On the Implications of Heterogeneous Memory Tiering on Spark In-Memory Analytics

Manolis Katsaragakis[⋆†], Dimosthenis Masouros[⋆], Lazaros Papadopoulos[⋆], Francky Catthoor[†α], Dimitrios Soudris[⋆]
[⋆]*Microprocessors and Digital Systems Laboratory, ECE , National Technical University of Athens, Greece*
[†]*Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium*, [α]*IMEC-BE, Leuven, Belgium*
[⋆]{mkatsaragakis, dmasouros, lpapadop, dsoudris}@microlab.ntua.gr
[α]francky.catthoor@esat.kuleuven.be

***Abstract*—Today, the rise of big data has driven a growing demand for efficient and scalable computing solutions that can handle the massive amounts of data generated by modern applications. To address this challenge, application developers have embraced the use of novel distributed frameworks, such as Apache Spark, which enable efficient, in-memory processing for large amounts of data. Moreover, providers are seeking for new alternatives to cope with this increasing need for "infinite" memory resources, that can provide analogous performance efficiency, while also reducing operational costs. In this direction, novel multi-tier and disaggregated memory architectures emerge, which combine heterogeneous memory technologies that trade-off between performance, cost and energy efficiency. This diptych of evolution imposes new challenges and open questions on how to optimally configure software and hardware as a whole, for maximizing resource efficiency.**

In this paper, we examine the implications of heterogeneous memory tiering on Spark in-memory analytics. Our study considers a multi-tier heterogeneous DRAM/NVM memory system with contrasting access latency, bandwidth and energy consumption capabilities. By using a set of 7 diverse applications from the HiBench benchmarking suite, we first explore the impact of these memory configuration setups on their performance and energy efficiency. Then, we perform a detailed analysis on how the system's low-level performance metrics correlate to higher level metrics of interest (i.e., performance/energy), which aims to provide deeper insights w.r.t. the relationship between software and hardware events. Driven by the obtained results, we identify a set of guidelines derived by deploying Spark in-memory analytics over heterogeneous memory tiered systems.

***Index Terms*—Disaggregated computing, in-memory analytics, Multi-tier architecture, DRAM, Intel Optane DC**

I. Introduction

Over the last years, the growth of applications that handle large datasets is rapidly increasing and becoming more and more complex. Machine Learning(ML) handling large datasets [1], data produced on the Edge that need to be processed and stored [2], as well as big-data and analytical processing applications [3] are just few examples of applications that generate enormous workloads. Such kind of applications impose increased requirements in terms of computational and memory requirements, leading to I/O bottlenecks, performance degradation and non-sustainable behavior.

This work has been partially funded by EU Horizon program NEPHELE under grant agreement No 101070487 (https://nephele-project.eu/).

From the application developers' perspective, the perpetual rise of data produced has led to the adoption of novel frameworks that enable the parallel processing of huge data volumes in a distributed manner [4]–[6]. Apache Spark [6] is one of the most widely used processing engines for large scale data analytics, offering a new way of computing with its resilient distributed dataset (RDD), which is stored in memory, while being computed on the latter, thus, avoiding expensive intermediate disk writes found in prior big data frameworks, such as Hadoop [5].

On the other hand, from the providers' standpoint, the increasing memory requirements of modern applications along with the rapid adoption of in-memory analytics frameworks have highlighted the need for novel system architectures, which are able to bypass the fixed resource proportionality of conventional servers, while maintaining performance and maximizing energy and cost efficiency. To this end, multi-tier [7]–[9] and disaggregated [10]–[12] memory architectures have been proposed as new design paradigms, where different memory technologies can be leveraged and/or combined according to the needs of running applications. Such architectures can include conventional DRAM technology and Non Volatile Memory(NVM) technologies, such as 3D-XPoint, PCM, 3D DRAM [13] and FeFET [14], which provide trade-offs between access latency, bandwidth, energy efficiency, data persistence and other [15]. In fact, such kind of solutions have been already deployed in various commercial platforms, such as the SAP HANA [16] database and the Aurora exascale supercomputer, which employs the DAOS architecture for storage [17], both taking advantage of the only commercial NVM technology up to now, i.e., Intel Optane DC Persistent Memory(DCPM). Moreover, upcoming technologies, such as Samsung Memory Expander [18] and Compute Express Link (CXL) [19] aim to further bridge existing performance gaps and exploit the limits of multi-tier memory disaggregated platforms, however with the cost of more complex hierarchies.

