2014 47th Annual IEEE/ACM International Symposium on Microarchitecture

SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers

Yunqi Zhang Michael A. Laurenzano Jason Mars Lingjia Tang
Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{*yunqi, mlaurenz, profmars, lingjia*}*@eecs.umich.edu*

***Abstract*—One of the key challenges for improving efficiency in warehouse scale computers (WSCs) is to improve server utilization while guaranteeing the quality of service (QoS) of latency-sensitive applications. To this end, prior work has proposed techniques to precisely predict performance and QoS interference to identify 'safe' application co-locations. However, such techniques are only applicable to resources shared across cores. Achieving such precise interference prediction on real-system simultaneous multithreading (SMT) architectures has been a significantly challenging open problem due to the complexity introduced by sharing resources within a core.**

In this paper, we demonstrate through a real-system investigation that the fundamental difference between resource sharing behaviors on CMP and SMT architectures calls for a redesign of the way we model interference. For SMT servers, the interference on different shared resources, including private caches, memory ports, as well as integer and floating-point functional units, do not correlate with each other. This insight suggests the necessity of decoupling interference into multiple resource sharing dimensions. In this work, we propose `SMiTe`, a methodology that enables precise performance prediction for SMT co-location on real-system commodity processors. With a set of `Rulers`, which are carefully designed software stressors that apply pressure to a multidimensional space of shared resources, we quantify application *sensitivity* and *contentiousness* in a decoupled manner. We then establish a regression model to combine the *sensitivity* and *contentiousness* in different dimensions to predict performance interference. Using this methodology, we are able to precisely predict the performance interference in SMT co-location with an average error of 2.80% on SPEC CPU2006 and 1.79% on CloudSuite. Our evaluation shows that `SMiTe` allows us to improve the utilization of WSCs by up to 42.57% while enforcing an application's QoS requirements.

Keywords-quality of service; simultaneous multithreading; datacenter; warehouse scale computer

I. Introduction

The geometric growth of computation in the cloud drives rapidly increasing costs in building and operating warehouse scale computers (WSCs). Unfortunately, most WSCs operate at fairly low utilization, often below 30% [1], which translates to low efficiency and high total cost of ownership (TCO). This inefficiency can be significantly improved by allowing co-locations of multiple applications on individual servers to share hardware resources. However, resource sharing introduces varying amounts of performance interference among applications. Therefore, the performance predictability is negatively affected, posing critical challenges for guaranteeing that user-facing latency-sensitive workloads such as web search can meet their strict quality of service (QoS) targets.

Recently, prior work has proposed techniques to precisely predict the QoS interference among co-located applications on chip multiprocessor (CMP) servers [2–6]. Based on this precise interference prediction, WSC cluster schedulers can identify 'safe' co-locations that bound performance degradation while improving server utilization. However, prior techniques only focus on predicting the interference caused by resource sharing across cores on a multicore processor. Despite the ubiquitous presence of simultaneous multithreading (SMT) processors [7, 8] in WSCs, an approach to perform precise interference prediction on real-system SMT processors has been an open problem.

Realizing precise prediction for SMT co-locations in addition to CMP co-locations is a particularly challenging problem due to significantly more complex interactions between shared resources within the core and in the uncore. In addition to the last-level cache (LLC) and memory bandwidth, which are shared across CMP cores, SMT cores provide much finer granularity resource sharing on core, among hardware contexts. The additional shared resources include private cache(s), memory ports, as well as integer and floating-point functional units. This fine-grained sharing across a large number of resources leads to greater performance variability and unpredictability. In addition, there is diversity in how resources are shared, which further increases the difficulty of precise interference prediction. For example, a functional unit cannot be shared concurrently by multiple hardware contexts, however a cache's capacity can be shared simultaneously. In this paper, we demonstrate that the difference between CMP and SMT resource sharing calls for a fundamental redesign in the way we model interference.

This paper first presents a real-system investigation to better understand how applications interfere on commodity SMT multicore processors. From the investigation, we have gained several insights that guide the design of an SMT interference prediction methodology.

- Firstly, across various shared resources (caches, functional units, memory ports, etc.), contention on each individual resource alone can cause significant performance degradation and the amounts of degradation exhibit high variability across applications and resources.
- Secondly, there is little correlation among application

1072-4451/14 $31.00 © 2014 IEEE
DOI 10.1109/MICRO.2014.53

contention characteristics for different shared resources. For example, an application being sensitive to contention for data caches does not necessarily mean that it is less (or more) contentious (or sensitive to contention) for the floating-point functional unit.

These observations indicate that a holistic approach such as those used in prior work for interference prediction on CMP servers is not suitable for SMT. For example, Bubble-Up [4] relies on a single monotonic metric to quantify interference in all shared resources, which fails to capture the multidimensionality of the resource sharing behavior on SMT. We must redesign a methodology to model interference for SMT co-locations in a manner that the sharing behavior is decoupled along multiple dimensions of various types of resources.

Based on these observations, we design `SMiTe`, a methodology that enables precise performance prediction on real-system SMT processors. `SMiTe` leverages a carefully designed suite of software stressors, called `Rulers`, to characterize an application's contention nature for each shared resource. Each `Ruler` in the suite is designed to maximize the pressure on one specific resource while minimizing the pressure on all other resources. By co-locating one application with a `Ruler`, we measure the performance degradation of the application as its *sensitivity*, and the performance degradation of the `Ruler` as the application's *contentiousness* on the corresponding resource. A regression model is then established, using application's *sensitivity* and *contentiousness* for different resources to precisely predict the performance interference in SMT co-locations. Based on the precise prediction, we are able to steer the cluster-level job scheduler to make co-location decisions that improve the utilization of a WSC without violating QoS requirements.

This is the first work that enables precise performance interference prediction on real-system SMT multicore processors to provide QoS-awareness and utilization improvement in WSCs. Specifically, this paper makes the following contributions:

- **In-depth Analysis of Performance Interference on Real-System SMT Processor –** Our investigation demonstrates the low correlation among application contention characteristics for various shared resources, which motivates our design of a multidimensional modeling methodology for SMT co-locations.
- **Design of Rulers to Decouple Interference Modeling –** We present `Rulers`, carefully designed software stressors to put maximum amount of pressure on each individual resource while incurring minimum pressure on other resources. These `Rulers` allow us to capture an application's *sensitivity* and *contentiousness* for each shared resource in a decoupled manner.
- **Precise Prediction of Performance Interference for SMT Co-locations –** We propose `SMiTe`, a methodology to establish a prediction model using `Ruler` measurements to combine the *sensitivity* and *contentiousness* characteristics of each application to precisely predict performance interference on multicore SMT processors.
- **Scale-out Study –** Through a scale-out study that uses `SMiTe`'s prediction of the QoS interference to steer cluster-level scheduling decisions, we demonstrate the effectiveness of `SMiTe` methodology in enabling co-locations to achieve higher server utilization.

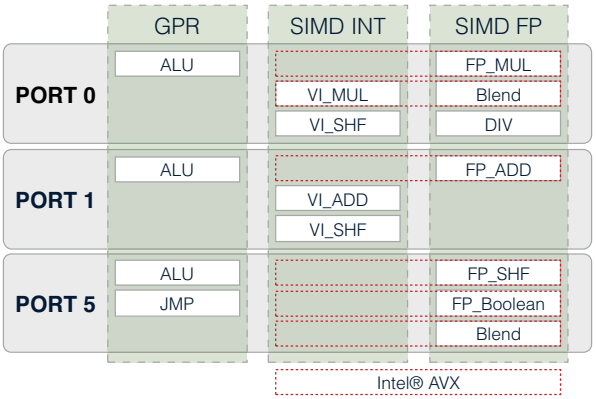

Figure 1. Execution cluster of Intel Sandy Bridge microarchitecture. Multiple operations are port-specific (e.g. `FP_MUL` can only execute on port 0).

Our evaluation demonstrates that `SMiTe` is able to precisely predict the performance interference caused by SMT co-locations. On average, `SMiTe` achieves 2.80% prediction error on SPEC CPU2006 [9], and 1.79% error on CloudSuite [10] workloads on Intel Sandy Bridge and Ivy Bridge servers. In addition, our scale-out study shows that `SMiTe`-steered QoS-aware cluster scheduling achieves utilization improvements ranging from 9.24% at 95% QoS target to 42.57% at 85% target.

