Received September 17, 2019, accepted October 17, 2019, date of publication November 19, 2019, date of current version December 2, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2954169

Survey of Memory Management Techniques for HPC and Cloud Computing

ANNA PUPYKINA AND GIOVANNI AGOSTA
Dipartimento di Elettronica e Informazione, Politecnico di Milano, 20133 Milan, Italy
Corresponding author: Anna Pupykina (anna.pupykina@polimi.it)

This work was supported in part by the H2020 EU Project RECIPE under Grant 801137.

ABSTRACT The emergence of new classes of HPC applications and usage models, such as real-time HPC and cloud HPC, coupled with the increasingly heterogeneous nature of HPC architectures, requires a renewed investigation of memory management solutions. Traditionally, memory is shared by an operating system using segmentation and paging techniques. At the same time, new classes of applications require Quality of Service (QoS) guarantees. As such, the typical practice of reserving a subset of the supercomputer to a single application becomes less attractive, leading to the exploration of cloud technologies. In this context, a viable scenario is that of multiple applications, with different QoS levels, coexisting on the same deeply heterogeneous HPC infrastructure and sharing resources. However, for this scenario to succeed in practice, resources, including memory, need to be allocated with a vision that includes both the application requirements and the current and future state of the overall system. In this survey, challenges of memory management in HPC and Cloud Computing, different memory management systems and optimisation techniques to increase memory utilisation are discussed in detail.

INDEX TERMS Clouds, high performance computing, memory management, resource management.

I. INTRODUCTION

The desire to achieve Exascale leads to the rapid development of high-performance computing at the hardware, software and application levels. At the hardware level, heterogeneity is becoming the main trend towards higher net performance and, more importantly for Exascale, performance per watt, as evidenced by the dominance of heterogeneous architectures in the Top500[1] and Green500[2] lists. On the application side, new domains appear that previously lacked access to HPC resources. While traditional applications of meteorology, oil and gas, seismology, or aero- and hydrodynamics still dominate, the Cloud HPC [1] is now accessed also by small and medium-sized enterprises [2]. Thus, the traditional HPC scenario with running batch jobs on separate nodes is replaced by a new scenario in which several applications with different QoS levels coexist in the same deeply heterogeneous HPC infrastructure and share resources. In this context, time-critical applications, such as financial analytics, online video transcoding and medical imaging, require predictable performance, which is contrary to the traditional need to maximise resource utilisation while minimising power consumption.

Moreover, moving HPC to the Cloud requires taking into account the different resource optimisation targets inherent to the HPC and Cloud Computing environments. The cloud aims to serve applications on a reduced amount of available hardware resources using virtualisation technologies in order to achieve economies of scale. Recent studies on the resource management in the Cloud have focused on the virtualisation techniques [3], such as isolation between Virtual Machines (VMs) and reducing VM overhead, to increase scalability and performance. Unlike cloud services, HPC environments need to maximise the interconnect performance. Thus, HPC cloud resources need to provide both scalability and high performance. Research in HPC aims at finding this balance and at using a unified manager for both Cloud and HPC targets [4].

In this survey, we first emphasise the importance of memory optimisation in supporting the Exascale of the future, and we then describe the scope of memory management in HPC and Cloud. We mainly focus on the analysis of different memory management techniques used to achieve the effective management of memory resources on the different layers of

The associate editor coordinating the review of this manuscript and approving it for publication was Shuiguang Deng.

[1]www.top500.org
[2]www.green500.org

hardware/software stack. Some challenges in current research are highlighted to motivate future research directions. This survey does not describe each algorithm in detail.

The rest of the paper is organised as follows. The role of memory management systems is discussed in Section II. Section III reports some past related surveys. Sections IV and V present a broad classification of memory management techniques in HPC and in Cloud Computing, respectively. Finally, we discuss open problems and future research directions of memory management in Section VI and offer some conclusions in Section VII.

II. SCOPE OF MEMORY MANAGEMENT

Traditionally, HPC systems are designed in a way that all nodes can run at peak performance simultaneously. It allows performing computational intensive applications and benchmarks, such as High-Performance Linpack, with maximum performance. Such systems are built on a close interaction between CPU and memory. When the system is scaled, the CPUs and the memory, since they are physically coupled, they get upgraded proportionally, thereby keeping the processor-memory ratio constant. However, the memory bandwidth based on the traditional processor-memory interconnection technology will limit also the performance of heterogeneous architectures with accelerators. In this case, the memory in fact consumed at a higher rate, proportionally to their processing capabilities [5]. The current trend in HPC system design is to increase the number of memory channels per CPU and the number of I/Os in each double data rate (DDR) memory generation. This will increase the price and the memory system power consumption [6]. Modern HPC architectures meet new challenges.

A. TOWARDS EXASCALE COMPUTING

In the last decade, various researchers have worked to overcome Exascale challenges. The request for more complex computational resources was indicated in the International Exascale Software Project roadmap [7]. Increasing the number of cores per node is a traditional way to increase computational performance. The roadmap indicated the need for a wider use of heterogeneous nodes that combine stream-based cores with traditional cores based on load/store. Also, it was noticed that in addition to increasing the complexity of the computational resources, the heterogeneous architecture complicates the programming models and languages in order to provide the efficient resources sharing (e.g. the memory bus). The study of data locality was appointed as one of the directions of the system software development since it can have a much significant impact on performance when used in systems with a complex memory hierarchy. Machines were expected to have deep and non-coherent memory hierarchies. The locality-aware scheduling in highly NUMA memory architectures would have demonstrated how to use them. By 2013, such runtime systems should have been developed. A dynamic runtime management system for all types of resources, such as cores, bandwidth, logical and physical memory or storage (i.e., data replication management, coherency and consistency, layout changes more suitable for specific cores/accelerators), was announced to be completed by 2019 year.

The importance of new memory management solutions has also been described in the report on crosscutting technologies for computing at the exascale level [8]. The memory management was seen as a significant challenge for exascale science applications due to deeper, complex memory hierarchies and relatively smaller capacities. A complex memory hierarchy expected at the exascale requires tools and interfaces to manage and control memory for runtime systems, and to provide information to assist the compiler in optimising memory management. A promising way to deal with memory complexity issues was to base it on dynamic, latency-tolerant approaches. The following two main challenges were highlighted. **On-Chip Memory:** Given the importance of minimising unnecessary data movement, the ability to allow software control of data movement for performance-critical kernels would be useful. Methods that can coexist with conventional, automatically managed caches are important because an incremental porting path is required. **Global Addressing:** Message passing probably can continue to be used for internode communication in exascale systems, but it is not practical for interprocessor communication at billion-way parallelism. Global memory addressing is preferred over cache-coherence for managing fine-grained computation and data movement on-chip [8].

According to the survey looking at the challenges of the high-performance computing scaling challenges [9], the complexity of an application and system software will continue to grow in several dimensions. In the specific case of memory management, it is expected that both homogeneous and heterogeneous designs would probably have multiple levels of memory; these hierarchical memories would affect the complexity of the software. The challenge proposed to the software developer is to find the most efficient way to utilise the different memory types, as well as to minimise the movement of data within the hierarchy, due to the NUMA nature of the hierarchy and the different types of memory. Moreover, there are clear similarities in HPC and Cloud Computing for what concerns different architectural and software issues, such as component rightsizing and optimisation, energy management and efficiency, and programming efficiency.

