Adrias: Interference-Aware Memory Orchestration for Disaggregated Cloud Infrastructures

Dimosthenis Masouros*, Christian Pinto†, Michele Gazzetti†, Sotirios Xydis*‡ and Dimitrios Soudris*
*Microprocessors and Digital Systems Laboratory, School of Electrical and Computer Engineering
National Technical University of Athens, Greece
Email: {dmasouros, sxydis, dsoudris}@microlab.ntua.gr
†IBM Research Europe, Dublin, Ireland
Email: {christian.pinto, michele.gazzetti1}@ibm.com
‡Department of Informatics and Telematics
Harokopio University of Athens, Greece

***Abstract*—Workload co-location has become the de-facto approach for hosting applications in Cloud environments, leading, however, to interference and fragmentation in shared resources of the system. To this end, hardware disaggregation is introduced as a novel paradigm, that allows fine-grained tailoring of cloud resources to the characteristics of the deployed applications. Towards the realization of hardware disaggregated clouds, novel orchestration frameworks must provide additional knobs to manage the increased scheduling complexity.**

We present Adrias, a memory orchestration framework for disaggregated cloud systems. Adrias exploits information from low-level performance events and applies deep learning techniques to effectively predict the system state and performance of arriving workloads on memory disaggregated systems, thus, driving cognitive scheduling between local and remote memory allocation modes. We evaluate Adrias on a state-of-art disaggregated testbed and show that it achieves 0.99 and 0.942 R^2 score for system state and application's performance prediction on average respectively. Moreover, Adrias manages to effectively utilize disaggregated memory, by offloading almost 1/3 of deployed applications with less than 15% performance overhead compared to a conventional local memory scheduling, while clearly outperforms naive scheduling approaches (random and round-robin), by providing up to $\times 2$ better performance.

***Index Terms*—Memory disaggregation, Data placement, Deep Learning, Cloud Infrastructures, Orchestration, Interference**

I. Introduction

Cloud computing has been established as the new standard for application deployment due to the flexibility and cost effectiveness it offers [60]. Cloud systems typically suffer from high resource under-utilization, due to the strict performance requirements of deployed applications, which push providers to dedicate machines for guaranteeing performance [34], [74], [82]. To tackle this under-utilization problem, multi-tenancy [18], [19] has been adopted as the de-facto deployment model for Cloud applications. However, multi-tenancy leads to interference in shared resources, which in turn induces variability and degradation in the performance of applications [25], [26], [59], [65]. As a result, intelligent orchestration and allocation of computing resources is required [25], [44], [58], [85], where complex cluster-wide software mechanisms control how hardware resources are assigned to applications.

Cloud systems typically involve two layers of resource management which are orthogonal to each other and can be applied independently, i.e., *L1)* The initial static allocation of resources and placement of incoming applications (so called resource orchestration) and *L2)* the dynamic adjustment of allocated resources to meet requirements of applications (so called runtime management). Mechanisms in the first category should be able to identify the resource requirements of incoming (possibly unknown) applications, and avoid placements that lead to resource interference, while also considering the underlying HW heterogeneity [9], [25]–[27], [61]. The second layer includes runtime mechanisms that dynamically optimize the performance of running applications, such as SW controllers [19], [29] and/or OS-integrated extensions [87], [91] that regulate resources of deployed applications.

Despite their sophistication, software only mechanisms have been proven incapable of fully resolving the resources under-utilization problem, that is mostly a bi-product of the diverse computational requirements of cloud workloads combined with the fixed resource proportionality of cloud-server systems. As a consequence, it is common in modern data-centers to observe a fragmentation of resources that are available yet not consumable by any workload [42], [46].

To overcome this resource wall challenge, *hardware disaggregation* has been proposed as a new design paradigm [4], [45], [75], [76]. In a disaggregated cloud, the underlying HW infrastructure is organized as pools of heterogeneous resources that can be composed on-demand into compute units tailored around workload-specific requirements. Recent scientific research has examined the applicability of the disaggregated concept to several components of modern Cloud and HPC infrastructures, including processing units [10], [21], [33], [73], memory [32], [57], [72], [84] and storage [10], [48], [53], [66]. In addition, rack-scale operating systems [81] and runtimes [14] have been proposed for dynamically managing disaggregated resources.

Need for memory orchestration: This paper focuses on memory-disaggregated infrastructures addressing the newly induced problem of interference-aware memory orchestration.

In a memory disaggregated system, orchestrated placement to memory resources is required to minimize the impact on applications performance due to the increased latency in accessing remote memory [50]. While prior research efforts have thoroughly examined dynamic runtime mechanisms (L2 – e.g., page migration/prefetching) for memory-disaggregated and multi-tiered memory systems [47], [62], [91], limited work has been conducted with respect to the problem of interference-aware memory placement in disaggregated clouds (L1), since existing scheduling approaches [25], [26], [41] neither target memory orchestration nor been have extensively examined for such composable systems. However, orchestration of memory resources in disaggregated environments is critical for two main reasons: *i)* In presence of interference, determining an efficient memory mapping can significantly improve the overall performance of the application and *ii)* Optimal initial allocation of memory can minimize the amount of data travelling back and forth through the network, e.g., in applications that present a low ratio of hot versus cold pages

Paper Contributions: We introduce *Adrias*[1], an interference-aware memory orchestration framework that enables effective/optimized data placement decisions on memory-disaggregated cloud infrastructures. The key features of Adrias could be summarized through: *i)* its ability to forecast the tendency of system-wide metrics in the future, thus driving proactive memory orchestration decisions; *ii)* its accurate performance predictions for deployed applications w.r.t. memory heterogeneity (local/fast vs. remote/slow DRAM) and interference and *iii)* its power to leverage disaggregated memory with minimal impact on the performance of deployed applications without the employment of dynamic memory management mechanisms. Adrias exploits system-level performance monitoring information and leverages deep learning approaches to place incoming applications on the pool of available memory resources. To the best of our knowledge, this is the first work tackling the problem of interference-aware memory orchestration, i.e. applications' data placement on memory-disaggregated cloud systems. Our main contributions are:

- We perform an in-depth exploration and provide new insights on the performance capabilities of the state-of-art ThymesisFlow disaggregated memory testbed [72]. We characterize ThymesisFlow under various interference scenarios for a set of in-memory cloud workloads, namely Redis, Memcached and several Spark analytics and analyze the impact of memory disaggregation w.r.t. their performance.
- We propose two deep learning models tailored to disaggregated memory systems; *i)* a system state prediction model that forecasts the tendency of monitored performance events in the future and *ii)* a performance prediction model that estimates the performance of applications, when deployed on memory disaggregated systems. Using these models, we are able to accurately predict the tendency of system metrics and performance of incoming applications, achieving up to 0.999 and 0.942 R^2 scores respectively.
- We present Adrias, an orchestration framework for memory disaggregated systems. By leveraging the developed prediction models and a simple, yet effective, scheduling logic, Adrias employs remote memory efficiently, by offloading up to 35% of best-effort applications with less than 15% performance degradation compared to a local DRAM memory allocation approach and also provides comparable QoS guarantees for latency-critical ones.

[1]Adrias was a WWII battleship that hit an underwater mine and was split in two parts but managed to survive and return to service.

II. Related Work

Memory Disaggregation: Memory disaggregation is not a new subject of study, with several approaches appearing over the years. There are fully software-based approaches that expose remote memory as Linux swap devices or rely on RDMA transfers to be explicitly programmed for moving memory blocks to/from remote memory [3], [32], [52], [68], [77], [81], [86]. Efficient use of RDMAs often involves having to reserve and pin chunks of memory beforehand, leading to inefficient utilization of memory resources. A number of full hardware solutions have also been proposed [14], [35], [72] that although different, they are all mostly based on intercepting low level CPU memory operations to process and forward them towards remote systems.

▸ In this work we focus on the ThymesisFlow open-source hardware [72] that requires zero modifications to existing applications and it integrates seamlessly with the Linux memory management system.

Cloud Resource Orchestration: Prior research works have proposed solutions both for static resource allocation of applications arriving on a cluster [25], [26], [31], [41], [44], [89], as well as dynamic regulation of allocated resources of workloads throughout execution [19], [43], [51], [67], [71], [80], [96]. Moreover, production ready resource orchestrators have also been presented [11], [23], [40], [83], [85], showcasing the need for efficient supervision of resources due to interference in shared resources. However, these works focus on typical cloud infrastructures, that consist of conventional servers. Other scientific approaches examine the problems of application orchestration on disaggregated memory systems [54], [69] and performance modeling [93], however they either rely on emulated prototypes [93] or totally neglect the implications of resource interference in shared resources [54], [69].

▸ Adrias goes a step further, by providing an interference-aware orchestrator for memory disaggregated systems, deployed and evaluated over a real, state-of-the-art testbed.

Memory Management of Tiered Systems: Lately, management of multi-tiered systems and/or disaggregated, heterogeneous resources inside data-center facilities is attracting more attention. Recent research efforts focus on mechanisms that provide dynamic disaggregated memory allocations for VMs [15], [16], [64], efficient prefetchers and replacement policies [47], [55], [56], [62], [63], [91] and cost/performance tradeoffs between heterogeneous memory pools [87].

▸ These works belong to runtime management techniques (L2), are orthogonal to Adrias, since they operate on different levels of optimization and can be utilized complementary.

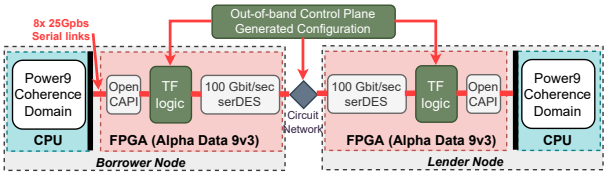


Fig. 1: ThymesisFlow [72] architecture overview

III. Disaggregated Memory Testbed

Prototype Setup: We replicate the ThymesisFlow [2], [72] memory-disaggregated prototype consisting of two AC922 IBM® POWER9™ servers, featuring two sockets with 64 logical cores, 10MB L3 cache per socket and 1.2TB DRAM @2666MHz. Both servers run RHEL (kernel `v5.8.0`) and are equipped with an Alpha Data 9V3 card, that features a Xilinx Ultrascale FPGA. The FPGAs are connected back-to-back using a single copper cable that models a 100Gbps point-to-point connection in a circuit switched network fabric.

Hardware Architecture: Fig. 1 depicts the hardware infrastructure of the ThymesisFlow prototype. On both servers the FPGAs are interfaced to the CPU bus using the OpenCAPI [22] prototcol, that enables coherent access to the CPU memory from an accelerator. The interface between CPU and FPGA is based on 8x25Gbps serial links for a total of 200Gbps. On the borrowing side, the OpenCAPI accelerator is mapped at a specific physical location in the CPU bus that can be then hot-plugged as regular memory. On the lender side the OpenCAPI accelerator accesses the memory on behalf of the borrower by totally bypassing the lender CPU, thus avoiding any unnecessary overhead. ThymesisFlow enables byte-addressable disaggregated memory that does not require any software support such as in the case of solutions based on RDMA [32], [62] and those enabling disaggregated memory by means of a Linux swap device [3], [86]. Every time a memory access performed at the borrower side causes a last level cache miss or flush, the cache line is refilled/flushed via remote memory. An OpenCAPI transaction (read or write) is generated by the borrowing CPU and received by the FPGA. Here the transaction address is modified to a valid one at the lender side and the operation traverses the circuit network (100Gbps) towards the FPGA at the remote node, where the OpenCAPI transaction is re-issued on the bus towards the memory. Responses in case of a read follow the reverse path.