II. Real-System Investigation

In contrast to CMP co-locations where only last-level cache (LLC) and memory bandwidth are shared among different cores, hardware contexts co-located on the same SMT core share a much wider range of resources including both functional units and the memory subsystem. In this section, we present an investigation to better understand application sharing behavior on these various resources and the resulting performance interference on commodity multicore SMT processors.

A. Experimental Methodology

One main difference between CMP co-locations and SMT co-locations is whether on-core resources are shared. In an Intel Sandy Bridge [11] processor, these resources are implemented as an execution cluster composed of 6 ports that perform sets of different operations. As illustrated in Figure 1, ports 0, 1 and 5 are used for functional units, whereas ports 2, 3 and 4 (not shown) are for memory accesses. Note that, in this design, there are many port-specific operations that can only execute on specific port(s).

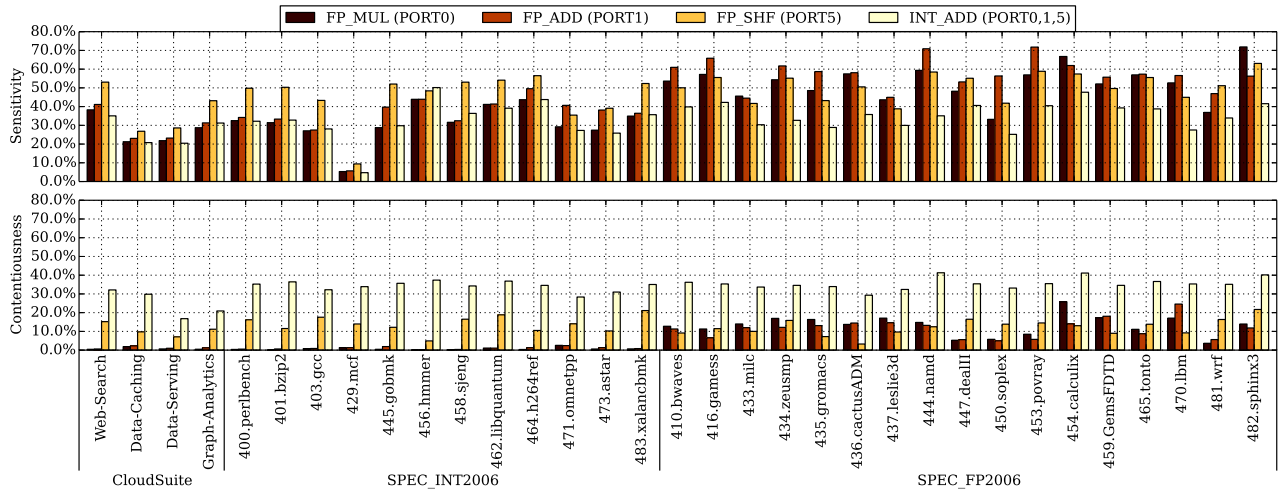

Figure 2. The *sensitivity* and *contentiousness* of different workloads on functional unit resources.

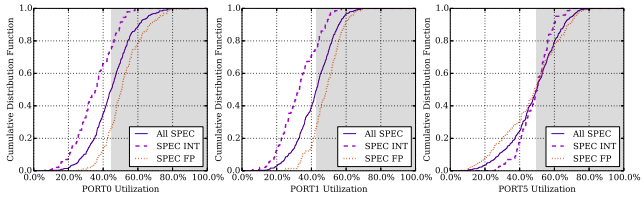

Figure 3. Aggregated functional unit utilization distributions across all the co-location pairs in SPEC CPU2006 for ports 0, 1 and 5.

For example, `FP_MUL` can only execute on port 0, `FP_ADD` on port 1, `FP_SHF` on port 5 and `INT_ADD` on ports 0, 1 and 5.

Taking advantage of these port-specific operations, we carefully designed a set of stressors to study application *sensitivity* and *contentiousness* on various functional units for SMT co-locations. In addition to functional units, we also designed a set of memory stressors to study the interfering behavior on various levels of cache. The design principles and details behind these stressors (`Rulers`) are presented in Section III-B1.

B. Contention for Functional Units

Modern microarchitectures often include a large number of functional units to exploit instruction-level parallelism (ILP). As illustrated in Figure 1, each functional unit is usually designed to only execute certain types of operations. When investigating the interference due to sharing functional units between multiple hardware contexts co-located on an SMT core, we aim to answer the following questions:

- What is the amount of performance degradation caused by contention for each type of functional unit?
- Are applications' *sensitivity* and *contentiousness* for the same resource correlated? The answer to this question will indicate whether they need to be modeled separately.
- What is the variability of an application's contention characteristics across different functional units? The answer to this would indicate whether we need to characterize each shared resource separately or one single unified metric is sufficient for all resources.
- Do emerging WSC workloads (e.g. CloudSuite) behave differently from traditional workloads (e.g. SPEC CPU2006) in terms of functional unit contention?

Functional Unit Contentiousness and Sensitivity - Figure 2 shows applications' *sensitivity* and *contentiousness* measured by a set of `Rulers`, each `Ruler` maximizing the pressure in one specific functional unit resource, including `FP_MUL` at port 0, `FP_ADD` at port 1, `FP_SHF` at port 5 and `INT_ADD` spreading across ports 0, 1, 5. We quantify an application's *sensitivity* as the degradation it suffers from co-locating with `Rulers`, while *contentiousness* is defined as the degradation it causes to the `Rulers`. Our findings are as follows:

- *Finding 1. Applications in general are sensitive to functional unit contention.* As shown in Figure 2, applications suffer 5% - 70% performance degradation when contending for only one type of functional unit.
- *Finding 2. The level of sensitivity to contention for each functional unit varies across applications*. For example, `429.mcf` suffers 6% performance degradation due to port 1 contention, while `444.namd` suffers as high as 71% degradation.
- *Finding 3. Sensitivity and contentiousness of each application for each shared resource do not correlate with each other, and thus need to be captured separately.*
- *Finding 4. Each application has various levels of sensitivity and contentiousness for different functional units*. For example, `454.calculix` is more contentious to port 0 while `470.lbm` is more contentious to port 1. *This suggests the need to capture the contention characteristics for each functional unit separately.*

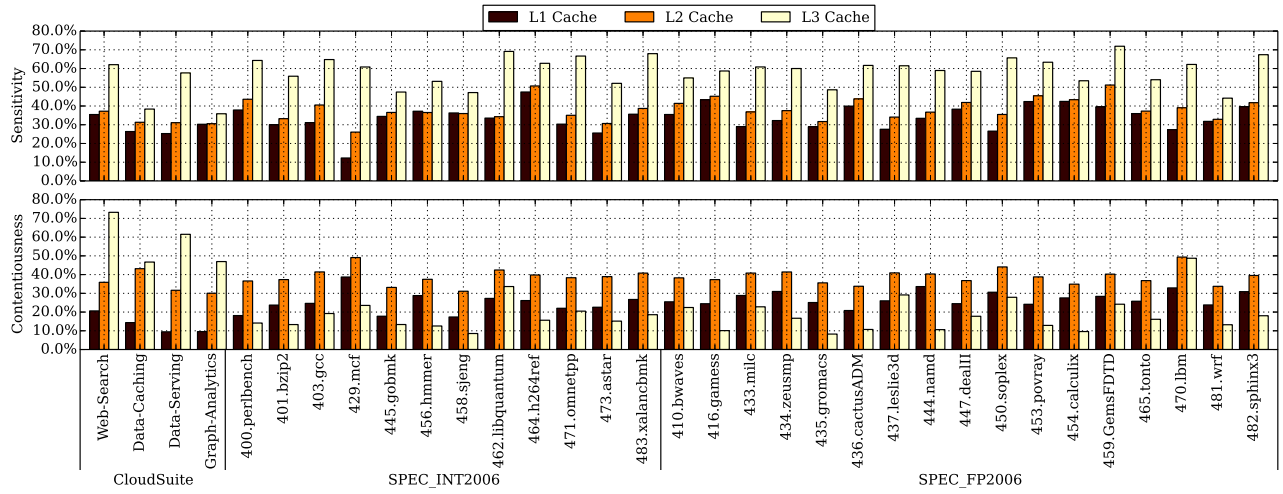

Figure 4. The *sensitivity* and *contentiousness* of different workloads on memory subsystem resources.

