DICER: Diligent Cache Partitioning for Efficient Workload Consolidation

Konstantinos Nikas
Computing Systems Laboratory
National Technical University of Athens
knikas@cslab.ece.ntua.gr

Nikela Papadopoulou
Computing Systems Laboratory
National Technical University of Athens
nikela@cslab.ece.ntua.gr

Dimitra Giantsidi
Computing Systems Laboratory
National Technical University of Athens
dgiantsidi@cslab.ece.ntua.gr

Vasileios Karakostas
Computing Systems Laboratory
National Technical University of Athens
vkarakos@cslab.ece.ntua.gr

Georgios Goumas
Computing Systems Laboratory
National Technical University of Athens
goumas@cslab.ece.ntua.gr

Nectarios Koziris
Computing Systems Laboratory
National Technical University of Athens
nkoziris@cslab.ece.ntua.gr

ABSTRACT

Workload consolidation has been shown to achieve improved resource utilisation in modern datacentres. In this paper we focus on the extended problem of allocating resources when co-locating *High-Priority (HP)* and *Best-Effort (BE)* applications. Current approaches either neglect this prioritisation and focus on maximising the utilisation of the server or favour HP execution resulting to severe performance degradation for BEs. We propose DICER, a novel, practical, dynamic cache partitioning scheme that adapts the LLC allocation to the needs of the HP and assigns spare cache resources to the BEs. Our evaluation reveals that DICER successfully increases the system's utilisation, while at the same time minimising the impact of co-location on HP's performance.

ACM Reference Format:
Konstantinos Nikas, Nikela Papadopoulou, Dimitra Giantsidi, Vasileios Karakostas, Georgios Goumas, and Nectarios Koziris. 2019. DICER: Diligent Cache Partitioning for Efficient Workload Consolidation. In *48th International Conference on Parallel Processing (ICPP 2019), August 5–8, 2019, Kyoto, Japan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3337821.3337891

1 INTRODUCTION

Multicore systems are the norm for high performance servers that are widely used in both cloud datacentres and supercomputers. These systems encapsulate several cores sharing several resources that are critical for application performance, such as the last-level cache (LLC), memory links, I/O controllers, etc. However, sharing resources on such systems without any regulation can be damaging; when multiple applications execute simultaneously, resource contention can lead to destructive interference, unfairness or starvation, and thus reduced and unpredictable performance.

Past work has focused intensely on the mitigation of this impact on performance. The proposed resource managers follow two orthogonal approaches. The first one investigates ways to efficiently partition the shared resources among the concurrently running applications [12, 18, 20, 21, 33, 34, 37, 41, 46]. The second approach extends system schedulers that operate inside the server [5, 15, 16, 22, 24, 27, 32, 42, 50] to account for contention and identify appropriate co-schedules that mitigate it and thus maximise the overall throughput or maintain performance fairness.

However, all these resource managers typically assume that co-located applications are equally important. Nevertheless, modern cloud datacentre operation is based on executing workloads that have different *Quality of Service* (QoS) (or *Service Level Objective - SLO*) requirements. In addition, these requirements can be expressed by different metrics, such as throughput [10], response latency [30], operations per second [11], and completion deadline [49], that ultimately lead to different application classes [25, 29, 49] and billing policies [14]. Therefore, resource managers need to adapt to prioritisation; they need to consider the performance of critical applications together with total system utilisation and throughput.

To address this, recent research efforts [7, 8, 10, 19, 29–31, 43, 49] have focused on an extended resource allocation problem: a *High Priority (HP)* application is co-located with a number of *Best Effort (BE)* applications, and the goal is to safeguard the QoS of HP while maximising the throughput of BEs. These schemes essentially attempt to maximise the utilisation of servers, accelerators, or warehouses through workload consolidation while ensuring or achieving a required QoS level for HP applications.

In this paper we work on the same extended resource allocation problem of consolidating an HP and BEs on the same server. We differentiate though from previous works, as our goal is to provide a practical scheme that operates transparently to the running applications, i.e. without any assumption on the information provided by the application itself [29] as well as any profiling information either pre-existing [42, 49] or extracted by accompanying profile-inferring tools [30]. This way, our scheme will be able to operate "out-of-the-box" in all typical execution platforms.

To cope with the matter at issue, one can follow a naive, conservative approach and avoid workload consolidation, safeguarding the performance of the HP application; this is clearly a suboptimal

ICPP 2019, August 5–8, 2019, Kyoto, Japan

ACM ISBN 978-1-4503-6295-5/19/08…$15.00
https://doi.org/10.1145/3337821.3337891

approach in terms of effective utilisation of resources. The other extreme approach would be to disregard the priorities of applications and co-locate them within the same server in an unmanaged way, taking though the risk of severely harming the HP application.

A more elaborate approach can leverage the recently introduced support for managing shared hardware resources on modern servers. Intel has released as part of its latest Xeon processors the Intel Resource Director Technology (RDT) [4], a framework that monitors and manages the shared last-level cache (LLC) and memory bandwidth. Similarly, Cavium has added support for managing the LLC in the ThunderX processors [1, 44]. Such technologies provide a straightforward mechanism to address our problem: the cache can be taken over almost entirely by the HP application leaving only a minimum fraction assigned to the BE applications.

We have experimented with both straightforward cache allocation policies, namely *Unmanaged (UM)* and *Cache Takeover (CT)*, and made three observations that drive our approach. First, in many multiprogrammed workloads, when allocated exclusively a portion of the cache, the HP application is able to achieve similar performance to when running alone in the system occupying the entire cache. Hence, there is ample opportunity for dynamically co-locating HP and BEs, increasing the platform's utilisation. Second, there exist multiprogrammed workloads for which the HP application performs better when allocated exclusively less cache space. While this seems to be counter-intuitive, it happens because the containment of BEs in minor portion of the cache introduces bandwidth saturation on the memory link that in turn impacts the HP application. Third, strict cache allocation policies that favour the HP application performance result in waste of resources, hurting system utilisation due to their unfairness. On the other hand, leaving resources unmanaged may increase the system's utilisation, at the expense though of frequently disrespecting the QoS requirements of the HP application.

We propose DICER, a dynamic cache partitioning scheme that targets both system utilisation increase and HP performance close to that of isolated execution, i.e. respect the HP performance requirements. This is achieved by diligently managing the LLC allocations of the co-located applications, trying to alleviate the effects of contention. We implement DICER based on Intel's RTD support for monitoring and partitioning LLC occupancy and monitoring memory bandwidth.

Our evaluation results on multiprogrammed workloads from the SPEC CPU 2006 [17] and Parsec 3.0 [6] suites show that DICER enables the HP application to achieve an SLO of 80% for more than 90% of our workloads and an SLO of 90% for 74% of our workloads; at the same time DICER maintains the effective system utilisation of a full server to 60% on average. Compared to CT, the static cache allocation policy that conservatively favours the HP application, DICER achieves similar or higher conformance to various HP SLOs, especially as more BEs are co-located on the same server. At the same time, by dynamically assigning spare cache space to BEs, DICER achieves in most cases comparable effective utilisation to UM, the unmanaged policy that does not isolate the HP application and enforces no priorities on the consolidated workloads. Finally, our evaluation of DICER on a combined index incorporating both SLO conformance and effective system utilisation, shows that DICER outperforms all other schemes and can benefit providers trying to maximise system utilisation while maintaining their Service Level Agreements (SLAs).