The recent crosscut report of exascale requirements reviews held by Advanced Scientific Computing Research program [10] highlighted the extreme-scale systems that will be delivered in 8 to 10 years, and that will offer unprecedented compute power and challenging parallelism. Relatively slow interconnects between nodes become the main challenge for exascale machines that will contain millions of heterogeneous cores with deep memory hierarchies. The report discusses memory issues that significantly complicate software development, such as the need for explicit control of data movement, depth and types of memory and burst buffers. At the same time, planning and distribution policies are also complicated by the need to support extra functionality in modern workflows, especially data analysis requirements: real-time, pseudo-real time, collaborative planning, variable work requirements, and allocations based on other resources, such as disk and memory. In real simulations in nuclear astrophysics, deep memory hierarchies were expected to present significant challenges because large, multidimensional data structures are ubiquitous in astrophysical implementations. So the way the data structure is organised in memory is essential to ensure effective memory accessibility (for example, for vectorisation) and memory allocation. Problems with memory allocation include the placement of data structures in certain levels of the memory hierarchy to ensure the best locality. It was noted that the nuclear physics community needs a portable way without direct compiler support to effectively manage the memory layout and the memory allocation. Memory management libraries were proposed as probably the best method. Fully ceiding memory management solutions in runtime systems were found ineffective since the requirements for layout and placement are physically motivated and can depend on time.

B. HPC IN THE CLOUD

Another challenging research direction in HPC is moving HPC to the Cloud, that is, the efficient use of cloud resources to run HPC applications. There are three ways in which the cloud abstraction can be used in the HPC context [11]: **HPC in the cloud:** moving HPC applications to current cloud platforms; **HPC plus cloud** addition HPC resources with cloud services; **HPC as a service (HPCaaS):** exposing HPC resources using cloud services. While HPC architectures have become more heterogeneous by employing various types of accelerators, such as NVIDIA GPUs, Intel Xeon Phis, FPGA based, cloud platforms mostly employ homogeneous architectures. Nowadays, vendors, such as Amazon (P2, P3, G3, F1 instances of Amazon EC2 [3]), provide users with access to heterogeneous cloud architectures.

The main issue of porting HPC applications to the cloud environment depends on the difference between the requirements and workflow of the traditional cloud services applications and those of HPC applications. Traditionally, HPC applications tend to require a higher amount of computing resources (CPU, memory, network bandwidth) than cloud services. Regarding the difference in the workflow, HPC applications are often executed in batches, rather than launching individual jobs in a 24/7 environment, which is instead typical for Clouds. Therefore, moving HPC applications to cloud platforms requires an optimisation of the resource allocation that takes into account the specific HPC application features.

Efficient management of shared resources, such as memory, CPU, storage, is a crucial issue in cloud computing. Here, the client requires resources on-demand, thus making the behaviour highly dynamic. Resource management in cloud is a complex process that combines matching resources to applications according to the requested requirements, scheduling, allocating those resources, monitoring them over time and load balancing in order to run applications as efficiently as possible. The major memory challenge in cloud computing consists in the imbalance of memory usages in each node. Cloud systems serve applications with dynamic, heterogeneous and complex memory usages. Due to the presence of data-intensive workloads, such as in-memory databases, data caching, bioinformatics, and graph processing, available free memory may vary widely across cloud nodes. The trade-off between significantly increasing the memory capacity requirements for memory-intensive applications and free memory on distributed nodes is addressed by rightsizing memory allocation and exposing a global memory bank to all machines. This increases the effective amount of memory.

C. MEMORY MANAGEMENT FUNCTIONALITY

The memory management (MM) system runs as a separate service or as a part of the runtime management system on Service Node (SN) and it controls memory allocation on the Computational Nodes (CN). It deals with the following issues:

- choosing the most suitable memory according to the allocated processing elements;
- enabling concurrent, thread-safe memory allocation and deallocation while avoiding fragmentation;
- performing translation from virtual to physical addresses, and vice versa;
- performing runtime optimisation.

Choosing the most suitable memory In order to select the best memory modules, the memory manager takes into account the following search criteria:

- bandwidth between the allocated processing elements and the memory module;
- latency of the memory module;
- direction of data transfer (in/out);
- space available on the module;
- load on routing and ports.

Actual criteria are defined depending on the QoS requirements, which are provided to the memory manager by the runtime resource manager. Furthermore, depending on the target architecture, certain characteristics may change at runtime. In general, memory management systems are based on an algorithm that takes runtime decisions on the basis of continuously updated information about the state of the resources and that aims to:

- allocate the buffer in memory unit near the processing unit and leave memory units free near unused processing units;
- leave free spaces for allocation of high priority requests.

Concurrent, thread-safe memory allocation and deallocation. At present, there are many implementations of the malloc function, each one with its own strengths and weaknesses. In general, memory allocation in Clouds is managed

[3]https://aws.amazon.com/ec2/instance-types/

by the OS when it is executed on the virtual machine. In HPC systems, control on memory allocation can be taken by the OS or by the platform-dependent library.

Runtime optimisation. If there is free memory, the memory manager could evaluate the possibility to migrate the data of running kernels to more suitable memory modules. This would allow the memory allocator to optimise the performance of already running kernels and to free memory for higher priority applications; however, the benefits of migration should be carefully evaluated, because the kernel that uses the to-be-migrated data must be stopped while the data is moved.

III. RELEVANT SURVEYS

The importance of resource management in HPC, HPC in the Cloud and Cloud Computing and the relevant solutions were discussed in several surveys. The Energy-Efficient High-Performance-Computing Working-Group (EE HPC WG) Energy and Power Aware Job Scheduling and Resource Management (EPA JSRM) team conducted comprehensive interviews and identified the energy- and power-aware scheduling and resource management solutions that are used on the major HPC sites [12]. This report showed different trends in power consumption minimisation, such as the job-killing when the power limit is exceeded, the analysis of information obtained from the power system monitoring and the forecast of energy consumption for each job. The impact of the memory system on the power consumption is out of scope here.

Research challenges, including resource management aspects for HPC Cloud, were surveyed in [4]. Overall most works described in this survey were motivated by network and virtualisation issues from current cloud platforms. Moreover, it was indicated that performance prediction is key to allocate resources.

Several studies on memory and device disaggregation are surveyed in [13]. In addition, an alternative vision on the future directions for devices disaggregation is proposed.

Survey [14], for instance, emphasises the importance of the power management solutions in the Cloud and it shows the place of memory components in the total power consumption. Power Scalable Memories are less commonly viewed as components designed to minimise the energy consumption. However, works on memory management energy reduction are also mentioned in this survey.

Techniques for efficient resource provision on the cloud platforms were discussed in [15]. The problem was reduced to that of placing the virtual machines to support the requested services. More than 150 articles were surveyed and the state of art of the algorithms, such as virtual machine migration, forecast methods, stability and availability, to realise these objectives, was presented.

Forecast models for resources provisioning were surveyed in [16]. Prediction models take into account different types of resources, which include physical resources such as memory, storage, servers, processors and networking. Researches suggest not to ignore the correlation among resources by focusing on one or two of them. [17] discusses and classifies popular prediction techniques in the general context of computing systems with an emphasis on multicore processors. These prediction techniques are used widely from predicting simple attributes, such as buffer utilisation in networks-onchip, to predicting complex relationship affecting the power usage effectiveness in data centres.

Survey [18] focuses on different techniques of memory partitioning and management across multiple guest OSs in a virtualised environment where a hypervisor controls and manages the memory sub-system of the machine. Memory virtualisation solutions, implemented in software and with hardware assistance, are evaluated against virtualisation requirements.

The particular case of resource management for hard and soft real-time multi-/many-core systems are described in [19]. The resource allocation strategies that satisfy real-time constraints were analysed, and experiments were carried out to compare execution time, energy consumption and utilisation.

In contrast to the surveys mentioned above, this survey provides a broad summary of the state-of-art techniques in HPC and Cloud used for memory optimisation, concentrating on the wide number of objectives to address the different layers of resource management. To the best of our knowledge, this survey is the first attempt to put together all direction of the memory optimisation techniques both in HPC and in the Cloud.