- *Finding 5. Emerging WSC workloads' contention behaviors for functional units are similar to SPEC_INT benchmarks.*

Due to the variability across applications and across each type of functional unit, we conclude that an ideal interference model needs to capture application *contentiousness* and *sensitivity* separately along each resource sharing dimension.

Functional Unit Utilization - In addition to the *sensitivity* and *contentiousness*, we also profile the utilization of various functional units when applications co-locate on an SMT core using hardware performance monitoring units (PMUs). Figure 3 presents the cumulative distribution function (CDF) for the utilization of ports 0, 1, and 5 respectively, across all pairs of co-located applications. In Figure 3, utilization is measured as the aggregated utilization of two co-located applications on an SMT core, where the shaded area illustrates the percentage of the co-located pairs that have higher utilization than the median. As shown in the figure, SPEC_FP benchmarks tend to have higher utilization for ports 0 and 1 than SPEC_INT. On the contrary, for port 5, SPEC_INT has higher utilization, and this is due to the higher branch instruction counts in SPEC_INT, which are executed on port 5.

- *Finding 6. Ports 0 and 1 have similar utilization distributions, which are distinctly different from the utilization distribution of port 5.* This also indicates that applications' contention behaviors at different functional units need to be measured separately to capture the variability across ports.

C. Interference in Memory Subsystem

In addition to functional units, private caches (L1 and L2), shared LLC and memory bandwidth are also shared among SMT contexts. Compared to CMP co-locations, sharing private caches adds additional complexity to the SMT co-locations.

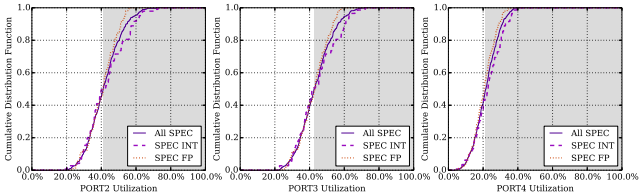

Figure 5. Aggregated memory port utilization distributions across all the co-location pairs in SPEC CPU2006.

Memory Subsystem Contentiousness and Sensitivity - Figure 4 presents application *sensitivity* and *contentiousness* measured using a set of memory `Rulers` co-located with the applications. We design our L1 and L2 cache `Rulers` to be the same binary with different working set sizes. As we increase the working set size, there is a monotonic increase in the `Ruler`'s performance impact.

- *Finding 7. Contention behaviors in the memory subsystem are more monolithic than functional units, demonstrating the basis for prior work to quantify the memory subsystem pressure using a unified metric. In addition, there is noticeable variability across applications.* Some applications' performance heavily relies on one specific cache level. For example, applications such as `454.calculix` have very similar *sensitivity* to contention in L1 and L2 caches, which indicates their high reliance on the L1 cache and low utilization of the L2 cache.
- *Finding 8. CloudSuite workloads are much more contentious at the L3 cache than SPEC applications, although they exhibit very similar levels of sensitivity across three levels of cache.*

Memory Port Utilization - Similarly, we also profile the aggregated utilization for memory ports across all the co-location pairs in SPEC as shown in Figure 5, in which port 2 and port 3 are used for memory loads and port 4 for memory stores. In the figure, we find that memory store port (port 4) is heavily underutilized, compared to the load ports. This is supplementary to our finding 6 that applications' behaviors across ports vary, and thus need to be captured separately.

D. Correlation Among Sharing Dimensions

We summarize our measurement of application *sensitivity* and *contentiousness* for different resources in Figure 6. As illustrated in the figure, *there is a large variance in sensitivity and contentiousness in each sharing dimension across applications.* For example, application *sensitivity* to port 0 or port 1 ranges from negligible to above 70%. However, *on average, contention in each dimension can cause significant interference and thus all these dimensions need to be considered in our interference model*. In addition, *there is also a significant difference in terms of contention behaviors across different sharing dimensions*. For example, applications are more sensitive to contention for port 5 because branch instructions, which can only be executed on port 5, are critical for performance. In addition, most applications tend to generate more pressure on L2 than on L1 and L3 caches.

Furthermore, we quantify the correlation among different sharing dimensions using Pearson correlation coefficients. The absolute Pearson correlation coefficients of the *contentiousness* and *sensitivity* across all benchmarks in each of the 7 shared resources are presented in Figure 7. The absolute value of Pearson correlation coefficient ranges from 0 to 1, where 1 indicates the perfect correlation (both positive and negative) and 0 indicates no correlations.

- *Finding 9. As demonstrated in this figure, there is little correlation for application sensitivity and contentiousness among different sharing dimensions.* For example, as shown in the figure, an application being sensitive to contention for caches does not necessarily mean that it is less (or more) contentious or sensitive to contention for the floating-point functional unit. In fact, 97.96% of the pairs of sharing dimensions have a correlation coefficient lower than 0.80, and for the majority of the pairs the coefficient is lower than 0.50. These low correlations further suggest the necessity to decouple and measure each sharing dimension separately.

III. SMiTe Methodology

In this section, we present `SMiTe` methodology. Designed based on the findings summarized in Section II, `SMiTe` enables precise performance prediction for SMT co-locations on real-system multicore processors.

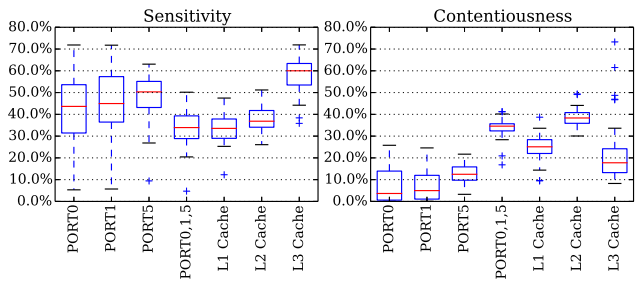

Figure 6. The *sensitivity* and *contentiousness* of all SPEC CPU2006 and CloudSuite applications. Applications' contention characteristics have a large variance both for the same resource and across different resources.

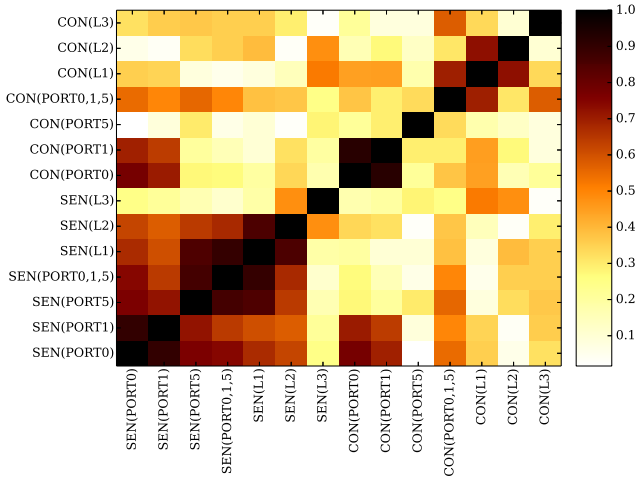

Figure 7. The absolute values of Pearson correlation coefficient among all the *sensitivity* and *contentiousness* dimensions. 97.96% of the pairs have a correlation coefficient lower than 0.80 and the majority of the pairs, lower than 0.50.

A. Overview

The overview of the `SMiTe` methodology is presented in Figure 8, which consists of three main steps.

