Report on Online tracking of Working Sets, and Locality Study of Dirty and Reference Bits

Swapnil Haria
Department of Computer Sciences
University of Wisconsin-Madison
Madison, WI, USA
swapnilh@cs.wisc.edu

***Abstract*—Effective management of a machine's memory resources needs comprehensive information regarding the real-time memory consumption of applications. Memory demands can be estimated accurately by tracking the working set sizes of programs. We propose a software approach for monitoring the working set size of applications, based on Denning's mathematical model of working sets. This technique is implemented using an existing tool, BadgerTrap, and its ability to accurately track working set sizes is demonstrated using memory-intensive workloads.**

Large pages are increasingly being used for improving the performance of big-memory applications. However, current implementations fail to accommodate page swapping adequately, and thus have limited use in conditions of memory pressure. We observe the distribution of dirty and reference bits for several workloads to study the feasibility of tracking these at a large 2MB page granularity. These observations are made in the presence and absence of memory pressure, to observe the effects of page swapping.

I. Introduction

Understanding the runtime memory demands of applications is critical for the efficient management of memory resources. Working sets were proposed by Denning in the 1970s as a convenient and intuitive method for modelling a program's memory demand [1]. At a high level, a program's working set comprises of the set of recently referenced pages from its entire address space. The presence of high correlation in working sets for intervals closely spaced in time make this an attractive method of tracking memory requirements of applications. As a result, working set estimation has been employed for diverse purposes, such as to enable lazy restore of checkpointed memory and to increase the density of virtual machines on a physical node without compromising performance [2], [3].

There have been previous efforts which have tracked working sets using performance counters, cache and page miss ratio curves (MRC) and reference bit scanning [2]–[4]. These methods suffer from the problems of aliasing, low accuracy, or involve hardware modifications. We propose a software-only technique of estimating working set sizes (WSS) during the runtime execution of an application, which is based on principles from Denning's mathematical model of working sets [1]. This enables highly accurate tracking of working sets without any changes to the hardware or the application being monitored. This functionality is added to BadgerTrap, an existing tool that allows online instrumentation of TLB misses [5].

Large page sizes are increasingly being supported to allow big memory applications to work with 2MB and 1GB page sizes [6]. This increases program performance chiefly by increasing the coverage of TLBs. However, huge page support in the linux kernel suffers from the limitation that such pages cannot be swapped out on memory pressure. This is slightly overcome through Transparent Huge Pages (THP) [7], but at a significant performance cost. Dirty and reference bits maintained at the granularity of huge pages could enable simple swapping mechanisms, but also lead to increased memory traffic. We study the spatial locality of dirty and reference bits to estimate the inflation in memory traffic due to their courser tracking, for big-memory workloads.

In this work, we extend the BadgerTrap tool to support the runtime estimation of an application's working set. We propose two different approaches for tracking the working set of the application. The effectiveness of this functionality is illustrated by using it to observe the working set behavior of several memory-intensive applications from various benchmark suites across their runtime. This report examines the spatial locality of dirty and reference bits to study the increased memory traffic caused by the tracking of these bits on a courser large-page level.

The remainder of the report is structured as follows: Section II presents a brief description of the working set model, BadgerTrap as well as Transparent Huge Pages. The initial hypotheses made regarding the estimation of working sets, and the distribution of dirty and reference bits are discussed in Section III. The enhancements made in BadgerTrap to allow working set estimation are presented in Section IV. The working set behavior as well as the observations on the spatial locality of dirty bits from simulated benchmarks results are compiled and discussed in Section V. Section VI summarizes the previous literature tackling relevant issues, and finally Section VII concludes the report and describes further work to be performed in this domain.

II. Background

A. BadgerTrap

BadgerTrap (BT) [5] is a tool that enables real-time analysis of Translation Lookaside Buffer(TLB) misses for workloads running on actual hardware. We prefer BadgerTrap over cycle-accurate simulators for our analysis as big-memory workloads run several orders of magnitude slower in such simulators as compared to actual hardware. Also, most simulators

only model OS-level events to a moderate degree of detail, and hence can not be trusted for a comprehensive analysis.

BadgerTrap is designed to intercept the hardware-assisted page walk performed on an x86-64 TLB miss, and transform it into a page table fault. It does this by setting the reserved bits of all PTEs of the process under observation during the initialization phase. The page fault handler is modified to pass along these BT-induced faults to a new software-assisted TLB miss handler. The handler installs a clean version of the PTE entry into the TLB in order to service the TLB miss correctly. The TLB miss handler is also responsible for monitoring the number of TLB misses.

