CInC: Workload Characterization In Context of Resource Contention

Cesar Gomes and Mark Hempstead

Tufts University

Abstract

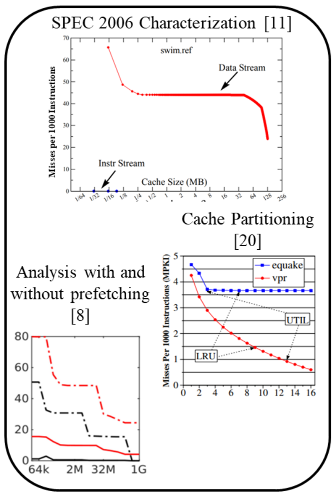
Increased contention for hardware resources is a consequence of more, distinct workloads running on the same system; this is especially true for a shared cache. However, shared cache analysis is dominated by methods that center on how a workload behaves when run alone, or in isolation. *Though evaluating a workload in isolation is important, it is increasingly irresponsible to not evaluate a workload under contention as a standard part of the analysis.*

Characterizing in Context, or CInC is both a position paper on modern performance analysis and a framework of contention-forward performance analysis. This work provides evidence that it is no longer responsible to make decisions using a workload's isolation behavior. Additionally, we provide a way to talk about the contention that is formal and clear. Further, we provide two methods for cache sensitivity analysis built on top of the CInC framework: one simplifies multi-capacity-curve analysis via a Condensed Representation Model and the other distills capacity curves into Curve Analysis Features. We also expand curve analysis metrics by expressing the rich information available in a workload's contention behavior through a novel measurement of stability under contention. The tools enable a contention-forward characterization of a subset of SPEC 2017 rate workloads run on a real system. Additionally, we present a case study exploring a CInC-based SVM classifier which we apply towards co-scheduling (accuracy 88% vs 78% with a clustering tool).

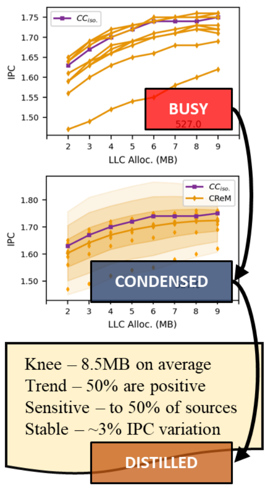
1. Motivation: Studying Contention is Messy

Contention analysis is necessary with increasing hardware complexity and system utilization. However, standard analysis tools like capacity curves that can be found across the literature [9, 11, 14, 22] are always generated from isolation experiments. Further, designers tacitly justify not investigating workloads in highly utilized systems when designing systems to optimize individual workload performance. The logic is on display in architectural designs that include per workload profiling architecture [12, 19, 20], or frameworks to study workloads in facsimile isolation. Here, we motivate the need for standard analysis tools to understand contention behavior.

What is Standard for Cache Analysis? Capacity Curves in Literature. Analyzing how a metric varies with a change in configuration is a standard tool in cache sensitivity studies; when cache size is changed we consider this a capacity curve. Figure 1 (a) shows cache-sensitivity capacity curves from different published works on varying topics: a workload characterization [11]; cache partitioning paper [20]; and work on architecturally agnostic curve generation [8]. While there exists varieties of capacity curves [9, 14, 22], analyzing many curves concurrently is overwhelming, and we exacerbate this problem with the addition of inter-workload contention. Figure 1 (b) shows a family of performance capacity curves generated for the 527.cam4 workload. The x-axis reflects the range of last level cache allocations, and the y-axis shows Instructions Per Cycle (IPC). It's clear that quick analysis is difficult with this view, but *condensing* representation enables easier insight. Further, *distilling* that insight into formal features allows the data to be described in text. Both of these methods are a contribution of CInC.



(a) Capacity Curves in prior art



(b) 527.cam4 vs 11 different sources of contention

Figure 1: Capacity Curves from the literature: (a) Prior art uses isolation capacity curves in various characterizations and use cases [8, 11, 20], but rarely do they use contention curves; (b) Focusing on 527.cam4, we see capacity curves shift under different sources of contention, but condensing and distilling insights makes for easier analysis.

2. Characterizing In Context

We present Characterization In Context or **CInC**, a framework for capacity curve analysis under an ever-growing likelihood of resource contention in real systems.