In summary, the main contributions of this paper are:

- We perform a detailed analysis of the impact of simple co-location policies on the performance of applications and the utilisation of the server (Section 2).
- Based on our analysis, we design and implement DICER, a novel, practical, dynamic cache partitioning scheme that adapts to the actual cache requirements of the HP application and assigns any spare cache to BEs (Section 3).
- We evaluate DICER using various multiprogrammed workloads, and show that it is able to maximise the server's utilisation while minimising the impact of co-location on HP's performance (Section 4).

2 WORKLOAD CONSOLIDATION

2.1 Execution Scenario

Today, datacentres and supercomputers typically avoid consolidating multiple applications on the same multicore server [11, 43, 47]. This co-location could compromise the performance of applications, due to contention for the shared resources of the server, making providers unable to guarantee Quality-of-Service (QoS). On the other hand though, workload consolidation would allow providers to maximise server utilisation, in terms of core utilisation and system throughput.

To render co-location efficient and practical, we study the extended resource allocation problem of executing concurrently *High Priority (HP)* and *Best Effort (BE)* applications. We focus on a multiprogrammed scenario that has been considered by prior studies [19, 29] as well: one HP application executes on one core of the server, while multiple instances of another application run on the remaining cores and are considered as BE workloads. However, in contrast to other works that rely on a known target for QoS and focus on guaranteeing QoS at the expense of utilisation, we focus on minimising the impact of co-location on HP while maximising the server utilisation.

In this section we present two simple co-location policies and analyse their impact on the performance of HP and the utilisation of the server. The findings of this analysis are then used to drive the design of DICER.

2.2 Baseline Co-location Policies

Unmanaged (UM). In this scheme, all applications are executed in a typical fashion, i.e., there is no control on sharing resources or any QoS enforcement. UM is a contention-unaware co-location policy; all HP and BEs experience full contention on the LLC and memory bandwidth.

Cache-Takeover (CT). On the contrary, *"Cache-Takeover"* (CT) is a co-location scheme that tries to mitigate interference in an intuitive, yet conservative, way. CT controls the LLC space that each application receives and conservatively allocates the maximum possible isolated portion of the LLC to HP, leaving the minimum possible LLC portion for all the BEs.

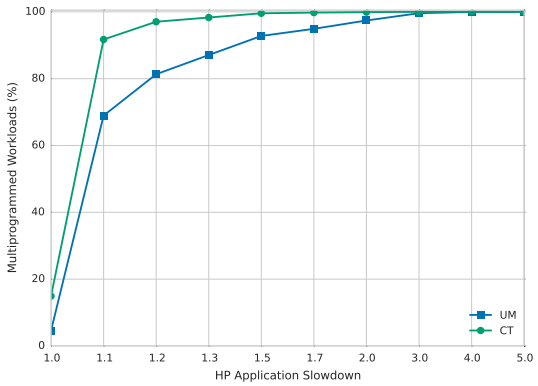


Figure 1: Cumulative distribution of HP's slowdown when running together with 9 BEs.

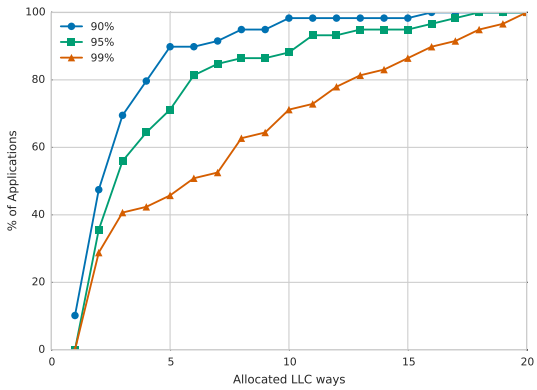


Figure 2: Cumulative distribution of HP's LLC allocation required when running alone to achieve 90%, 95%, and 99% of the performance achieved using all the 20 ways of the LLC.

2.3 Impact on HP Performance

To assess the impact of the baseline policies on HP's performance, we execute 3481 multiprogrammed workloads in total, composed of one HP and 9 BEs (Section 4.1 provides details regarding our methodology and experimentation platform). We calculate the slowdown HP suffers with respect to its execution time when run alone on the system and plot in Figure 1 its cumulative distribution.

For UM, we observe that for the majority of the workloads (64%) HP runs around 1.1× slower compared to running alone, while its performance is unaffected by the co-located BEs for only less than 5% of the workloads. In contrast, for around 29% of the workloads, HP suffers a significant slowdown between 1.1× and 2×, while there exist a few workloads (around 2.5%) in which HP is more than 2× slower compared to when running alone. Hence, it is evident that in order to make co-location practical, the impact of interference due to resource contention needs to be mitigated.

On the other hand, CT preserves HPs' performance for more workloads than UM; HP is unaffected by the co-located BEs for 15% of the workloads, compared to less than 5% for UM. Similarly, the percentage of workloads in which HP runs between 1.1× and 2× slower is reduced from 29% to 8%. These improvements are made possible through greatly reducing the resources allocated to BEs. However, as explained next, this reduction is usually excessive, and in some cases it can even harm HP's performance.

2.3.1 Suboptimal use of resources. Figure 2 shows the cumulative distribution of the minimum number of LLC ways HP needs when running alone, in order to perform within 90%, 95%, and 99% of its maximum performance achieved using the full LLC. The majority of the applications require less than the entire LLC or even the 19 ways that would be assigned to them by CT; 50% of them achieve 99% of their maximum performance when allocated only 6 LLC ways. If performance requirements are relaxed, cache requirements are reduced further; 90% of the applications achieve 90% of their maximum performance when allocated only 5 LLC ways.

Key Observation 1. CT's conservative choice to allocate almost the whole LLC to HP leads to a suboptimal partitioning of the cache resources. In addition, as the cache requirements of an application typically vary during its execution [40], CT's static allocation of cache space exacerbates the problem.

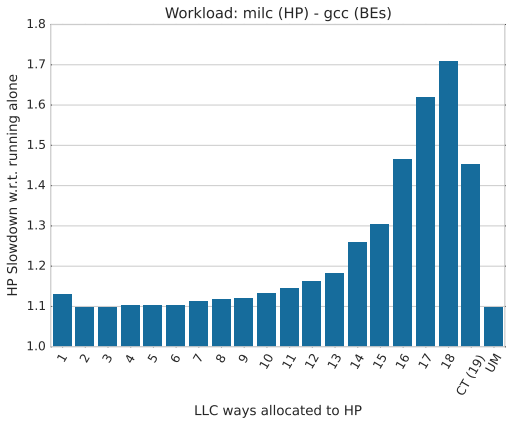


Figure 3: HP's Slowdown for different static LLC configurations. The x-axis represents the LLC ways assigned to HP. The rest of the 20 LLC ways are assigned to BEs.