1) Working Sets: Denning observed that the stream of references made to its allocated pages by a program exhibits locality, with a subset of the pages being favorably referenced during any time interval. He also found that there is great inertia in the composition of this subset of preferred pages, and that a high correlation exists between reference patterns closely related in time. Denning proposed the concept of working sets to better study these observations and their implications on program behavior. He defined a program's working set W(t,T) as the set of unique pages accessed in the time interval [t-T+1,t], with T being referred to as the window size. The window size must be large enough to include references to most currently useful pages, and small enough to avoid being affected by program's phase changes. Consequently, the working set size w(t,T) is quite simply the number of pages present in W(t,T).

B. Huge Pages

Translation Lookaside Buffers (TLBs) are provided by most architectures to help minimize address translation overheads. The absence of valid translations in the TLB results in TLB misses and thus degrades the performance of the program. Translations stored in the TLB are at the page level granularity, and thus the page size along with the number of TLB entries directly affects the effectiveness of a TLB. The number of TLB entries is constrained by the latency requirements associated with TLB accesses. While the page size on x86 based machines has conventionally been 4KB, there is increasing support for larger pages (such as 2MB pages in x86-64) to reduce the number of TLB misses, improve the TLB miss latency, and thus improve the performance of big-memory applications.

However, the use of huge pages with the linux kernel needs elevated privileges, boot time allocation to guarantee success and requires considerable effort from developers as well as system administrators. Also, pages marked as huge pages are reserved by the kernel and cannot be allocated for other uses. Furthermore, they cannot be swapped out to disk under memory pressure. Large page support was improved upon by the introduction of Transparent Huge Pages (THP), which aims to enable easy access to huge pages, without the need of explicit developer or administrator attention. THP works by preferentially allocating a huge page on a page fault if a huge page is available. On swapping, the huge page is split into its component 4KB pages, and then swapping is performed on a 4K page size granularity. Thus, THP allows easy adoption of huge pages when needed, but regresses to smaller page sizes to facilitate swapping.

III. Problem Statement and Initial Hypotheses

A. Working Set Sizes

Working sets have been proposed as an effective means of analysing the memory demands of applications. The working set size w(t,T) depends on two factors- the window size T as well as the observation time t. The the influence of observation time t is removed by averaging out the results from many random observations. Denning emphasized the importance of the size of the observation window, which should be chosen large enough to witness references to all currently needed pages, and small enough to avoid being affected by phase changes of the program. Thus, our initial hypothesis is that the WSS should increase linearly with the window size initially, till the window is large enough that all necessary pages are found in the working set. Once this is achieved, the WSS will be constant inspite of increasing window sizes till the window size approaches the duration of the program phases.

B. Dirty/Reference study

Huge pages help improve program performance, especially for big-memory applications, but current implementations suffer from limitations related to page swapping. Swapping is important especially in the case of big-memory workloads to alleviate memory pressure. While swapping of huge pages was initially not supported by the linux kernel, the introduction of Transparent Huge Pages solves this to an extent by dividing the large 2MB page into its constituent 4KB pages when swapping is needed. This approach erodes the performance benefits of supporting huge pages. Swapping techniques for 4KB pages based on dirty and reference bits can be directly extended for large 2MB pages. However, naively tracking dirty and reference bits at the coarser 2MB granularity could increase disk I/O traffic significantly. This additional traffic arises from the fact that in the extreme case, an entire 2MB page could be written back to disk even though only a 4KB chunk of it was actually dirty.

In order to observe the costs associated with this, we intend to study the spatial locality present in dirty and reference bits in the case of big-memory workloads. We consider spatial locality to be the similarity of values of dirty and reference bits of smaller 4KB pages constituting a larger 2MB page. The existence of significant spatial locality in dirty and reference bits implies that there is minimal additional I/O traffic caused by the coarse, 2MB level tracking of dirty and reference bits, as most 4KB chunks making up a 2MB page are dirty, or all of them are clean.

A list of relevant hypotheses has been compiled here, to direct our current study. In this report, we only tackle the hypotheses considering 2MB pages as large pages.

Hypothesis 1 *At most times, most dirty bits are set.*

Hypothesis 2 *At most times, most (above 75%) of the recently accessed pages are dirty.*

Hypothesis 3 *Dirty bits show spatial locality at a 2MB page granularity (above 75% of 4K pages making up a 2MB page exhibit similar behavior)*

Hypothesis 4 *Large 1GB pages aren't majorly dirty (below 25% of 4K pages making up a 1GB page are dirty)*