Software Architecture: From the software standpoint, the two servers in the prototype are not symmetric. ThymesisFlow enables the borrower-lender model, where the lender gives away part of its local memory for access from a remote borrower. On the borrower server, ThymesisFlow exposes disaggregated memory as a CPU-less NUMA node, that can be attached using the `libthymesisflow` library. This allows users to avail of all Linux default NUMA-aware functionalities. Alternatively, users can either hotplug the disaggregated memory to the running Linux system in the borrower server, or keep it out from the Linux kernel memory management system and use custom memory allocators. In this paper, we hotplug the disaggregated memory and we control how applications access it by means of ad-hoc Linux cgroups.

IV. Characterizing Memory Disaggregation

We analyze the impact of resource contention on the ThymesisFlow memory disaggregation testbed. We unveil important insights concerning both potential hardware limits of memory disaggregated systems and the impact of interference on the performance of applications leveraging remote memory.

A. Examined Workloads

Cloud infrastructures typically host two types of workloads, best-effort (BE) and latency-critical (LC) ones. The former require the highest possible throughput while the latter have strict QoS guarantees. To cover both types of applications, we examine the following open-source, in-memory applications:

- **Redis (LC):** Redis [17] is a NoSQL key/value store that keeps data in memory. We examine the impact of local and remote memory allocation modes on Redis server instances, serving user requests.
- **Memcached (LC):** Memcached [28] is a distributed memory object caching system, used to cache data and objects in the main memory.
- **Spark in-memory analytics (BE):** Apache Spark [94] is an analytics engine for large-scale data processing. We evaluate 17 different spark applications derived from the HiBench benchmark suite [39], using the default Spark parameter configuration and the *small* dataset as described in [39]. We study the impact of disaggregated memory only on the executor processes, which perform all the task computations.

Load generation: We use asymmetric load generation on the underlying system, by co-locating LC, BE and iBench [24] interference microbenchmarks, as described in SectionV-B1. For LC workloads, we utilize the `memtier_benchmark` [1], i.e. the official Redis Labs' utility for load generation of NoSQL key-value databases. Tail latency is measured using a set of closed-loop memtier clients [79] over asymmetric co-located workloads. More in detail, our setup spawns 4 threads, where each thread spawns 200 clients, i.e. eliminating client-bias [49], [97]. We use a SET:GET ratio of 1:10 and generate a constant load of 10000 and 40000 requests per client. This configuration leads to a total of approximately 30.000 and 100.000 operations served per second for Redis and Memcached respectively, which closely relate to realistic loads found in production, e.g., Facebook services [5], [90].

B. Limits of HW memory disaggregation on ThymesisFlow

We first assess the capacity of our testbed while performing memory operations, by examining the effect of repeated data movement between the local and remote system. In more detail, we spawn an increasing number (1 to 32) of memory bandwidth interfering micro-benchmarks [24] and force them to use memory borrowed from the remote node, thus generating traffic on the link between the FPGA devices in the ThymesisFlow testbed. To highlight interference effects, we monitor system-level metrics related to the local system and

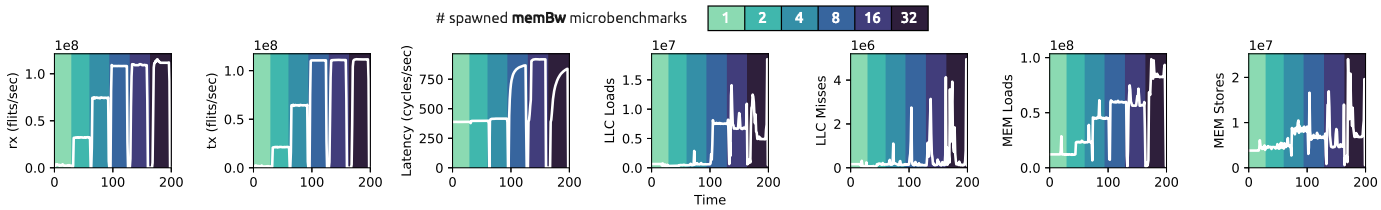


Fig. 2: Capacity of our memory disaggregated testbed for different stressing scenarios. Different color shades denote different number of memory bandwidth stressing microbenchmarks [39] deployed on remote memory mode.

the network interconnection. Specifically, for the local system we gather the Last Level Cache (LLC) loads and misses, and the memory loads and stores, whereas for the FPGAs we measure the number of flits (32B) received (`rx`) and transmitted (`tx`), and the average latency on the ThymesisFlow communication channel. Fig. 2 shows the results and also reveals three important remarks (R1-R3).

R1) Bounded throughput: The throughput of the disaggregated memory reveals an upper bound in the amount of transferred data, with a cap of approximately 2.5Gbps[2]. This value reveals that remote memory has approximately a three orders of magnitude lower bandwidth threshold compared to conventional DDR4 memory systems, which can support a theoretical of 120GBps sustained memory bandwidth [78].

R2) Communication latency: For low to medium amounts of generated traffic (1 up to 4 memory bandwidth microbenchmarks) the average latency on the communication channel follows a steady state of approximately 350 cycles/sec. However, in cases of increased traffic (8 up to 32 microbenchmarks) the latency is almost tripled, reaching a yield plateau of 900 cycles/sec. Until the 4 micro-benchmark, the prototype is capable of handling the memory requests received and the latency remains constant across all executions, while bandwidth increases steadily. From 8 micro-benchmarks onwards, the channel is saturated (bandwidth plateau) and the back-pressure mechanism implemented in the FPGAs starts delaying memory transactions, hence the step in latency.

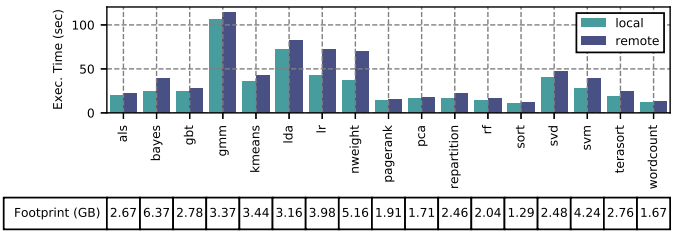
R3) Local system interference: Application deployment on remote memory also induces interference on the memory hierarchy of the local system. This is expected for chip-level metrics (e.g., LLC Loads and Misses), as the cache memory hierarchy lies beneath the abstraction layer of local/remote memory accesses. Regarding memory loads and stores, remote pages are memory-mapped and handled through an enhanced `numactl` memory controller and, thus, all remote traffic is handled on-chip by memory controllers of the local node.

C. Workload characterization

We further quantify the impact of local and remote allocations on the performance of our applications when executed in isolation and under different interference scenarios.

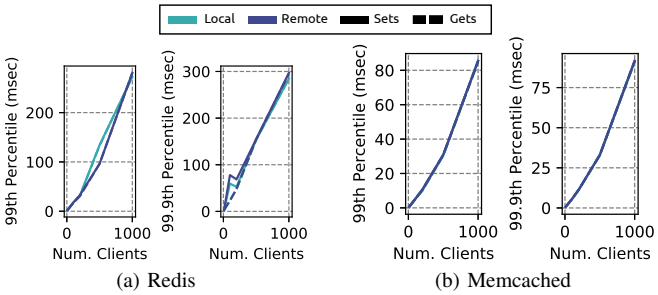
Execution in isolation: For LC applications (Redis, Memcached), we examine the 99*th* and 99.9*th* response percentiles

[2]Given that each flit is 32B, we compute `rx` and `tx` bandwidth in b/s by multiplying with 32*8 [72].



Footprint (GB)	2.67	6.37	2.78	3.37	3.44	3.16	3.98	5.16	1.91	1.71	2.46	2.04	1.29	2.48	4.24	2.76	1.67

Fig. 3: Spark performance characterization (total execution time) when executed in isolation on local and remote memory.



(a) Redis (b) Memcached

Fig. 4: Tail latency of LC applications with increasing RPS when executed in isolation on local and remote memory.

(tail latency) under different loads by scaling the number of clients requesting `Set` and `Get` operations. For BE (Spark) ones, we examine the execution time of applications for the two memory allocation modes. Fig. 3 and Fig. 4 show the results for LC and BE applications respectively.

R4) Non-uniform performance variation: For Redis and Memcached applications we confirm the results reported in prior work [72] that local and remote memory provide almost identical curves in terms of tail latency, for all the configurations examined. This similarity in performance occurs due to the fact that in-memory caches perform many small read and write memory accesses with minimal bandwidth pressure requirements, which can be easily attained based on the specifications of our disaggregated system. For Spark applications, we notice an average degradation of 20% over all our examined benchmarks, which, however, is not uniform across all benchmarks tested. For example in Fig. 3, we observe that `nweight` and `lr` suffer almost a ×2 slowdown when ran on remote memory, whereas others, such as `gmm` and `pca` experience less than 10% performance degradation. This reveals that remote memory is suitable for some applications

while it is not the best options for others.

Execution under interference: Last, we investigate the relative performance impact of resource contention on different levels of the system hierarchy, between local and remote memory. We deploy the same application and measure its performance on local and remote memory under the four interference scenarios namely: `cpu`, `l2`, `l3` and `memBw`. For both modes (local and remote), we spawn a different number (1 up to 16) of resource trashing micro-benchmarks [24], targeting different resources on the system (CPU, L2-cache, Last-Level-Cache and Memory Bandwidth), i.e., if the application is deployed on local memory, so are the ibench microbenchmarks and vice-versa. Fig. 5 shows the respective results, where the density of each cell depicts the performance slowdown of the respective scenario between local and remote memory.

R5) Performance chasm under contention: After a certain threshold (typically 16 for `L3` and > 8 for `memBw` micro-benchmarks), the same amount of interference results in much higher performance degradation on the remote memory, reaching up to ×4 additional slowdown in certain cases. Combined with the results presented in Fig.2, we observe that this threshold corresponds to the saturation point in the communication channel of the FPGAs. This is true both for BE and LC applications, with the latter appearing to be more resistant to interference effects. This shows that remote memory gets saturated much more easily than local DRAM, which also confirms our observation regarding the limitation of remote memory bandwidth made in Section IV-B.

R6) LLC vitality: Contention on the LLC has the greatest negative impact for the majority of BE applications. Interference on the LLC leads to consecutive misses, which in turn are translated to increased memory bandwidth due to read/write accesses from/to the main memory. While both LLC and memory bandwidth end up in generating traffic on the channel, we see that data locality and caching is of paramount importance, as intense LLC contention (16 spawned microbenchmarks) leads to the worst possible performance degradation for the majority of the Spark applications. Moreover, a sustained and increased interference effect on the memory network bus, leads to gradual performance degradation, relative to the extent of the underlying interference. Last, since in-memory databases rely heavily on pointer chasing operations, which introduce poor on-chip spatial locality [20], [38], they appear to be less cache sensitive, revealing higher response times only on memory bandwidth interference scenarios.