1) **Characterizing Sensitivity and Contentiousness (Section III-B) –** For each application, we quantify its contention characteristics for shared SMT resources. A set of `Rulers` is designed to sense an application's *sensitivity* and *contentiousness* along various sharing dimensions, including functional units and the memory subsystem. By co-locating the application of interest with a `Ruler`, we measure the application's performance degradation as its *sensitivity* to contention in the corresponding sharing dimension, and the degradation of the `Rulers` as the application's *contentiousness* in the same dimension.
2) **Performance Prediction Model (Section III-C) –** To predict the performance interference between applications when they co-locate on an SMT core or CMP cores, we establish a regression-based prediction model, which combines each application's multidi-

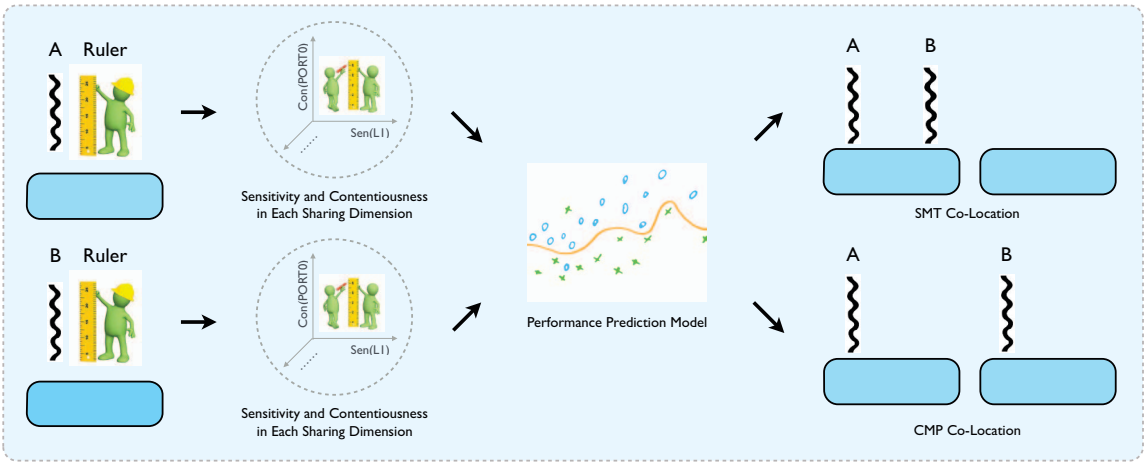

Figure 8. Overview of SMiTe methodology. Based on our insight that there is little correlation among applications' contention characteristics across multiple resource sharing dimensions, we design a set of Rulers to quantify an application's *sensitivity* and *contentiousness* in a decoupled manner (e.g., in each sharing dimension). A regression-based prediction model is then established to use an application's *sensitivity* and *contentiousness* characterizations to make performance prediction for SMT and CMP co-locations.

mensional characteristics quantified by the Rulers to precisely predict the performance degradation in both CMP and SMT co-location scenarios.

3) **Steering towards Safe Co-locations (Section III-D) –** SMiTe allows us to quickly profile an application, and precisely predict the level of performance degradation that applications may suffer from the co-location. With this prediction ability, a cluster scheduler in a WSC can identify 'safe' job co-locations that would not violate applications' QoS requirements, achieving high server utilization.

B. Quantifying Sensitivity and Contentiousness

In order to make precise performance predictions, we first characterize an application's *sensitivity* and *contentiousness*. Several key factors determine the characterization quality, including our Rulers design and the methodology to quantify the *sensitivity* and *contentiousness*.

1) Ruler Design: We design a set of Rulers to sense an application's interfering behavior in each sharing dimension in a decoupled manner. A good Ruler design needs to maximize the measurement accuracy while minimizing the profiling overhead. Here are two key principles that guide our Ruler design:

- **Each Ruler needs to maximize the pressure in the targeted sharing dimension while minimizing the impact in all other dimensions.** For example, a Ruler that targets port 0 needs to achieve maximum pressure on that port while minimizing its pressure on other functional units and the memory subsystem. As demonstrated in Figure 7, there is little correlation across all sharing dimensions. Therefore, minimizing the overlapping resources that each Ruler stresses helps decouple the interfering behaviors into independent dimensions.
- **A linear relationship between the intensity of the Ruler and the amount of interference it causes on the corresponding resource is desirable.** To characterize an application's *sensitivity* to contention for a given resource, we need to measure its performance degradation under a range of pressure intensities generated by the Ruler. Having a linear relationship between the

```
loop:
    mulps   %xmm0, %xmm0
    ......
    mulps   %xmm7, %xmm7
    ......
    jmp loop
```
(a) FP_MUL (PORT0)

```
loop:
    addps   %xmm0, %xmm0
    ......
    addps   %xmm7, %xmm7
    ......
    jmp loop
```
(b) FP_ADD (PORT1)

```
loop:
    shufps  %xmm0, %xmm0
    ......
    shufps  %xmm7, %xmm7
    ......
    jmp loop
```
(c) FP_SHF (PORT5)

```
loop:
    addl    %eax, %eax
    ......
    addl    %edx, %edx
    ......
    jmp loop
```
(d) INT_ADD (PORT0,1,5)

```
#define MASK 0xd0000001u
#define RAND (lfsr = (lfsr >> 1) ^ (unsigned int)(0 - (lfsr & 1u) & MASK))
......
    while (1) {
        data_chunk[RAND % FOOTPRINT]++;
        ......
        data_chunk[RAND % FOOTPRINT]++;
    }
```
(e) MEM (L1, L2 Cache)

```
......
    first_chunk = data_chunk;
    second_chunk = data_chunk + FOOTPRINT / 2;
    while (1) {
        for (i = 0; i < FOOTPRINT / 2; i += 64) {
            first_chunk[i] = second_chunk[i] + 1;
        }
        for (i = 0; i < FOOTPRINT / 2; i += 64) {
            second_chunk[i] = first_chunk[i] + 1;
        }
    }
```
(f) MEM (L3 Cache)

Figure 9. Implementation of Rulers.

intensity and the resulting interference is highly useful for reducing the profiling overhead. Instead of profiling the entire *sensitivity* curve by sampling the degradation under various intensity points, a linear relationship requires only two samples at both end points of the *sensitivity* curve.

It is very challenging to achieve these principles on real-system SMT processors due to the complexity in a commodity processor. Here we present our carefully designed `Rulers`.

Functional Unit `Rulers` - As presented in Figure 9(a-d), in order to design decoupled `Rulers` that stress each resource independently, we design our functional unit `Rulers` using port-specific instructions [11] (see Figure 1). In addition, we remove all data dependencies between consecutive instructions and unroll the loops to maximize the functional unit utilization. By doing so, we achieve higher than 99.99% utilization for the targeted resource, validated using the hardware performance counters `UOPS_DISPATCHED_PORT:PORT0,1,5`. In addition, this design allows us to achieve the desirable linear relationship between the `Ruler` intensity and the interference it causes, because the intensity of our functional unit `Ruler` directly translates to the port utilization. Note that because specialized functional units are commonly used in modern processors, the design principle of the port-specific functional unit `Ruler` can be applicable to other microarchitectures such as IBM Power7 [12].

Memory Subsystem `Rulers` - Compared to functional unit `Rulers`, it is more difficult to completely decouple the interference in the memory subsystem because multiple levels of caches can be inclusive. In addition, to issue memory accesses, a certain amount of computation is unavoidable. Thus, we design our `Rulers` to maximize the pressure on the targeted cache level as an approximation, and rely on the regression-based prediction model to decouple the overlapping impact.

As shown in Figure 9(e), the L1 and L2 cache `Rulers` randomly access a chunk of data using a lightweight random number generator: linear-feedback shift register (LFSR). For the L3 cache `Ruler` as shown in Figure 9(f), we use stride access with a 64-byte offset, the size of the cache line, to maximize the amount of pressure. For both designs, we also unroll the loops to minimize the number of branch instructions. The intensity of our memory subsystem `Ruler` is defined as the working set size of each `Ruler`. We measure the average Pearson correlation coefficient between the working set size of our `Ruler` at each cache level and the performance degradation of all SPEC applications when co-located with the `Ruler`, and we observe strong linear correlations. The Pearson coefficients are 0.92 for L1, 0.89 for L2 and 0.95 for L3 cache. This linear relationship significantly reduces our profiling overhead, because the entire *sensitivity* curve for all working set sizes can be accurately approximated by interpolating between 3 `Rulers` whose working set sizes being the L1, L2 and L3 cache sizes.