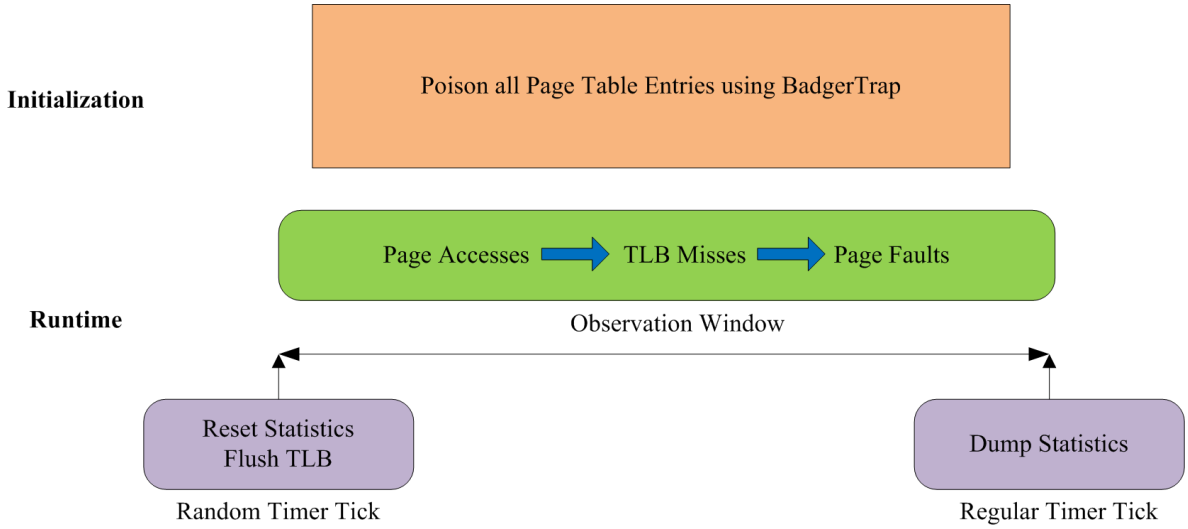

Fig. 1: Overview of our approach for WSS tracking

Hypothesis 5 *At most times, most reference bits are set.*

Hypothesis 6 *Reference bits show spatial locality at a 2MB page granularity (above 75% of 4K pages making up a 2MB page have similar referenced behavior)*

IV. Implementation

The actual implementation of the kernel-level tools modified or built for the observations in this report are discussed in this section.

A. Working Set Size tracking

We present three techniques for the online measurement of the working set size w(t,T) of an application. The first technique is discussed as a naive measurement, while the two bit-vector based approaches are grouped together as accurate measurement techniques. These have been integrated into BadgerTrap to leverage its ability to instrument TLB misses. The window size T and the upper limit for the random timer are configured by means of a new system call added to the linux kernel. Interrupts from a randomly triggering timer are used to determine various start times for the tracking intervals. At the start of every tracking interval, the TLB is flushed to ensure that every future page reference results in a TLB miss. BadgerTrap allows us to handle these TLB misses in a custom software-assisted fault handler, which also keeps count of these misses. The entire sequence of operations following a TLB miss with modified BT enabled is depicted in Figure 2, while an overview of the tool itself is shown in Figure 1.

1) Naive Measurement: The WSS can be estimated by tracking the total number of TLB misses in the predetermined window size T. The naive count is the upper bound of the WSS of the program at the given time. This approach is the simplest to implement, but the count may be inflated due to duplicate TLB misses caused by large working sets as it lacks the ability to track the number of unique page accesses. Thus, to maintain the count of distinct page references or equivalently, TLB misses, we used two methods of greater complexity.

2) Accurate Measurement: We achieve greater accuracy by keeping track of pages already included in our working set using a bit vector of size 1MB. Thus for every TLB miss observed, we index into this bit vector using the 20 least significant bits of the virtual page number. This counts as a unique access only if the indexed bit is unset. Finally, we set this bit to weed out future accesses to the same page from being counted. The bit vector approach presents us with a lower bound for the working set size, as two distinct pages with the same lower 20 bits will be counted only once.

To improve the accuracy of our estimation further, we implemented a two-level bloom filter to identify unique page references. The first level is the same as the simple address-based indexing described in the previous paragraph. For the second level, MurmurHash is used to index into the bit vector. MurmurHash is a non-cyptographic hash chosen here primarily for its speed and ease of implementation. The actual tracking of page references is the same as in the simple bit vector approach.

3) Comparison: These above implementations were thoroughly tested using various self-written micro-benchmarks which exercised different possible scenarios with deterministic results. The results of all three approaches for one such micro-benchmark can be seen in Figure 3. The micro-benchmark iteratively made regular, strided accesses to an array much larger than the spread of the TLB, essentially thrashing the TLB. The naive estimation, without any support for detecting duplicates, considered each TLB miss to be distinct. Thus, the WSS increased almost linearly with the window size using naive estimation. Both the simple bit-vector approach as well as the two level bloom-filter approach perform much better. These two techniques only count the TLB misses from the first iteration as distinct page accesses, and correctly weed out TLB misses from subsequent iterations as duplicate access to pages already part of the working set. WSS estimates from the two bit-vector based techniques were within 1% of each other for all experiments. Hence, we only consider the WSS estimates from the simple bit-vector based technique in the results section.