R7) Stacking interference effects: For certain benchmarks (e.g., `nweight`, `sort`, `kmeans`), we also notice a performance gap between local and remote memory when imposing interference on lower levels of the system hierarchy (i.e., CPU and L2 cache). We call this a *stacking interference effect*. For such applications, we expect the remote memory allocation mode to be a prohibitive option under realistic scenarios, where different resources are congested simultaneously.

D. Affinity of system & workload metrics

Section IV-B revealed that low-level system metrics can provide insightful information regarding the state of the system. Taking also into account the high performance variability shown in Fig. 5, it is evident that being able to project low-level performance events to higher-level metrics of interest (e.g., application slowdown), would allow us to estimate performance solely through the assessment of lower-level metrics. To investigate whether such a relationship exists, we examine the correlation between low-level system and high-level application metrics prior and during execution when deployed using the remote memory allocation mode. Specifically, we generate several deployment scenarios (similar to the ones described in Section V-B1) by randomly co-locating different ibench workloads with the examined benchmarks, and we keep track of the underlying system metrics during execution.

We evaluate the linear correlation between the average system performance metrics 120 seconds prior to application scheduling (x), as well as during execution ($\tilde{x}$), with the application performance, using the Pearson's correlation coefficient. For Spark applications we consider as performance the total execution time, whereas for Redis and Memcached we study the end-to-end latency, the 99^{th} and the 99.9^{th} percentiles. Fig. 6 shows the respective results and clearly reveals the existence of a correlation between certain metrics and the performance of applications. What is of great interest is that runtime metrics reveal a much higher correlation compared to the historical ones, forming our concluding remark:

R8) Predictive monitoring capability: Proactive runtime assessment of the state of the underlying system is feasible and provides useful insights both regarding the system itself, as well as the performance of deployed applications.

V. Adrias Design

The main goal of Adrias is to efficiently orchestrate applications arriving in a disaggregated system, by deciding between local and remote memory modes. Fig. 7 shows an overview of Adrias' architecture. The *Watcher* component continuously monitors and gathers performance events of the underlying system. The *Predictor* exploits Long Short-Term Memory (LSTM) models for forecasting the future state of the system and performance of deployed applications. Finally, the *Orchestrator* utilizes the predictions to decide the memory allocation policies accordingly.

A. Watcher

The *Watcher* component is responsible for gathering performance events from the underlying hardware infrastructure, providing insights on the data flowing through the memory hierarchy of the system. Focus is given on cache- and memory-related performance counters, as well as metrics that depict the status of the communication channel between the local and the remote memory sub-system. Driven by our analysis in Section IV-B, we monitor the following metrics: Last-level cache misses (LLC_{mis}), Last-level cache loads (LLC_{ld}), Local DRAM memory loads (MEM_{ld}), Local DRAM memory

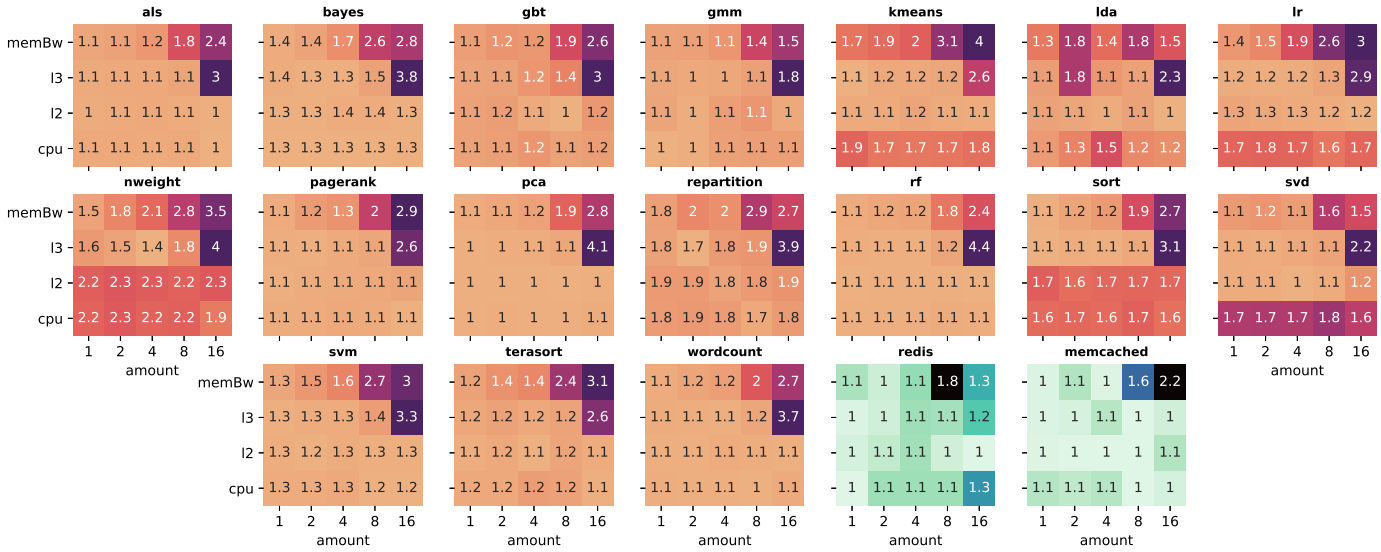


Fig. 5: Benchmark performance characterization under interference on local and remote memory. The heatmap density denotes the execution time and 99^{th} percentile slowdown of remote vs. local execution for BE (orange) and LC (teal) applications respectively, under different interference scenarios (`cpu`, `l2`, `l3`, `memBw`) using iBench [24] contention microbenchmarks.

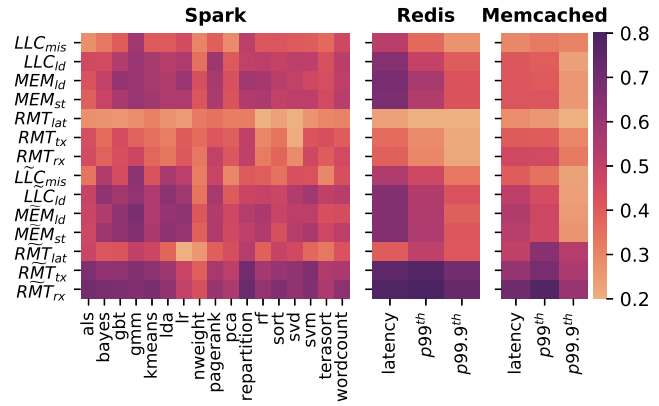


Fig. 6: Correlation of historical (x) and runtime ($\tilde{x}$) system performance events with performance of deployed applications on remote memory.

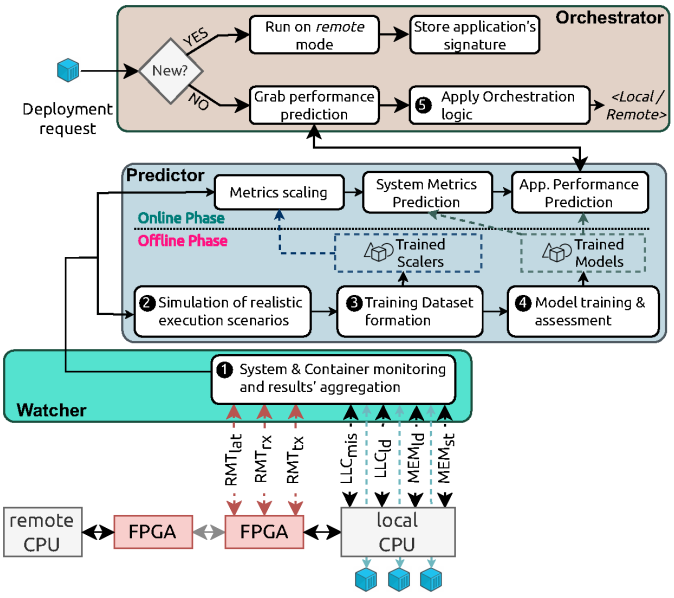


Fig. 7: Overview of Adrias architecture

stores (MEM_{st}), FPGAs communication's channel average latency (RMT_{lat}) and FPGAs receive (RMT_{rx}) and transmit (RMT_{tx}) throughtput. For the CPU performance events of the local system, we utilize Linux's `perf` tool, while the events related to the FPGA channel are directly provided by the ThymesisFlow framework [72]. We set the monitoring interval equal to 1 sec, which, as shown in [30], is a "sweet spot" between inference overhead and QoS violations increment.

B. Stacked-LSTM Predictor

The *Predictor* forecasts the future state of the disaggregated system and predicts the performance of incoming applications w.r.t. the memory allocation mode (local vs remote). The prediction process consists of two phases, *offline* and *online*. The offline phase (design-time) involves three main activities: *1)* collection of representative system metrics' traces that correspond to "realistic" execution scenarios (❷), *2)* generation of the dataset used for training and testing (❸) and *3)* design, train and validation of the prediction models (❹). In the online phase (run-time), the Predictor utilizes the trained models to predict the aforementioned prediction metrics of interest.

1) Offline phase: Interference-Aware trace collection: The first step of the offline phase concerns the collection of interference-aware traces of low-level system metrics (by utilizing the Watcher component of Section V-A). The data collected is used as input dataset for training our deep learning models, described later in this section. This step is vital for the overall functionality of Adrias, since gathering representative

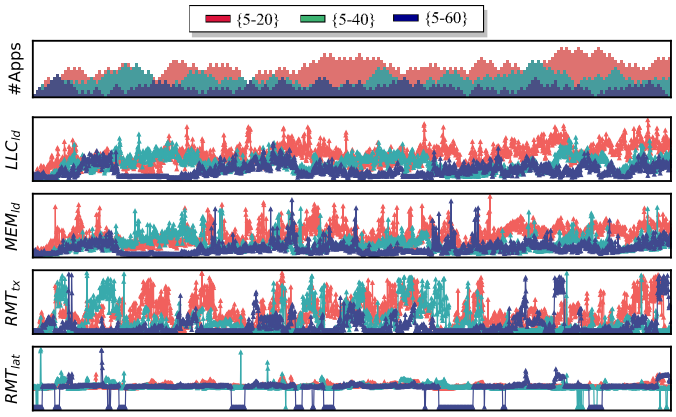


Fig. 8: Number of concurrent applications (top) and performance metrics over time for three representative scenarios. Adrias' data acquisition scheme captures different congestion phases, both within the same and among different scenarios.

data is essential to increase accuracy of any DNN model. **Scenario generation:** We simulate different execution scenarios by employing a random scenario generation approach. Each scenario is characterized by a *spawn interval* $\{t_1, t_2\}$, which denotes the arrival time range of consecutive application deployments on the system. For instance, a spawn interval of $\{5, 40\}$ means that each new application arrives after a random interval between 5 and 40 seconds. Within each interval we pick a random benchmark either from the examined applications, or from the iBench pool and we deploy it randomly on local or remote memory. Through iBench micro-benchmarks, we aim to replicate supplementary interference scenarios that cannot be directly generated by our examined LC and BE workloads. To capture the high dynamicity found in Cloud environments [74], we examine different arrival rates, with sets ranging from $\{5, 20\}$ up to $\{5, 60\}$, where the former imitate more congested scenarios and the latter indicate a more relaxed application arrival pattern. Fig. 8 depicts three exemplary but representative scenarios, assuming heavy ($\{5, 20\}$), moderate ($\{5, 40\}$) and less ($\{5, 60\}$) congested application deployment distributions[3]. As shown, the specific setup exposes a wide variety of phases, both regarding the number of concurrent applications and the spectrum of the monitored metrics.