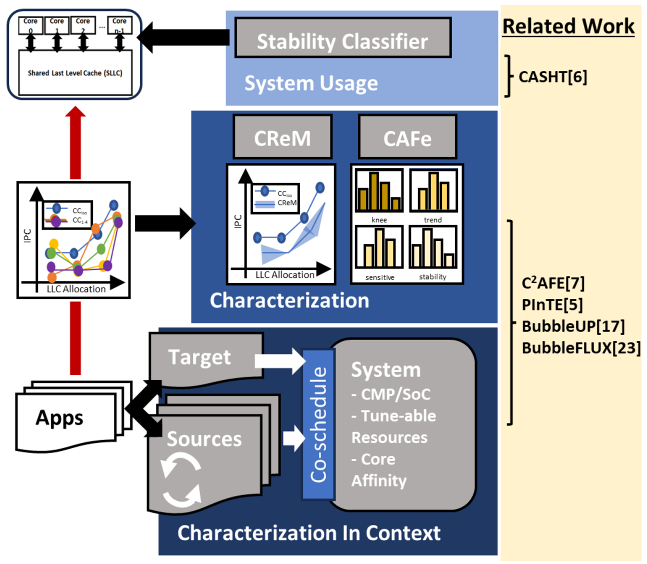


Figure 2: Characterization In Context: High-level illustration of CInC forming a foundation for characterization and tools for use in a system.

Capacity curves are a tool to inspect target workloads for insights that will drive system designs and configurations. Building a single capacity curve starts with capturing **performance metrics** for a **target** workload at each, potential resource configuration — the result is the performance metric (y-axis) as a function of resource configuration (x-axis). CInC proposes to evaluate workloads concurrently in order to study the impact of different **sources** of contention. Making CInC a standard of performance analysis yields multiple curves for one workload, and our framework addresses this data explosion in two ways: condensing the visualization, and distilling each curve into formally-defined features.

2.1. CInC-ing Capacity Curves at a High-Level

CInC formalizes contention analyses seen in prior work [5–7, 17, 23], and expands the field with methods addressing the resulting complexity of studying experiments across different **contexts**. Figure 2 shows that data generated via CInC (co-scheduling in this case) can be a foundation that enables novel characterization and system usage paradigms.

2.2. Condensing and Distilling

Condensing Capacity Curves. We present Condensed Representation Modeling, or **CReM** as a solution to visualizing numerous capacity curves in a *condensed* and insightful format. In Figure 1 (b), the plot labeled **CONDENSED** shows distinct characteristics of CReM: a regression line [10], and shaded intervals derived via standard deviation that envelope this line. Listing 1 shows how to construct a CReM.

Distilling Capacity Curves. We present Curve Analysis Features, or **CAFe** to distill capacity curves into formulaic features to answer the following questions to best describe the workload behavior:

- How much capacity is needed to minimize performance loss, or where is the **KNEE** of our curve?;
- Does the capacity curve show a positive, negative, or mixed **TREND** with more capacity?;
- Does performance change significantly as capacity changes, or how **SENSITIVE** is the target workload to changes in capacity?
- Does performance change with different sources of contention, or is it **STABLE**[1]?

The power of CAFe resides in providing formulaic solutions, rather than empiricism, to the above questions in order to mitigate error.

Listing 1: Generating CReM: Code assumes pandas data frame that is filtered for contention data and for a specific target workload, and that matplotlib and numpy are imported; NOTE: m refers to the performance metric.

```
...
# Plot scatter plot of capacity curve data
x    =   contention['Cache Allocation']
y    =   [round(I,2) for I in contention[m]]
ax.scatter(x, y)
# Generate & Plot LOESS
...
cinc_loess = [...]
ax.plot(x, cinc_loess)
# Compute standard deviation per allocation
std =   []
for i in x:
    tmp =    contention['COS'==i]
    std.append(numpy.std(tmp[m].tolist()))
# Compute lower and upper interval bounds
lowr=   [[] for i in range(0,3)]
uppr=   [[] for i in range(0,3)]
for i in range(0,3):
    for j,l in enumerate(cinc_loess)
        lowr[i].append(l-(std[j]*(k+1)))
        uppr[i].append(l+(std[j]*(k+1)))
# Plot standard deviation intervals
for i in numpy.linspace(2,0,3,dtype=int):
    ax.fill_between(x,lowr[i],uppr[i])
...
```