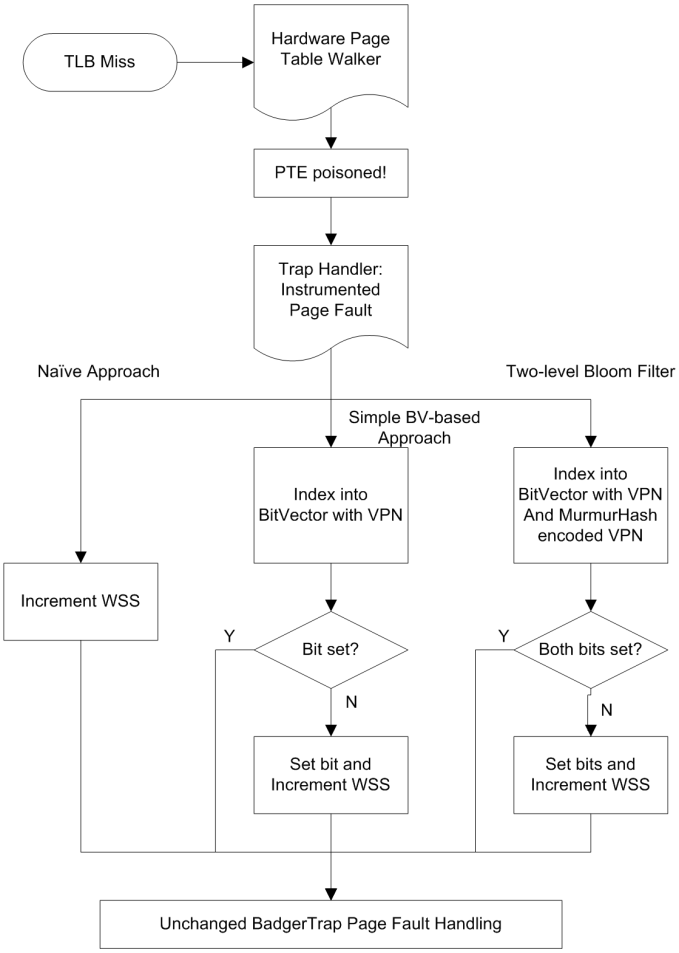

Fig. 2: Flowchart with our modified BadgerTrap for each instrumented TLB miss

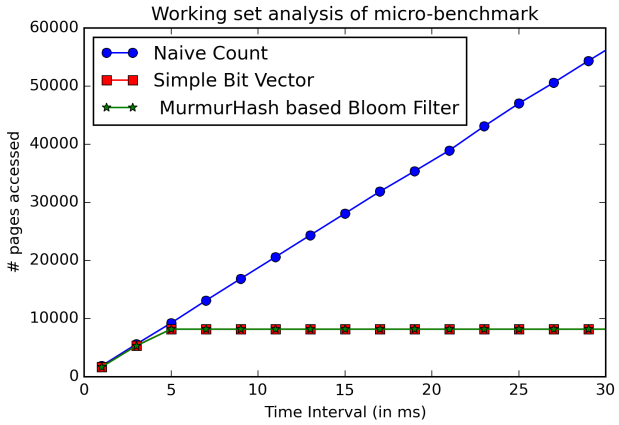

Fig. 3: Working Set Size estimation of micro-benchmark thrashing the TLB

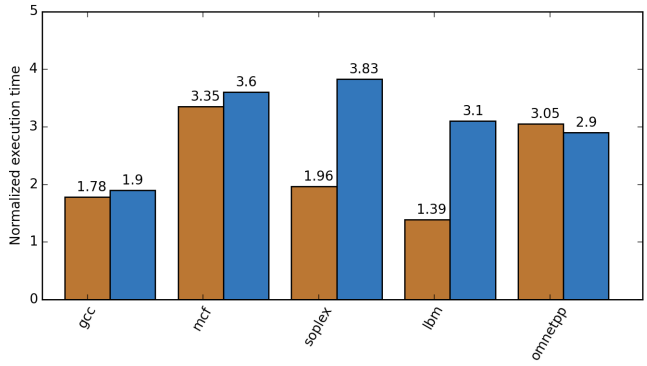

Fig. 4: Overheads in terms of increased run times for SPEC workloads

4) Tool overheads: Running unmodified BadgerTrap slows down observed program by a factor of 2x to 40x, depending on the number of TLB misses observed. Our modifications to this tool further increase this overhead, chiefly due to greater TLB misses on account of the TLB flush performed at the beginning of every tracking interval. The cost of indexing into the bit vector and computing the MurmurHash are insignificant when compared to the cost of servicing the TLB miss. The overheads introduced by our observation methods influences the results slightly by degrading the performance of the program due to the increased number of TLB misses. Our interest lies in the high-level patterns and these are unaffected by our observation methods. As seen in Figure 4, the execution times of various SPEC benchmarks running with our modified BadgerTrap tool are about 1.4-3.8 times the standalone execution times. These results were calculated with the random timer limit of up to 10s, and window size ranging from 1-50ms. The tracking overheads will increase with decreasing random timer limits, and decreasing window sizes.