This simultaneous evolution of the software and hardware "worlds" imposes new challenges on how to efficiently manage the deployment of such applications on heterogeneous memory systems. The co-existence of heterogeneous multi-tier memory architectures with the in-memory application alternatives composes an extended set of exploration for efficient deploy-

ment in terms of performance, energy consumption and other aspects. Thus, an efficient exploration of the alternative configurations is required, in order to efficiently deploy application workloads over multi-tier hybrid memory systems [20]. In-depth analysis for the sufficient memory tier and technology selection is crucial, while the Spark internal configuration is also dominant for the efficacy of the application. The efficient co-selection of hardware and software parameters can fully exploit performance potential, minimize overhead and provide trade-off among hardware and software metrics. However, such kind of selection and efficient correlation among these characteristics is not staightforward.

In this work, we present an extensive exploration and characterization of various in-memory Spark applications and workloads over heterogeneous multi-tier memory systems. We present an extended analysis in terms of performance, energy consumption and scalability for in-memory Spark analytics derived from different application domains, i.e, micro-operations, machine learning and websearch. Through our experimental analysis: i) we indicate key aspects that are crucial for the overall performance/energy of the execution, ii) we provide a set of guidelines and key takeaways for efficient deployment over and iii) we indicate current limitations of recent multi-tier heterogeneous memory systems.

II. Related Work

Several works have been conducted aiming to evaluate alternative Spark workloads over disaggregated memory systems. Authors of [21] and [22] focus on the characterization and exploration of database-related applications over Apache Spark, while authors of [23] present an analysis of batch and stream processing workloads from micro-architectural perspective. Furthermore, research conducted in [24] aims to investigate similarities and patterns among known Spark workloads, while authors of [25] investigate the design of distributed file system over multi-tier storage. Authors of [26] analyze the performance of Spark applications and propose a performance autotuning framework.

Moreover, aiming to combine heterogeneous memories over Spark analytics, authors of [27] integrate persistent memory technologies as an low-cost alternative to extend capacity of existing computer clusters. Additionally, in [28] the potential benefits of Optane DC Persistent Memory (DCPM) for neuroimaging data processing and storage are investigated. Furthermore, in [29] authors present and analysis for in-memory analytical database workloads over Optane DCPM, while the authors of [30] investigate the impact of data and object placement on heterogeneous memory configurations over various in-memory applications. Moreover, in [31] an exploration of the key-value stores is investigated over persistent disaggregated memory systems.

Although research has illuminated the potential impact of in-memory analytics and the integration of various memory systems, no study to date, to the best of our knowledge, has provided an in-depth characterization and guidelines for Apache Spark in-memory analytics across multi-tier heterogeneous memory systems. In contrast to existing literature, we provide a thorough investigation on the underlying factors and aspects that drastically affect the overall behavior of Spark applications over multi-tier heterogeneous memory systems.

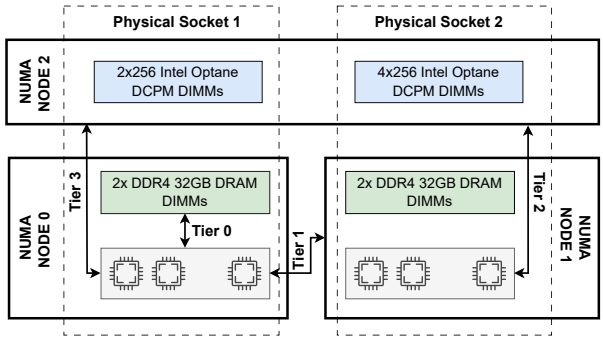


Fig. 1: Overview of the target multi-tier system architecture and testbed

III. Hardware & Software Testbed

A. Multi-tier Disaggregated Memory System

Since the integration and implementation of disaggregated memory systems decoupled from the processor in not trivial, due to absence or existence of complex prototypes [10], [32], we emulate our disaggregated memory system over a multi-socket server with heterogeneous DRAM/NVM memory system. Specifically, our experiments were conducted on a single node with a 2x20 core Intel Xeon Gold 5218R CPU @2.10GHz with 4x32GB DDR4 DIMMs and 6×256GB Optane DC NVDIMMs. Aiming to provide alternative access latency and bandwidth across nodes, we provide asymmetry between Optane DIMMs over the physical sockets, i.e., we deploy 2 DIMMs on socket 1 and 4 DIMMs on socket 2. Last, we configure the Intel Optane DC memory to *App Direct* mode with ext4-DAX file system, which allows the PMEM capacity to be used as byte-addressable persistent memory that is mapped into the system's physical address space (SPA) and directly accessible by applications.