IV. MEMORY MANAGEMENT TECHNIQUES IN HPC

OS (Linux) based MM is a traditional technology for many modern HPC systems. It offers wide functionalities, general applicability and easy of use for application developers. Linux based MM allows to achieve sufficient performance improvement in properly configured HPC systems and requires less effort to adjust various runtime management systems and standalone MM. However, as it was mentioned in the previous section, there is a trend to complicate the architecture of the HPC system, which requires more OS functionality. Moreover, the goal of traditional OS resources optimisation, that is increasing performance, directly conflicts with one of the HPC traditional target scenarios that provide consistent performance to deal with a heavy workload. To satisfy special functional needs, complex heterogeneous and configurable architectures require software support at the runtime system level. Recent deeply heterogeneous memory systems require additional hardware support.

Dynamic memory allocation is commonly used within traditional HPC systems and it is usually performed by the OS. Furthermore, OS-based dynamic memory allocation does not require changes in common applications. Nevertheless, heterogeneous programming models, such as OpenCL, require applications to declare memory needs before offloading a computation. This is caused by heterogeneity in both processing units and memory devices.

To cope with the limited number of memory accesses, which causes the decrease of the modern heterogeneous systems performance, NUMA architectures can allocate a separate memory to perform computational processes and, at the same time, provide a unification of the use of different kinds of memory devices. There is ongoing research on cache optimisation and on the use of scratchpad memory.

Surveyed studies have a combination of different objectives and hardware/software technologies that depends on the target architecture. The brief overview of the surveyed memory management techniques is presented in Figure 1.

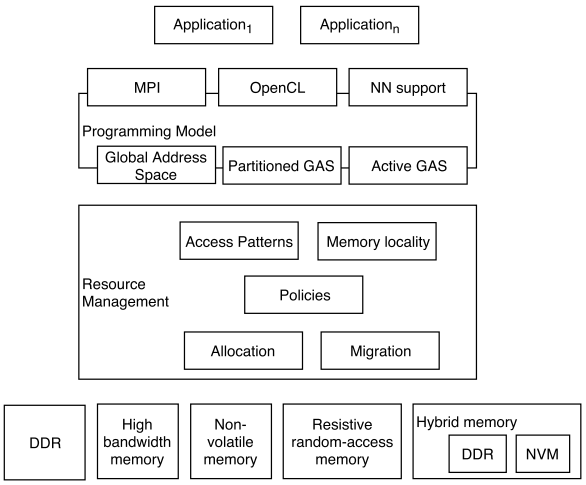

FIGURE 1. Layers of the surveyed memory management techniques in HPC.

A. ACCESS PATTERNS

Regarding the limitation of the data read and write latency for the eDRAM integrated to the same silicon chip with the processing logic, [20] aims at hiding the on-chip communication mechanism between the applications local memory management. The local memory management system called Pattern Aware Memory System (PAMS) operates independently from the master core. It accelerates both static and dynamic data structures and their access patterns based on the information provided by pre-compiled pattern descriptors. In addition to this, PAMS controls data movements between the main and the scratchpad memories. Sharing-Aware Memory Management Unit (SAMMU) [21] analyses the memory access behaviour on the hardware level. It also provides information to the operating system to perform an online thread- and data-mapping to increase the locality, thus improving performance. SAMMU extends the operations of the MMU to gather information about the threads and NUMA nodes that accessed the particular physical pages without stalling the application execution.

A general scheme of the access pattern usage is shown in Figure 2. The mechanism that utilises hardware event counters or pre-compiled pattern descriptors to track specific events allows estimating the impact caused by contention on share hardware resources. Pattern Aware Memory Controller is in charge to optimise memory allocation and can be considered on the different levels of the memory hierarchy.

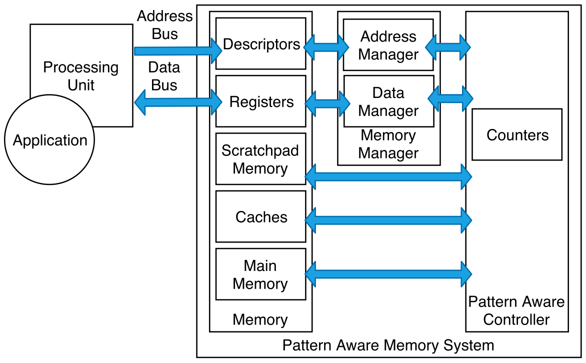

FIGURE 2. General scheme of the pattern-aware memory systems.

B. MIGRATION

The memory pages management approach for various hybrid memories based on page migrations is presented in [22]. The utility-based hybrid memory management (UH-MEM) systematically estimates both the single application benefits and the overall system benefits of migrating a page between different memory types. Decisions on data placement are based on the estimated benefits. NUMA, that is an abstraction used to expose heterogeneous memory, has one major issue. Data migration between devices is only resolved to page granularity. Other data allocated on the same pages are migrated as well. A solution based on re-purposing arena-based heap management to keep locality among related data structures that are used together is discussed in [23]. Two mechanisms available on modern Linux systems for migrating data between physical locations are analysed in [24]. Memory migration mechanisms within the Linux kernel are far behind a simple user-space memory copy, and, with additional software degradation, a regular NUMA system can reasonably approach the bandwidth of a deep-memory architecture. Experiments carried out on this approximation platform show that decreasing the number of worker threads and using them for data migration instead allows getting a significant speed-up. Reference [25] proposed an operating system to assist the hierarchical memory management system designed for the manycore co-processor based heterogeneous architectures. The co-processor's RAM is controlled entirely by the OS kernel, so the OS is in charge of orchestrating the data movement between the host and the device and of updating the process virtual memory address space.

The main motivation for the memory migration is that if a memory page is not migrated along with a task that is migrated to another node, accessing the page on the original node incurs remote memory access. Besides, memory migration is applied between different types of memory, which is especially essential for memory-intensive applications. A general scheme of the memory migration with page granularity is shown in Figure 3. The migration can be managed both by the OS, as in [25] and [24], and by the hardware, as in [24] and [22]. However, page migration is an expensive operation and can lead to additional overhead

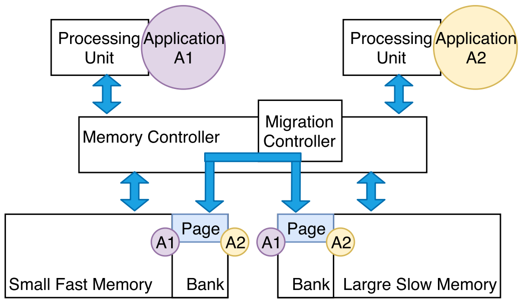

FIGURE 3. **General scheme of the memory migration.**

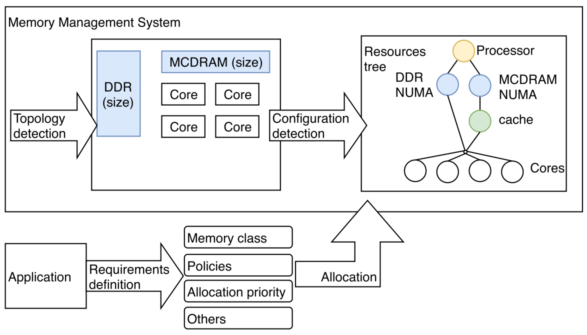

FIGURE 4. **Overview of the memory locality in HPC.**

for memory access due to page granularity. The proposed approaches consider various ways to reduce migration costs.

C. MEMORY LOCALITY

A way to expose processor and memory locality information in the Linux kernel and in user-space libraries such as the hwloc software project was discussed in [26]. An extension of the hierarchical tree to include upcoming heterogeneous memory architectures and memory-side caches was proposed. This solution is convenient to find local resources and to apply recursive top-down placement algorithms while correctly exposing the topology of these new architectures. Flexible and portable memory allocation toolkit hexe [27] divides memory allocation into three parts: allocating a virtual address, mapping a virtual address to a physical memory class that is under control of the toolkit, and mapping a virtual address to an exact physical memory page that will be used by the OS. The initialisation of the hexe runtime system and the topology detection are initially performed on the OS level using hwloc library. Hexe is responsible for a more comprehensive topology and for architectural feature detection. An extension of the heap management system jemalloc in conjunction with memkind library was proposed in [28]. The goal here is to enable fine-grained client control over memory properties and to reuse memory modified by expensive system calls for the highly threaded environment. A minor modification of the Linux kernel for more intuitive control of the placement of memory pages was proposed in [29]. Generally, NUMA is based on a single distance metric between all domains. In addition, other metrics characterising relationships between nodes were included, such as latency, bandwidth, capacity. The NUMA distance becomes a two-dimensional metric, since this approach matches the source node with the CPU performing a querying process with the ordered list of NUMA nodes with memory devices.