TABLE 1: Curve Analysis Features: I = configurable sensitivity parameter; m = array of performance metrics captured from each resource configuration experiment; K = Knee; i,j = cache allocation low and high bounds; hmean = harmonic mean; [:] = value list; Pos. = Positive; Neg. = Negative; IPC.len = number of elements in curve; *assumes single curve; +assumes multiple curves

Feature	Definition
Knee*	$K\ if\ \frac{hmean(m_{K:j})}{metric_K} < 1 + (I/100)\%,$ $K\ in\ [i...j],\ i < j$
Trend*	$Pos.\ if\ \sum_{K=i} int(m_K \geq m_{K-1}) = m_K.length$ $Neg.\ if\ \sum_{K=i} int(m_K < m_{K-1}) = m_K.length$ $Mix\ otherwise$
Sensitivity*	$sensitive\ if \frac{max(m)}{min(m)} > 1 + (I/100))$
Stability+	$\sum_{K=n} \frac{stdv(m_K[:])}{hmean(m_K[:])} / m_K.len$

1. Measuring in this way is novel to the contention context

TABLE 2: CAFe for a representative subset of SPEC 2017 Workloads: Data is sorted according to the stability feature; Please note that CAFe detected no negative trends.

	Knee (MB)		Trend Distribution		Sensitivity Distribution		Stability (%)		
workload	mean	min	% positive	% mix	% sensitive	% insensitive	mean	max	min
538.imagick_r.0	7.25	2	0.00	100.00	8.33	91.67	0.17	0.19	0.13
548.exchange2_r.0	5.08	2	0.00	100.00	0.00	100.00	0.17	0.33	0.12
541.leela_r.0	8.25	7	0.00	100.00	0.00	100.00	0.30	0.33	0.27
511.povray_r.0	7.17	4	0.00	100.00	0.00	100.00	0.34	0.84	0.13
525.x264_r.0	8.25	5	25.00	75.00	0.00	100.00	0.83	0.95	0.70
527.cam4_r.0	8.50	7	50.00	50.00	50.00	50.00	2.98	3.40	2.26
531.deepsjeng_r.0	7.92	5	25.00	75.00	0.00	100.00	3.39	3.76	3.11
507.cactuBSSN_r.0	9.00	9	75.00	25.00	16.67	83.33	5.73	6.60	5.28
519.lbm_r.0	9.00	9	100.00	0.00	100.00	0.00	5.77	6.49	4.66
502.gcc_r.3	8.83	8	75.00	25.00	100.00	0.00	7.66	8.50	6.90
520.omnetpp_r.0	7.58	4	16.67	83.33	100.00	0.00	9.43	11.89	6.86
500.perlbench_r.2	8.75	7	25.00	75.00	83.33	16.67	9.55	11.46	7.18

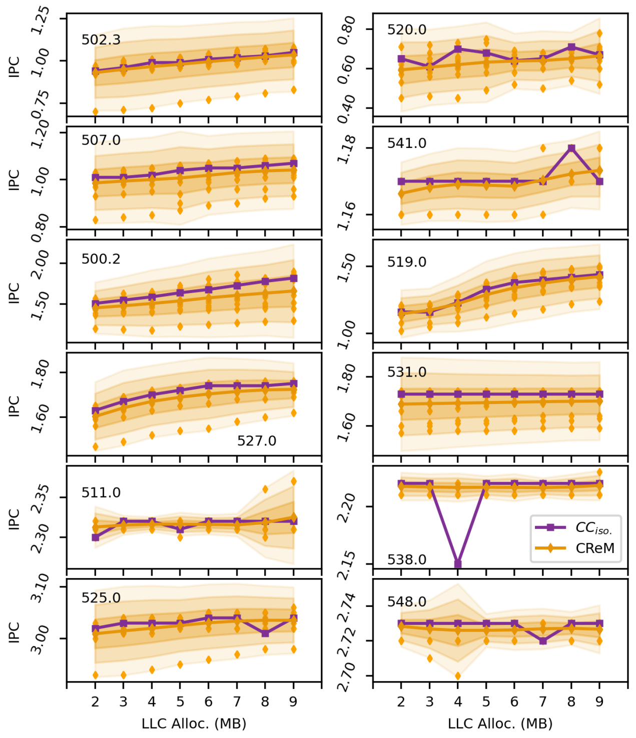


Figure 3: SPEC 2017 CReM: The CReM per workload, where x is the cache allocation in MB and y is IPC; Isolation ($CC_{iso.}$) and contention-context data points included for contrast; y is unique per plot for clear analysis per workload.