2.3.2 Bandwidth saturation. CT is based on the intuitive assumption that HP performs best when it receives as much cache as possible and BEs are contained in the minimum possible cache space. However, our analysis has unveiled several cases where CT degrades HP's performance while other less conservative static LLC allocations, or even UM, perform better. Focusing further on these cases, we have deduced that as CT contains BEs in a single LLC way, it causes them to experience very frequent cache misses and thus saturate the memory link. If HP happens to depend on memory bandwidth, this saturation directly impacts its execution time.

We present one such example in Figure 3, which plots the slowdown of HP (milc) when running together with 9 BEs (gcc) for all possible static cache partitions between HP and BEs. We observe that: (i) HP performs best (1.09× slowdown compared to when running alone) when it is assigned 2 LLC ways; (ii) HP's performance remains close to the best one when allocated 3 to 6 LLC ways; (iii)

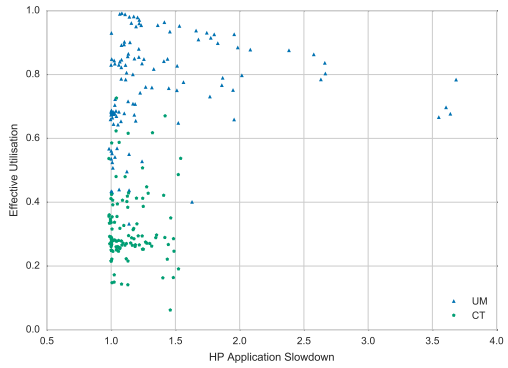


Figure 4: System's effective utilisation vs HP's slowdown when running with 9 BEs with the UM and CT policies.

CT causes a slowdown of 1.45×, and (iv) UM performs similar to the best static configuration; however, as UM imposes no restrictions on cache usage, HP gains control of around 26% of the LLC instead of the minimum required 2 out of the 20 ways.

Key Observation 2. In contrast to its core assumption, CT's static, conservative and suboptimal utilisation of resources can become detrimental to HP's performance.

2.3.3 Classification of multiprogrammed workloads. Motivated by these two key observations, we classify the multiprogrammed workloads into two classes based on CT's impact on HP's performance:

(i) *CT-Favoured (CT-F)*, for which CT improves HP's performance over UM, and
(ii) *CT-Thwarted (CT-T)*, for which CT offers no improvement or even degrades HP's performance compared to UM.

Out of our 3481 workloads, around 60% fall into the CT-T class, i.e. in the large majority of cases a static, conservative scheme such as CT is not suitable for co-locating applications, as it cannot improve HP's performance compared to UM. In addition, for the remaining CT-F workloads, even though CT succeeds in improving HP's performance, resources are partitioned suboptimally.

2.4 Impact on Utilisation

To assess the impact of the baseline co-location policies on the server's utilisation, we select a representative sample of 120 multiprogrammed workloads from our set of 3481; 50 of them belong to CT-F, while the rest belong to CT-T. We measure the system's Effective Utilisation (EFU) employing the IPC_{norm_hmean} metric [37], shown in Equation 1. The harmonic mean of normalised IPCs is a metric that balances performance and fairness, and takes values between 0 and 1 to reflect the impact of co-location; a value of 1 means no performance loss, i.e. no impact.

$$EFU = IPC_{norm\_hmean} = \frac{n}{\frac{IPC^{HP}_{alone}}{IPC^{HP}} + \sum_{i=0}^{n-1} \frac{IPC^{BE}_{alone,i}}{IPC^{BE}_{i}}} \quad (1)$$

Figure 4 presents a scatter plot of the EFU and the slowdown suffered by HP for each one of the 120 multiprogrammed workloads for both UM and CT. It is evident that the suboptimal resource allocations made by CT take their toll on the utilisation of the system. On the other hand, UM achieves significantly higher utilisation than CT but fails to safeguard HP's performance.

Key Observation 3. To make co-location practical, a scheme is needed that will increase the system's utilisation similar to UM and at the same time minimise the impact on HP, like CT.

In the next section, we propose DICER, an intelligent dynamic scheme that dynamically identifies the proper amount of cache space that HP needs, and assigns the rest of the LLC to the BEs. Unlike *UM*, which does not manage the shared resources in favour of any application, DICER prioritises the performance of HP. At the same time, unlike *CT*, which statically, naively and conservatively attempts to favour HP, DICER fairly allocates cache space to the BEs to increase the effective system utilisation.

3 DICER: DILIGENT CACHE PARTITIONING

3.1 Overview

DICER is a new scheme that dynamically adapts HP's LLC allocation, trying to match its cache requirements at any moment of its execution. DICER strives to safeguard HP's performance for both CT-F and CT-T workloads, and at the same time assign the maximum possible cache space to BEs to boost their performance and increase the system's utilisation as much as possible.

DICER implements the following high-level execution flow. Execution time is split in monitoring periods of length T, during which DICER monitors HP's IPC as well as the memory bandwidth consumed by HP and BEs. At the end of each period, based on these metrics, DICER assesses the performance and behaviour of the HP application, and decides how to better partition the LLC among the co-located applications.

```
current_allocation.HP = N-1
current_allocation.BE = 1
optimal_allocation = current_allocation
CT_Favoured = True

dicer_driver():
  while(1):
    monitor()
    if BW_saturated:
      allocation_sampling()
    else:
      allocation_optimisation()
  return

monitor():
  BW_saturated = False
  measure_IPC_HP()
  measure_MemBW_HP()
  measure_MemBW()
  if MemBW > MemBW_threshold:
    BW_saturated = True
  return

allocation_sampling():
  CT_Favoured = False
  (optimal_allocation, IPC_opt) = sampling()
  current_allocation = optimal_allocation
  return
```

Listing 1: DICER - Main driver.

3.2 Detailed Design

Listing 1 shows the high-level DICER driver. DICER begins similar to CT, i.e. it assumes that the system executes a CT-F multiprogrammed workload and, therefore, the best resource partition for alleviating any impact on HP's performance is the one imposed

by CT. Thus, DICER starts by allocating all but one LLC ways to HP and containing all BEs in a single LLC way. Then, at the end of each monitoring period, DICER has two options, depending on whether bandwidth saturation has been detected on the memory link or not: eliminate bandwidth saturation or optimise the allocation of the LLC.

3.2.1 Eliminating bandwidth saturation. During each monitoring period, DICER tracks the bandwidth consumed by the co-located workloads on the memory link. If it surpasses a certain threshold, DICER detects bandwidth saturation.

The first time bandwidth saturation is detected, DICER recognises that the multiprogrammed workload belongs to the CT-T class and not CT-F, as initially assumed. Hence, as explained in Section 2.3.2, CT's conservative cache partitioning can no longer be considered the best one for HP. Instead, DICER needs to identify the allocation that suits HP the best; for that it resorts to sampling.