B. Locality Analysis of Dirty and Reference bits

The study of the locality of dirty and reference bits was performed using a new tool, currently called SwapTrap. SwapTrap allows online analysis of page table entries for applications running on actual hardware. SwapTrap interrupts the program under observation at random times in its execution, using a random timer, and then scans the entire process page table to identify patterns and observe the distribution of dirty, present and reference bits. Thus, SwapTrap facilitates high level analysis of process page tables. For the current study, we configured SwapTrap to check for spatial locality of dirty and reference bits of 4K pages aligned on 2MB page boundaries.

V. Evaluation

In this section, we describe our evaluation methodology for observing the working set sizes of big-memory workloads, as well as studying the spatial locality of dirty and reference bits. The use of BadgerTrap and SwapTrap enable us to sidestep the long simulation times of running big-memory workloads using cycle-level simulators.

A. Methodology

The results reported in the following sub-section were collected by running workloads alongside our enhanced BadgerTrap and SwapTrap on an quad-core, x86-64 machine with 4GB of memory. Our workloads comprised of several memory-intensive SPEC CPU2006 workloads, as well as big-memory

applications like graph500 and memcached. The selection of SPEC CPU2006 workloads is based on the performance characterization of these benchmarks in [8].

B. Results

We first present the analysis of working set sizes for some SPEC benchmarks, and then discuss the distribution of dirty, accessed and valid pages for these workloads.

1) Working Set Size Estimation: The influence of window size on the WSS of an application is studied by measuring the WSS at many different random times in its execution and repeating this for increasing window sizes. Here, we only present the results from the simple bit-vector approach, as the results are within 1% of the results obtained by the two-level bloom filter approach. However, the latter technique significantly increases the run time for a few benchmarks, as seen in Figure 4.

As seen in Figure 5, the WSS, in terms of the number of unique page accesses, increases linearly with increasing window sizes before stabilizing. We consider lbm and omnetpp separately on a longer time frame to demonstrate two interesting patterns. These results are presented in Figure 6. The rationale behind Denning's lower and upper limits on the window size can be understood from the WSS analysis of omnetpp. The WSS increases with the window size from 1ms to 50ms, then stays constant for over 100ms. The increase in WSS above the window size of 160ms is brought about by a phase change in the program. Another pattern is seen in the case of lbm, which implements the Lattice Boltzmann Method for the 3D simulation of incompressible fluids. It makes continuous streaming accesses to a huge number of memory locations, and thus has a working set which always increases with the window size.

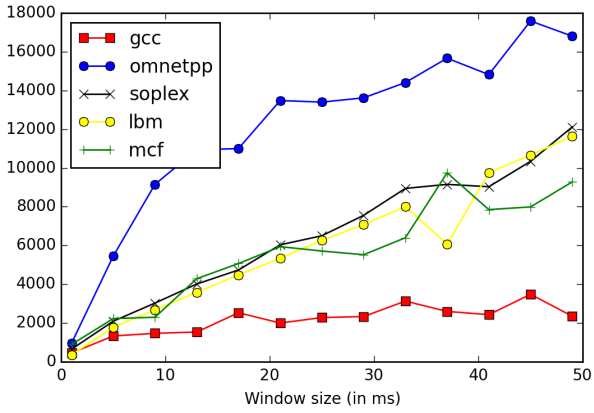

Fig. 5: Working Set Size analysis of some SPEC benchmarks

2) Locality Analysis of dirty and accessed pages: We now present the results of the study of the distribution of valid, accessed and dirty pages for SPEC benchmarks. As explained in Section III, we are interested in locality in terms of behavior across each set of 512 4KB pages aligned on a 2MB page boundary. Three sets of observations are presented here to clearly depict the trends on increasing memory pressure. The

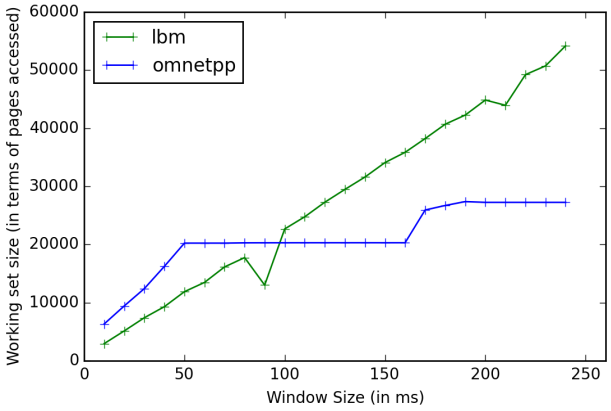

Fig. 6: Longer WSS analysis of omnetpp and lbm benchmarks

first group of graphs (Figures 7, 8 and 9) correspond to the benchmarks running on a relatively idle machine, with enough memory to avoid the need for swapping. The next two sets of observations (Figures 10- 15) are obtained in conditions of memory pressure, with only about 600MB and 200MB of memory free on the machine. These conditions were caused in a controlled manner by running a program which pinned a large amount of memory.