3. CInC-ing SPEC 2017

We generate CReM and CAFe data for a subset of SPEC 2017 which was previously identified as representative [15].

3.1. Classifying with CReM

CReM allows for simultaneous cache and contention analysis despite the number of capacity curves, as shown in Figure 3. Analysis yields the classes defined in Table 3, and workloads are grouped accordingly:

TABLE 3: CInC Classification: CAFe-based classifications for capacity curves.

Class	Definition
Cache Agnostic	workload(s) are *insensitive* to capacity changes
Cache Satisfied	workload(s) have *knees* below the max capacity
Cache Starved	workload(s) are *sensitive* and *knees* equal max capacity
Contention Stable	workload(s) have low *stability* value, or are stable
Contention Unstable	workload(s) have a *stability* measure of at least 1%

- Cache Agnostic - 511.povray, 525.x264, 531.deepsjeng, 538.imagick, 541.leela, and 548 exchange2;
- Cache Satisfied - 527.cam4
- Cache Starved - 500.perlbench, 502.gcc, 507.cactuBSSN, 519.lbm, and 520.omnetpp;
- Contention Stable - covers the cache agnostic workloads except for 531.deepsjeng;
- Contention Unstable - all other workloads.

3.2. Classifying with CAFe

Distilling curves into formal quantities makes a comparison between curves simple, and Table 2 lists data based on CAFe. We sort rows by the average stability column where lower means the workload behavior is *stable* across experiments. Sorting in this way also shows correlations between other features and stability. For instance, a *stability* value less than 1 indicates that a workload is *cache insensitive*. Further, *stable* workloads are shown to have *knees* further from the maximum capacity which supports the insight that such workloads do not depend on shared cache. Additionally, the correlation between the *stability* and the *mixed trend* CAFe suggests there is performance variation as the cache allocation changes, though this is largely noise. For stability above 1, correlations are difficult to determine since these workloads are classified as unstable and cache-satisfied/starved.

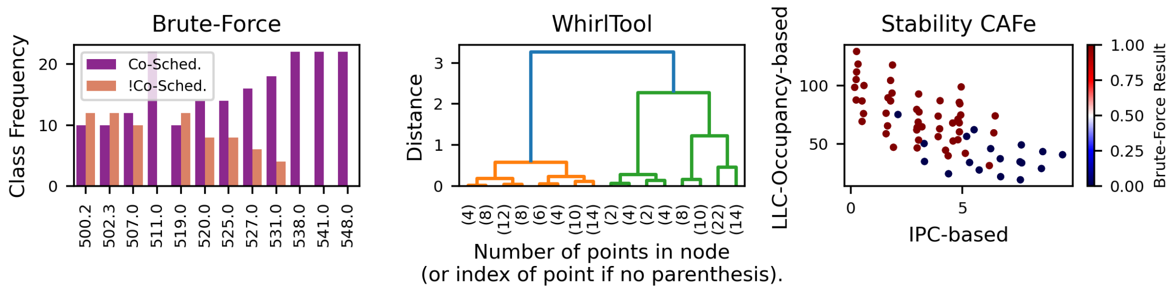


Figure 4: Co-scheduling Outcomes Observed Three ways: Different classification methods for experiment results generated with our CInC method; (left) A brute-force classification of co-scheduling; (center) Dendrogram from WhirlTool which clusters workload capacity curve distances; (right) Observable clusters in a scatter plot generated with CAFe.