Insights from scenario execution: Overall, we have simulated 72 diverse 1-hour scenarios with different spawning intervals. Fig. 9 and Fig. 10 show the performance distribution of our examined benchmarks over all the 72 execution scenarios.

▸ Regarding Spark benchmarks (Fig. 9), the use of remote memory has a substantial performance impact compared local DRAM, since the distributions for the scenarios using remote memory expose a tendency towards higher values. However, certain benchmarks (e.g., `gmm`) present overlapping performance distributions, between local and remote memory.

[3]Through the random scenario generation, the maximum number of applications running simultaneously on the disaggregated testbed is 35, with Spark applications spawning 2 worker instances with 4 threads each.

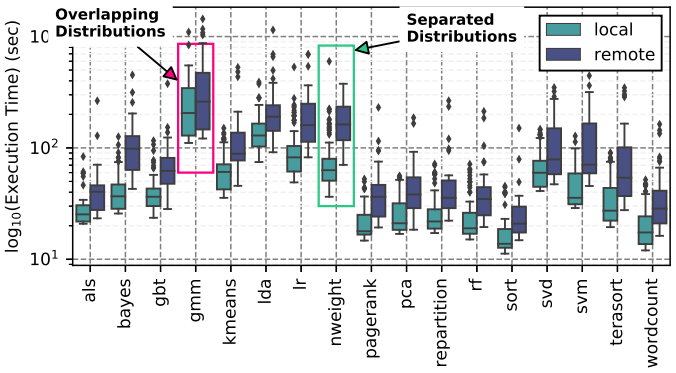
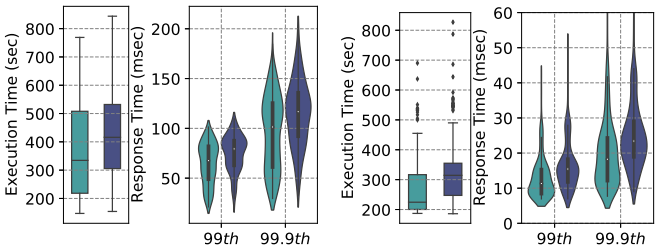


Fig. 9: Spark performance distribution over different execution scenarios.



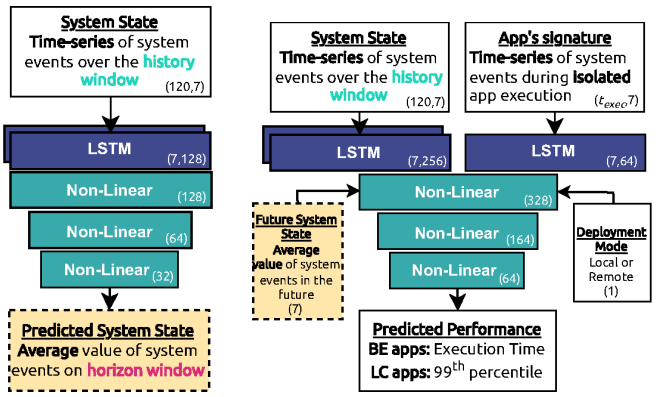
(a) Redis (b) Memcached

Fig. 10: Distribution of total execution time to serve 10.000 requests (left) and of 99*th* and 99.9*th* percentile of response time per request (right) for Redis and Memcached over difference execution scenarios.

This denotes that there might be certain cases where remote memory could provide better performance than local (even if local memory latency is in the order of ~ 80*ns* and remote memory latency ~ 900*ns*), due to high contention in the memory bus of the latter compared to low interference in the communication channel of the former. Moreover, considering that BE applications typically do not have strict performance requirements, we would be able to sacrifice performance to take advantage of the disaggregated memory. In contrast, there are benchmarks (e.g., `nweight`), which, as also described in Sec. IV-C, do not take advantage of remote memory at all, due to stacking interference effects.

▸ For Redis and Memcached, we examine the 99^{th} and 99.9^{th} percentiles, which form typical QoS requirements of LC applications. We observe that remote memory provides higher response times, however we again notice overlapping performance distributions between the two modes. Overall, we anticipate disaggregated memory to be prohibitive for stricter QoS constraints, especially in the Redis case. However, when more relaxed QoS requirements are set, remote memory could be leveraged without violating any constraints.

2) Offline phase - Prediction models training: Adrias utilizes a stacked-model architecture by combining two prediction models one after another: a *system state* prediction model, that forecasts the future values of low-level system



(a) System's state model architecture

(b) Performance prediction model architecture

Fig. 11: DNN architecture of (a) system's state and (b) application's performance prediction models. Parentheses shows the #features and size of each layer.

metrics, and a *performance prediction* model, that receives these predictions (among others) and infers the performance of an application if deployed on local or remote memory. Fig. 11a and Fig. 11b show an abstract view of the models' architecture.

System State Model: The rationale behind this model originates from the observations made in Section IV-D that runtime system metrics are more closely correlated to applications' performance. The model receives as input the *System State – S*, a two-dimensional feature vector containing the time-series values for each examined metric over a history window of length r. We define the system state by considering the metrics monitored by the Watcher component (V-A). As output, the model provides the *Predicted System State* – $\hat{S}$, a vector that corresponds to the predicted mean value of system performance events over a horizon window z. After evaluating different values for r and z , we have determined that a value of 120 seconds provides useful insights both regarding the history and the horizon windows.

Due to the sequential nature of the input, we utilize Long Short-Term Memory (LSTM) [37] layers as the backbone of the model, which has been proven to be extremely accurate in forecasting system level metrics under interference and for deep horizons [30], [65]. The input feature vector is first processed by 2 LSTM layers that identify dependencies between the time-series data and the results are passed to a triplet of non-linear blocks, that combine fully-connected layers with ReLU activation functions, batch normalization and dropout layers to expose non-linearity and avoid overfit.

Performance Prediction Model: This model forecasts the performance of incoming applications, if deployed on local or remote memory. We follow a universal modeling approach, i.e., we build a unique model for all the BE and one for all the LC applications, where the former predicts the execution time and the latter the 99^{th} response time percentile. Although prior research works typically follow a per-application modeling approach [8], [25], [92], we argue that this is not efficient (yet could be more accurate), since building a performance model per application is too time-consuming and requires simulating scenarios for each new application and maintaining a single model per workload imposes serious scalability issues.

Modeling and predicting the performance of applications is a non-trivial task and requires awareness regarding: *i)* the dynamics and sources of interference on the underlying system and *ii)* the inherent characteristics of the application itself and how these characteristics get affected from the current and future status of the system. To uncover this information, the performance models receive as inputs four parameters: The past and predicted system state feature vectors S and $\hat{S}$, the deployment mode (local/remote) and the *application's signature* – k, which is a unique identifier per application, that contains the sequences of monitored metrics during application's execution in isolation on remote memory mode.

The time-series inputs (S,k) are individually processed by two LSTM layers that identify important features in the sequential data. The output results are then concatenated with the deployment mode and the future system state vector $\hat{S}$ to form the hidden representation, which is then processed again by a triplet of non-linear blocks to provide the final performance prediction.

C. Orchestrator

The *Orchestrator* leverages the predicted metrics to proactively assess the state of the system and choose the disaggregated memory policy for deployed applications accordingly. Its design is driven by two fundamentals: *i)* Cloud policies typically apply QoS guarantees for LC workloads and best-effort execution for batch applications [25], [58] and *ii)* Disaggregated memory systems hide dangerous pitfalls (characterization process of Section IV), i.e., disaggregated memory imposes significant performance overhead if utilized recklessly, especially when multiple applications compete over the available resources and, thus, should be leveraged wisely, targeting applications that benefit the most out of it. We tackle the first aspect directly, by introducing a straightforward, yet effective, orchestration logic for BE and LC applications, while the second one is addressed indirectly, through the automatic feature extraction of Adrias' prediction models.

When a new workload is deployed on the system, if Adrias does not own any prior information regarding its *application signature*, it schedules it on the remote memory, captures and stores the respective metrics. Otherwise, it communicates with the Predictor and receives the estimated execution time (for BE) or 99^{th} percentile (for LC) for local and remote memory modes. For BE, we utilize the following discrete function to decide the deployment mode:

$$mode_{BE} = \begin{cases} local, & \text{if } \hat{t}_{local} < \beta * \hat{t}_{remote}. \\ remote, & \text{otherwise.} \end{cases} \quad (1)$$

where $\hat{t}$ is the predicted execution time and β is a slack parameter depicting the maximum performance loss margin that we are willing to sacrifice to leverage remote memory. Choosing β

Event	R^2
LLC_{mis}	0.9969
LLC_{ld}	0.9995
MEM_{ld}	0.9641
MEM_{st}	0.9983
RMT_{lat}	0.9977
RMT_{rx}	0.9871
RMT_{tx}	0.9876
Avg.	0.9932

TABLE I: R^2 score per performance event

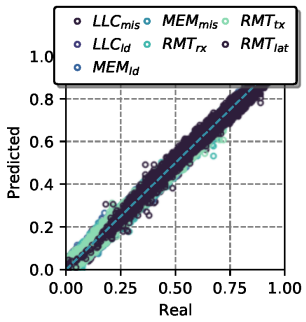


Fig. 12: Performance events predictions' regression residuals

depends on two factors. First, the application itself, as various applications present diverse performance characteristics when deployed on remote memory (Sec. IV). Second, the underlying interference, as overwhelming the remote memory requires greater performance degradation tolerance.

For LC workloads, Adrias aims to utilize remote memory without violating a pre-established QoS constraint. The memory allocation mode is chosen as follows:

$$mode_{LC} = \begin{cases} remote, & \text{if } \hat{p}^{99th}_{remote} \leq QoS. \\ local, & \text{otherwise.} \end{cases} \quad (2)$$

where $\hat{p}^{99th}$ is the predicted 99^{th} response time percentile. Particularly for LC applications, achieving QoS requirements solely through performance assessment during deployment can turn out to be infeasible, due to the unpredictability of the system's future load. In such cases, Adrias can be utilized complementary with other runtime management frameworks, e.g., [19], [67], [91], to dynamically adjust resources.

VI. Evaluation

We implement Adrias[4] using Python (`v3.7.0`), ZeroMQ [36] and the PyTorch library [70]. Adrias imposes minimal overhead on the system, with an average of 300MB RAM utilization and occasional CPU usage spikes of $\approx 15\%$, that correspond to predictions for newly deployed applications.

A. Accuracy Evaluation

We partition the datasets produced during simulation (Section V-B1) in two subsets of 60% (training set) and 40% (test set) of the samples. We examine the accuracy of the system state and performance models for predicting the mean value of monitored metrics over the horizon window and the performance (execution time/tail latency) of applications.