Based on the above hardware configuration, our target disaggregated multi-tier architecture is illustrated in Figure 1. From an operating system's perspective, the available memory is organized as three distinct, asymmetric NUMA nodes, where NUMA 0 and NUMA 1 are symmetric to each other and include the DRAM memory capacity of socket 0 and socket 1 respectively, whereas NUMA 2 is assymetric and includes the memory capacity of the NVM memory. We define four memory access scenarios (henceforth referred to as different Tiers), that emulate different local (intra-NUMA) and remote (inter-NUMA) allocation modes, as shown in Figure 1. Specifically, Tier 0 forms a local allocation mode, where memory is obtained directly from the same physical socket as the cores, whereas Tiers 1-3 are considered as remote modes with heterogeneous memory types each. Traditionally, remote memory accesses reveal higher latency and lower bandwidth per operation [33], due to the overhead of network and data

exchange, which is implicitly entailed through our physical hardware configuration described previously. Moreover, the imbalance of NVM DIMMs on each socket allows us to exploit different latency and bandwidth. The idle latency and bandwidth characteristics of the local and remote memory access for our experimental setup are shown in Table I.

B. Spark Engine Configuration

We configure the Spark engine to run in a pseudo-distributed, standalone mode, where both Spark Master and Spark Executors reside on the same physical machine. Each computing unit acts independently and is bound with a set of CPU resources either on NUMA node 0 or NUMA node 1 and memory resources from all the available NUMA regions. To achieve this, we force Spark executors to be deployed and access specific compute/memory tiers, by specifying the `cpunodebind` and `membind` flags of Linux's `numactl` tool. By default, in standalone mode, Spark deploys one executor instance that utilizes all the available cores of the bound node (40 hyperthreads in our case). Last, we use Hadoop's Distributed File System (HDFS) [5] instead of the local file system to store input and output data of Spark. Both HDFS and Spark are configured with their default parameters.

C. Examined Spark applications

We utilize a subset of benchmarks derived from HiBench suite [34]. We study 7 Spark applications derived from the HiBench benchmark suite [34], which is widely used to evaluate Spark applications, listed in Table II. These applications form representative examples out of three different workload categories, i.e., micro-operations, machine learning and web searching. Moreover, we examine a diverse set of input dataset sizes per application, i.e. tiny, small and large.

IV. Characterization & Analysis

In this section, we provide an extensive profiling and characterization of our examined examined Spark applications over the different memory configurations considered, as described in Sec. III. Following a Q&A approach, we debate over different aspects that form interesting study factors and, based on our experimental analyses, we provide a set of valuable insights and outcomes to consider when deploying Spark analytics over heterogeneous memory configurations.

A. How do the different memory tiers affect the performance of our examined applications?

First, we examine how allocating memory from different tiers affects the execution time of applications, by using the default Spark configuration (1 executor - 40 cores). Figure 2 (top) illustrates the execution time of all the alternative benchmarks, where the Y axis indicates the execution time and X axis the corresponding input dataset size, respectively. We observe that for the majority of the workloads, local execution provides optimized performance compared to remote execution, due to the inherent inferior performance of the remote memory Tiers compared to the local one. Overall, the local execution (Tier 0) achieves 44.2%, 66.4% and 90.1% better execution time on average, compared to Tier 1, Tier 2 and 3 remote execution, respectively. However, there exist certain cases where allocating memory from local and remote nodes lead to similar execution time (e.g., repartition-tiny, pagerank-tiny, pagerank-small), which reveals the potential of certain applications for exploiting remote memory, without sacrificing performance. Moreover, while most of the applications exhibit linear or superlinear execution time increment across the different dataset sizes, `als` shows an almost constant execution time regardless of the input workload and the Tier considered.

⋆ ***Takeaway 1:*** *Performance degradation due to remote memory highly depends on the nature of each application and its workload size, with certain combinations revealing tolerance to the inherent inferior efficiency of remote Tiers.*

Next, we also investigate the impact of each memory technology, i.e., DRAM and Intel Optane DCPM, across the different tiers. Executions bound with DRAM technology (Tiers 1 & 2) exhibit almost similar performance, regardless of the latency and bandwidth overhead due to remote memory fetching. On the other hand, executions that are bound with Intel Optane DCPM DIMMs (Tiers 3 & 4) require 76.7% more execution time, compared to execution bound with DRAM DIMMs, because accessing Optane DCPM requires higher access latency and provides limited bandwidth compared to the latter. Overall, remote memory accesses across different nodes impose an extra communication overhead due to the physical distance of the nodes. Thus, an extra latency penalty is added to the already high access latency of persistent memory,

TABLE I: Idle access latency and memory bandwidth per tier

		Idle Latency (ns)	Bandwidth (GB/s)
Tier	**Tier 0**	77.8	39.3
	Tier 1	130.9	31.6
	Tier 2	172.1	10.7
	Tier 3	231.3	0.47