Allocation sampling. During sampling, DICER applies decreasing LLC partition sizes to HP, allocating the rest of the cache to BEs, similar to [13]. Every sample is applied for a fixed interval, long enough to make the effects of the partitioning visible. DICER monitors HP's performance and identifies the *optimal_allocation* as the one that results to the highest IPC for HP, *IPC_opt*. Then *optimal_allocation* is enforced and a new monitoring period starts.

3.2.2 Optimising cache allocation. DICER's target is to maximise the utilisation of the system while minimising the impact of co-location on HP. To achieve that, DICER carefully adapts HP's LLC allocation based on the measurements of the last monitoring period, i.e. the effects of the last enforced partitioning on HP. The cache allocation optimisation mechanism is presented in Listing 2.

```
allocation_optimisation():
  if phase_change():
    allocation_reset()
    return
  if performance_stable():
    current_allocation.HP = current_allocation.HP − 1
    current_allocation.BE = current_allocation.BE + 1
  else:
    if performance_better():
      return
    else:
      allocation_reset()
  return
```

Listing 2: DICER - Cache allocation optimisation.

Phase change. As DICER tunes the LLC partitioning based on HP's IPC, it is imperative to differentiate between IPC changes caused by the resource allocation and those caused by the fact that all applications typically go through various phases with different performance characteristics. Hence, before optimising the cache allocation, DICER first checks whether a phase change in the HP application's execution has occurred.

DICER denotes as phase changes only points where the application suddenly exhibits a need for significantly more cache space. These points are identified when the memory link bandwidth consumed by HP becomes higher than the geometric mean of the bandwidth consumed in the previous three monitoring periods, as shown in Equation 2:

$$MemBW_t^{HP} > (1 + phase\_threshold) \times \sqrt[3]{\prod_{i=t-1}^{t-3} MemBW_i^{HP}} \quad (2)$$

Once a phase change has been detected, DICER needs to reset the optimisation process as the current allocation is not suitable for the new phase. Note that application phase changes related only to the computational behaviour of HP that do not exhibit different cache requirements, are not detected and do not trigger a reset. DICER treats them similar to all other execution points, making decisions based solely on any change observed on HP's IPC.

Performance validation. When DICER does not detect a phase change, it assesses whether HP's performance is stable or it has been affected by the last allocation decision. The performance is assumed to remain stable as long as the IPC remains within a percentage a close to the IPC of the previous monitoring period, as shown in Equation 3:

$$(1 - a) \times IPC_{t-1} \leq IPC_t \leq (1 + a) \times IPC_{t-1} \quad (3)$$

If the performance is *stable*, DICER presumes that the current allocation surpasses HP's actual cache requirements. Consequently, in its attempt to maximise the system's utilisation, it slightly reduces HP's cache portion and increases BEs' allocation accordingly to boost their performance. The validity of this presumption and the efficacy of the new allocation will be evaluated at the end of the next monitoring period, based on the impact on HP's IPC.

On the other hand, if the performance has *improved* compared to the previous period, DICER assumes that HP has entered into a new phase with higher IPC but with the same cache requirements, as otherwise a phase change would have been detected. Therefore, DICER selects not to alter the LLC allocation and proceeds to the next monitoring period.

Finally, if the performance has *decreased* compared to the previous period, DICER infers that either the gradual decrease of HP's cache portion has eventually harmed HP's IPC, or HP has entered a new phase with lower IPC but with the same cache requirements. As DICER cannot straightforwardly distinguish between these two cases, it resets the optimisation process, even though it is unnecessary for the latter.

```
allocation_reset():
  if CT_Favoured:
    rollback_allocation = current_allocation
    current_allocation  = optimal_allocation
    monitor()
    if BW_saturated:
      allocation_sampling()
      return
    if performance_better():
      return
    else:
      current_allocation = rollback_allocation
      return
  else:
    current_allocation  = optimal_allocation
    monitor()
    if BW_saturated:
      allocation_sampling()
      return
    if performance_near_opt():
      return
    else:
      allocation_sampling()
  return
```

Listing 3: DICER - Cache allocation reset

3.2.3 Resetting cache allocation. Listing 3 presents DICER's cache allocation reset mechanism. DICER needs to reset the cache allocation optimisation process when a phase with different cache requirements has been detected or when HP's IPC has degraded. In

both cases, the current LLC allocation is detrimental to HP's performance, so DICER needs to change it to the one that ensures the best performance for HP. This configuration however, depends on the class of the multiprogrammed workload.

CT-Favoured workloads. For CT-F workloads, the allocation is reverted back to the one imposed by CT, i.e. all but one LLC ways are assigned to HP. This change is followed by a monitoring period, at the end of which the correctness of the new allocation is validated:

(i) If bandwidth saturation is detected on the memory link, then the workload has changed to CT-T and DICER needs to sample partitions in order to identify the best one for HP.
(ii) If HP's performance improves, then the decision to reset the cache allocation was correct and the optimisation process can proceed again from this point.
(iii) Otherwise, if HP's performance does not improve, DICER infers that the lower IPC that triggered the reset mechanism was not caused by the reduction of the allocated cache space. Instead, it was caused by the application entering a new phase with a lower IPC and thus, DICER reverts back to the allocation that triggered the reset process.

CT-Thwarted workloads. For CT-T workloads, the LLC allocation is reverted back to the last *optimal_allocation*, i.e. the one discovered during the last partition sampling. Similar to CT-F, this change is followed by a monitoring period at the end of which the efficacy of the new allocation is validated:

(i) If bandwidth saturation is detected on the memory link, then partition sampling has to be performed again.
(ii) If HP's IPC is close to *IPC_opt*, i.e. the performance is similar to when the allocation was identified as optimal, then DICER is deemed to have reverted to the best possible configuration and the optimisation process can proceed again from this point.
(iii) Otherwise, if HP's IPC is not close to *IPC_opt*, DICER assumes that the best possible configuration has changed and partition sampling has to be performed again.

3.3 Implementation

We implement DICER by extending the Intel® RDT Software Package (v1.1.0) [3], an open source stand-alone library which allows control of the LLC and memory bandwidth monitoring and allocation mechanisms, namely CMT, CAT, MBM and MBA. As our server though does not support MBA, DICER employs only the former three.

DICER partitions the LLC in an isolated fashion, i.e., there are non-overlapping portions of LLC between the HP and the BEs. Further, as modern servers [1, 4, 44] offer way-based LLC partitioning, DICER modifies allocations in way granularity. When the allocations are altered, the contents of the LLC are not affected; they remain intact until they are evicted by future LLC misses.

4 EVALUATION

4.1 Platform & Methodology

Platform configuration. We perform our experiments and evaluation on an Intel Xeon E5-2630 v4 processor equipped with 10 cores, 25MB 20-way LLC, and 128GB of memory. Table 1 presents the details of the system together with the configuration parameters of DICER. It should be noted that all the parameter values have been selected after performing a sensitivity analysis which for the sake of space is not included in this paper.

Hardware	**Processor**	Intel Xeon E5-2630 v4 (Broadwell) 10 cores, 2.2GHz, SMT disabled
	LLC	25MB, 20-way set associative
	Memory Bandwidth	68.3 Gbps per channel
	Memory	128GB DDR4
DICER	**Monitoring period**	T = 1 sec
	BW saturation threshold	MemBW_threshold = 50 Gbps
	Phase detection threshold	phase_threshold = 30% (Equation 2)
	IPC stability percentage	a = 5% (Equation 3)

Table 1: System configuration.