2) Characterizing Contentiousness and Sensitivity: To quantify an application's *sensitivity* and *contentiousness*, we co-locate the application with the `Rulers` on the neighboring hardware context on an SMT core. For each resource i, we measure the application A's performance degradation as its *sensitivity* Sen_i^A via the following equation:

$$Sen_i^A = \frac{IPC_{solo}^A - IPC_{co-location/Ruler_i}^A}{IPC_{solo}^A} \quad (1)$$

Similarly, we define application A's *contentiousness* Con_i^A as the corresponding `Ruler`'s performance degradation.

$$Con_i^A = \frac{IPC_{solo}^{Ruler_i} - IPC_{co-location/A}^{Ruler_i}}{IPC_{solo}^{Ruler_i}} \quad (2)$$

C. Performance Prediction Model

1) Prediction Model: After characterizing each application, to predict the performance degradation of application A when co-located with application B on an SMT core, we combine both A's *sensitivity* and B's *contentiousness* on each sharing dimension i, using a linear model. The prediction model is shown in Equation 3.

$$Deg_{co-locate/B}^A = \sum_i^N (c_i \times Sen_i^A \times Con_i^B) + c_0 \quad (3)$$

In this model, the degradation for A in each dimension is proportional to measured application A's *sensitivity* and the co-located application B's *contentiousness* on that dimension. The linear model reflects the assumption that an application's performance degradation from each shared dimension is additive. The amount (weight) that each sharing dimension contributes to the total performance degradation is captured by the coefficient c_i. The constant term c_0 is introduced to approximate the performance interference caused by other resources not captured in the model. A constant is used because the impact of other resources should have a small variance across applications, based our assumption that functional units and memory subsystem are the main contributors for the degradation.

2) Manage Prediction Error: There are two main sources of potential prediction errors. Firstly, the model can only capture the interference in a limited number of dimensions. Other shared resources such as the branch predictor might also cause performance interference, which are approximated by the constant c_0 in our model. Secondly, in order to reduce the profiling overhead, we take advantage of the approximately linear relationship between the intensity of a `Ruler` and the performance interference. This approximation might introduce errors in performance prediction. However, as we will show in our evaluation (Section IV), our model achieves high precision, demonstrating that the model has captured the significant resource dimensions.

3) Predicting Tail Latency: In addition to the average performance, many modern web service workloads in WSCs have certain requirements on the percentile latency, often the tail latency [13]. For example, QoS requirements can be specified as 90% of the queries need to achieve under-100ms latency. In addition to service time, the time a query waits in the job queue before it gets processed also contributes to the latency. Thus, the percentile latency does not linearly correlate with the average performance due to this queueing effect, and needs to be modeled differently on top of the average performance prediction calculated using Equation 3.

To address this, we model the web service workload using a simple first-come first-served (FCFS) `M/M/1` queueing system [14], which has a closed-form solution. We use the `M/M/1` model based on two observations:

- Both the service time distribution and the inter-arrival distribution usually have small coefficients of variance in practice. This indicates that we can approximate these distributions using the exponential distribution and Poisson distribution, respectively, without losing much precision [15].
- The queueing and the processing usually happen at the same level (e.g. a per thread queueing strategy often implies that each job in the queue is handled by one thread), which indicates that we can model the system with a single-server model [16]. For example, rather than having a global queue for all the worker threads, each thread has its own processing queue in `Memcached`. This allows us to model the response time distribution using the single-server model, because we are essentially just instantiating multiple copies of a single-server queueing system, one copy per worker thread.

In FCFS `M/M/1` queueing model, the response time probability density function (PDF), $f(t)$, can be modeled as shown in Equation 4, in which λ is the mean value of the arrival rate distribution $Poisson(\lambda)$, and μ is the average rate of the servicing time distribution $Exp(\mu)$.

$$f(t) = (\mu - \lambda)e^{-(\mu-\lambda)t} \tag{4}$$

Based on the average performance degradation (Deg) in Equation 3, we can extrapolate the degraded average service rate μ'.

$$\mu' = (1 - Deg)\mu \tag{5}$$

Taking the integral of the PDF in Equation 4, we can calculate the cumulative distribution function (CDF) of the response time. Using the inversion of the CDF and combining it with the degraded service time estimation in Equation 5, we estimate the p-th percentile latency t_p with Equation 6.

$$t_p = -\frac{ln(1-p)}{(1 - Deg)\mu - \lambda} \tag{6}$$

D. SMiTe in Action

With the ability to precisely predict the average performance and percentile latency interference, `SMiTe` can identify 'safe' co-locations of applications so that the QoS interference for latency-sensitive applications due to co-locations is under a given threshold. The advantages of `SMiTe` over exhaustive pairwise offline profiling are twofold: 1) `SMiTe` characterizes each application individually once and uses the characterization for performance prediction. This allows the WSC operators to avoid the complexity of cross-product characterization for all possible co-locating applications. Many latency-sensitive applications in WSCs are long running and well suited for the type of profiling [4]. 2) In addition to much more efficient offline profiling, `SMiTe` has carefully controlled profiling complexity so that each application's characterization can be completed in the order of seconds. This allows us to conduct quick online profiling for any new application when it arrives at the cluster-level scheduler before getting scheduled to a suitable server.

IV. Evaluation

A. Experimental Setup

Table I. Machine specifications in our experimental setup.

Processor	Microarchitecture	Kernel
Intel Xeon E5-2420 @ 1.90GHz	Sandy Bridge-EN	3.8.0
Intel i7-3770 @ 3.40GHz	Ivy Bridge	3.8.0

We evaluate our `SMiTe` methodology on two commodity multicore SMT processors summarized in Table I. The Linux `perf` tool is used to measure the hardware performance monitoring units (PMUs). We use CloudSuite [10] and SPEC CPU2006 [9] with `ref` inputs as our workloads. To construct the training and testing sets for our prediction model, we divide 29 SPEC benchmarks into 2 sets based on their even/odd numbering. Four applications from CloudSuite, including `Web-Search`, `Data-Caching`, `Data-Serving` and `Graph-Analytics`, are used to represent latency-sensitive workloads in modern WSCs.

Throughout this section, the performance degradation caused by the co-location $Deg_{co-location}$ is defined as in Equation 7, where IPC_{solo} is the instructions per cycle (IPC) measurement when the application is running alone and $IPC_{co-location}$ is the IPC when co-located with other applications.

$$Deg_{co-location} = \frac{IPC_{solo} - IPC_{co-location}}{IPC_{solo}} \tag{7}$$

The performance prediction error is reported as the absolute error between the measured performance degradation and the predicted degradation as shown in Equation 8.

$$Error = \left| Deg^{predicted}_{co-location} - Deg^{actual}_{co-location} \right| \tag{8}$$

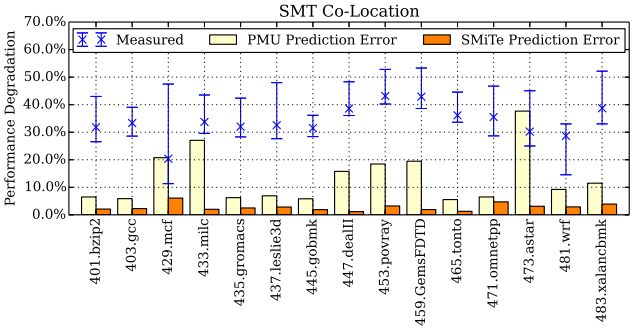

Figure 10. Performance prediction accuracy for SMT co-location on SPEC CPU2006 benchmarks, where the average prediction error of PMU based approach is 13.55% and SMiTe is 2.80%.

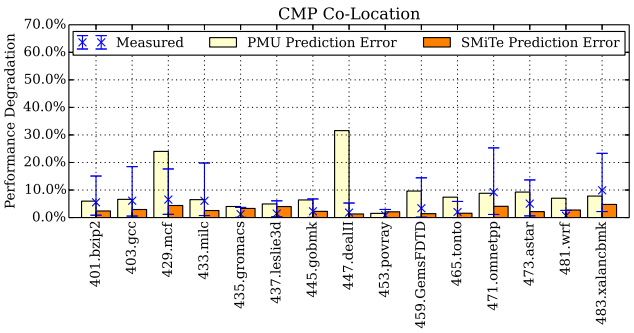

Figure 11. Performance prediction accuracy for CMP co-location on SPEC CPU2006 benchmarks, where the average prediction error of PMU based approach is 9.43% and SMiTe is 2.80%.