The performance of the application is significally affected by the explicit control of the locality of physical memory. Furthermore, being able to allocate memory by taking into account memory locality becomes imperative with upcoming new memory technologies. A general scheme of the memory locality usage is shown in Figure 3. Memory allocation requires not only knowledge about memory hierarchy and configuration but also detailed application analysis. Current research shows that the automatic use of memory locality is still not as efficient as having explicit instructions from the application developer that might require rewriting the code for each different architecture.

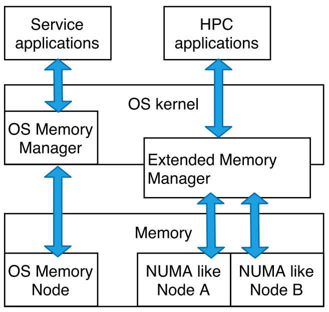

FIGURE 5. **Overview of the memory partitioning in HPC.**

D. PARTITIONS

Adapting the Linux MM in order to subdivide NUMA memory nodes,thus improving the partitioning of resources among processes running on the same node, was done by [30]. They extend the Linux kernel with additional resource control capabilities based on control groups (cgroups) to maximise resource utilisation with HPC workloads. Linux lacks control over physical memory location, making it non-effective for deeply heterogeneous hierarchical memory. To overcome this limitation, physical NUMA nodes were partitioned into arbitrary logical blocks at a user-defined granularity and they were then presented to the rest of the system as separate NUMA nodes. High Performance Memory Mapping and Allocation Platform based on bypassing Linux memory management presented in [31] allow to dynamically partition a node physical memory and independently manage partitions in a separate and isolated resource management layer. HPC applications running on a consolidated platform use a specialised lightweight memory management framework explicitly designed around the requirements of HPC applications.

E. POLICIES

In [32] it was noticed that significant performance improvements could be achieved with data mapping in NUMA architectures. Generally, locality-based policies improve performance more than policies that are based on memory balance. A mixed policy was proposed to address the trade-off between optimising locality and balancing memory controllers. In [33] it was demonstrated that the performance-focused data placement technique might result in lower overall reliability. The static, dynamic, and program annotation-based reliability-aware data placement techniques were proposed. The reliability-aware data placement approach analyses memory pages for their hotness (e.g., how 'active' the page is currently in the cache) and vulnerability at runtime. To avoid prior profiling, the reliability-aware dynamic migration schemes were developed. The co-designed hardware-software mechanism HpMC [34] implements a policy switching engine based on the temporal locality of applications and several new components that extend a single-level memory controller to facilitate switching policies and migrating pages. The periodical analysis of the temporal locality guides the switch between a hierarchical inclusive memory system PCache and a flat exclusive memory system HRank to deliver high performance and energy efficiency.

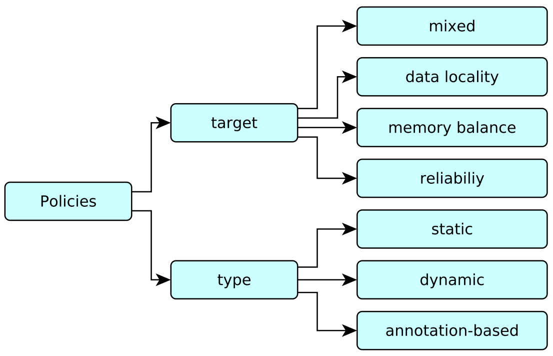

FIGURE 6. **Taxonomy of the proposed policies.**

Traditional performance-enhancing mechanisms rely on increasing the locality of memory accesses. In addition to the locality-based techniques, recent studies have also proposed to consider several factors when performing memory mapping, such as balance and reliability, to avoid overloading and guarantee fault tolerance. A brief taxonomy of the proposed policies is presented in Figure 6. Taking all factors into account while performing the mapping can result in more significant improvements than when relying on a single metric only. However, it makes policy computationally expensive and error-prone. Besides, most of the current techniques consider a single application. Since real HPC systems execute several applications at the same time, the mapping decisions can conflict between them.

F. HIGH BANDWIDTH MEMORY

High bandwidth memory (HBM) is a new memory technology that allows a significant performance improvement on bandwidth-bound applications. However, more bandwidth does not always bring a better execution time, especially when it is allocated without enough concurrency in the memory system to support it [35]. In [36], the HBM-DRAM memory system was analysed in terms of performance benefits of representative HPC applications executions. Such memory architecture consists in the HBM implemented with 3D-stacked Multi-Channel DRAM (MCDRAM) and conventional DRAM. These are significantly different in terms of capacity, bandwidth and latency. The HBM-DRAM system was configured in flat (as an additional NUMA node), cache and hybrid mode. If the intra-node communicator includes a sufficient number of processes or if the buffer size requested is large enough, Enhanced memory management schemes for HBM based on MCDRAM presented in [37] allocate user and shared buffers to the high-bandwidth on-package memory. In addition to this, they reduce the overheads of memory mapping. A runtime-aware prefetching mechanism presented in [38] allows within the CHARM++ runtime system to prefetch and to evict data heterogeneous memory nodes when the working set does not fit within HBM. The proposed mechanism was investigated for bandwidth-sensitive HPC applications on Intel Xeon Phi KNL. Allocating memory-intensive workloads among MCDRAM and DDR memory resources in Intel's manycore processor KNL using different heterogeneous memory modes is described in [39]. An intelligent scheduling approach based on profiled resource usage patterns showed that the actual performance of multiple message passing interface workloads could be improved in terms of the execution time and system utilisation.

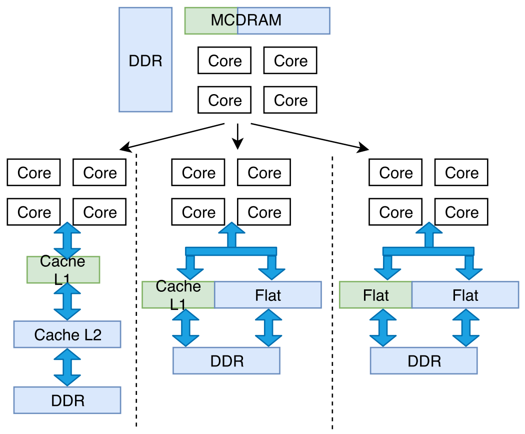

FIGURE 7. **Overview of the high bandwidth memory modes.**

Knowing the application and the architecture's topology is needed to use different memory modes (shown in Figure 7). The configuration which are better fine-tuned, have a significant improvement in performance. This, however, comes at the cost of non-portable optimisations. At the same time, the basic configurations that do not require application or runtime changes can offer less performance. In addition to the configuration, HBM is challenging in terms of using in programming models. For instance, communication libraries such as MPI can use HBM to allocate and manage internal memory objects for achieving efficient inter-node communication support. However, this affects the available memory budget, reducing the amount of memory accessible to the user data and, potentially, reducing the overall performance. In some studies, HBM has been used as a cache for the slower memory. One of the critical advantages of a caching approach is that existing applications can run unmodified. Accessing and storing a large number of tags are challenging in the multi-gigabyte cache managing.