In the absence of memory pressure, there is significant spatial locality in the distribution of dirty bits. From Figure 7, 97% of the larger 2MB pages either have all of their constituent 4KB pages simultaneously dirty, or clean. Similar spatial locality is seen in the case of reference bits for over 97.8% of the large 2MB pages. Over 99.5% of the valid, allocated pages are dirty and referenced. These results corroborate our initial hypotheses.

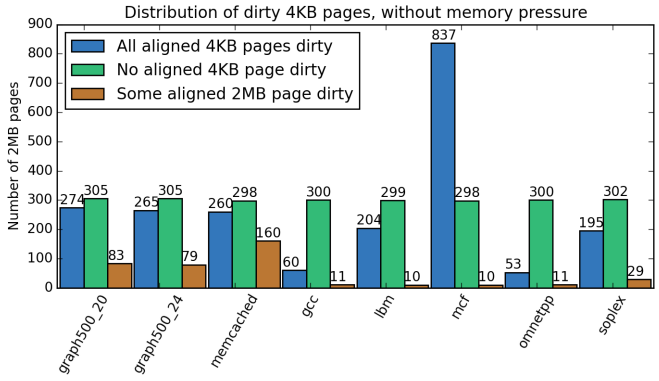

Fig. 7: Distribution of dirty 4KB pages, without memory pressure

Increased memory pressure leads to swapping, and negatively affects the spatial locality of dirty and referenced bits. We include the non-memory intensive gcc benchmark in our study as its working set is small enough to be unaffected by the low availability of memory. Thus, its results remain constant across our experiments. The memory-intensive benchmarks

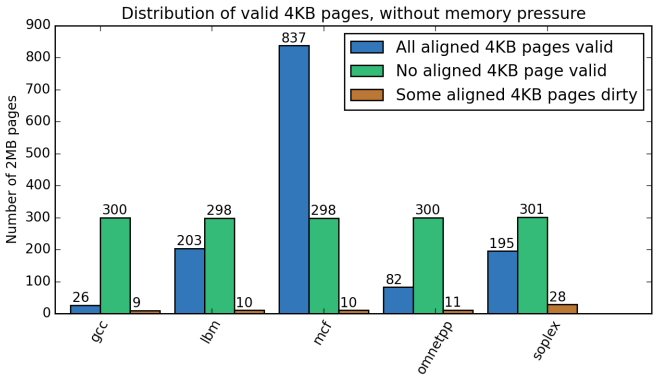

Fig. 8: Distribution of valid 4KB pages, without memory pressure

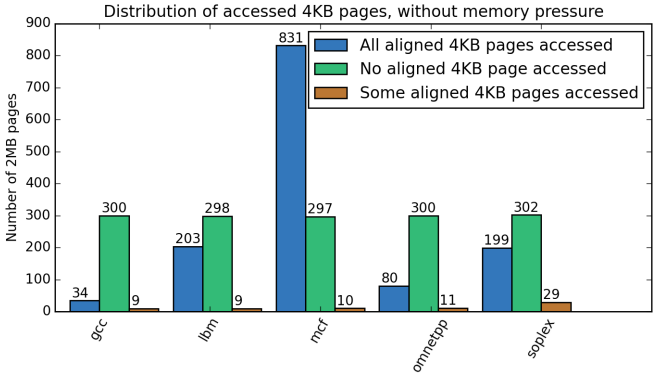

Fig. 9: Distribution of accessed 4KB pages, without memory pressure

suffer greatly from the effects of increasing memory pressure, and the number of 2MB pages with only some of their component 4KB pages dirty increases. The mcf benchmark with its large memory requirements shows the effects of memory pressure with 600MB of available memory. The other benchmarks maintain their initial page distributions before succumbing to memory pressure at 200MB of available memory.