B. Performance Interference Prediction

1) SMiTe Prediction for General Purpose Workloads: We first investigate the prediction accuracy of SMiTe using SPEC benchmarks on an Intel Ivy Bridge server. All even-numbered benchmarks are used as training set for the performance prediction model, and odd-numbered benchmarks as testing set. The model is trained using the *sensitivity* and *contentiousness* measurements of each application as well as the performance degradation profiling of each co-locating pairs in the training set. Then the trained prediction model takes application's *sensitivity* and *contentiousness* from the testing set to predict the performance degradation of co-locating pairs.

In this experiment, we also compare our Ruler based prediction model against PMU based models. PMU based models have been commonly used for scheduling optimization [17] and power modeling [18] on SMT processors. Since there is no prior work providing techniques for precise performance prediction on real-system SMT processors, we carefully designed several PMU based model for predicting performance and selected the best one as our baseline to evaluate the viability of a PMU based model. Specifically, after experimenting with a number of PMUs and various regression strategies including linear regression, decision tree, higher order polynomial regression, we found the best performing model to be a linear regression model using 11 PMU measurements: *instructions/cycle*, *iTLB-misses/cycle*, *dTLB-load-misses/cycle*, *dTLB-store-misses/cycle*, *i-cache-misses/cycle*, *L1D-hits/cycle*, *L2-hits/cycle*, *L2-misses/cycle*, *L3-hits/cycle*, *MEM-hits/cycle*, *branch-mispredictions/cycle*. The linear regression is established using Equation 9 to predict the performance degradation on SMT and CMP co-locations.

$$Deg^{A}_{co-locate/B} = \sum_{i}^{N}(c^{A}_{i}PMU^{A}_{i} + c^{B}_{i}PMU^{B}_{i}) + c_0 \quad (9)$$

The prediction accuracy of SMiTe and the PMU based model is reported in Figure 10 for SMT co-locations and Figure 11 for CMP co-locations. As shown in the figure, the average measured performance degradations of each benchmark when co-located span a wide range, from 11.74% to 53.14%. In the figure, the bars labeled as PMU Prediction Error present the average prediction error of the PMU based prediction model when each benchmark co-locates with all the other benchmarks in the testing set. The average error for the PMU based model is 13.55% for SMT co-locations and 9.43% for CMP co-locations. Compared to PMU based approach, SMiTe provides significantly higher precision, predicting both SMT and CMP co-locations with an average error of 2.80%.

2) SMiTe Prediction for Cloud Workloads: In this section, we evaluate our methodology on CloudSuite benchmarks, which are used to represent latency-sensitive applications running in modern WSCs.

In contrast to the SPEC workloads, Cloudsuite applications are usually multithreaded and span more than one core. Thus, we set up this experiment differently on our Sandy Bridge-EN machine, which has 6 cores with 12 SMT hardware contexts on each socket. To half load the server as a baseline, we configure the cloud applications to run with 6 threads for SMT co-location experiment such that each core has one SMT context busy and the other one idle. Similarly, 3 threads are used for the CMP co-location experiment with 3 out of 6 cores are left completely idle as the baseline. Accordingly, we use 6 instances of the same Ruler for SMT experiment and 3 instances for CMP experiment when measuring the *sensitivity* and *contentiousness* of the cloud applications. We use odd-numbered benchmarks from SPEC as the training set and even-numbered benchmarks as the testing set. Both PMU based and SMiTe prediction models are trained using the SPEC training set, and tested on co-locations between CloudSuite applications as latency-sensitive applications and SPEC testing set as batch applications.

The prediction errors for both SMiTe and PMU based approaches are presented in Figure 12. The bars labeled as Measured present the maximum, average and minimum measured performance degradation, ranging from co-locating with 1 instance to 6 instances of the batch applications (x-

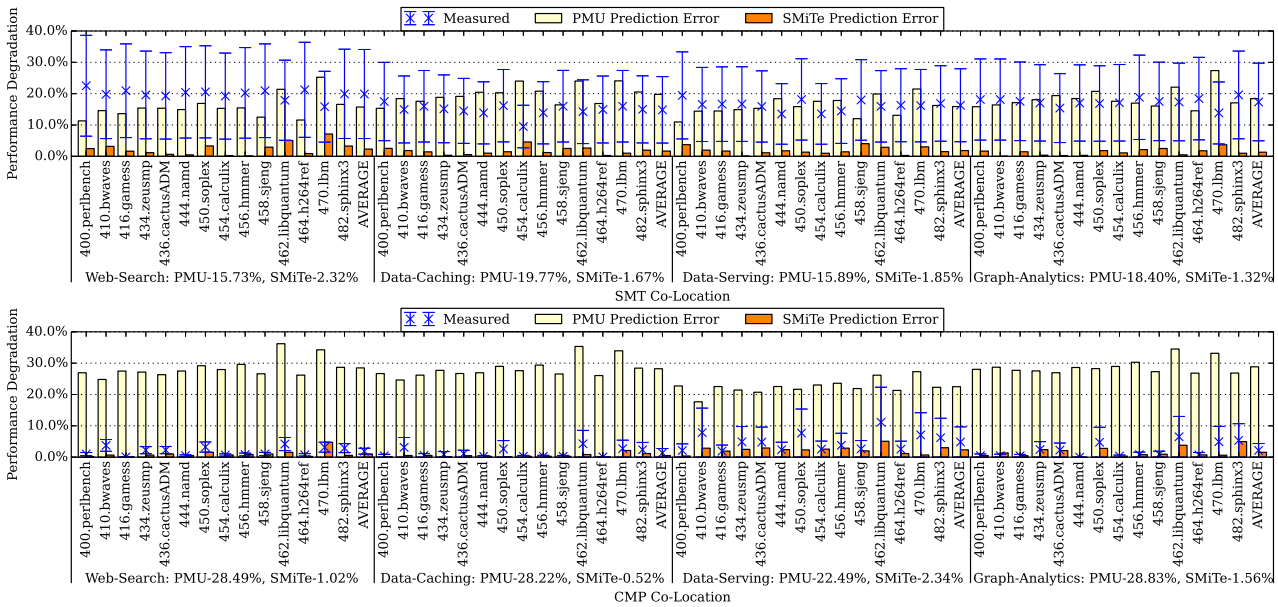

Figure 12. Performance prediction accuracy for SMT and CMP co-location on CloudSuite benchmarks (`Web-Search`, `Data-Caching`, `Data-Serving` and `Graph-Analytics`), where the average prediction error of PMU based approach is 17.45% for SMT co-location, 27.01% for CMP co-location and `SMiTe` is 1.79% and 1.36% respectively.

axis) for SMT co-locations, and 1 to 3 instances for CMP co-locations. As shown in the figure, the PMU based model has an average prediction error of 17.45% for SMT co-locations and 27.01% for CMP co-locations, while `SMiTe` can precisely predict the performance degradation with 1.79% and 1.36% average errors respectively.

As demonstrated by our experiments, the PMU based prediction model performs poorly on both SPEC and CloudSuite applications. We observed a few possible sources that may contribute to the inaccuracy:

- Some PMUs are designed to be core counters, e.g. `UOPS_EXECUTED.PORT2_CORE`, and there are no counters available to measure the corresponding events at per SMT context granularity [19].
- Some PMUs are known to contain bugs and may report inaccurate measurements [20].
- There are limited numbers of PMUs available on the real system, and they may not fully expose the resource usage information that is needed for the precise prediction.

3) SMiTe's Prediction Accuracy for Tail Latency: We evaluate our prediction model for 90th percentile latency using `Web-Search` and `Data-Caching` (`Data-Serving` and `Graph-Analytics` do not report percentile latency statistics). We use the profiled performance degradation and 90th percentile latency of CloudSuite application when co-located with `Rulers` to train our latency prediction model using Equation 6. In Figure 13, the measured performance degradation and 90th percentile latency are measured when `Web-Search` and `Data-Caching` are co-located

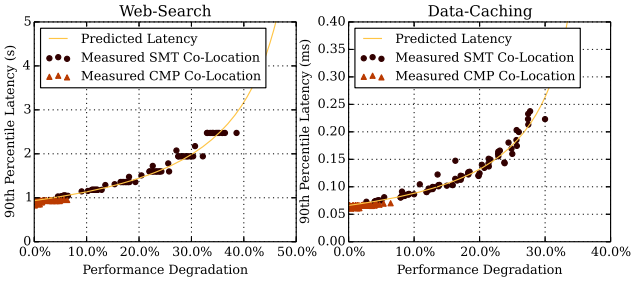

Figure 13. Prediction accuracy for 90th percentile latency when latency-sensitive application represented by CloudSuite co-locate with batch applications. The average absolute prediction error on `Web-Search` is 4.61% and 6.17% for `Data-Caching`.

with applications from the SPEC testing set. The figure demonstrates that our queueing model is able to capture the correlation between the performance degradation and the 90th percentile latency. The average prediction error of our model is 4.61% for `Web-Search` and 6.17% for `Data-Caching`.