Evaluation methodology. We employ in total 59 applications, 9 from the Parsec 3.0 [6] (serial versions) and 25 from the SPEC CPU 2006 [17] (8 of them with multiple inputs) suites. We create multiprogrammed workloads by nominating one application as HP and one as BE. Thus, each multiprogrammed workload comprises 1 HP co-located with multiple BE instances, depending on the number of cores used. For example, if all the cores of the system are used, the workload comprises 1 HP and 9 BEs.

Overall, we create $59 \times 59 = 3481$ possible multiprogrammed workloads, which we use for our initial study and the evaluation of the baseline co-location policies. To evaluate DICER, we use a representative sample of 120 multiprogrammed workloads from our original set of 3481; 50 of them belong to CT-F and 70 to CT-T.

Each experiment starts simultaneously the co-located HP and BEs, pinned on separate cores of the same socket. When an application finishes, it is restarted until all of them have executed at least once. That way, we ensure that HP is constantly executed experiencing full contention on the shared resources.

As we intentionally approach the resource allocation problem transparently to the running applications, without any application-specific information, we focus on their throughput in terms of Instructions per Cycle (IPC) to measure QoS. Therefore, if an application requires a QoS level of 90%, we consider that it achieves its Service-Level Objective (SLO) if its IPC is equal or greater than 90% of the IPC achieved when executing alone on the system (IPC_{alone}); otherwise the SLO is missed and the Service-Level Agreement (SLA) is violated.

4.2 DICER Evaluation

4.2.1 HP's performance. Figure 5 compares the performance of HP and BEs for the different co-location policies, namely DICER, UM and CT, for the two identified classes of multiprogrammed workloads. It is evident that regardless of the workload class, DICER provides either the best or close to the best performance for HP.

Specifically, for CT-F workloads (shown in the leftmost part of Figure 5) DICER performs similar to CT. In contrast, for the CT-T class that, as explained in Section 2.3.3, includes workloads for

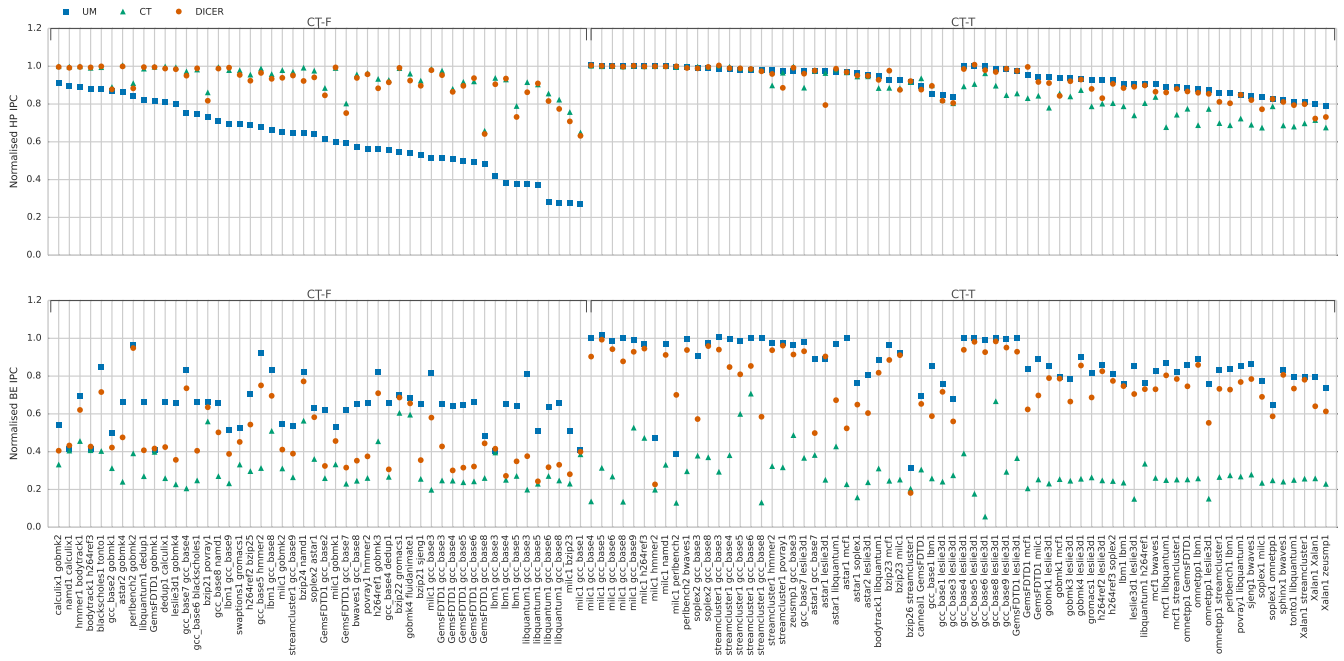


Figure 5: Performance of HP (top) and BEs (bottom) normalised to when each application runs alone (higher is better). DICER provides consistently the best HP performance for both workload types while improving BEs' performance compared to CT.

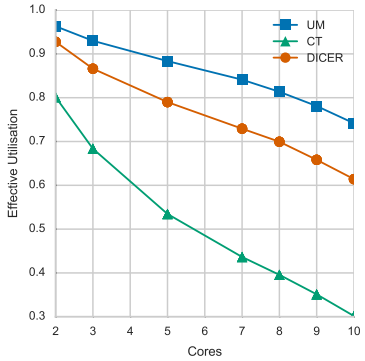


Figure 6: Geometric mean of the effective utilisation achieved for the three co-location policies.The x-axis represents the employed cores of the server, one of which is assigned to HP and the rest to BEs.

which CT either offers no improvement or even harms HP's performance compared to UM, DICER performs similar to UM. At the same time, by allocating more cache space to BEs, DICER improves their performance significantly across all cases compared to CT.

4.2.2 Effective utilisation. To evaluate whether DICER achieves its goal of increasing the server's utilisation, we measure the EFU while varying the number of co-located BEs and plot its geometric mean across our 120 multiprogrammed workloads in Figure 6.

As expected, DICER significantly outperforms CT in terms of effective utilisation. In particular, as the number of co-located BEs increases, the effective system utilisation drops significantly when using CT, as it restricts all the BEs in a single LLC way. On the other hand, UM provides the best effective utilisation. However, as it does not exert any control on the way resources are shared, UM impacts the HP application severely; not only it degrades HP's performance significantly for CT-F workloads, as shown in Figure 5, but in general it exhibits a low success rate in achieving performance SLOs, as explained next.

Figure 7 presents the percentage of workloads for which the co-location policies succeed in achieving a given SLO for HP, when varying the number of co-located BEs, for various SLOs. An SLO of 80% corresponds to a QoS level for HP of at least 80%, i.e. a performance degradation of maximum 20% compared to when running alone can be tolerated for the HP before the SLO is missed.