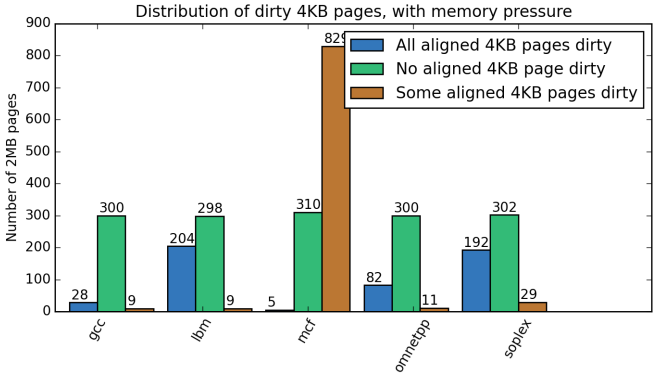

Fig. 10: Distribution of dirty 4KB pages, with 600MB of available memory

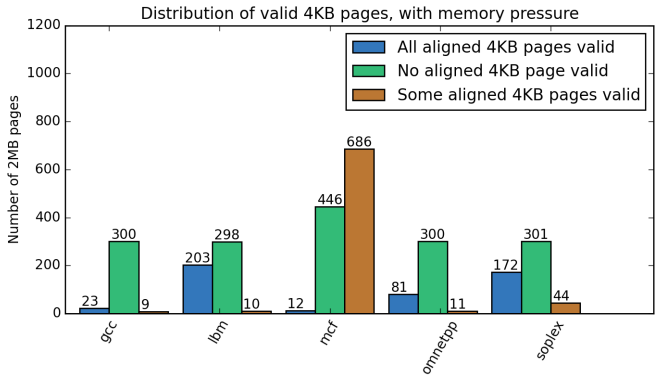

Fig. 11: Distribution of valid 4KB pages, with 600MB of available memory

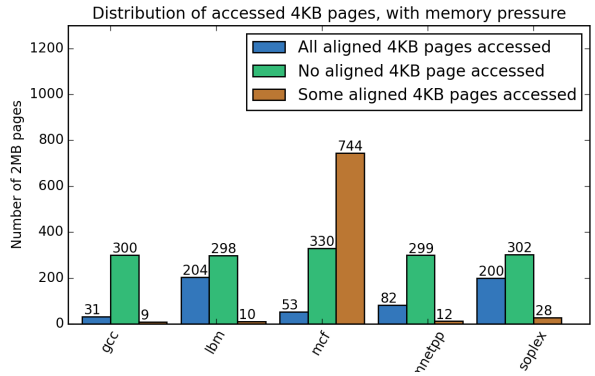

Fig. 12: Distribution of accessed 4KB pages, with 600MB of available memory

VI. Related Work

WSS estimation for the purposes of power savings through dynamic cache resizing in CMPs is discussed in [2]. Aparna et al consider the WSS on a cache line granularity, and propose the Tagged WSS Estimation (TWSS) technique. TWSS counts the number of recently accessed entries in the cache, by adding an active bit in each line and a counter in each L2 slice. The active bits are set on cache hits, and the counter tracks the active evicted lines using the tag addresses. The monitoring intervals are measured in terms of clock cycles. This approach incurs the hardware cost of adding an extra bit to the tag metadata, and a counter per L2 slice.

Another novel method of tracking the WSS with low overheads is proposed in [4]. The overheads of monitoring the working sets are minimized through intermittent tracking, with tracking disabled when steady memory behavior is seen. The page miss ratio is plotted against the amount of available memory to form the Miss Ratio Curve. The WSS is estimated from this curve as the minimum memory size corresponding to page miss ratio below a predetermined miss rate.

[9] utilizes regression analysis to compute the working set size of virtual machines. Such working set estimation facilitates accurate monitoring of memory pressure in virtualized environments, which is needed to enable high density of virtual

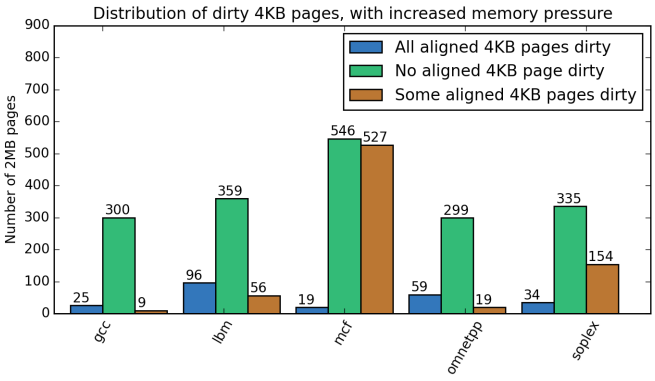

Fig. 13: Distribution of dirty 4KB pages, with 200MB of available memory

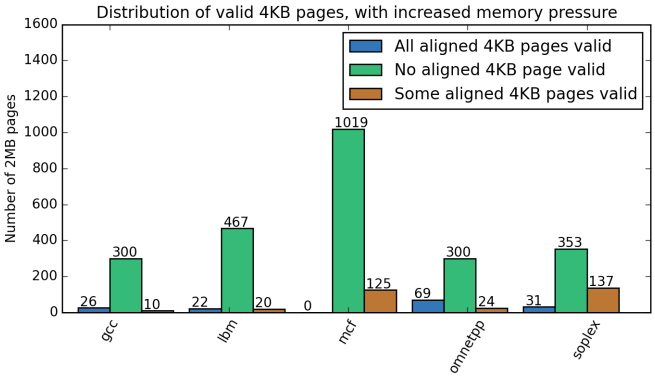

Fig. 14: Distribution of valid 4KB pages, with 200MB of available memory

machines while guaranteeing steady performance. This work seeks to correlate the real-time memory demands of a virtual machine with virtualization events such as page-in or writes to the cr3 register. A parametric regression model is built using independent predictors related to these virtualization events to predict the memory consumption of virtual machines.