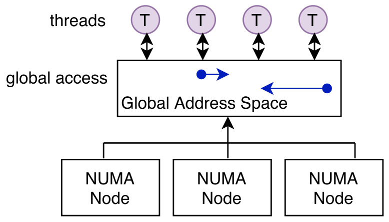

FIGURE 8. **General scheme of the global address space.**

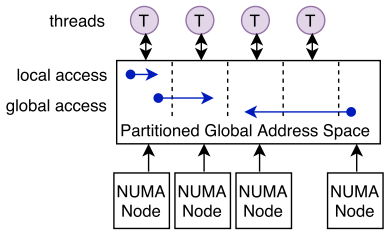

FIGURE 9. **General scheme of the partitioned global address space.**

G. GLOBAL ADDRESS SPACE

Collaborating European projects united by a common goal: ExaNoDe, ExaNeSt and ECOSCALE, propose the memory model that enabled a real global address space (GAS), where data are actually shared between multiple partitions without the need to perform memory transfers([40]–[43]). The ExaNoDe operating system is a NUMA-aware Linux that supports dynamic distributed memory management and user-level socket-based communication across servers to offer the benefits of UNIMEM system architecture. UNIMEM provides data movement mechanisms and integration with peripheral devices, allowing to use memory hierarchy for the OS by accessing the light RDMA operations. A general scheme of the GAS is shown in Figure 8. A partitioned GAS (PGAS) introduced in [44] is a non-uniform global address space where the representation of a global address statically encodes the physical address of the memory. PGAS approaches can be seen as a compromise between exposing locality and presenting global access to data. A general scheme of the PGAS is shown in Figure 9. Address translation in PGAS uses static address mappings that are efficient for RDMA, but that limit the ability for the computation and communication load balancing. Highly-scalable PGAS memory-centric system architecture where threads migrate to the data they access is presented in [45]. A portion of the address space marked as "Replicated" is used to ensure data

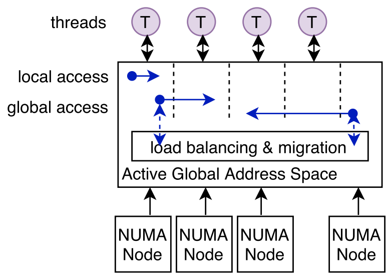

FIGURE 10. **General scheme of the active global address space.**

locality and it is accessible via a programming model. The instruction code is always maintained in replicated memory so that a thread can always find its code locally without knowing where the thread is executing. Active GAS (AGAS) was designed by [46] for a message-driven runtime system to dynamically balance load and data during execution. A general scheme of the AGAS is shown in Figure 10. AGAS allows performing hardware-controlled dynamic mapping, thanks to the fact that distributed shared memory systems create logical shared address space using distributed memory hardware at the OS level. Additionally, a network-managed global virtual memory where the AGAS runtime set-ups the network fabric to adaptively route messages based on memory locations rather than host identifiers was proposed.

For the memory accessed via GAS the need for cache coherency is eliminated since the data does not move. However, GAS complicates memory access because all memory accesses are remote, even if memory is located on a local node. On the other hand, PGAS provide a more-scalable solution by tightly coupling threads to memory locations. However, application classes that can utilise PGAS efficiently are limited due to difficulty in designing algorithms around the concept of stationary-threads.

H. ALLOCATION

The resource allocation and scheduling techniques can be applied as a part of MM in order to choose a specific memory unit for the task. Recent studies on the shared resource-aware scheduling and task allocation methods for HPC systems have focused on locality- [47], energy- [48], performance- [49], resilience- [50] aware usage planning of the processors and accelerators. The power-aware threads co-scheduling approach for the multicore processor presented in [51] takes into account an effect of the shared last-level cache on the performance degradation. Recent advances in memory allocation for homogeneous multicore architectures have focused on removing the need for application-specific allocators and on improving scalability and allocation speed [52]. The work presented in [53] addresses the adoption of dynamic memory management for the design of many-accelerator FPGA-based systems, thus allowing each accelerator to dynamically adapt its allocated memory according to the runtime memory requirements. It supports fully-parallel memory access paths by grouping block RAM (BRAM) modules into unique memory banks, named heaps, each managed by an allocator. The approach proves effective in increasing FPGA density.

The current research trend in memory allocation is to perform an online characterisation of the memory access behaviour and performing page migrations to improve the behaviour. Data placement for software-controlled on-chip memory requires more complex design solutions, where the characteristics of FPGA computation, memory access patterns, as well as the architectural properties, need to be explicitly considered. FPGAs come with large on-chip memory resource these days. However, the limited dynamic memory management (DMM) support does not allow application developers to maximise the utilisation of the on-chip memory resources during run time and thus achieve higher performance. CPU-based solutions, such as garbage collection and slab allocation, lead to significant resource overhead if use on FPGA directly. Recent research on DMM support for FPGA deal with reducing high latency of memory allocation. Another challenge in on-chip memory allocation is synthesising dynamic data structures that require either particular implementation of an application-specific memory management scheme or resolving the dynamic memory usage using static allocations.

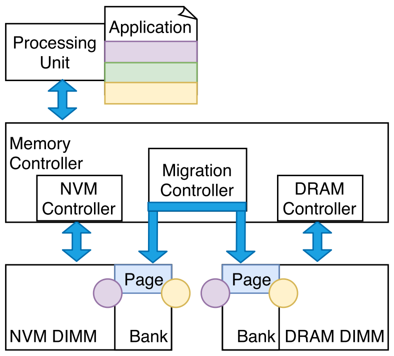

FIGURE 11. General scheme of the hybrid memory.

I. HYBRID MEMORIES

Hybrid memories that include DRAM and non-volatile memory (NVM) to extend the main memory capacity are used for large-memory applications. A general scheme of the hybrid memory is presented in Figure 11. In order to eliminate longer write latencies and higher write energies in NVMs, an offline/online placement scheme that uses access profiling was proposed in [54]. Since it was noticed that heap objects, allocated in the same place in the code, share similar access behaviours, in this approach, the global access characteristics of heap objects guide initial data placement.

Recent research evaluates the speed and endurance of hybrid memories with DRAM and NVM and mainly focuses on the identification of the hot/cold pages to move frequently accessed (hot) pages from the slow NVM to the fast DRAM to achieve higher performance and energy efficiency. However, a large number of invalid page migrations, where the cost to migrate a page is larger than the performance gain is prone not only to bring a huge overhead but also to generate unnecessary writes on NVM. Most existing NVM allocators focus on improving space utilisation and ignore the wear-leveling problem of NVM cells, especially for the wear leveling of memory cells within a page. Consequently, NVM is still suffering from imbalanced wear of memory cells in the same page.

J. NEURAL NETWORK AND DEEP LEARNING

In order to address the memory bottleneck that arises while training deep learning (DL) models on HPC systems, Memory-centric deep learning system architecture was proposed in [55]. The high-level deep learning framework analyses the neural network (NN) data dependencies' structure at compile-time and derives the data-dependencies of memory-hungry deep neural network data. This information is utilised by the runtime MM to schedule performance-aware, software-managed memory overlaying operations across the host-device memory to expand the reach of memory available for the training. The optimisation of the memory needed for executing NN applications was considered in [56]. A software/hardware interface allows Resistive random-access memory (RRAM) arrays [57] to be configured as accelerators for NN applications or as normal memory for a larger memory space dynamically. This solution provides the advantages of using RRAM efficiency for NN computation and the processing in-memory architecture to reduce the data movement overhead.

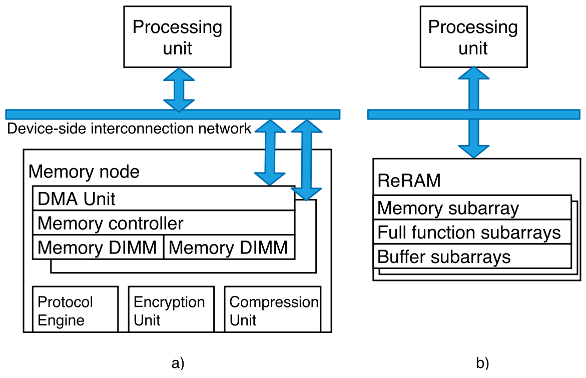

FIGURE 12. General scheme of the memory architectures for NN applications (a - memory hierarchy model, b - RRAM based architectures).