TABLE II: Examined Spark applications and dataset sizes

Application	Abbr.	Data size range (tiny,small,large)
Sorting of text input data	sort	32KB, 320MB, 3.2GB
Performs shuffle operations	repartition	3.2KB, 3.2MB, 32MB
Alternating Least Squares	als	100, 1.000, 10.000 (users) 100, 1.000, 10.000 (products) 200, 2.000, 20.000 (ratings)
Naive Bayes classification	bayes	25.000, 30.000, 100.000 (pages) 10, 100, 100 (classes)
Random forest	rf	10, 100, 1.000 (examples) 100, 500, 1.000 (features)
Latent Dirichlet Allocation	lda	2.000, 5.000, 10.000 (docs) 1.000, 2.000, 3.000 (vocabulary) 10, 20, 30 (topics)
PageRank	pagerank	50, 5.000, 500.000 (pages)

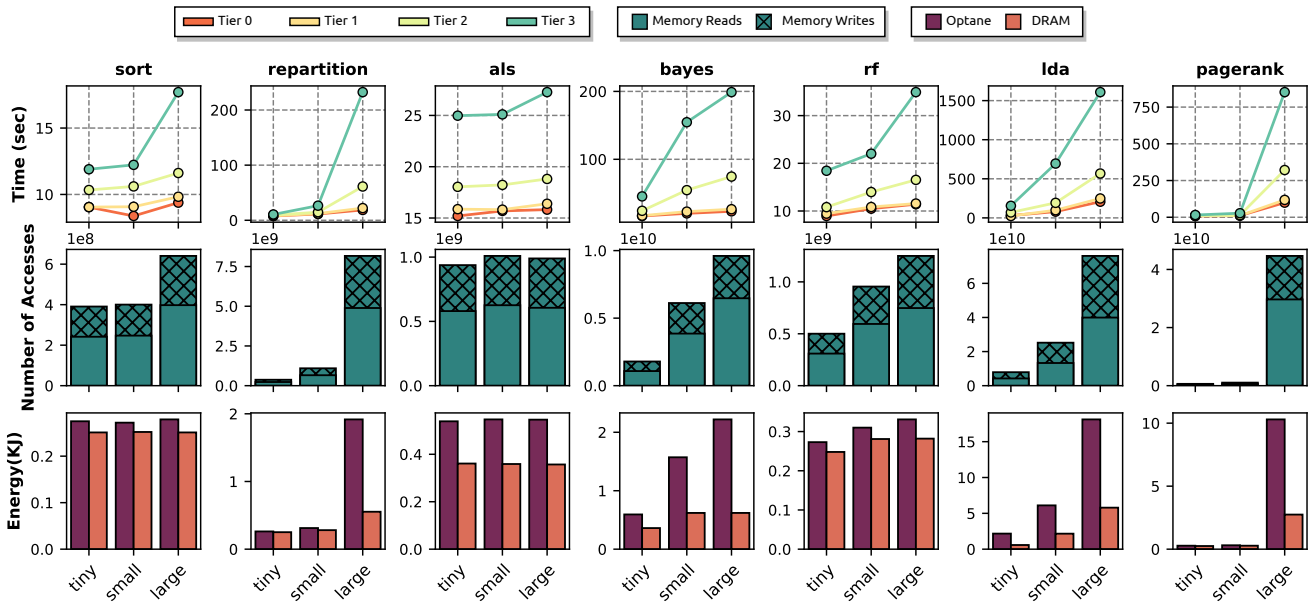


Fig. 2: Execution time for all the different memory tiers (top), number of accesses (reads and writes) to memory (middle) and DRAM/NVM energy comparison (bottom) for all the different dataset sizes and benchmarks examined.

leading to further performance degradation. Applications with higher execution requirements, such as `repartition`, `bayes`, `lda` and `pagerank` are more sensitive to performance degradation, where up to 96.7% more execution time is observed on average for all workloads when deployed on Tier 2 and as the input workload increases, in contrast to `sort`, `als` and `rf`, where we observe 31.1% degradation on average respectively.

⋆ ***Takeaway 2:*** *The combination of remote memory access penalty along with the inherent latency/bandwidth inefficiency of Optane DCPM compared to DRAM, leads to a disproportional increment on the performance gap between the two technologies as the time of execution increases.*

B. What are the core bottleneck factors for the observed performance degradation?

NVDIMMs are slower compared to DRAM, thus we focus on the insights of the degradation of applications bound with Intel Optane DC. To this end, we monitor the number of read and write memory accesses over the NVDIMMs by utilizing Intel's `ipmctl` tool. Figure 2 (middle row) illustrates the respective results. Similar to execution time experiments, as the number of read/write accesses increases, the performance drops significantly. In applications such as `bayes`, `lda` and `pagerank`, where the number of total accesses is an order of magnitude higher compared to the other benchmarks, we notice severe degradation both as the input workload increases and as the application is bound to more distant tiers.