Although UM provides the best effective utilisation, it is evident that it manages to achieve the desired SLO for a lot fewer workloads compared to CT and DICER. Especially as the number of co-located BEs increases and the SLO targets rise, SLO misses for UM proliferate. On the other hand, for SLOs from 75% up to 90%, DICER sustains HP's QoS for a significantly higher percentage of workloads compared to CT, particularly when more than half of the server cores are occupied. This outcome is the result of DICER's dynamic adaptation to contention on both the LLC and memory bandwidth, which aggravates under high core utilisation. For higher SLOs (95%), DICER and CT achieve the given target for about the same percentage of workloads.

Our evaluation reveals that DICER in general optimises both HP's performance and the effective system utilisation across both classes of co-located workloads. To further elaborate and quantify how well DICER manages to optimise both targets, we follow the

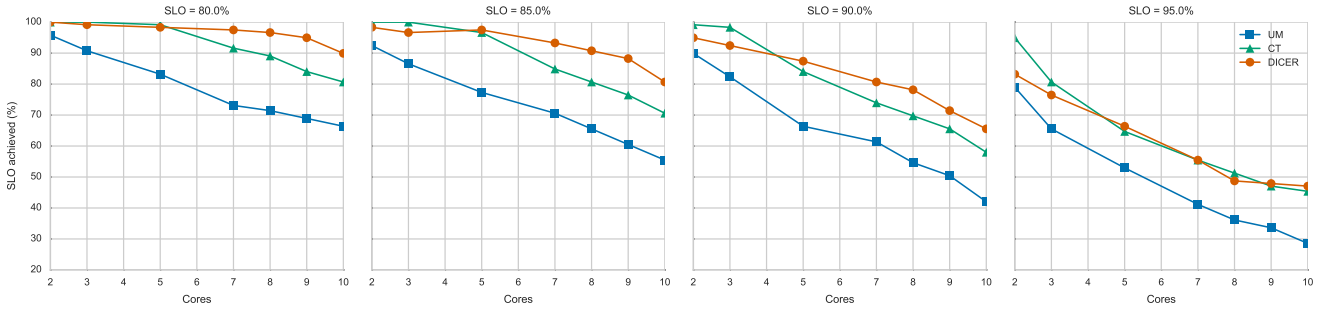
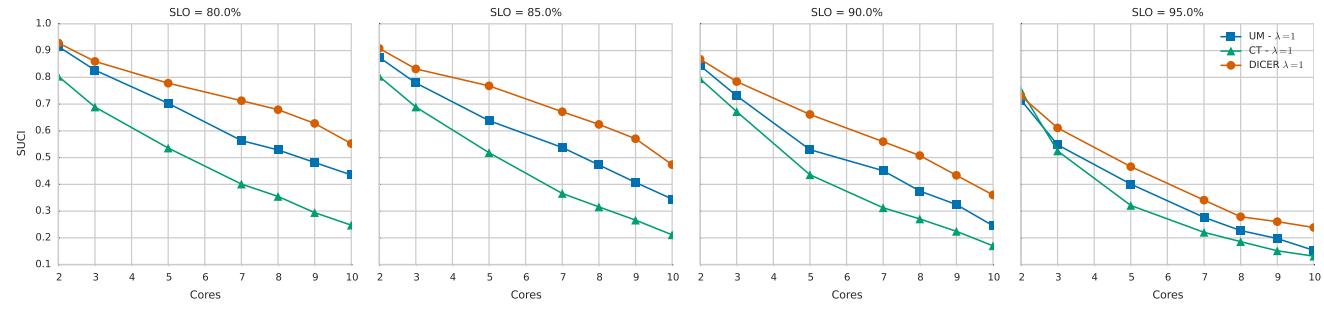
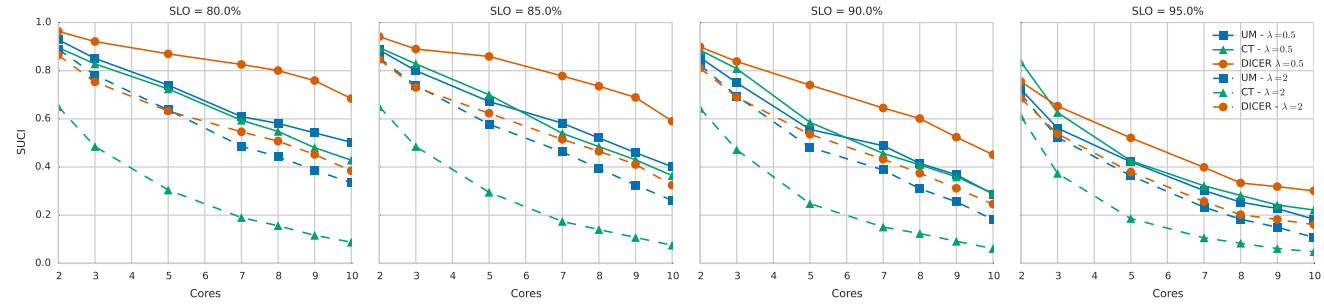


Figure 7: Percentage of workloads that achieve the given SLO for HP for the three co-location policies. The x-axis represents the employed cores of the server, one of which is assigned to HP and the rest to BEs. DICER performs equally well or better than other mechanisms especially for high number of employed cores.



(a) $\lambda = 1$



(b) $\lambda = 0.5$ **and** $\lambda = 2$

Figure 8: Comparison of co-location policies in the joint optimisation problem of achieving SLOs while maximising EFU.

example of [39] and define a combined metric, namely the *SLO-Effective Utilisation Combined Index (SUCI)*.

For a multiprogrammed workload comprising k applications, SUCI is defined as follows:

$$SUCI = c_{SLO} * EFU^{\lambda} \tag{4}$$

where c_{SLO} denotes whether the the requested SLO was achieved:

$$c_{SLO} = \begin{cases} 1, & \text{if } \frac{IPC^{HP}}{IPC^{HP}_{alone}} >= SLO \\ 0, & else \end{cases} \tag{5}$$

SUCI takes a value between 0 and 1, with 0 indicating that the SLO was missed and an SLA violation has occurred; this is done on purpose to disregard any BE improvements that materialise causing HP to miss its SLO. SUCI increases when both the HP's SLO is achieved and the effective system utilisation of the server increases. When $\lambda = 1$, the two terms of SUCI are equally weighted, i.e. achieving SLOs is equally important to system utilisation. If though one of the terms is deemed to be more important than the other, λ needs to be adjusted accordingly. If utilisation is more important, then λ needs a value greater than 1; otherwise, a value less than 1 will make conformance to SLOs more significant.

Figure 8 illustrates how UM, CT and DICER perform in terms of SUCI, for various numbers of co-located workloads and SLOs. We plot the geometric mean of SUCI to summarise results across our 120 workloads. Overall, DICER significantly outperforms the other

two policies for all SLOs, regardless of whether the conformance to SLOs is equally important to the the server's utilisation or not. Therefore, we deduce that DICER is an efficient and effective policy that can maximise the server's utilisation, while minimising the possibility of QoS violations for HP.

