected interaction aspects individually. In this context, abstracting away from certain interactions is essential to keep the research problems tractable. At the same time, evaluating the impact of such simplification on research results becomes equally necessary—a recent case study by Corradi et al. [8] in fact warns that the growing distance between the theoretical assumptions and the real system behavior may make some of the existing research results rather difficult to apply.

A straightforward method for assessing the impact of theoretical simplifications is experimental validation. Unfortunately, evaluating resource interactions in realistic conditions is an expensive undertaking. Resource utilization depends on workload type in complex ways and even if we can control the workload, it is not always clear what workload intensity to use to achieve a particular resource utilization. Our work contributes a tool that simplifies experiments with varying partial utilization of shared resources. Given an arbitrary executing workload, the tool observes a resource utilization metric of choice – processor utilization in our case – and throttles the workload to achieve the target utilization. This removes the need for implementing workload generation harnesses with configurable workload intensity, and makes it

DOI of original article: http://dx.doi.org/10.1016/j.future.2016.04.020.
* Corresponding author at: Department of Distributed and Dependable Systems, Charles University in Prague, Czech Republic.
*E-mail address:* podzimek@d3s.mff.cuni.cz (A. Podzimek).

http://dx.doi.org/10.1016/j.future.2016.11.013
0167-739X/© 2016 Elsevier B.V. All rights reserved.

possible to examine system behavior at a particular utilization directly. The modular tool design and the available experiment configuration options are described in Section 3, the ability to enforce configured partial utilization is validated in Section 4.

After introducing and validating the tool design, we use it to present novel results on the behavior of computing systems under partial utilization. In Section 5, we look at the complex relationship between workload intensity and processor utilization. Our results provide a more accurate empirical alternative to operational analysis, which is typically used to relate utilization and throughput [9]. In Section 6, we look at how partial utilization impacts the practice of pinning workloads to selected processor cores. Against common intuition, our results show that the most efficient and most performant pinning configurations change with utilization. Finally, in Section 7, we briefly show how partial utilization influences virtualization overhead. These results are particularly relevant to cloud computing environments, where partial utilization and virtualization is often the norm.

This article constitutes a comprehensive presentation of our work on experimental evaluation of partially utilized systems, and combines and extends previously published conference material. Our extensions focus both on adding more depth to existing results, as well as on entirely new experiments—ultimately providing a more complete picture that explains the need for partial-load experiments along with compelling experimental results.

Our contributions can be summarized as follows:

- We present a comprehensive description of the Showstopper tool used to conduct experiments at partial CPU utilization. The basic concepts and the high-level architecture of Showstopper were introduced in our prior work [10,11], based on a preliminary version of the tool. Thanks to its modular architecture, the tool has been significantly extended to provide a wider range of control and dithering mechanisms, and to provide support for measurement of system load inside Linux LXC containers.
  Here we present a detailed description of the architecture and the various building blocks of the control algorithm, along with guidelines for choosing the values of critical parameters. We validate the ability of Showstopper to enforce and maintain the desired partial utilization, and extend the previously published results with quantitative evaluation of accuracy of different configurations of the load-control algorithm.
- We introduce a novel methodology to examine the relationship between processor utilization and application throughput, and present experimental results that illustrate the complex nature of this relationship. We extend our prior work [10] by introducing fast-forwarding of load traces replayed by Showstopper, and evaluate the effect of fast-forwarding on the accuracy of throughput measurements. We also significantly extend the workload and platform coverage.
- We provide a novel study of the impact of CPU pinning on performance and energy efficiency for pairs of colocated workloads. Our experiments with different CPU pinning configurations at varying levels of background load expose a trade-off between workload isolation and overall system performance. We also show that the performance increase due to pinning configuration at certain background loads increases the overall energy efficiency of the system. Here we complement the results presented in our prior work [12] with a study of the overhead observed in the employed colocation solutions (KVM virtual machines and LXC containers) at partial loads.

## 2. Motivating challenges

In this section, we demonstrate several challenges that motivate our work on experiments with partial utilization. While some of our examples aim for brevity, we would like to stress that the presented challenges get even more complex in typical computing systems with a deep software stack and sharing of hardware among colocated virtual machines.

### 2.1. Controlling utilization

Resource utilization metrics serve as the basis for monitoring and managing the quality of service provided by a computer system. Services such as Amazon Cloudwatch [13] are commonly available to clients of cloud computing providers, and many resource management policies are defined in terms of resource utilization. Combined traces of virtual and physical resource utilization are commonly collected on contemporary production systems for both physical and virtual machines. These traces provide insight into allocation and sharing of resources and aid in developing resource management policies [14,15]. Fig. 1 shows an example of such a trace, capturing the average processor load over a four-hour period, sampled in 15-min intervals.

Given a trace that describes realistic resource utilization conditions, an experiment that seeks to validate theoretical results and assumptions in the same conditions will need to achieve the same utilization trace. Typically, the experiment would also rely on available benchmarks as examples of realistic workloads [16–20]. Many benchmarks, however, are designed to execute in isolation, with all system resources at their disposal. In contrast, reproducing realistic utilization conditions requires a partial and coordinated use of system resources. Even benchmarks that aim to limit the system load [21] do so by controlling application level metrics (typically problem size or request rate) rather than by observing and responding to system level resource utilization. The whole problem becomes even more complex if we consider sustaining a given partial load (see footnote 1) while executing multiple benchmarks in parallel.

To illustrate the difficulties, we explore two solutions applicable to achieving partial loads with existing benchmarks: (1) executing multiple benchmarks in parallel so that the number of benchmarks divided by the number of processors equals the desired load, and (2) executing a benchmark with a constant duty cycle[1] corresponding to the desired load. These solutions also serve as baselines for our approach.

We evaluate the two baseline solutions on six DaCapo [22] benchmarks with a default iteration time short enough to make throughput estimates from short time intervals (tens of seconds) feasible. The selected benchmarks capture various types of multi-threaded behavior—ranging from an almost single-threaded execution (fop, luindex), through a moderate multi-threadedness failing to saturate all available processors (avrora, h2), to heavily multi-threaded computations capable of consuming all CPU capacity (sunflow, xalan). Generally speaking, the key difference between moderately and heavily multi-threaded benchmarks lies in the overall frequency of synchronization among threads. While threads in moderately multi-threaded benchmarks synchronize and block each other frequently (forming e.g. producer–consumer relationships or participating in *rendezvous* events), threads in heavily multi-threaded benchmarks spend most of their time working on well isolated subproblems of a parallel computation, without the need to block and synchronize with each other frequently.

Our goal is to achieve partial loads of 33% and 75% using the above methods. For method (1), this means executing 4 and

[1] We use the terms *duty cycle* and *partial load* in an intuitive fashion throughout this section. A more formal definition of both quantities and their interpretation by our load control tool is given in Section 3.