Moreover, in cases where the ratio of write to read accesses increases, we observe a non-linear degradation on the performance of applications. This is due to the asymmetry of read and write operations on the Intel Optane DCPM in terms of access latency, which is known to perform worse when write accesses increase [29]. This is clearly observed for the `lda` benchmark under the `large` workload, whose execution time skyrockets proportionally to the number of write operations performed in the Intel Optane DCPM. Similar observations can be derived for the other benchmarks. Last but not least, apart from the impact of the high number of accesses on the target application, increased number of write operations reduces the lifetime of persistent memory [35], thus in the long-term further performance degradation may occur due to potential hardware failures.

⋆ ***Takeaway 3:*** *Application's performance is highly affected by the number of read and write operations on the persistent memory, with the latter having even more impact by design.*

C. Does bandwidth or latency dominate performance?

In addition to the undeniable overhead added by increased accesses on the NVDIMMs, we aim to investigate further overheads regarding the utilization of persistent memory as a remote tier. For this scope, we take advantage of Intel's Memory Bandwidth Allocation(MBA) tool [36] and we limit the maximum possible bandwidth utilization to different thresholds. Our experiments were conducted for 1 single executor with 40 cores. Violin plots in Figure 3 illustrate the

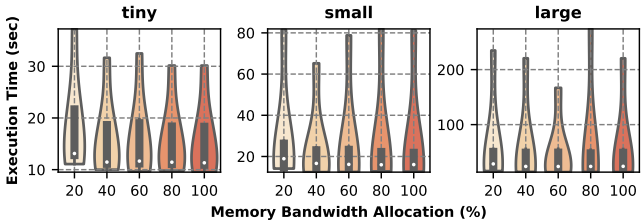
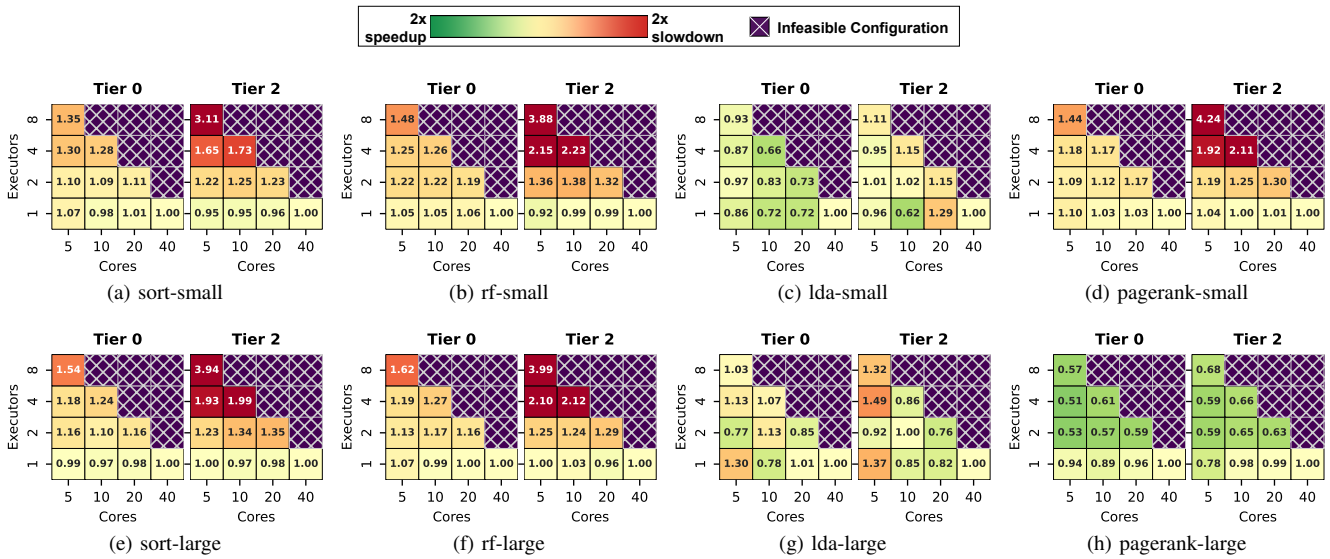


Fig. 3: Execution time over various memory bandwidth levels for all benchmarks and tiny,small and large input workloads



(a) sort-small (b) rf-small (c) lda-small (d) pagerank-small

(e) sort-large (f) rf-large (g) lda-large (h) pagerank-large

Fig. 4: Speedup/Slowdown of `sort`, `rf`, `lda` and `pagerank` applications for all workloads for varying number of cores and executors. The default configuration is 1 executor with 40 cores.