1) System state prediction model: Table I shows the results per metric, by evaluating the coefficient of determination [7] - R^2. Adrias achieves pretty high accuracy, ranging from 0.964 up to 0.999 R^2 score and with an average of 0.993 R^2 overall, illustrating its capability to proactively assess the tendency of system metrics in the future. Fig.12 also presents the actual versus predicted metrics, as there exist cases where a high R^2 score could be counter-intuitive [6]. This plot verifies the strong prediction capabilities regarding the system's state, since the majority of the points lie on the 45^o residual line.

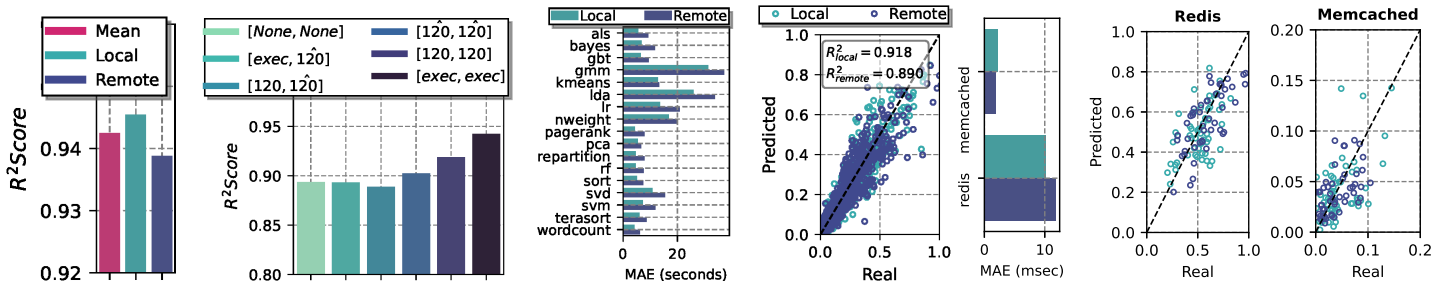
2) Application performance prediction models: As a first step, we train and test the performance models for BE applications, using as the future system state ($\hat{S}$) the actual monitored metrics, gathered during the trace collection process. Fig.13a depicts the respective results, showing that Adrias is able to achieve a $0.942R^2$ score on average, with a slightly higher accuracy for predicting the execution time on local mode ($R^2 = 0.945$) compared to the remote ($R^2 = 0.939$).

Impact of stacked models to overall accuracy: Apparently, the actual future metrics are not available a priori during a realistic inference step. Thus, a reasonable question is: *Should we train the performance models using as "future system state" the actual system metrics, or train using propagated predictions from the system state model?* To answer, we examine the prediction accuracy for different input vectors ($\hat{S}$) during training and testing. Fig.13b shows the results, where the pair $\{x_1,x_2\}$ maps to {train,test} and denotes the type of vector $\hat{S}$ used in each phase. Specifically, *None* implies that $\hat{S}$ was not fed to the model, while $\{120,1\hat{2}0\}$ and $\{exec, e\hat{x}ec\}$ indicate that $\hat{S}$ is calculated from the actual metrics or propagated from the system state model for a window of 120s or for the full duration of the application respectively. The pairs $\{120,120\}$ and $\{exec, exec\}$ give the best accuracy, which, however, are not pragmatic. In practice, feeding the predicted vector $\hat{S}$ to the performance model in the training phase ($\{1\hat{2}0,1\hat{2}0\}$) is the best approach. It is interesting that while the system state model provided very accurate predictions ($0.99R^2$), we still experience an accuracy drop of 3% compared to the theoretical maximum ($\{exec, exec\}$). Moreover, leveraging the predicted future system yields a 2% higher accuracy compared to historical only data ($[None, None]$), thus, verifying the advantage of predictive monitoring.

Runtime accuracy: By employing the $\{1\hat{2}0,1\hat{2}0\}$ approach, we also show the Mean Absolute Error for BE (Fig. 13c) and LC (Fig. 14a) applications and the actual vs. predicted residuals (Figures 13d and 14b). Comparing the MAE with the median performance presented in Fig. 9, the employed DNN models are able to provide accurate performance predictions for both BE and LC applications. Even in cases where we observe high MAEs (e.g., `gmm`, `lda`), these errors correspond to approximately 10% variation compared to the median values of their performance distribution. Overall, we are able to achieve an R^2 score equal to 0.905 for BE and 0.874 for LC.

Generalization on unseen applications: Last, we test Adrias' universal modeling approach ability, by evaluating accuracy using an application-granular leave-one-out validation. Fig.15a shows the R^2 score per benchmark, when excluded during training phase. The model is able to generalize adequately for certain benchmarks (e.g., `gbt`) whereas it fails for others (e.g., `lda`) yielding 0.72 and 0.30 R^2 scores respectively. This suggests that a continuous collection of representative *application signatures* and retraining is crucial for unknown applications. We explore the

[4]https://github.com/pl4tinum/Adrias



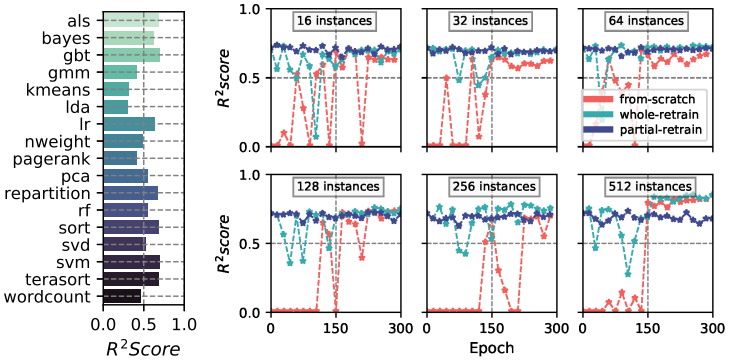
(a) Overall prediction accuracy (b) Impact of real vs. predicted system state on perf. accuracy (c) MAE per Spark benchmark (d) Prediction residuals

Fig. 13: Evaluation of performance prediction model (execution latency) for Best-Effort (BE) applications

(a) MAE (b) Prediction residuals for LC applications

Fig. 14: Evaluation of prediction model (99^{th} percentile) for LC apps

effectiveness of three retraining approaches to improve the accuracy on newly collected data of unseen applications, i.e., *i) from-scratch:* train the whole model from the beginning, *ii) whole-retrain:* retrain all the layers of the model using data from the unseen application and *iii) partial-retrain:* update only the weights of the model that correspond to the application's signature. To avoid bias on the new data during retraining, we set a one-order of magnitude lower learning rate and we feed batches of mixed samples from seen and unseen applications. Fig.15b shows the accuracy achieved and time needed (0.8sec per epoch on a V100 GPU) for different number of training samples for `gbt` application. For lower number of available samples (16-64), whole-retrain results to better accuracy compared to a ground-up training, whereas for higher ones (128-512) it achieves comparable results with less time (epochs) needed. Partial retraining does not improve accuracy over the unseen application, showing the close interrelationship between the application's characteristics (signature) with the dynamics of the system (system state).



(a) Application-granular leave-one-out accuracy (b) Model (re)training accuracy over time (epochs) for various number of samples for `gbt` benchmark

Fig. 15: Performance prediction accuracy for unseen applications (a) and different model retraining approaches (b).

B. Orchestration Evaluation

We compare Adrias with three other scheduling schemes, *i) Random:* mode is chosen randomly, *ii) Round-Robin:* mode chosen in turn between local and remote *iii) All-Local:* all applications allocate local memory.

Adrias' impact on BEs' performance: First, we evaluate the ability of Adrias to utilize the remote memory, without violating the performance threshold introduced through the slack parameter β. Fig.16 (top) shows the execution time distribution of all the examined BE benchmarks and Fig.16 (bottom) the number of times each application got scheduled on the local and remote memory when using the three schedulers and Adrias with different β slack values. For the majority of the workloads, *Random* and *Round-Robin* schedulers provide the worst performance distributions, confirming the need for intelligent orchestration mechanisms. For high betas ($\beta = 1$ and $\beta = 0.9$), Adrias provides identical scheduling decisions with the *All-Local* approach, due to the explicit performance deterioration of the remote memory combined with the implicit accuracy errors of the prediction models. For β equal to 0.8 and 0.7 Adrias achieves to effectively utilize the remote memory, managing to offload approximately 10% and 35% of deployed applications with an average drop of 0.5% and 15% in the median performance over all applications. While the values 0.8 and 0.7 for β would imply an equivalent degradation on the performance of applications, we observe that this is not the case, which is attributed to the accuracy error of the performance prediction model. Finally, for lower slack values (i.e., β=0.6) Adrias offloads the majority of deployed applications to remote memory, which, however, induces significant performance degradation.

Adrias' impact on LCs' performance: Next, we explore Adrias' capability to schedule Redis and Memcached on the remote memory, without violating a pre-established QoS constraint. Based on Fig.10, we define five different QoS levels ($p99^{th}$ response time) of various strictness, i.e., 0 up to 4 per LC application, that correspond to the 87.5^{th}, 75^{th}, 50^{th}, 25^{th}, 12.5^{th} distribution percentiles, where Level 0 denotes the most relaxed and Level 4 the strictest QoS constraint. Fig.17 shows the total number of violations (left) and offloads (right) for all the schedulers. *Random* and *Round-Robin* schedulers provide the worst possible results, since they introduce the highest numbers of QoS violations both for Redis and Memcached, whereas *All-Local* outperforms the

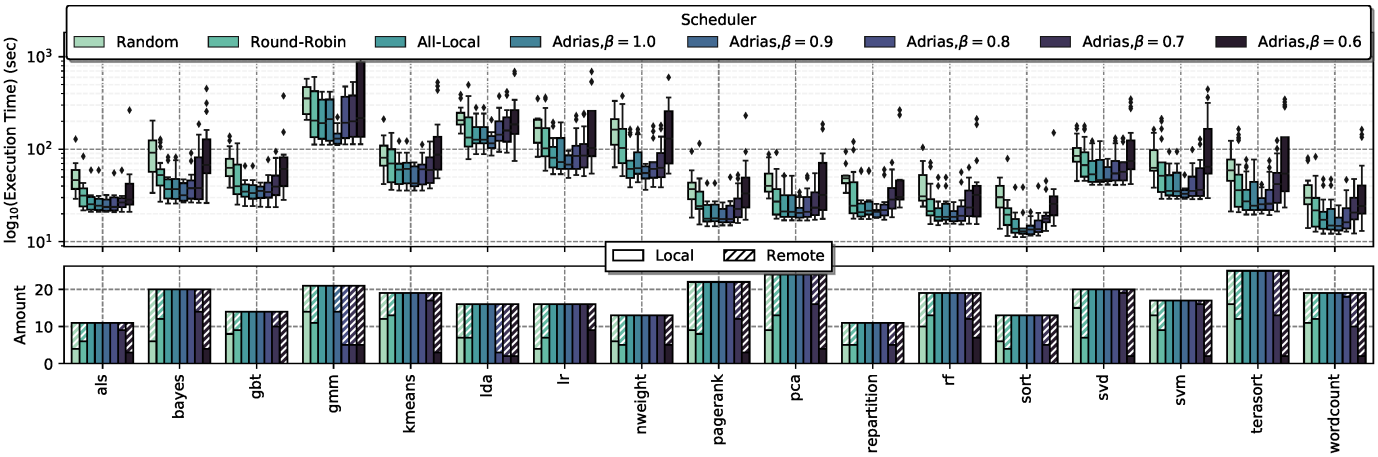
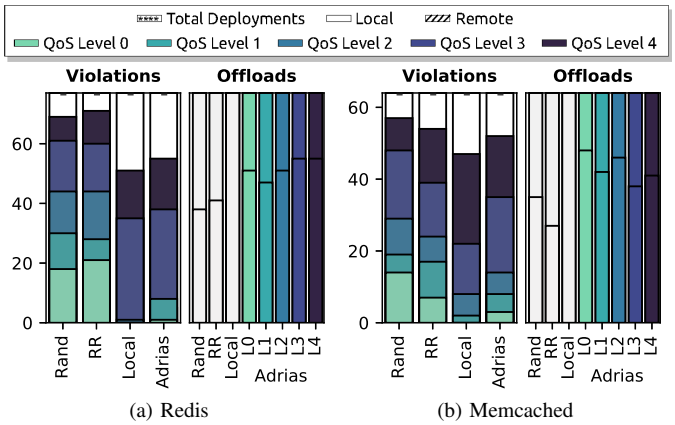


Fig. 16: Execution latency distribution of BE applications (top) and number of times that the application was deployed on local and remote memory (bottom) for different scheduling logics.