5 RELATED WORK

DICER leverages existing support for cache partitioning on modern servers, targeting the joint optimisation problem of maintaining the QoS of an HP application while maximising the effective system utilisation by co-locating multiple BEs. DICER operates in a practical, application transparent manner and is agnostic to the target performance of the HP.

Cook et al. [9] were the first to evaluate the potential of hardware cache partitioning for co-locating HP and BEs on a real system, and proposed a dynamic cache partitioning scheme based on the metric of LLC misses. However, that scheme lacks support for identifying and mitigating memory bandwidth saturation. Lo et al. [29] proposed Heracles, a resource manager that takes into account the QoS status provided by HP itself together with low-level system metrics, and applies cache partitioning, frequency scaling, thread packing, and network control. Kasture and Sanchez [25] proposed Ubik, a cache management scheme that targets latency-critical workloads with strict QoS requirements. However, both schemes rely on application-specific metrics that are provided by the application itself. In addition, Ubik requires extra hardware support for gathering miss curves [37]. Dirigent [49] reduces the variation in execution time of HP applications when co-located with BEs, by dynamically predicting applications' completion time and tuning cache shares and frequency. However, Dirigent requires offline profiling of HPs in isolation. Papadakis et al. [35] proposed DCP-QoS, a dynamic cache partitioning scheme for co-locating HP and BEs that is similar to DICER. While DCP-QoS follows a black-box approach, it lacks support for identifying and mitigating memory bandwidth saturation. Finally, Funaro et al. [14] proposed Ginseng, a market driven cloud mechanism for allocating LLC portions to guest virtual machines. Each guest VM bids for cache ways through an economic agent that states a valuation for each number of ways. DICER is orthogonal to Ginseng as it could be used to increase revenue from the cloud provider perspective.

LLC partitioning has also been recently used as a mechanism to enhance system fairness and throughput. Selfa et al. [38] introduced various clustering policies to improve system fairness based on Intel's CAT. Similarly, El-Sayed et al.[13] proposed a hybrid cache partitioning and sharing scheme that groups applications into clusters and then partitions the LLC among these clusters, to increase throughput. Park et al. [36] proposed CoPart, a resource manager that dynamically characterises applications and partitions the LLC and memory bandwidth to the applications, using Intel CAT and MBA, to improve system fairness. Finally, Wang et al. [44] proposed SWAP, a fine-grained LLC management scheme that combines set and way partitioning through page colouring [2, 26, 28, 48] and hardware support. However, all these schemes treat applications as of equal priority and lack any support for QoS, unless applications provide explicitly their metric of interest.

Besides commercial hardware mechanisms for resource partitioning, several hardware/software solutions have been proposed to partition or manage hardware resources for QoS and/or fairness [12, 18, 20, 33], and several cache management policies and mechanisms have been proposed for increasing throughput and mitigating interference in LLC [21, 23, 34, 37, 41, 45, 46]. Finally, prior works focus on application co-scheduling, targeting to reduce interference and mitigate the effects of contention on application and system performance, both for applications of equal priorities [5, 15, 16, 22, 24, 27, 42, 50] and when the QoS of latency-critical applications needs to be safeguarded [11, 47]. Co-scheduling is orthogonal to our work.

6 CONCLUSIONS

In this paper, we have presented DICER, a novel, practical, dynamic cache partitioning scheme for the extended resource allocation problem of co-locating one HP and multiple BE applications, where the HP operates under a given SLO. We have based DICER's design on a detailed analysis of the impact of simple co-location policies, which revealed that: (i) an HP application can maintain its performance using only a portion of LLC ways; (ii) the performance of an HP application that utilises all the available LLC space can be harmed by contention and/or interference on the memory bandwidth, and (iii) dynamic management of shared resources can help accommodate more BEs to increase the effective system utilisation. DICER as a mechanism extends Intel RDT and takes advantage of modern servers' capabilities for cache monitoring, cache allocation and memory bandwidth monitoring. DICER operates agnostically to the workload and the target performance of HP and uses resource monitoring to adapt the LLC allocation to the current needs of HP and the current resource utilisation on the server. The co-located BEs benefit from cache space that is unnecessary or harmful for the performance of the HP. In this way, DICER achieves both high performance for the HP application and high effective system utilisation.

We have evaluated DICER on 120 multiprogrammed workloads, in comparison to an unmanaged (UM) cache allocation policy and a static cache allocation policy, where the HP application takes over the largest possible part of the LLC (CT). DICER achieves high conformance of the HP to given SLOs (more than 90% of our workloads achieve an SLO of 80% and 74% of our workloads achieve an SLO of 90%), successfully adapting to the needs of the HP application and safeguarding its performance. In comparison to CT, DICER achieves equally high or better performance for the HP, successfully detecting the needs of the HP in terms of LLC cache space and mitigating contention effects on the memory bandwidth. In comparison to UM, which does not distinguish between HP and BEs, DICER only mildly reduces system utilisation, in favour of the performance of HP, while ensuring progress of the BEs. Finally, using a combined index for SLO and effective utilisation, we have showed that DICER is an effective policy for a provider that seeks to maximise system utilisation without compromising the performance of the HP and violating their SLAs with their clients.

We are extending DICER to explicitly, dynamically control the memory bandwidth, using Intel's MBA, which, at the same time, will allow DICER to be more diligent to its current cache partitioning policies. To better safeguard the performance of the HP

application, we intend to extend DICER to dynamically manage the number of co-located BEs. Further analysis is also necessary to investigate whether assigning overlapping cache partitions to the HP and the BEs can benefit some workloads. Finally, we will explore how DICER can be more effective under various user-faced or provider-faced scenarios of achieving a given SLO while maximising the effective system utilisation.

ACKNOWLEDGMENTS

This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 732366 (ACTiCLOUD).