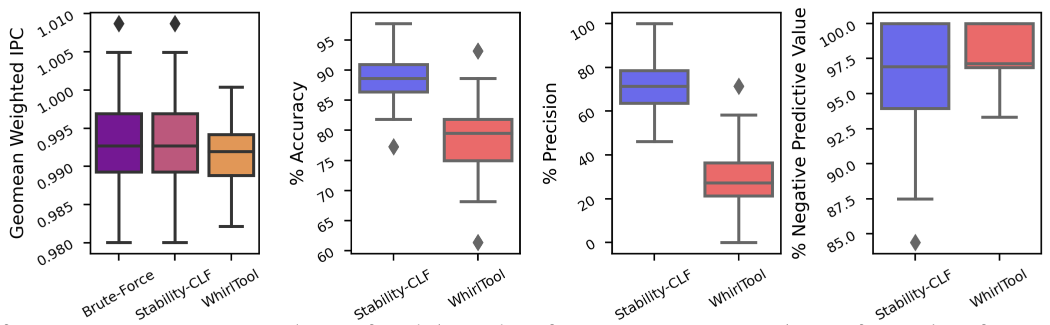


Figure 5: Performance & Accuracy Analysis of Stability Classifier: Comparative analysis of our classifier against prior art and brute force; (first) Weighted IPC is comparable between our classifier and Brute-Force classification, and trends higher than WhirlTool; (second) The stability classifier is more accurate than WhirlTool on average; Stability Classifier has an advantage classifying co-scheduling workloads, but a lower bound on no-co-scheduling classifications than WhirlTool.

4. Case Study: CInC-designed Co-Scheduling

Multi-tenancy, or the condition that different workloads run concurrently on a system incurs optimization problems like whether or not to co-schedule [2, 13, 16, 24]. Solutions to co-scheduling either have prior knowledge of how workloads behave under contention, or make estimates based on isolation behavior [1, 4, 18, 21]. Here, we investigate applying a CInC-driven classifier towards co-scheduling.

4.1. CAFe-based Scheduler

We generate CAFe for each metric we captured via Intel RDT (IPC, Misses, BW[MB/s], and LLC occupancy in kB), and exhaustively plot each CAFe against each other. Co-scheduling can be reduced to a binary classification (**co-schedule** or **not**), and we label each experiment by whether the weighted sum IPC reduces by at least 1%.

Stability provides clear clusters. Figure 4 shows our experiments classified with three methods: Brute-Force, WhirlTool [18], and clusters apparent when plotting Stability CAFe against each other. For CAFe, we select the IPC- and L3-Occupancy-based stability since they yield the clearest clusters once labeled. We train a support vector machine with the cluster data to create the Stability-Classifier (-CLF).

4.2. Experimental Setup

CInC-generated Data. WhirlTool uses isolation data while Stability-CLF uses contention data. We use contention-context IPC in weighted IPC calculations to reflect co-scheduled performance amd isolation-context IPC to model running on separate systems (not co-scheduled).

Clustering in Isolation. WhirlTool is trained with isolation-IPC-based capacity curves, configure agglomerative clustering to compute the average distance between clusters as the distance metric, set the number of clusters (n) to 2, and classify based on which clusters experiments fall in.

Evaluation. We generate 100, random test and training sets from 132 experiments split 88 in test and 52 in training. Accuracy is computed as the number of accurate predictions over the number of experiments tested, while performance is the geometric mean of weighted IPC [3] for all experiments.

4.3. Results

Figure 5 shows box plots representing the range of performance and accuracy for the stability classifier and WhirlTool. Results show our classifier has a higher ceiling in performance and accuracy. Average performance for each method are in noise margins, but our classifier has a higher upper bound. Additionally, the variability suggests sensitivity to the split of training and testing set, which means more workloads should minimize the sensitivity and potentially yield consistently higher values for the stability classifier Regarding accuracy, the Stability Classifier has higher accuracy (88%) than WhirlTool (78%).

5. Conclusion

CInC proposes we elevate contention to the first-class in computer architecture analysis, and presents frameworks to enable this elevation for engineers and architects. CInC-driven potential is exemplified by an accurate co-scheduling classifier trained on novel contention metrics.

References

[1] N. Beckmann and D. Sanchez, “Jigsaw: Scalable software-defined caches,” in *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on.* IEEE, 2013, pp. 213–224.

[2] D. Byrne, N. Onder, and Z. Wang, “Mpart: Miss-ratio curve guided partitioning in key-value stores,” in *Proceedings of the 2018 ACM SIGPLAN International Symposium on Memory Management*, ser. ISMM 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 84–95. [Online]. Available: https://doi.org/10.1145/3210563.3210571

[3] L. Eeckhout, *Computer architecture performance evaluation methods*, ser. Synthesis lectures on computer architecture, #10. San Rafael, Calif: Morgan & Claypool Publishers, 2010.