(a) Redis (b) Memcached

Fig. 17: Number of QoS Violations (left) and offloads (right) for Redis (a) and Memcached (b) benchmarks.

others by introducing almost no violations for looser QoS constraints and minimum number for stricter ones. For looser QoS levels (0-2), Adrias achieves similar results to the *All-Local* approach, by eliminating the majority of QoS violations, while offloading almost 1/3 of the applications to the remote memory. For stricter ones, Adrias provides comparable results with *All-Local* by introducing an average of 5% and 20% more QoS violations for Redis and Memcached respectively.

Adrias' impact on data traffic: Last, we quantify the amount of transmitted data over the FPGAs' network interconnection. Among all the examined scenarios, Adrias reduces the amount of transmitted data by 45% (β = 0.8) and 23% (β = 0.7) on average compared to Random and Round-Robin schedulers respectively. We note that in cases where Adrias offloads similar number of applications with the other schedulers, it generates up to 55% less traffic on the channel, revealing its tendency to favor less memory-intensive applications to be allocated on the disaggregated memory.

VII. Further Discussion Points

Why Deep Learning? Modern cloud data centers suffer from extensive and non-deterministic performance variability due to interference, workload diversity and HW heterogeneity [74], thus mechanistic or model predictive control approaches form highly expensive solutions due to the extensive simulations required to capture all the possible deployment scenarios [12]. To this end, ML-centric cloud platforms are attracting a lot of attention [9], [88], [95]. The rationale behind the employment of DL techniques is the automatic pattern discovery of neural networks; notably such patterns, able to discriminate performance indicators, could not be effectively set by typical human modeling or would require extensive scenario analysis. In particular, LSTMs have been proven to be extremely efficient on interpreting temporal patterns, i.e., interpreting system monitor time-series to actual performance metrics [30], [65].

Ability of Adrias to unveil human-driven remarks. During evaluation (Fig.16) we noticed that Adrias favors offloading certain applications to the remote memory (e.g., `gmm`, `lda`). While these applications present overlapping performance distributions between local and remote modes (Sec. V-B1), Adrias avoids offloading "non-overlapping" ones (e.g., `nweight`), which also suffer from stacked interference effects (R7). This verifies that Adrias properly models the inherent characteristics of the examined applications.

Adrias & HW heterogeneity. Adrias assumes no prior knowledge on the HW infrastructure, as any performance variability due to heterogeneity will directly affect the monitored metrics. For example, in case a system avails of both remote DRAM and NVMe, these would be considered by Adrias as two different memory tiers, with different latency characteristics. There is no requirement for Adrias to be aware of the actual medium backing each tier.

Adrias scalability. Due to the inherent HW limitations of the ThymesisFlow prototype, Adrias was evaluated on a single-node cluster. However, by design, Adrias is able to

scale on multiple nodes, where the monitoring (Watcher) and performance prediction (Predictor) components are distributed across the nodes of the cluster. The orchestration logic could be centralized (e.g., be integrated directly in the control plane of Kubernetes [13]), however, it should be adjusted in a straightforward manner to account for cluster-level efficiency in case of iso-QoS predictions between different nodes.

VIII. Conclusion

Hardware disaggregation is the next big step for efficient management of cloud infrastructures. In this work, we performed an extensive, interference-aware characterization for a set of cloud applications and highlighted the hidden pitfalls on a real memory disaggregated testbed. Driven by this analysis, we designed Adrias, a resource orchestrator for memory disaggregated cloud systems. Adrias leverages deep learning techniques to decide the memory mode of deployed applications. We showed that Adrias can efficiently utilize remote memory with minimal performance overheads.

Acknowledgements

NTUA authors have been partially supported for this research by the European Union's Horizon 2020 Research and Innovation programme PRIVATEER under Grant Agreement No 101096110. The authors would also like to thank Dr. Georgios Retsinas for his helpful insights and comments regarding the manuscript.

References

[1] "memtier_benchmark: A high-throughput benchmarking tool for redis & memcached," https://github.com/RedisLabs/memtier_benchmark, (Last accessed: 08/02/2022).

[2] "Welcome to the home of thymesisflow," https://github.com/OpenCAPI/ThymesisFlow/, (Last accessed: 16/02/2022).

[3] E. Amaro, C. Branner-Augmon, Z. Luo, A. Ousterhout, M. K. Aguilera, A. Panda, S. Ratnasamy, and S. Shenker, "Can far memory improve job throughput?" in *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, A. Bilas, K. Magoutis, E. P. Markatos, D. Kostic, and M. I. Seltzer, Eds. ACM, 2020, pp. 14:1–14:16. [Online]. Available: https://doi.org/10.1145/3342195.3387522

[4] S. Angel, M. Nanavati, and S. Sen, "Disaggregation and the application," in *12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, July 13-14, 2020*, A. Phanishayee and R. Stutsman, Eds. USENIX Association, 2020. [Online]. Available: https://www.usenix.org/conference/hotcloud20/presentation/angel

[5] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," in *ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '12, London, United Kingdom, June 11-15, 2012*, P. G. Harrison, M. F. Arlitt, and G. Casale, Eds. ACM, 2012, pp. 53–64. [Online]. Available: https://doi.org/10.1145/2254756.2254766

[6] J. P. Barrett, "The coefficient of determination—some limitations," *The American Statistician*, vol. 28, no. 1, pp. 19–20, 1974.

[7] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*. Springer, 2009, pp. 1–4.

[8] A. Bhattacharyya, C. Amza, and E. de Lara, "Phase aware performance modeling for cloud applications," in *13th IEEE International Conference on Cloud Computing, CLOUD 2020, Virtual Event, 18-24 October 2020*. IEEE, 2020, pp. 507–511. [Online]. Available: https://doi.org/10.1109/CLOUD49709.2020.00075

[9] R. Bianchini, M. Fontoura, E. Cortez, A. Bonde, A. Muzio, A. Constantin, T. Moscibroda, G. Magalhães, G. Bablani, and M. Russinovich, "Toward ml-centric cloud platforms," *Commun. ACM*, vol. 63, no. 2, pp. 50–59, 2020. [Online]. Available: https://doi.org/10.1145/3364684

[10] L. Bindschaedler, A. Goel, and W. Zwaenepoel, "Hailstorm: Disaggregated compute and storage for distributed lsm-based databases," in *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 301–316. [Online]. Available: https://doi.org/10.1145/3373376.3378504

[11] E. Boutin, J. Ekanayake, W. Lin, B. Shi, J. Zhou, Z. Qian, M. Wu, and L. Zhou, "Apollo: Scalable and coordinated scheduling for cloud-scale computing," in *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014*, J. Flinn and H. Levy, Eds. USENIX Association, 2014, pp. 285–300. [Online]. Available: https://www.usenix.org/conference/osdi14/technical-sessions/presentation/boutin

[12] M. Breughe, S. Eyerman, and L. Eeckhout, "Mechanistic analytical modeling of superscalar in-order processor performance," *ACM Trans. Archit. Code Optim.*, vol. 11, no. 4, pp. 50:1–50:26, 2014. [Online]. Available: https://doi.org/10.1145/2678277

[13] B. Burns, B. Grant, D. Oppenheimer, E. A. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, 2016. [Online]. Available: https://doi.org/10.1145/2890784

[14] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, T. Sherwood, E. D. Berger, and C. Kozyrakis, Eds. ACM, 2021, pp. 79–92. [Online]. Available: https://doi.org/10.1145/3445814.3446713

[15] B. Caldwell, S. Goodarzy, S. Ha, R. Han, E. Keller, E. Rozner, and Y. Im, "Fluidmem: Full, flexible, and fast memory disaggregation for the cloud," in *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020*. IEEE, 2020, pp. 665–677. [Online]. Available: https://doi.org/10.1109/ICDCS47774.2020.00090

[16] W. Cao and L. Liu, "Hierarchical orchestration of disaggregated memory," *IEEE Trans. Computers*, vol. 69, no. 6, pp. 844–855, 2020. [Online]. Available: https://doi.org/10.1109/TC.2020.2968525

[17] J. Carlson, *Redis in action*. Simon and Schuster, 2013.

[18] Q. Chen, Z. Wang, J. Leng, C. Li, W. Zheng, and M. Guo, "Avalon: towards qos awareness and improved utilization through multi-resource management in datacenters," in *Proceedings of the ACM International Conference on Supercomputing, ICS 2019, Phoenix, AZ, USA, June 26-28, 2019*, R. Eigenmann, C. Ding, and S. A. McKee, Eds. ACM, 2019, pp. 272–283. [Online]. Available: https://doi.org/10.1145/3330345.3330370

[19] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 107–120. [Online]. Available: https://doi.org/10.1145/3297858.3304005

[20] S. Chen, S. GalOn, C. Delimitrou, S. Manne, and J. F. Martínez, "Workload characterization of interactive cloud services on big and small server platforms," in *2017 IEEE International Symposium on Workload Characterization, IISWC 2017, Seattle, WA, USA, October 1-3, 2017*. IEEE Computer Society, 2017, pp. 125–134. [Online]. Available: https://doi.org/10.1109/IISWC.2017.8167770

[21] I. Chung, B. Abali, and P. Crumley, "Towards a composable computer system," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2018, Chiyoda, Tokyo, Japan, January 28-31, 2018*. ACM, 2018, pp. 137–147. [Online]. Available: https://doi.org/10.1145/3149457.3149466

[22] O. Consortium, "OpenCAPI Specification," Online: http://opencapi.org, 2017, accessed: January 2019.