REFERENCES

[1] [n. d.]. Cavium ThunderX family of workload optimized processors. https://cavium.com/pdfFiles/ThunderX_PB_p12_Rev1.pdf.
[2] [n. d.]. Free BSD Page Coloring. https://www.freebsd.org/doc/en/articles/vm-design/page-coloring-optimizations.html.
[3] [n. d.]. Intel RDT Software Package. https://github.com/01org/intel-cmt-cat.
[4] [n. d.]. Intel Resource Director Technology. https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html.
[5] Major Bhadauria and Sally A. McKee. 2010. An Approach to Resource-aware Co-scheduling for CMPs. In *ICS'10*. 189–199.
[6] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University.
[7] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. In *ASPLOS '17*. 17–32.
[8] Quan Chen, Hailong Yang, Jason Mars, and Lingjia Tang. 2016. Baymax: QoS Awareness and Increased Utilization for Non-Preemptive Accelerators in Warehouse Scale Computers. In *ASPLOS '16*. 681–696.
[9] Henry Cook, Miquel Moreto, Sarah Bird, Khanh Dao, David A. Patterson, and Krste Asanovic. 2013. A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-efficiency While Preserving Responsiveness. In *ISCA '13*. 308–319.
[10] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS'13*. 77–88.
[11] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. In *ASPLOS '14*. 127–144.
[12] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt. 2010. Fairness via Source Throttling: A Configurable and High-performance Fairness Substrate for Multi-core Memory Systems. In *ASPLOS'10*. 335–346.
[13] N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sanchez. 2018. KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores. In *HPCA '18*. 104–117.
[14] Liran Funaro, Orna Agmon Ben-Yehuda, and Assaf Schuster. 2016. Ginseng: Market-driven LLC Allocation. In *USENIX ATC '16*. 295–308.
[15] Alexandros-Herodotos Haritatos, Georgios Gourmas, Konstantinos Nikas, and Nectarios Koziris. 2016. A resource-centric Application Classification Approach. In *COSH'16*. 7–12.
[16] Alexandros-Herodotos Haritatos, Georgios Goumas, Nikos Anastopoulos, Konstantinos Nikas, Kornilios Kourtis, and Nectarios Koziris. 2014. LCA: A Memory Link and Cache-aware Co-scheduling Approach for CMPs. In *PACT'14*. 469–470.
[17] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.
[18] Andrew Herdrich, Ramesh Illikkal, Ravi Iyer, Don Newell, Vineet Chadha, and Jaideep Moses. 2009. Rate-based QoS Techniques for Cache/Memory in CMP Platforms. In *ICS'09*. 479–488.
[19] Andrew Herdrich, Edwin Verplanke, Priya Autee, Ramesh Illikkal, Chris Gianos, Ronak Singhal, and Ravi Iyer. 2016. Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family. In *HPCA'16*. 657–668.
[20] Ravi R. Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Donald Newell, Yan Solihin, Lisa R. Hsu, and Steven K. Reinhardt. 2007. QoS policies and architecture for cache/memory in CMP platforms. In *SIGMETRICS'07*. 25–36.
[21] Aamer Jaleel, William Hasenplaugh, Moinuddin K. Qureshi, Julien Sebot, Simon C. Steely Jr., and Joel S. Emer. 2008. Adaptive insertion policies for managing shared caches. In *PACT'08*. 208–219.
[22] Aamer Jaleel, Hashem H. Najaf-abadi, Samantika Subramaniam, Simon C. Steely, and Joel Emer. 2012. CRUISE: Cache Replacement and Utility-aware Scheduling. In *ASPLOS'12*. 249–260.
[23] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. 2010. High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP). In *ISCA '10*. 60–71.
[24] Yunlian Jiang, Xipeng Shen, Jie Chen, and Rahul Tripathi. 2008. Analysis and Approximation of Optimal Co-scheduling on Chip Multiprocessors. In *PACT '08*. 220–229.
[25] Harshad Kasture and Daniel Sanchez. 2014. Ubik: Efficient Cache Sharing with Strict Qos for Latency-critical Workloads. In *ASPLOS '14*. 729–742.
[26] R. E. Kessler and Mark D. Hill. 1992. Page Placement Algorithms for Large Real-indexed Caches. *ACM Trans. Comput. Syst.* 10, 4 (Nov. 1992), 338–359.
[27] Kishore Kumar Pusukuri, Rajiv Gupta, and Laxmi N. Bhuyan. 2013. ADAPT: A framework for coscheduling multithreaded programs. *ACM Trans. Archit. Code Optim.* 9, 4, Article 45 (Jan. 2013), 24 pages.
[28] Jiang Lin, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang, and P. Sadayappan. 2008. Gaining Insights into Multicore Cache Partitioning: Bridging the Gap between Simulation and Real Systems. In *HPCA'08*. 367–378.
[29] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *ISCA'15*. 450–462.
[30] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations. In *MICRO-44*. 248–259.
[31] Paul Marshall, Kate Keahey, and Tim Freeman. 2011. Improving utilization of infrastructure clouds. In *CCGrid'11*. 205–214.
[32] Andreas Merkel, Jan Stoess, and Frank Bellosa. 2010. Resource-conscious scheduling for energy efficiency on multicore processors. In *EuroSys'10*. 153–166.
[33] Kyle J. Nesbit, Miquel Moretó, Francisco J. Cazorla, Alex Ramírez, Mateo Valero, and James E.i Smith. 2008. Multicore Resource Management. *IEEE Micro* 28, 3 (2008), 6–16.
[34] Konstantinos Nikas, Matthew Horsnell, and Jim D. Garside. 2008. An adaptive bloom filter cache partitioning scheme for multicore architectures. In *ICSAMOS'08*. 25–32.
[35] Ioannis Papadakis, Konstantinos Nikas, Vasileios Karakostas, Georgios I. Goumas, and Nectarios Koziris. [n. d.]. Improving QoS and Utilisation in modern multi-core servers with Dynamic Cache Partitioning.
[36] Jinsu Park, Seongbeom Park, and Woongki Baek. 2019. CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers. In *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 10.
[37] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO-39*. 423–432.
[38] V. Selfa, J. Sahuquillo, L. Eeckhout, S. Petit, and M. E. Gómez. 2017. Application Clustering Policies to Address System Fairness with Intel's Cache Allocation Technology. In *PACT '17*. 194–205.
[39] Yannis Sfakianakis, Christos Kozanitis, Christos Kozyrakis, and Angelos Bilas. 2018. QuMan: Profile-based Improvement of Cluster Utilization. *ACM Transactions on Architecture and Code Optimization (TACO)* 15, 3 (2018), 27.
[40] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. 2003. Discovering and Exploiting Program Phases. *IEEE Micro* 23, 6 (Nov. 2003), 84–93.
[41] Shekhar Srikantaiah, Mahmut T. Kandemir, and Mary Jane Irwin. 2008. Adaptive set pinning: managing shared caches in chip multiprocessors. In *ASPLOS'08*. 135–144.
[42] Lingjia Tang, Jason Mars, and Mary Lou Soffa. 2011. Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures. In *EXADAPT '11*. 12–21.
[43] Lingjia Tang, Jason Mars, Wei Wang, Tanima Dey, and Mary Lou Soffa. [n. d.]. ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computers. In *ASPLOS '13*. 89–100.
[44] X. Wang, S. Chen, J. Setter, and J. F. Martínez. 2017. SWAP: Effective Fine-Grain Management of Shared Last-Level Caches with Minimum Hardware Support. In *HPCA '17*. 121–132.
[45] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. 2011. SHiP: Signature-based Hit Predictor for High Performance Caching. In *MICRO '11*. 430–441.
[46] Yuejian Xie and Gabriel H. Loh. 2009. PIPP: Promotion/Insertion Pseudo-partitioning of Multi-core Shared Caches. In *ISCA'09*. 174–183.
[47] Hailong Yang, Alex D. Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers. In *ISCA'13*. 607–618.
[48] Xiao Zhang, Sandhya Dwarkadas, and Kai Shen. 2009. Towards Practical Page Coloring-based Multicore Cache Management. In *EuroSys '09*. 89–102.
[49] Haishan Zhu and Mattan Erez. 2016. Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems. In *ASPLOS '16*. 33–47.
[50] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. 2010. Addressing Shared Resource Contention in Multicore Processors via Scheduling. In *ASPLOS '10*. 129–142.