The trend in DL is to employ larger and deeper networks that lead to significant memory footprints. Modern DL frameworks require users to fine-tune their memory usage so that the training data of a deep neural network (DNN) fits within the physical memory. Virtualising the memory usage of DNNs enables all memory hierarchy to be utilised for memory allocations. A general scheme of the memory architecture for NN applications is shown in Figure 12 (a). Despite its merits, virtualising can acquire significant performance overheads due to huge communication latency between memory layers. State of the art solutions commonly employ bandwidth-optimised 3D stacked memories (such as HBM). However, increasing the capacity of this on-package memory is challenging. Furthermore, deploying DNN in embedded systems requires the memory system to meet the energy restrictions. As a memory solution for DNN, RRAM has the advantage of non-volatility, low power consumption and fast access speed. A general scheme of the RRAM memory architecture is shown in Figure 12 (b). Although the RRAM-based architectures achieved excellent results, there is still room for improvement, such as the resolution of the RRAM cells and the negative value storing.

K. PROGRAMMING MODELS

Being aware of the OpenCL performance portability issue, OpenCL software optimisation techniques, including memory allocation and mapping, for efficient execution HPC workload on embedded systems-on-chip (SoC) ARM Mali GPU Compute Architecture was proposed in [58]. The programmer effort required to manage NUMA-like two-level memory organisation manually is described in [59]. The application programmer is encouraged to invoke the TLM-malloc API for the subset of data structures and memory objects that are important. For the remaining memory objects (including the function call stack, the code segments, and other automatic memory items) a simple OS-based mechanism is assumed.

Recent research showed that programmer-driven memory management is the quickest way to reach the target bandwidth for the Exascale architecture. The high performance is achieved only through manual code optimisation and tuning and can not be reached by using only auto configurable memory systems. The main goal in optimising program models for HPC is to provide optimal performance for all applications in various workflows without any user intervention requires, such as application changes based on the knowledge of the low-level details of the hardware.

TABLE 1. Survey of memory management objectives in HPC.

Choosing the memory unit	[26] [29] [30] [33] [34] [40] [45] [54] [59]
Memory allocation and deallocation	[20] [27] [28] [30] [31] [36] [37] [38] [40] [52] [53] [58]
Memory mapping	[20] [21] [27] [30] [31] [32] [40] [44] [58]
Memory access	[20] [40] [44] [45] [46] [52] [53] [56]
Memory migration	[22] [23] [24] [25] [33] [38] [40] [54]
Manage cache or scratchpad memory	[20] [26] [30] [36] [37]
Scheduling	[25] [30] [38] [39] [40] [51] [55]

Surveyed research on Memory Management (MM) techniques was summarised with respect to the functionality objectives (Table 1), the memory methodology (Table 2) and the evaluated outcome (Table 3). The overall taxonomy is presented in the Figure 13.

TABLE 2. Survey of memory management techniques in HPC.

OS modification or/and extension	[21] [24] [25] [28] [29] [30] [31] [32] [36] [37] [40] [52]
Middleware (runtime systems or/and platform dependent libraries)	[20] [23] [26] [27] [33] [34] [36] [38] [40] [45] [51] [53] [54] [55] [56] [58] [59]
Hardware	[22] [33] [34] [36] [40] [46] [54] [55] [56]
Dynamic allocation	[27] [28] [31] [36] [37] [38] [46]
Static allocation	[40] [58]
Multi-pattern allocation	[20] [30]
NUMA architecture	[21] [23] [24] [25] [27] [29] [30] [31] [32] [36] [37] [38] [39] [40] [44] [59]
Unified memory access	[22] [52] [53] [54] [56] [58]
Hybrid memory access	[26] [34] [39] [44]
Cache	[20] [36] [37] [39] [44] [51]

TABLE 3. Evaluated outcome.

performance/ runtime/ speed-up	[20] [21] [22] [23] [24] [25] [30] [32] [33] [36] [38] [42] [46] [51] [52] [54] [55] [56] [58]
energy/ power	[20] [22] [34] [45] [51] [54] [55] [56] [58]
bandwidth/ latency	[24] [27] [34] [37] [41] [44]
memory utilisation/ traffic	[20] [52] [53] [59]

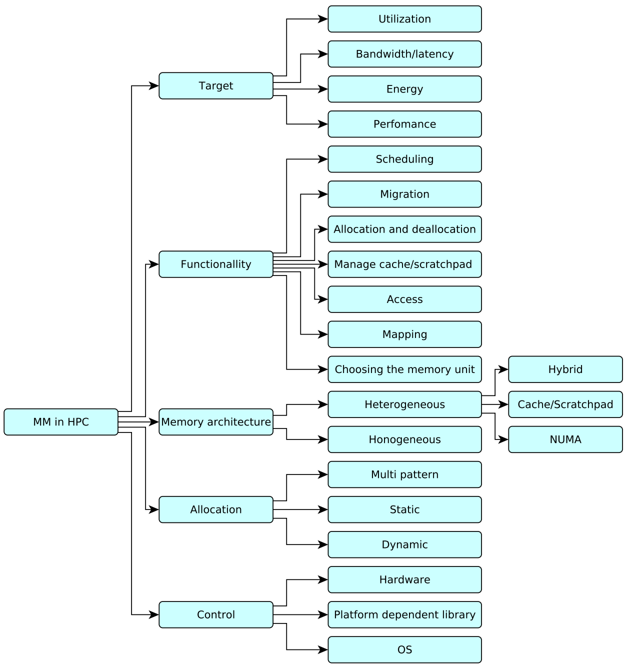

FIGURE 13. Memory management in HPC taxonomy.

V. MEMORY MANAGEMENT TECHNIQUES IN CLOUD COMPUTING

The growth of the scientific workflows and of the datasets and the complexity of computational multi-parametric models is pushing us to use the Cloud solutions. This requires

MM adaptation for the efficient use of the novel workload characteristics and resources. In contrast to the traditional HPC architectures, the main resources provided by clouds are VMs and storage areas. To perform an HPC application in the Cloud, the execution of each task has to be scheduled to an appropriate VMs.

Surveyed studies have a combination of different objectives and used on various stage of resource management. The brief overview of the survived memory management techniques is presented in Figure 14.

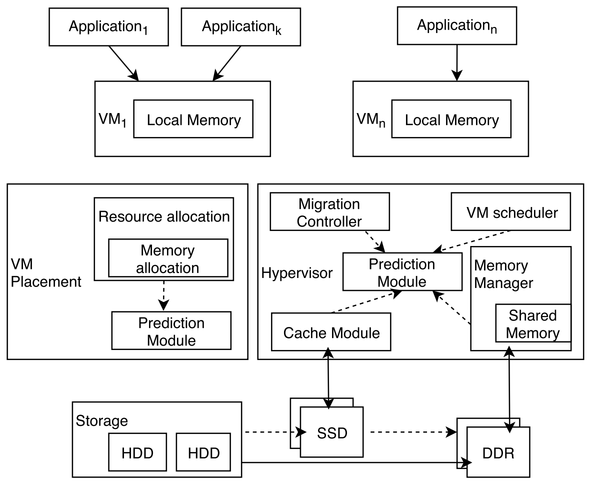

FIGURE 14. Brief overview of the memory optimisation points in cloud.

A. SCHEDULING

Generally, scheduling is a decision-making process that allows to distribute available resources among a number of tasks by determining the order of execution. Scheduling is a NP-complete problem, even for the simple scenarios. Moreover scheduling in Cloud architectures is a bit more complicated due to the heterogeneity of VMs and negative impact of the transfers of the huge data between VMs.