[4] N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sanchez, “Kpart: A hybrid cache partitioning-sharing technique for commodity multicores,” in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 02 2018, pp. 104–117.

[5] C. Gomes, X. Chen, and M. Hempstead, “PInTE: Probabilistic induction of theft evictions,” in *2022 IEEE International Symposium on Workload Characterization (IISWC).* Los Alamitos, CA, USA: IEEE Computer Society, 11 2022, pp. 1–13. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IISWC55918.2022.00011

[6] C. Gomes, M. Amiraski, and M. Hempstead, “CASHT: Contention analysis in shared hierarchies with thefts,” *ACM Trans. Archit. Code Optim.*, vol. 19, no. 1, 03 2022. [Online]. Available: https://doi.org/10.1145/3494538

[7] C. Gomes and M. Hempstead, “c^2afe: Capacity curve annotation and feature extraction for shared cache analysis,” in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020, pp. 113–115.

[8] M. Hassan, C. H. Park, and D. Black-Schaffer, “A reusable characterization of the memory system behavior of spec2017 and spec2006,” *ACM Trans. Archit. Code Optim.*, vol. 18, no. 2, 03 2021. [Online]. Available: https://doi.org/10.1145/3446200

[9] X. Hu, X. Wang, L. Zhou, Y. Luo, Z. Wang, C. Ding, and C. Ye, “Fast miss ratio curve modeling for storage cache,” *ACM Trans. Storage*, vol. 14, no. 2, 04 2018. [Online]. Available: https://doi.org/10.1145/3185751

[10] W. G. Jacoby, “Loess:: a nonparametric, graphical tool for depicting relationships between variables,” *Electoral Studies*, vol. 19, no. 4, pp. 577–613, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0261379499000281

[11] A. Jaleel, “Memory characterization of workloads using instrumentation-driven simulation a pin-based memory characterization of the spec cpu 2000 and spec cpu 2006 benchmark suites,” in *VSSAD Technical Report 2007*, 2007.

[12] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, “High performance cache replacement using re-reference interval prediction (rrip),” in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA ’10. New York, NY, USA: ACM, 2010, pp. 60–71. [Online]. Available: http://doi.acm.org/10.1145/1815961.1815971

[13] X. Jia, J. Jiang, T. Zhao, S. Qi, and M. Zhang, “Towards online application cache behaviors identification in cmps,” in *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, Sept 2010, pp. 1–8.

[14] R. Koller, A. Verma, and R. Rangaswami, “Estimating application cache requirement for provisioning caches in virtualized systems,” in *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, 07 2011, pp. 55–62.

[15] A. Limaye and T. Adegbija, “A workload characterization of the spec cpu2017 benchmark suite,” *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 149–158, 2018.

[16] J. Mars and M. L. Soffa, “Synthesizing contention,” in *Proceedings of the Workshop on Binary Instrumentation and Applications*, ser. WBIA ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 17–25. [Online]. Available: https://doi.org/10.1145/1791194.1791197

[17] 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*, ser. MICRO-44. New York, NY, USA: Association for Computing Machinery, 2011, p. 248–259. [Online]. Available: https://doi.org/10.1145/2155620.2155650

[18] A. Mukkara, N. Beckmann, and D. Sanchez, “Whirlpool: Improving dynamic cache management with static data classification,” in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 113–127. [Online]. Available: https://doi.org/10.1145/2872362.2872363

[19] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, “Adaptive insertion policies for high performance caching,” in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA ’07. New York, NY, USA: ACM, 2007, pp. 381–391. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250709

[20] M. K. Qureshi and Y. N. Patt, “Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,” in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 39. Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432.

[21] P. Tsai, N. Beckmann, and D. Sanchez, “Jenga: Software-defined cache hierarchies,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 06 2017, pp. 652–665.

[22] C. Waldspurger, N. Park, A. Garthwaite, and I. Ahmad, “Efficient mrc construction with shards,” 02 2015.

[23] 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*, ser. ISCA ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 607–618. [Online]. Available: https://doi.org/10.1145/2485922.2485974

[24] D. Zhan, H. Jiang, and S. C. Seth, “Clu: Co-optimizing locality and utility in thread-aware capacity management for shared last level caches,” *IEEE transactions on computers*, vol. 63, no. 7, pp. 1656–1667, 2014.