C. Scale-out Study: Improving Utilization while Guaranteeing QoS

With `SMiTe`'s precise prediction, we can enable 'safe' co-locations in order to improve utilization without violating the QoS requirement. In this experiment, we assume a cluster composed of 4,000 servers and each 1,000 of them are running one of the four latency-sensitive applications from Cloudsuite (`Web-Search`, `Data-Caching`,

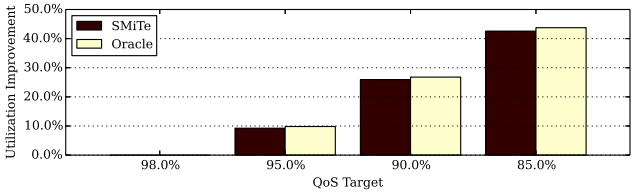

Figure 14. Utilization improvement when we allow SMT co-location under different QoS targets defined as average performance. SMiTe improves the utilization by 9.24%, 25.90% and 42.97%, at 95%, 90% and 85% QoS target respectively, which is very close to the Oracle co-location policy as 9.82%, 26.78% and 43.75%.

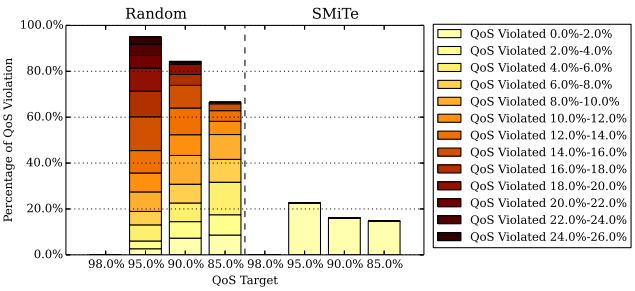

Figure 15. Percentage of QoS violation in all scheduled co-locations under SMiTe co-location policy and Random co-location policy when QoS is defined as average performance. In order to achieve same amount of utilization gain, Random co-location policy violates up to 26% QoS requirement while the largest violation from SMiTe is only 1.67%.

Data-Serving and Graph-Analytics). We use our performance prediction model to guide the cluster-level scheduler to co-locate latency-sensitive applications with batch SPEC applications. Our evaluation baseline disallows SMT co-locations, which is the state-of-the-art approach to guarantee QoS in modern WSCs without a precise prediction mechanism, leaving one out of the two SMT contexts on each core idle. Thus, we have 6 latency-sensitive application threads running on 6 cores, and we could potentially co-locate from 0 to 6 instances of batch applications on each server. In addition to SMiTe, we measure application's actual performance degradation and use these measurements to construct an Oracle co-location policy for comparison.

Figure 14 shows the utilization improvement when applying different co-location policies. SMiTe achieves 9.24%, 25.90% and 42.97% utilization improvement at 95%, 90% and 85% QoS targets respectively. Compared to the Oracle policy, which improves the utilization by 9.82%, 26.78% and 43.75%, SMiTe is very efficient and achieves utilization that is very close to the Oracle.

Due to the potential inaccuracy the prediction model has, in rare cases, the co-location decisions made by SMiTe might slightly violate the targeted QoS requirement. We quantify the violations compare it against an interference-oblivious policy that achieves exactly the same amount of utilization gain through randomly co-locating applications. In this experiment, the percentage of QoS violations is defined as the number of violations divided by the number of co-locations ($\frac{server_{violated}}{server_{co-located}}$), and the amount of violations

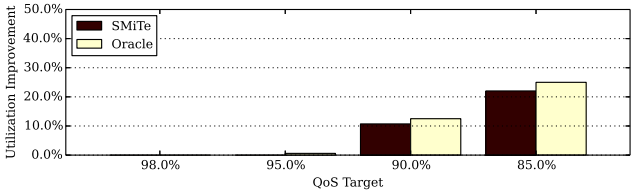

Figure 16. Utilization improvement when we allow SMT co-location under different QoS targets defined as 90th percentile latency. SMiTe improves the utilization by 0%, 10.72% and 22.03% at 95%, 90% and 85% QoS target respectively, which is relatively close to the Oracle co-location policy as 0.59%, 12.50% and 24.99%.

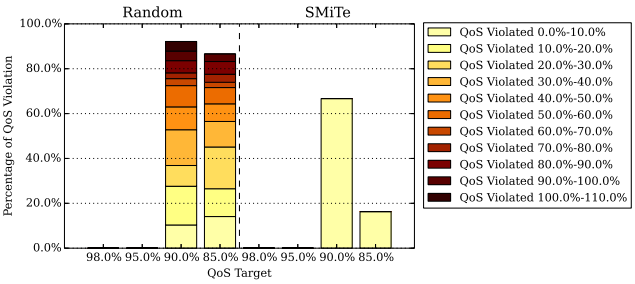

Figure 17. Percentage of QoS violation in all scheduled co-locations under SMiTe and Random co-location policy when QoS is defined as 90th percentile latency. To improve the same amount of server utilization, Random policy suffers from up to 110% QoS violation while the largest violation from SMiTe is only 0.96%.

is defined as the normalized violation ($\frac{QoS_{target}-QoS_{actual}}{QoS_{target}}$).

The QoS violations are shown in Figure 15. To achieve the same amount of utilization gain as SMiTe at each QoS target, the Random policy suffers from up to 26% QoS violation while the biggest violation using SMiTe is only 1.67%. In addition, as shown in the figure, SMiTe reduces 78.57% QoS violations on average compared to the Random policy.

D. Scale-out Study: Tail Latency

In this section, we evaluate the utilization improvement and QoS when the cluster-level scheduler uses SMiTe's prediction for tail latency to steer scheduling. Similar to the previous scale-out experiment, we assume a cluster composed of 4,000 machines with half-loaded latency-sensitive workloads composed of Web-Search and Data-Caching. The QoS requirement in this experiment is defined as the 90th percentile latency, which is more challenging to meet than the average performance. This is because the tail latency grows super-linearly with the average performance degradation due to the queueing effect. However, SMiTe is able to achieve 10.72% utilization improvement at 90% QoS requirement and 22.03% at 85% QoS requirement (90th percentile query latency is affected by 10% and 15% respectively), which is relatively close to the Oracle policy of 12.50% and 24.99% improvement as shown in Figure 16. In addition, compared to the Random policy shown in Figure 17, which suffers up to 110% QoS violations, the most serious violation SMiTe experiences is only 0.96%.

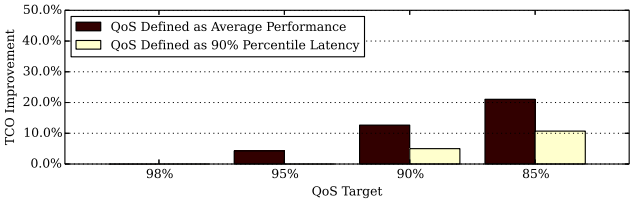

Figure 18. Total cost of ownership (TCO) improvement under different QoS requirements normalized by disallowing SMT co-location. SMiTe can save up to 21.05% cost under average performance requirement and up to 10.70% under 90th percentile latency requirement.

E. TCO Analysis

By improving the server utilization through co-locations, we improve the energy efficiency and also reduce the total cost of ownership (TCO) for building and operating the WSCs. Because we can provide the same amount of computation with fewer servers through co-locations, we reduce the number of servers needed, the required power provisioning, the datacenter area and the maintenance expenses consequently. In this section, we quantify the TCO saving by applying SMiTe methodology in WSCs under various QoS requirements.