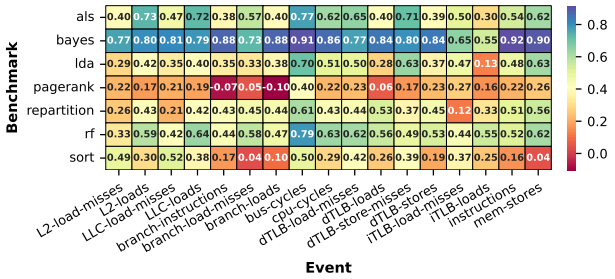


Fig. 5: Correlation of system-level metrics with execution time

impact of alternative maximum memory bandwidth allocation percentages(X-axis) on the execution time of the target applications(Y-axis) for all input workloads, on average. Both the average execution time and the variance of the distributions does not change as the bandwidth allocation percentage varies for all the input workloads, respectively. Thus, our target applications do not saturate the memory bandwidth, thus it is not a bottleneck for the application's performance. We conclude that the increased access latency to NVDIMMs and the overhead of remote access overhead is the most dominant factor for performance degradation.

⋆ ***Takeaway 4:*** *Performance is highly affected by the access latency of memory technologies, while memory bandwidth is not saturated, thus memory access latency and remote communication are the major bottlenecks.*

D. Despite its performance degradation, is persistent remote memory energy efficient?

Figure 2 (bottom) illustrates a comparison of the average energy consumption per DRAM(Tier 0) and Intel Optane DCPM DIMM(Tier 2), over the alternative benchmarks and workloads, respectively. Even though NVMs provide less power consumption per access compared to traditional DRAM technologies, we observe that Optane DIMMs consume higher energy consumption in total compared to the DRAM DIMMs. More specifically, DRAM execution provides 63.9% less energy consumption on average compared to Intel Optane DCPM execution. This is due to the fact that applications deployed on the NVDIMMs are utilized for higher amount of time, thus leading to increased accumulated energy consumption. Furthermore, we observe that the energy consumption, both on the DRAM and Intel Optane DCPM DIMMs is inline to the execution time scaling, as the input workload size increases. However, there exist workloads, such as `sort` and `als` can scale to larger workloads without significant energy overhead. Thus they can be considered as candidates for remote deployment without significant energy penalty.

⋆ **Takeaway 5:** *Energy consumption is inline with the execution time, however energy over-consumption can be avoided in some cases.*

E. How does software tunable Spark configuration parameters relate to and affect performance?

Next, we investigate how alternating the execution behavior of the Spark engine affects the performance of applications, by examining the performance for different number of executors and cores per executor. With this analysis, we revisit the long-standing debate between "fat" and "skinny" executors for disaggregated memory architectures, which have either isolated access to the memory or compete for shared resources (e.g., executors requesting memory on a rack-scale memory disaggregated system [11]). Figure 4 illustrates the relative speedup/slowdown for a subset of the representative benchmark under small and large workloads. X-axis indicates the number of cores per NUMA node and Y-axis shows the number of corresponding executors. As baseline, we consider a single executor over 40 cores (bottom-right square). We observe that `sort`(Fig. 4a), `rf` (Fig. 4b) and `pagerank`

(Fig. 4d) benchmarks provide significant slowdown when they are bound to the NVM memory tier by getting down to 3.11× slowdown. The increase of allocated CPU cores per executor does not necessarily provide performance optimization, due to contention in the shared resources of the system and especially the memory bus. Moreover, increased number of executors leads to extra number of accesses on the persistent memory DIMMs for executors co-operation, thus leading to performance degradation. In contrast to the examined benchmarks, the `lda` (Fig. 4c) benchmark is not significantly affected as the number of cores/executors ranges.

⋆ **Takeaway 6:** *Increased number of executors that compete over shared memory resources leads to further performance degradation, with persistent memory being even more susceptible to resource contention.*

Moreover, we investigate how alternative combinations of cores and executors behave over larger amounts of workload. Figures 4e- 4h illustrate the corresponding speedup/slowdown for the examined applications under the large workload. For the `sort`, `rf` and `lda` the qualitative comparison under small and large workloads is quite similar. However, regarding the `pagerank`(Fig. 4h) benchmark, speedup is observed as the number of executors increases, in contrast to small workload(Fig. 4d), where significant slowdown is observed as the number of executors rises. This is due to the fact that for the large workload, efficient data partitioning across the executors and sufficient resource utilization is occurred and executors are not underutilized. As the input dataset size increases, the benefits of parallel processing and distribution across multiple executors become more pronounced, leading to a speedup in the overall processing time. Similar observations can be derived for the other benchmarks.