[23] C. Curino, S. Krishnan, K. Karanasos, S. Rao, G. M. Fumarola, B. Huang, K. Chaliparambil, A. Suresh, Y. Chen, S. Heddaya, R. Burd, S. Sakalanaga, C. Douglas, B. Ramsey, and R. Ramakrishnan, "Hydra: a federated resource manager for data-center scale analytics," in *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, J. R. Lorch and M. Yu,

Eds. USENIX Association, 2019, pp. 177–192. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/curino

[24] C. Delimitrou and C. Kozyrakis, "ibench: Quantifying interference for datacenter applications," in *Proceedings of the IEEE International Symposium on Workload Characterization, IISWC 2013, Portland, OR, USA, September 22-24, 2013*. IEEE Computer Society, 2013, pp. 23–33. [Online]. Available: https://doi.org/10.1109/IISWC.2013.6704667

[25] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2013, Houston, TX, USA, March 16-20, 2013*, V. Sarkar and R. Bodík, Eds. ACM, 2013, pp. 77–88. [Online]. Available: https://doi.org/10.1145/2451116.2451125

[26] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*, R. Balasubramonian, A. Davis, and S. V. Adve, Eds. ACM, 2014, pp. 127–144. [Online]. Available: https://doi.org/10.1145/2541940.2541941

[27] C. Delimitrou, D. Sánchez, and C. Kozyrakis, "Tarcil: reconciling scheduling speed and quality in large shared clusters," in *Proceedings of the Sixth ACM Symposium on Cloud Computing, SoCC 2015, Kohala Coast, Hawaii, USA, August 27-29, 2015*, S. Ghandeharizadeh, S. Barahmand, M. Balazinska, and M. J. Freedman, Eds. ACM, 2015, pp. 97–110. [Online]. Available: https://doi.org/10.1145/2806777.2806779

[28] B. Fitzpatrick, "Distributed caching with memcached," *Linux journal*, vol. 124, 2004.

[29] J. Fried, Z. Ruan, A. Ousterhout, and A. Belay, "Caladan: Mitigating interference at microsecond timescales," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 281–297. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/fried

[30] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 19–33. [Online]. Available: https://doi.org/10.1145/3297858.3304004

[31] I. Gog, M. Schwarzkopf, A. Gleave, R. N. M. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, K. Keeton and T. Roscoe, Eds. USENIX Association, 2016, pp. 99–115. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/gog

[32] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with infiniswap," in *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, A. Akella and J. Howell, Eds. USENIX Association, 2017, pp. 649–667. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[33] A. Guleria, J. Lakshmi, and C. Padala, "Quadd: Quantifying accelerator disaggregated datacenter efficiency," in *12th IEEE International Conference on Cloud Computing, CLOUD 2019, Milan, Italy, July 8-13, 2019*, E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds. IEEE, 2019, pp. 349–357. [Online]. Available: https://doi.org/10.1109/CLOUD.2019.00064

[34] J. Guo, Z. Chang, S. Wang, H. Ding, Y. Feng, L. Mao, and Y. Bao, "Who limits the resource efficiency of my datacenter: an analysis of alibaba datacenter traces," in *Proceedings of the International Symposium on Quality of Service, IWQoS 2019, Phoenix, AZ, USA, June 24-25, 2019*. ACM, 2019, pp. 39:1–39:10. [Online]. Available: https://doi.org/10.1145/3326285.3329074

[35] Z. Guo, Y. Shan, X. Luo, Y. Huang, and Y. Zhang, "Clio: a hardware-software co-designed disaggregated memory system," in *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, B. Falsafi, M. Ferdman, S. Lu, and T. F. Wenisch, Eds. ACM, 2022, pp. 417–433. [Online]. Available: https://doi.org/10.1145/3503222.3507762

[36] P. Hintjens, *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[38] K. Hsieh, S. M. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3d-stacked memory: Challenges, mechanisms, evaluation," in *34th IEEE International Conference on Computer Design, ICCD 2016, Scottsdale, AZ, USA, October 2-5, 2016*. IEEE Computer Society, 2016, pp. 25–32. [Online]. Available: https://doi.org/10.1109/ICCD.2016.7753257

[39] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *Workshops Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*. IEEE Computer Society, 2010, pp. 41–51. [Online]. Available: https://doi.org/10.1109/ICDEW.2010.5452747

[40] Y. Huang, X. Yan, G. Jiang, T. Jin, J. Cheng, A. Xu, Z. Liu, and S. Tu, "Tangram: Bridging immutable and mutable abstractions for distributed data analytics," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, D. Malkhi and D. Tsafrir, Eds. USENIX Association, 2019, pp. 191–206. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/huang

[41] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A black-box batch workload resource manager for improving utilization in cloud environments," in *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*. ACM, 2019, pp. 272–285. [Online]. Available: https://doi.org/10.1145/3357223.3362734

[42] S. S. K, J. Lakshmi, and N. Bisht, "Towards improving data center utilisation by reducing fragmentation," in *11th IEEE International Conference on Cloud Computing, CLOUD 2018, San Francisco, CA, USA, July 2-7, 2018*. IEEE Computer Society, 2018, pp. 941–945. [Online]. Available: https://doi.org/10.1109/CLOUD.2018.00141

[43] R. S. Kannan, M. Laurenzano, J. Ahn, J. Mars, and L. Tang, "Caliper: Interference estimator for multi-tenant environments sharing architectural resources," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 3, pp. 22:1–22:25, 2019. [Online]. Available: https://doi.org/10.1145/3323090

[44] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of the Fourteenth EuroSys Conference 2019, Dresden, Germany, March 25-28, 2019*, G. Candea, R. van Renesse, and C. Fetzer, Eds. ACM, 2019, pp. 34:1–34:16. [Online]. Available: https://doi.org/10.1145/3302424.3303958

[45] K. Katrinis, D. Syrivelis, D. N. Pnevmatikatos, G. Zervas, D. Theodoropoulos, I. Koutsopoulos, K. Hasharoni, D. Raho, C. Pinto, F. Espina, S. López-Buedo, Q. Chen, M. Nemirovsky, D. Roca, H. Klos, and T. Berends, "Rack-scale disaggregated cloud data centers: The dredbox project vision," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany, March 14-18, 2016*, L. Fanucci and J. Teich, Eds. IEEE, 2016, pp. 690–695. [Online]. Available: https://ieeexplore.ieee.org/document/7459397/

[46] X. Ke, C. Guo, S. Ji, S. Bergsma, Z. Hu, and L. Guo, "Fundy: A scalable and extensible resource manager for cloud resources," in *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, C. A. Ardagna, C. K. Chang, E. Daminai, R. Ranjan, Z. Wang, R. Ward, J. Zhang, and W. Zhang, Eds. IEEE, 2021, pp. 540–550. [Online]. Available: https://doi.org/10.1109/CLOUD53861.2021.00070

[47] J. Kim, W. Choe, and J. Ahn, "Exploring the design space of page management for multi-tiered memory systems," in *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, I. Calciu and G. Kuenning, Eds. USENIX Association, 2021, pp. 715–728. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/kim-jonghyeon

[48] A. Klimovic, H. Litz, and C. Kozyrakis, "Reflex: Remote flash ≈ local flash," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2017, Xi'an, China, April 8-12, 2017*, Y. Chen, O. Temam, and J. Carter, Eds. ACM, 2017, pp. 345–359. [Online]. Available: https://doi.org/10.1145/3037697.3037732

[49] M. Kogias, S. Mallon, and E. Bugnion, "Lancet: A self-correcting latency measuring tool," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA,*

July 10-12, 2019, D. Malkhi and D. Tsafrir, Eds. USENIX Association, 2019, pp. 881–896. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/kogias-lancet

[50] P. K. Koutsovasilis, M. Gazzetti, and C. Pinto, "A holistic system software integration of disaggregated memory for next-generation cloud infrastructures," in *21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021, Melbourne, Australia, May 10-13, 2021*, L. Lefèvre, S. Patterson, Y. C. Lee, H. Shen, S. Ilager, M. Goudarzi, A. N. Toosi, and R. Buyya, Eds. IEEE, 2021, pp. 576–585. [Online]. Available: https://doi.org/10.1109/CCGrid51090.2021.00067

[51] N. Kulkarni, G. Gonzalez-Pumariega, A. Khurana, C. A. Shoemaker, C. Delimitrou, and D. H. Albonesi, "Cuttlesys: Data-driven resource management for interactive services on reconfigurable multicores," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 2020, pp. 650–664. [Online]. Available: https://doi.org/10.1109/MICRO50266.2020.00060

[52] H. A. Lagar-Cavilla, J. Ahn, S. Souhlal, N. Agarwal, R. Burny, S. Butt, J. Chang, A. Chaugule, N. Deng, J. Shahid, G. Thelen, K. A. Yurtsever, Y. Zhao, and P. Ranganathan, "Software-defined far memory in warehouse-scale computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 317–330. [Online]. Available: https://doi.org/10.1145/3297858.3304053

[53] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown, A. Klimovic, A. Slowey, and A. I. T. Rowstron, "Understanding rack-scale disaggregated storage," in *9th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2017, Santa Clara, CA, USA, July 10-11, 2017*. USENIX Association, 2017. [Online]. Available: https://www.usenix.org/conference/hotstorage17/program/presentation/legtchenko

[54] H. Li, D. S. Berger, S. Novakovic, L. Hsu, D. Ernst, P. Zardoshti, M. Shah, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "First-generation memory disaggregation for cloud platforms," *CoRR*, vol. abs/2203.00241, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.00241

[55] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management," in *2017 IEEE International Conference on Cluster Computing, CLUSTER 2017, Honolulu, HI, USA, September 5-8, 2017*. IEEE Computer Society, 2017, pp. 152–165. [Online]. Available: https://doi.org/10.1109/CLUSTER.2017.130

[56] K. T. Lim, Y. Turner, J. R. Santos, A. AuYoung, J. Chang, P. Ranganathan, and T. F. Wenisch, "System-level implications of disaggregated memory," in *18th IEEE International Symposium on High Performance Computer Architecture, HPCA 2012, New Orleans, LA, USA, 25-29 February, 2012*. IEEE Computer Society, 2012, pp. 189–200. [Online]. Available: https://doi.org/10.1109/HPCA.2012.6168955

[57] L. Liu, W. Cao, S. Sahin, Q. Zhang, J. Bae, and Y. Wu, "Memory disaggregation: Research problems and opportunities," in *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 1664–1673. [Online]. Available: https://doi.org/10.1109/ICDCS.2019.00165

[58] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture, Portland, OR, USA, June 13-17, 2015*, D. T. Marr and D. H. Albonesi, Eds. ACM, 2015, pp. 450–462. [Online]. Available: https://doi.org/10.1145/2749469.2749475

[59] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Improving resource efficiency at scale with heracles," *ACM Trans. Comput. Syst.*, vol. 34, no. 2, pp. 6:1–6:33, 2016. [Online]. Available: https://doi.org/10.1145/2882783

[60] MarketsandMarkets, "Cloud computing market by service, deployment model, organization size, vertical and region - global forecast to 2026," 2021.