Recent studies on scheduling in the Cloud were analysed from the MM techniques point of view. In [60] data distribution and task distribution are defined as dependent problems, and they are analysed together for efficient scheduling. Data communication requirements between VMs and storage nodes in addition to the computing and memory resource demands are taken into account in the scheduling approach proposed in [61]. Layers of the data communication requirements is presented in Figure 15. Scheduling interdependent tasks and intermediate data within a system with four level cache/memory hierarchies introduced in [62] aims to optimise performance and energy consumption. The scheduling approach introduced in [63] uses profiles which are created on the machine learning models based on the workflow profiling information along with resources that were used during execution, such as CPU, memory, a number of compute nodes, input file size. Virtual resources scheduling techniques described in [64] use an energy consumption model to perform VMs migration from the overloaded hosts

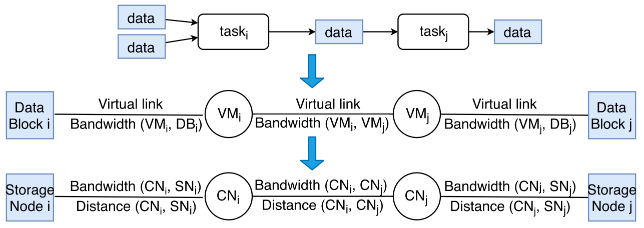
data
data
task$_i$
data
task$_j$
data
Data Block i
Virtual link
Bandwidth (VM$_i$, DB$_i$)
VM$_i$
Virtual link
Bandwidth (VM$_i$, VM$_j$)
VM$_j$
Virtual link
Bandwidth (VM$_j$, DB$_j$)
Data Block j
Storage Node i
Bandwidth (CN$_i$, SN$_i$)
Distance (CN$_i$, SN$_i$)
CN$_i$
Bandwidth (CN$_i$, CN$_j$)
Distance (CN$_i$, CN$_j$)
CN$_j$
Bandwidth (CN$_j$, SN$_j$)
Distance (CN$_j$, SN$_j$)
Storage Node j

FIGURE 15. Data communication requirements (CN - computation node).

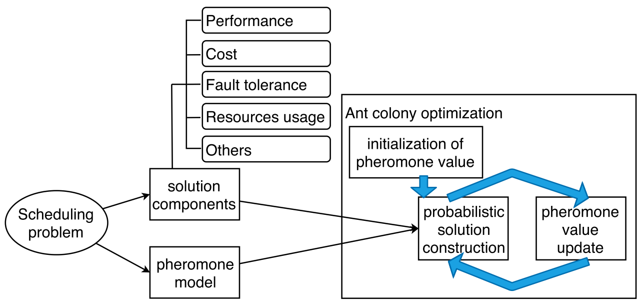

FIGURE 16. Ant colony optimisation in scheduling.

to other hosts with lower resource utilisation or to idled hosts. The responsibility of the scheduler in the VMs migration is also evident in the approach presented in [65]. Prediction model instructions are used to decide whether migration is needed, and the destination is determined using an improved ant colony algorithm. A multi-objective optimisation scheduling method (PBACO) proposed in [66] is also based on the ant colony algorithm. A resource cost model includes two parts of the CPU and the memory, such as the base cost and the cost associated with the transmission. The proposed approach achieves the multi-objective optimisation for the optimal span, deadline, resource utilisation and user costs. This work focuses on two types of resources, CPU and disk I/O bandwidth, but the approach allows to scale the number of resources. The ant colony optimisation method, along with VM dynamic forecast scheduling, was also used in [67]. Ant colony optimisation with slave ants for scheduling tasks in cloud computing environments described in [68] aims at maximising the utilisation of cloud servers. The proposed approach uses stored information about CPU utilisation and memory usage provided by the resource manager. Diversification and reinforcement strategies with slave ants allow avoiding long paths whose pheromones are wrongly accumulated by leading ants. Here, to evaluate the pheromone level, solutions generated by the first fit decreasing (FFD) algorithm are presented as the function of the memory loss of the particular solution. A general scheme of the ant colony optimisation in cloud scheduling is shown in Figure 16. Scheduling for Distributed Stream Processing Systems (DSPS) in [69] was considered as two interdependent parts, namely threads and resources allocation and mapping threads to resources. The allocation part is in charge of identifying the number of threads per task and the number of resource slots allocated. The mapping part aims at assigning these task to the specific resource slots and to meet the requirements. A dynamic resource scheduling model based on the Max Min Fuzzy Inference is designed in [70] for executing different types of cloud user requests. Reference [71] shows that task scheduling heuristic with the memory consumption balancing has a positive effect on the number of machines required to complete a multi-frontal solver at a given time. They propose the task agglomeration heuristics where the amount of memory required to perform a given subtree of the element partition tree is estimated. If this falls below a defined threshold, then the subtree is agglomerated into a single task. The aim of a task scheduling policy based on the memory accesses distribution in the time proposed in [72] is to prevent performance degradation. This approach uses per-task memory access monitoring for memory access classification in order to schedule a latency-sensitive task before bandwidth-sensitive tasks. A genetic-based optimisation algorithm presented in [73] uses a genetic-based algorithm for the task-core scheduling along with Phase-change memory (PCM) configuration. Aiming at green cloud computing, this approach provides a PCM configuration that balanced the PCM memory performance as well as the efficiency.

Scheduling has been traditionally considered as an NP-complete optimisation problem with the execution time as a single objective of interest. The complexity of the current heterogeneous cloud systems, brought additional objectives, such as energy consumption, operational costs, throughput, makespan, fault tolerance, reliability, security, predictability, elasticity. The challenge is that these objectives are often in conflict with each other, requiring new complex algorithms to obtain an optimal solution with regards to all conflicting objectives. Heuristic and metaheuristic algorithms for task scheduling in Cloud reduce the solution search space, consequently, significantly improve efficiency. However, these algorithms are time-consuming and, in some cases, return local optimum solution due to imbalance between local and global search. For instance, the common used ACO algorithms still have the slow convergence speed and are easy to fall into local optimum value. Besides, the quality of scheduling produced by these algorithms still tightly depends on the problem size and the number of objectives to be optimised. Therefore, it is necessary to further and deeply study and improve heuristic and metaheuristic algorithms along with algorithms inspired by nature with better optimisation performance. The majority of the scheduling algorithms focused on the scheduling problem of jobs/tasks, but ignore the deployment of input data. However, the management of memory resources and data movement are becoming more critical, as the applications are becoming data-intensive. The scheduling algorithms with data placement strategy will improve the data locality.

B. RESOURCE AND USER ALLOCATION

The complexity of provisioning algorithms for homogeneous resource allocation is already high and the computational time is long [74]. Further considerations on deeply heterogeneous resources add additional dimensions to the computation, which will significantly increase the complexity.

The stage of resources allocation in [64] is considered to be responsible for allocating VMs to the hosts represented as three dimensional cuboid with CPU, network bandwidth and memory capacity of the host as axes. Heuristics based on multi-dimensional vector bin packing problem for virtual resource allocation was proposed. The allocation model that deals with dispatching a new task to a VM, and with allocating appropriate resources for the task is described in [65]. Authors proposed to use the improved ant colony algorithm to allocate running VM to those hosts that have a large capacity of resources while running on low operational power. This approach allows to shut down redundant hosts, thereby reducing energy consumption and improving resource utilisation. In this study, two types of resources, CPU and disk I/O bandwidth, were considered. However, it has been illustrated that the number of resources can be increased. In article [69] efficient resources allocation and mapping for DSPSs over nodes or VMs of commodity clusters or Clouds is addressed. For simplicity, the analysis, that is limited to CPU and memory resources, can be scaled. The main idea is to preferably map the full available slot for a bundle of tasks, rather than allocate a single resource for each task. For all partial bundle of unmapped tasks, the best-fit slot is selected, such as the one whose sum of available CPUs and memory is the smallest, albeit sufficient, to satisfy resource requests. The resource provisioning through Markov Decision model proposed in [70] only assigns the value as zero or one with respect to the availability of resources and requirements of resources. The input variables are the resource requirements (i.e. bandwidth, memory and CPU) and the resource availability of VMs, respectively. The schedules algorithm selects particular resources that are allocated to the cloud users from the corresponding VM based on fuzzy theory that was mentioned above. Resources allocation algorithm presented in [75] starts by partitioning the workflow into several task groups. Then the resource allocation step divides VMs between determined partitions such that the resultant allocation is envy-free Pareto-efficient by employing a weighted version of the max-min fairness allocation policy. The approach proposed in [76] is a three-stage methodology that utilises a fully connected Neural Network to classify a given application, calculates the performance-cost sensitivity of application and solves a bounded knapsack problem using dynamic programming and finding a configuration that maximises the performance per cost ratio. An autonomic resource controller with a coordination technique based on the fuzzy control approach is presented in [77]. The controller dynamically adjusts the right amount of CPU and memory required to meet the response time using the estimated coefficient that determines the degree of influence of CPU, memory, or both on the application performance change. An approach described in [78] aims at increasing the efficiency of memory utilisation through an efficient user allocation mechanism. The profile-oriented clustering algorithm (POCA) positioned users with similar profiles into the same application servers to share and save memory space by reusing the application programs.