Working set estimation is also discussed in [3], to reduce restore time for checkpointed memory. Partial lazy restore is done by fetching the page from the active working set of an application, before restoring the rest of the memory. Two techniques are proposed to construct the working set of an application. The first method involves periodically scanning and clearing the reference bits in the page tables, while the other traces memory access immediately following the saving of state.

VII. Conclusion, Limitations and Future Work

The working set size estimation of several memory-intensive workloads aligns well with the hypotheses made in Section III. These results also illustrate the dependence of working sets on the size of the observation window, and validate the constraints placed on window size in Denning's work in this domain [1]. The window size should be large enough to encompass all currently accessed pages, and small enough to counter the effects of program's phase changes.

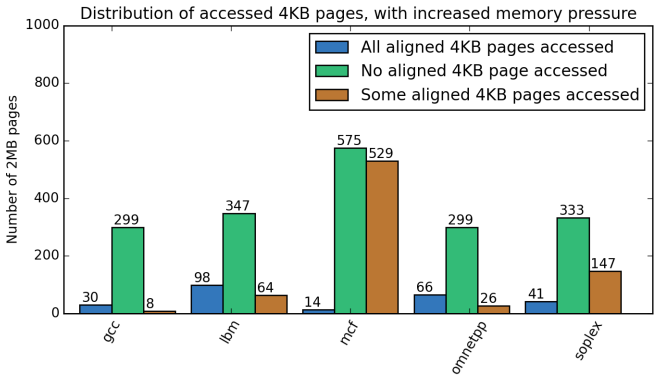

Fig. 15: Distribution of accessed 4KB pages, with 200MB of available memory

The study of the spatial locality of dirty and reference bits is still a work in progress. In this report, we analyzed the distribution of valid, dirty and referenced pages in the presence and absence of memory pressure. Initial observations suggest that there is significant spatial locality of dirty and reference bits without memory pressure. However, page swapping due to low memory availability disturbs this locality to some extent. We intend to extend these studies to other big-memory workloads, once the limitations in our analysis tools are rectified.

The modified version of BadgerTrap and our new tool, SwapTrap have only been tested with Linux Kernel v3.12.13. The modified BadgerTrap is prone to sporadic kernel crashes, particularly when changing between the three modes of WSS estimation. SwapTrap currently faces problem scanning the page tables of multi-threaded processes, and dumps results individually for each spawned process. All of these issues are being worked upon actively, and are expected to be fixed shortly. As expected, the modified version of BadgerTrap also suffers from the limitations of BadgerTrap itself, which are listed in [5].

Acknowledgment

The author would like to thank Professors Mark Hill and Michael Swift for their guidance through the course of this project. Jayneel Gandhi deserves a special mention for helping me settle into the Multifacet group, and for always being ready with valuable advice essential for forward progress. I would also like to thank Chris Feilbach for helping me set up the infrastructure needed for this independent study, and for letting me use his microbenchmark code. Finally, I thank Urmish Thakker and Lokesh Jindal for reviewing this report.

References

[1] P. J. Denning, "The working set model for program behavior," *Commun. ACM*, vol. 11, no. 5, pp. 323–333, May 1968. [Online]. Available: http://doi.acm.org/10.1145/363095.363141

[2] A. Dani, B. Amrutur, and Y. Srikant, "Toward a scalable working set size estimation method and its application for chip multiprocessors," *Computers, IEEE Transactions on*, vol. 63, no. 6, pp. 1567–1579, June 2014.

[3] I. Zhang, A. Garthwaite, Y. Baskakov, and K. C. Barr, "Fast restore of checkpointed memory using working set estimation," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '11. New York, NY, USA: ACM, 2011, pp. 87–98. [Online]. Available: http://doi.acm.org/10.1145/1952682.1952695

[4] W. Zhao, X. Jin, Z. Wang, X. Wang, Y. Luo, and X. Li, "Low cost working set size tracking," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 17–17. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002198

[5] J. Gandhi, A. Basu, M. D. Hill, and M. M. Swift, "Badgertrap: A tool to instrument x86-64 tlb misses," *SIGARCH Comput. Archit. News*, vol. 42, no. 2, pp. 20–23, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2669594.2669599

[6] R. Krishnakumar, "Hugetlb - large page support in the linux kernel," Oct. 2008. [Online]. Available: http://linuxgazette.net/155/krishnakumar.html

[7] R. H. E. L. Documentation, "Huge pages and transparent huge pages." [Online]. Available: https://access.redhat.com/documentation/

[8] T. K. Prakash and L. Peng, "Performance characterization of spec cpu2006 benchmarks on intel core 2 duo processor," in *ISAST Transactions on Computers and Software Engineering*.

[9] A. Melekhova, "Machine learning in virtualization: Estimate a virtual machine's working set size," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, ser. CLOUD '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 863–870. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2013.91