[61] J. Mars and L. Tang, "Whare-map: heterogeneity in "homogeneous" warehouse-scale computers," in *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, A. Mendelson, Ed. ACM, 2013, pp. 619–630. [Online]. Available: https://doi.org/10.1145/2485922.2485975

[62] H. A. Maruf and M. Chowdhury, "Effectively prefetching remote memory with leap," in *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, A. Gavrilovska and E. Zadok, Eds. USENIX Association, 2020, pp. 843–857. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/al-maruf

[63] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "Tpp: Transparent page placement for cxl-enabled tiered memory," 2022. [Online]. Available: https://arxiv.org/abs/2206.02878

[64] H. A. Maruf, Y. Zhong, H. Wang, M. Chowdhury, A. Cidon, and C. A. Waldspurger, "Memtrade: A disaggregated-memory marketplace for public clouds," *CoRR*, vol. abs/2108.06893, 2021. [Online]. Available: https://arxiv.org/abs/2108.06893

[65] D. Masouros, S. Xydis, and D. Soudris, "Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 1, pp. 184–198, 2021. [Online]. Available: https://doi.org/10.1109/TPDS.2020.3013948

[66] M. Nanavati, J. Wires, and A. Warfield, "Decibel: Isolation and sharing in disaggregated rack-scale storage," in *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, A. Akella and J. Howell, Eds. USENIX Association, 2017, pp. 17–33. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/nanavati

[67] R. Nishtala, V. Petrucci, P. M. Carpenter, and M. Själander, "Twig: Multi-agent task management for colocated latency-critical cloud services," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 167–179. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00023

[68] V. Nitu, B. Teabe, A. Tchana, C. Isci, and D. Hagimont, "Welcome to zombieland: practical and energy-efficient memory disaggregation in a datacenter," in *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, R. Oliveira, P. Felber, and Y. C. Hu, Eds. ACM, 2018, pp. 16:1–16:12. [Online]. Available: https://doi.org/10.1145/3190508.3190537

[69] N. Nordlund, V. Vassiliadis, M. Gazzetti, D. Syrivelis, and L. Tassiulas, "Energy-aware learning agent (EALA) for disaggregated cloud scheduling," in *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, C. A. Ardagna, C. K. Chang, E. Daminai, R. Ranjan, Z. Wang, R. Ward, J. Zhang, and W. Zhang, Eds. IEEE, 2021, pp. 578–583. [Online]. Available: https://doi.org/10.1109/CLOUD53861.2021.00075

[70] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abs

[71] T. Patel and D. Tiwari, "CLITE: efficient and qos-aware co-location of multiple latency-critical jobs for warehouse scale computers," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 193–206. [Online]. Available: https://doi.org/10.1109/HPCA47549.2020.00025

[72] C. Pinto, D. Syrivelis, M. Gazzetti, P. K. Koutsovasilis, A. Reale, K. Katrinis, and H. P. Hofstee, "Thymesisflow: A software-defined, HW/SW co-designed interconnect stack for rack-scale memory disaggregation," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2020, Athens, Greece, October 17-21, 2020*. IEEE, 2020, pp. 868–880. [Online]. Available: https://doi.org/10.1109/MICRO50266.2020.00075

[73] C. Reaño, F. Silla, G. Shainer, and S. Schultz, "Local and remote gpus perform similar with EDR 100g infiniband," in *Proceedings of the Industrial Track of the 16th International Middleware Conference, Middleware Industry 2015, Vancouver, BC, Canada, December 7-11, 2015*, K. R. Jayaram and M. A. Kozuch, Eds. ACM, 2015, pp. 4:1–4:7. [Online]. Available: https://doi.org/10.1145/2830013.2830015

[74] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *ACM Symposium on Cloud Computing, SOCC '12, San Jose,*

CA, USA, October 14-17, 2012, M. J. Carey and S. Hand, Eds. ACM, 2012, p. 7. [Online]. Available: https://doi.org/10.1145/2391229.2391236

[75] A. Roozbeh, "Realizing next-generation data centers via software-defined "hardware" infrastructures and resource disaggregation: Exploiting your cache," Ph.D. dissertation, KTH Royal Institute of Technology, 2021.

[76] A. Roozbeh, J. M. Soares, G. Q. M. Jr., F. Wuhib, C. Padala, M. Mahloo, D. Turull, V. Yadhav, and D. Kostic, "Software-defined "hardware" infrastructures: A survey on enabling technologies and open research directions," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 3, pp. 2454–2485, 2018. [Online]. Available: https://doi.org/10.1109/COMST.2018.2834731

[77] Z. Ruan, M. Schwarzkopf, M. K. Aguilera, and A. Belay, "AIFM: high-performance, application-integrated far memory," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 315–332. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/ruan

[78] S. K. Sadasivam, B. W. Thompto, R. N. Kalla, and W. J. Starke, "IBM power9 processor architecture," *IEEE Micro*, vol. 37, no. 2, pp. 40–51, 2017. [Online]. Available: https://doi.org/10.1109/MM.2017.40

[79] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: A cautionary tale," in *3rd Symposium on Networked Systems Design and Implementation (NSDI 2006), May 8-10, 2007, San Jose, California, USA, Proceedings*, L. L. Peterson and T. Roscoe, Eds. USENIX, 2006. [Online]. Available: http://www.usenix.org/events/nsdi06/tech/schroeder.html

[80] Y. Sfakianakis, M. Marazakis, and A. Bilas, "Skynet: Performance-driven resource management for dynamic workloads," in *14th IEEE International Conference on Cloud Computing, CLOUD 2021, Chicago, IL, USA, September 5-10, 2021*, C. A. Ardagna, C. K. Chang, E. Daminai, R. Ranjan, Z. Wang, R. Ward, J. Zhang, and W. Zhang, Eds. IEEE, 2021, pp. 527–539. [Online]. Available: https://doi.org/10.1109/CLOUD53861.2021.00069

[81] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang, "Legoos: A disseminated, distributed OS for hardware resource disaggregation," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, D. Malkhi and D. Tsafrir, Eds. USENIX Association, 2019. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/shan

[82] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "ROSE: cluster resource scheduling via speculative over-subscription," in *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*. IEEE Computer Society, 2018, pp. 949–960. [Online]. Available: https://doi.org/10.1109/ICDCS.2018.00096

[83] C. Tang, K. Yu, K. Veeraraghavan, J. Kaldor, S. Michelson, T. Kooburat, A. Anbudurai, M. Clark, K. Gogia, L. Cheng, B. Christensen, A. Gartrell, M. Khutornenko, S. Kulkarni, M. Pawlowski, T. Pelkonen, A. Rodrigues, R. Tibrewal, V. Venkatesan, and P. Zhang, "Twine: A unified cluster management system for shared infrastructure," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 787–803. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/tang

[84] S. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores," in *2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, A. Gavrilovska and E. Zadok, Eds. USENIX Association, 2020, pp. 33–48. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/tsai

[85] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015*, L. Réveillère, T. Harris, and M. Herlihy, Eds. ACM, 2015, pp. 18:1–18:17. [Online]. Available: https://doi.org/10.1145/2741948.2741964

[86] C. Wang, H. Ma, S. Liu, Y. Li, Z. Ruan, K. Nguyen, M. D. Bond, R. Netravali, M. Kim, and G. H. Xu, "Semeru: A memory-disaggregated managed runtime," in *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 2020, pp. 261–280. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/wang

[87] J. Weiner, N. Agarwal, D. Schatzberg, L. Yang, H. Wang, B. Sanouillet, B. Sharma, T. Heo, M. Jain, C. Tang, and D. Skarlatos, "TMO: transparent memory offloading in datacenters," in *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, B. Falsafi, M. Ferdman, S. Lu, and T. F. Wenisch, Eds. ACM, 2022, pp. 609–621. [Online]. Available: https://doi.org/10.1145/3503222.3507731

[88] N. Wu and Y. Xie, "A survey of machine learning for computer architecture and systems," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 54:1–54:39, 2023. [Online]. Available: https://doi.org/10.1145/3494523

[89] R. Xu, S. Mitra, J. Rahman, P. Bai, B. Zhou, G. Bronevetsky, and S. Bagchi, "Pythia: Improving datacenter utilization via precise contention prediction for multiple co-located workloads," in *Proceedings of the 19th International Middleware Conference, Middleware 2018, Rennes, France, December 10-14, 2018*, P. Ferreira and L. Shrira, Eds. ACM, 2018, pp. 146–160. [Online]. Available: https://doi.org/10.1145/3274808.3274820

[90] Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Characterizing facebook's memcached workload," *IEEE Internet Comput.*, vol. 18, no. 2, pp. 41–49, 2014. [Online]. Available: https://doi.org/10.1109/MIC.2013.80

[91] Z. Yan, D. Lustig, D. W. Nellans, and A. Bhattacharjee, "Nimble page management for tiered memory systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, I. Bahar, M. Herlihy, E. Witchel, and A. R. Lebeck, Eds. ACM, 2019, pp. 331–345. [Online]. Available: https://doi.org/10.1145/3297858.3304024

[92] Z. Yu, Z. Bei, and X. Qian, "Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, X. Shen, J. Tuck, R. Bianchini, and V. Sarkar, Eds. ACM, 2018, pp. 564–577. [Online]. Available: https://doi.org/10.1145/3173162.3173187

[93] F. V. Zacarias, R. Nishtala, and P. M. Carpenter, "Contention-aware application performance prediction for disaggregated memory systems," in *Proceedings of the 17th ACM International Conference on Computing Frontiers, CF 2020, Catania, Sicily, Italy, May 11-13, 2020*, M. Palesi, G. Palermo, C. Graves, and E. Arima, Eds. ACM, 2020, pp. 49–59. [Online]. Available: https://doi.org/10.1145/3387902.3392625

[94] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: a unified engine for big data processing," *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016. [Online]. Available: http://doi.acm.org/10.1145/2934664

[95] B. Zhang, Y. J. Ong, and T. Nakamura, "Simpo: Simultaneous prediction and optimization," in *IEEE International Conference on Services Computing, SCC 2022, Barcelona, Spain, July 10-16, 2022*, C. A. Ardagna, H. Bian, C. K. Chang, R. N. Chang, E. Damiani, S. Dustdar, J. Marco, M. P. Singh, E. Teniente, R. Ward, Z. Wang, F. Xhafa, and J. Zhang, Eds. IEEE, 2022, pp. 120–122. [Online]. Available: https://doi.org/10.1109/SCC55611.2022.00028

[96] Y. Zhang, W. Hua, Z. Zhou, G. E. Suh, and C. Delimitrou, "Sinan: Ml-based and qos-aware resource management for cloud microservices," in *ASPLOS '21: 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Virtual Event, USA, April 19-23, 2021*, T. Sherwood, E. D. Berger, and C. Kozyrakis, Eds. ACM, 2021, pp. 167–181. [Online]. Available: https://doi.org/10.1145/3445814.3446693

[97] Y. Zhang, D. Meisner, J. Mars, and L. Tang, "Treadmill: Attributing the source of tail latency through precise load testing and statistical inference," in *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*. IEEE Computer Society, 2016, pp. 456–468. [Online]. Available: https://doi.org/10.1109/ISCA.2016.47