⋆ **Takeaway 7:** *Workload size affects the performance, however there exist benchmarks who handle better high workloads.*

F. Can we somehow obtain a rough estimation of performance degradation on remote memory tiers?

Last, we investigate the potential of exploiting different metrics for estimating the performance of applications across the several tiers. Specifically, we examine two directions *i)* how system-level events correlate with execution time when deployed on local memory and *ii)* how performance correlates with the hardware specifications (Communication Latency/Memory Bandwidth) of each tier. Inspired by prior research [37], we investigate the relationship between system-level metrics to higher-level metrics of interest, i.e., execution time. Towards this direction we evaluate the system-level event Pearson correlation [38] with execution time, as depicted in Figure 5. We observe that `bayes` benchmark has the highest correlation(near-linear) with almost all system-level metrics, thus linear prediction models are expected to perform efficiently in execution time prediction over new tiers deployed. On the contrary, benchmarks such as `pagerank`, have low correlation to system-level metrics, thus more complex models are required for providing efficient time prediction on alternative tiers. Moreover, Figure 6 indicates the Pearson correlation

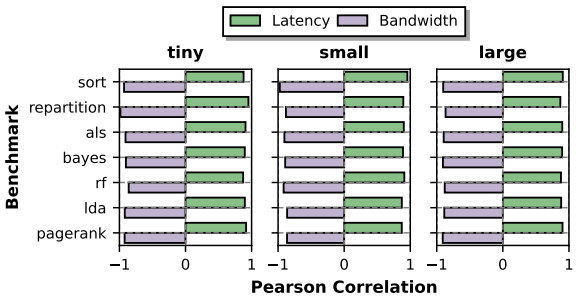


Fig. 6: Correlation of hardware specs (latency and bandwidth) with execution time for all applications and workloads

of execution time with bandwidth and latency for each individual application and workload, across all tiers. We observe that the execution time converges to near perfect positive(1) and negative(-1) correlation for both latency and bandwidth, respectively. Thus, execution time is highly correlated with both latency and bandwidth, therefore linear prediction models are expected to perform efficiently. Overall, we expect that by combining the hardware-related specifications along with system-level metrics, we can create accurate predictions of performance degradation across the different tiers, by using analytical models and/or Machine Learning techniques.

⋆ **Takeaway 8:** *The inherent latency/bandwidth specifications and system-level events reveal high correlation with performance over different tiers, thus, they can be utilized for prediction of execution time on heterogeneous memory tiers.*

G. Discussion and future perspectives

The outcomes of this paper reveal that Spark applications are highly affected by disaggregated memory architectures, mainly due to the high latency imposed by remote accesses. However, there are cases where remote memory can be employed without any discount in performance, whereas there is also plenty room for exploration w.r.t. determining the optimal memory tier per access type, especially in the case of persistent memory. On top of that, since Spark is designed to work over distributed nodes, further optimizations can be performed on the engine itself, to leverage a unified disaggregated memory architecture thus avoiding shuffling operations and minimize the overhead of remote memory access and scale across memory tiers. Last, the high correlation among system-level metrics and execution latency gives promising signs for employing Machine Learning techniques to effectively predict the performance of applications across different memory tiers.

V. Conclusion

In this work, we conduct an extensive characterization of Spark in-memory analytics over heterogeneous DRAM/NVM memory tiering. We investigate the impact of various factors that affect high-level characteristics, such as performance and energy consumption. We provide deployment guidelines and explore effective system and application configurations, while we denote potential future aspects. The major outcomes of this work can be exploited by developers who target Spark analytics over multi-tier heterogeneous memory systems.

REFERENCES

[1] L. Zhou, S. Pan, J. Wang, and A. V. Vasilakos, "Machine learning on big data: Opportunities and challenges," *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[2] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big iot data analytics: architecture, opportunities, and open research challenges," *ieee access*, vol. 5, pp. 5247–5261, 2017.

[3] L. Rodríguez-Mazahua, C.-A. Rodríguez-Enríquez, J. L. Sánchez-Cervantes, J. Cervantes, J. L. García-Alcaraz, and G. Alor-Hernández, "A general perspective of big data: applications, tools, challenges and trends," *The Journal of Supercomputing*, vol. 72, pp. 3073–3113, 2016.

[4] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10, Ieee, 2010.

[6] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[7] 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.), pp. 715–728, USENIX Association, 2021.

[8] 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.), pp. 331–345, ACM, 2019.

[9] A. Raybuck, T. Stamler, W. Zhang, M. Erez, and S. Peter, "Hemem: Scalable tiered memory management for big data applications and real NVM," in *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021* (R. van Renesse and N. Zeldovich, eds.), pp. 392–407, ACM, 2021.

[10] 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*, pp. 868–880, IEEE, 2020.

[11] 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.), pp. 690–695, IEEE, 2016.

[12] 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.