In the baseline configuration, we assume the WSC has half of the machines running latency-sensitive applications and the other half running batch applications. By applying SMiTe methodology, we can co-locate batch applications together with latency-sensitive applications on the same server if the QoS requirement can be met based on the prediction. The analytical methodology introduced in [21] is applied to study the impact of SMiTe on the 3-year TCO. We use the latest PUE statistics published by Google [22] as part of the input to the TCO model.

Figure 18 presents the results of our TCO analysis. SMiTe improves the TCO by up to 21.05% when targeting the average performance QoS requirement. Although 90th percentile QoS requirement is more challenging because the tail latency grows super-linearly due to queueing effect, SMiTe still achieves up to 10.70% improvement.

V. Related Work

There has been a large amount of work on resource management for multicore SMT processors [17, 23–28]. Feliu et al. [17] proposes a method to improve the overall throughput by balancing L1 bandwidth usage, however there is no performance guarantees. Eyerman et al. present probabilistic job symbiosis [23], which employs specialized performance accounting hardware to facilitate modeling the performance impact for SMT co-locations. Cazorla et al. [24, 25] propose a hardware mechanism to track and adjust shared resource usage, then leverage that mechanism to dynamically adjust the resources to meet application QoS targets.

As an alternative to predicting the impact of SMT co-locations, others have used competition heuristics to achieve efficient scheduling. Snavely and Tullsen [26] and De Vuyst et al. [27] use a sampling phase to discover the performance interference due to co-locations in order to schedule jobs on SMT processors. Vega et al. [28] present a competition heuristic to decide whether multiple threads should be consolidated to the SMT contexts on the same core for multithreaded workloads.

There are also a number of prior works on hardware or application characterization and modeling using micro-benchmarks [4, 18, 29–31]. Mars et al. [4] present a methodology that uses tunable memory micro-benchmarks to quantify the *sensitivity* and *contentiousness* of an application for the last level cache and memory bandwidth contention. However, their approach is designed only for uncore-level resource sharing. Bertran et al. [18] present an approach to automatically generate micro-benchmarks to study the energy-performance trade-offs for multicore SMT processors, in which they use the micro-benchmarks to obtain energy-related platform characterization. Delimitrou et al. [31] describe a micro-benchmark suite that can be used to detect resource contention for a number of shared resources in a CMP machine to facilitate intelligent cluster-level scheduling decisions.

Others have studied the performance interference on multicore processors without considering SMT co-locations. Delimitrou et al. [5] manage various co-location scenarios in order to improve the resource utilization without violating the QoS target. Tang et al. present a compiler [32] and a compiler-supported runtime framework [3] to control low-priority application's *contentiousness* and ensure the QoS of high-priority application, in order to improve the system throughput. Yang et al. [2] improve resource utilization by dynamically probing and controlling the execution of low-priority applications to guarantee the QoS of the high-priority applications.

VI. Conclusion

In this paper, we present SMiTe methodology, which enables precise performance interference prediction on multicore SMT processors. Based on our observation that there is very little correlation for an application's contention characteristics across different shared resources, we design a set of Rulers to quantify application's *sensitivity* and *contentiousness* in a decoupled manner. We then establish a regression model that combines the *sensitivity* and *contentiousness* measurements to predict the performance interference under various co-location scenarios. With SMiTe, we are able to predict SMT co-locations with 2.80% average error for SPEC CPU2006 benchmarks and 1.79% average error on CloudSuite. Based on the precise performance prediction our methodology provides, we can improve the server utilization by up to 42.57% through co-locations while enforcing the QoS requirement.

Acknowledgment

We thank the anonymous reviewers for their feedback. This research was supported by Google and by the National Science Foundation under grants CCF-SHF-1302682 and CNS-CSR-1321047.

REFERENCES

[1] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE computer*, vol. 40, no. 12, pp. 33–37, 2007.

[2] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2013, pp. 607–618.

[3] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, "Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2013, pp. 89–100.

[4] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. ACM, 2011, pp. 248–259.

[5] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2014.

[6] M. Laurenzano, Y. Zhang, L. Tang, and J. Mars, "Protean code: Achieving near-free online code transformations for warehouse scale computers," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Cambridge, UK: ACM, 2014.

[7] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, 1995, pp. 392–403.

[8] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture (ISCA)*, vol. 24, no. 2. ACM, 1996, pp. 191–202.

[9] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[10] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 40, no. 1. ACM, 2012, pp. 37–48.

[11] "Intel next generation microarchitecture codename sandy bridge: New processor innovations," Intel Developer Forum. 2010.

[12] B. Sinharoy, R. Kalla, W. J. Starke, H. Q. Le, R. Cargnoni, J. A. Van Norstrand, B. J. Ronchetti, J. Stuecheli, J. Leenstra, G. L. Guthrie, D. Q. Nguyen, B. Blaner, C. F. Marino, E. Retter, and P. Williams, "Ibm power7 multicore server processor," *IBM Journal of Research and Development*, vol. 55, no. 3, pp. 1:1–1:29, May 2011.

[13] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[14] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

[15] V. Gupta, M. Harchol-Balter, J. Dai, and B. Zwart, "On the inapproximability of m/g/k: why two moments of job size distribution are not enough," *Queueing Systems*, vol. 64, no. 1, pp. 5–48, 2010.

[16] J. Li, N. K. Sharma, D. R. Ports, and S. D. Gribble, "Tales of the tail: Hardware, os, and application-level sources of tail latency."

[17] J. Feliu, J. Sahuquillo, S. Petit, and J. Duato, "L1-bandwidth aware thread allocation in multicore smt processors," in *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques (PACT)*. IEEE Press, 2013, pp. 123–132.

[18] R. Bertran, A. Buyuktosunoglu, M. S. Gupta, M. Gonzàlez, and P. Bose, "Systematic energy characterization of cmp/smt processor systems via automated micro-benchmarks," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2012, pp. 199–211.

[19] P. Guide, "Intel® 64 and ia-32 architectures software developer's manual," 2010.

[20] S. Eranian, "Linux perf_events subsystem status update," in *Petascale Tools Workshop*, July 2013.

[21] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[22] "Google data center pue performance," http://www.google.com/about/datacenters/efficiency/internal/, accessed 30-May-2014.

[23] S. Eyerman and L. Eeckhout, "Probabilistic job symbiosis modeling for smt processor scheduling," in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2010, pp. 91–102.

[24] F. J. Cazorla, A. Ramirez, M. Valero, P. M. Knijnenburg, R. Sakellariou, and E. Fernández, "Qos for high-performance smt processors in embedded systems," *IEEE Micro*, vol. 24, no. 4, pp. 24–31, 2004.

[25] F. J. Cazorla, P. M. Knijnenburg, R. Sakellariou, E. Fernandez, A. Ramirez, and M. Valero, "Predictable performance in smt processors: Synergy between the os and smts," *IEEE Transactions on Computers*, vol. 55, no. 7, pp. 785–799, 2006.

[26] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous mutlithreading processor," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, vol. 35, no. 11. ACM, 2000, pp. 234–244.

[27] M. De Vuyst, R. Kumar, and D. M. Tullsen, "Exploiting unbalanced thread scheduling for energy and performance on a cmp of smt processors," in *Proceedings of the 20th International Conference on Parallel and Distributed Processing (IPDPS)*. IEEE, 2006, pp. 10–pp.

[28] A. Vega, A. Buyuktosunoglu, and P. Bose, "Smt-centric power-aware thread placement in chip multiprocessors," in *22nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2013, pp. 167–176.

[29] L. C. Carrington, M. Laurenzano, A. Snavely, R. L. Campbell, and L. P. Davis, "How well can simple metrics represent the performance of hpc applications?" in *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE, 2005, pp. 48–48.

[30] M. A. Laurenzano, M. Meswani, L. Carrington, A. Snavely, M. M. Tikir, and S. Poole, "Reducing energy usage with memory and computation-aware dynamic frequency scaling," in *European Conference on Parallel Processing (Euro-Par)*. Springer, 2011, pp. 79–90.

[31] C. Delimitrou and C. Kozyrakis, "ibench: Quantifying interference for datacenter applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 23–33.

[32] L. Tang, J. Mars, and M. L. Soffa, "Compiling for niceness: Mitigating contention for qos in warehouse scale computers," in *Proceedings of the 10th International Symposium on Code Generation and Optimization (CGO)*. ACM, 2012, pp. 1–12.