[13] A. Belmonte, H. Oh, N. Rassoul, G. Donadio, J. Mitard, H. Dekkers, R. Delhougne, S. Subhechha, A. Chasin, M. Van Setten, *et al.*, "Capacitor-less, long-retention (¿ 400s) dram cell paving the way towards low-power and high-density monolithic 3d dram," in *2020 IEEE International Electron Devices Meeting (IEDM)*, pp. 28–2, IEEE, 2020.

[14] T. Ali, P. Polakowski, S. Riedel, T. Büttner, T. Kämpfe, M. Rudolph, B. Pätzold, K. Seidel, D. Löhr, R. Hoffmann, *et al.*, "High endurance ferroelectric hafnium oxide-based fefet memory without retention penalty," *IEEE Transactions on Electron Devices*, vol. 65, no. 9, pp. 3769–3774, 2018.

[15] M. Katsaragakis, L. Papadopoulos, C. Baloukas, and D. Soudris, "Memory management methodology for application data structure refinement and placement on heterogeneous dram/nvm systems," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 748–753, IEEE, 2022.

[16] M. Andrei, C. Lemke, G. Radestock, R. Schulze, C. Thiel, R. Blanco, A. Meghlan, M. Sharique, S. Seifert, S. Vishnoi, *et al.*, "Sap hana adoption of non-volatile memory," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1754–1765, 2017.

[17] "Daos architecture."

[18] S. Park, H. Kim, K. Kim, J. So, J. Ahn, W. Lee, D. Kim, Y. Kim, J. Seok, J. Lee, *et al.*, "Scaling of memory performance and capacity with cxl memory expander," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, pp. 1–27, IEEE Computer Society, 2022.

[19] D. D. Sharma and S. Tavallaei, "Compute express link 2.0 white paper," *CXL. Retrieved October*, vol. 31, p. 2021, 2020.

[20] C. Lin, J. Zhuang, J. Feng, H. Li, X. Zhou, and G. Li, "Adaptive code learning for spark configuration tuning," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 1995–2007, IEEE, 2022.

[21] T. Chiba and T. Onodera, "Workload characterization and optimization of tpc-h queries on apache spark," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 112–121, IEEE, 2016.

[22] P. S. Rao and G. Porter, "Is memory disaggregation feasible? a case study with spark sql," in *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pp. 75–80, 2016.

[23] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Micro-architectural characterization of apache spark on batch and stream processing workloads," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pp. 59–66, IEEE, 2016.

[24] S. J. Jandaghi, A. Bhattacharyya, and C. Amza, "Phase annotated learning for apache spark: Workload recognition and characterization," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 9–16, IEEE, 2018.

[25] E. Kakoulli and H. Herodotou, "Octopusfs: A distributed file system with tiered storage management," in *Proceedings of the 2017 acm international conference on management of data*, pp. 65–78, 2017.

[26] D. Nikitopoulou, D. Masouros, S. Xydis, and D. Soudris, "Performance analysis and auto-tuning for spark in-memory analytics," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 76–81, IEEE, 2021.

[27] S. Chen, W. Wang, X. Wu, Z. Fan, K. Huang, P. Zhuang, Y. Li, I. Rodero, M. Parashar, and D. Weng, "Optimizing performance and computing resource management of in-memory big data analytics with disaggregated persistent memory," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp. 21–30, IEEE, 2019.

[28] V. Hayot-Sasson, S. T. Brown, and T. Glatard, "Performance benefits of intel® optane™ dc persistent memory for the parallel processing of large neuroimaging data," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 509–518, IEEE, 2020.

[29] A. Shanbhag, N. Tatbul, D. Cohen, and S. Madden, "Large-scale in-memory analytics on intel® optane™ dc persistent memory," in *Proceedings of the 16th International Workshop on Data Management on New Hardware*, pp. 1–8, 2020.

[30] S. Kannan, Y. Ren, and A. Bhattacharjee, "Klocs: Kernel-level object contexts for heterogeneous memory systems," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 65–78, 2021.

[31] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated key-value stores," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, pp. 33–48, 2020.

[32] F. V. Zacarias, R. Nishtala, and P. Carpenter, "Contention-aware application performance prediction for disaggregated memory systems," in *Proceedings of the 17th ACM International Conference on Computing Frontiers*, pp. 49–59, 2020.

[33] 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*, pp. 189–200, IEEE Computer Society, 2012.

[34] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *2010 IEEE 26th International conference on data engineering workshops (ICDEW 2010)*, pp. 41–51, IEEE, 2010.

[35] S. Akram, "Performance evaluation of intel optane memory for managed workloads," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 3, pp. 1–26, 2021.

[36] "Intel mba tool."
[37] D. Masouros, S. Xydis, and D. Soudris, "Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 184–198, 2020.
[38] I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," *Noise reduction in speech processing*, pp. 1–4, 2009.