The goal of resources provisioning is to use the minimum number of physical machines (PMs) to house a set of given VMs. Classical bin-packing algorithms for resource provisioning were modified to apply in the cloud environment to deal with the multi-dimensional resources (such as CPU, memory), the different bin sizes (to represent the heterogeneity) and multi-objective optimisation functions (such as energy, utilisation, reliability). Particularly, optimising all resources at the same time may result in exceeding a reasonable time for solving resource provisioning. Only for memory allocation, there can be considered such objectives as memory usage per instance (average and variances) and on the hypervisor (average, variances, the variance of situated virtual instances and their intersections). Several studies propose heuristics to address the pure optimisation problem for the integrated model of the considered resources. Most of the proposed algorithms allow finding the optimal resource allocation on the fly, which does not guarantee the performance and efficiency of the cloud in the long-term period. Recent studies use machine learning algorithms to provide efficient resources allocation for a long-term period by predicting future resource utilisation or classifying memory behaviour. The general scheme of the use of the classification is shown in Figure 17.

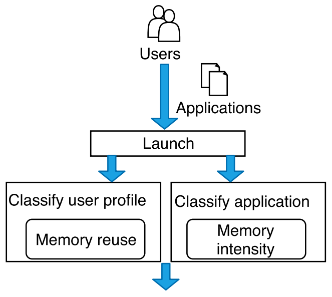

FIGURE 17. Usage of the classification in the memory provisioning.

C. MEMORY ALLOCATION

Some studies focus on the memory allocation problem as a separate part of resource allocation to highlight the memory allocation challenges. Appropriately to the memory requests behaviour, memory allocation in Cloud can be divided on the online and offline memory allocation. When the memory requests are known in advance, the memory allocation strategy can be configured optimally before computation. When memory requests arrive randomly, online memory allocation strategies are used. The well-known offline resource allocation solver is a problem of packing bins, where a finite number of objects of different volumes must be packed into a finite number of bins of fixed capacity. However, both offline and online memory allocations in Cloud require the optimal heuristic algorithms to effective manage the memory in an acceptable time.

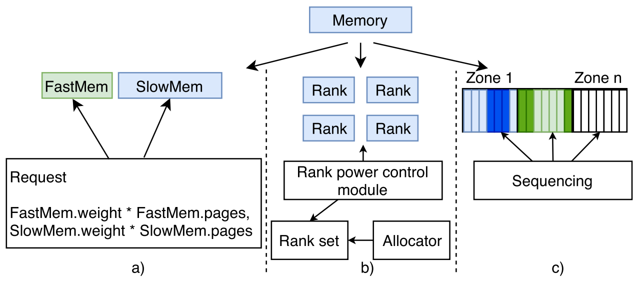

FIGURE 18. General scheme of the memory allocation in Cloud.

In order to manage memory heterogeneity in virtualised systems and to address max-min fairness across multiple resources, HeteroOS [79] uses extended Dominant Resource Fairness (DRF) algorithm. However, FastMem capacity is small, so that, according to this algorithm, most VMs always have SlowMem as the dominant resource. To deal with this limitation, weights for calculating the dominant memory, shown in Figure 18(a), were proposed to use. The main algorithm supporting Cost-Aware Heterogeneous Cloud Memory Model [80] is Dynamic Data Allocation Advance Algorithm that uses genetic programming to determine the data allocations on the memories. This approach focuses on a set of critical factors affecting the performance of cloud memory, such as communication costs, operating costs for moving data, energy performance and time constraints. Two heuristic algorithms of allocating memory for VMs are proposed in [81]. This approach, shown in Figure 18(b), provides the strategy to select the memory ranks for reducing the number of ranks occupied by VM and for minimising the global rank set for all VMs. An algorithm presented in [82] divides the memory into multiple zones, shown in Figure 18(c), where a subgroup of relative request sizes compete in reverse order. In addition to this algorithm, [83] proposes a cloud computing service that provides a continuous memory size. A mathematical and an approximation model for different request size distributions, traffic intensities, memory sizes, and granularity values allow calculating optimal quantised sizes.

D. MEMORY AGGREGATION AND DISAGGREGATION

Modern system architectures aim to improve resource isolation approach, allowing servers to share their resources through the memory hierarchy. An extension for the hypervisor to share the memory capacity of the nodes across the computing infrastructure called Globally Visible Tmem (GV-Tmem) was proposed in [84]. A software architecture allows to aggregate memory across nodes using Xen's Transcendent Memory (Tmem) [85]. The general scheme oh the memory aggregation is shown in Figure 19. This mechanism improves the utilisation of the node's memory but also increases pressure on it, especially if the VMs have varying demands for memory.

Due to an increase of in-memory data processing, to perform memory-intensive workloads cloud solutions have to

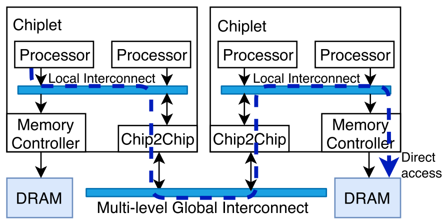

FIGURE 19. General scheme of the memory aggregation in Cloud.

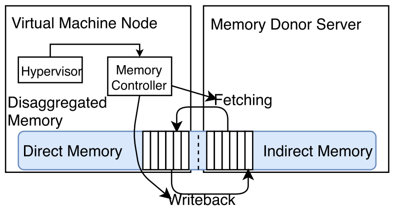

FIGURE 20. General scheme of the memory disaggregation in cloud.

have nodes with large memory. The efficiency of this kind of nodes may be much lower with respect to common hosts in terms of cost and power. However, memory disaggregation allows more memory than locally supported becomes available since more servers are being used for distributed memory. Reference [86] proposes an integrated hypervisor-based design for disaggregated memory. Instead of relying on the regular swap system designed for slow disks, the disaggregated memory support is directly added to the memory management in the Kernel-based Virtual Machine hypervisor. The general scheme oh the memory disaggregation is presented in Figure 19. The memory disaggregation can provide a cost-effective way to scale memory capacity while increasing the flexibility of resource allocation and energy efficiency for cloud providers. However, depending on the distribution of fast direct and slow indirect memory, application performance can vary significantly. This unpredictability caused by performance mismatch could potentially prevent the usage of disaggregated memory on cloud systems.

E. MEMORY PAGES MANAGEMENT

In virtualised systems, during the VM boot, the boot manager initialises a guest-VM's memory and adds the VM pages under the control of the OS allocator [87]. Often a situation arises when one virtual machine has free memory, while other virtual machines do not have enough memory. The workload of several VM applications changes over time, so memory requirements may suddenly decrease or increase dramatically. However, even if hypervisors have a large number of free memory pages, they cannot reallocate memory to a guest who needs urgent memory to use. When the memory requirements are less than the amount of free memory, the guest will necessarily start the guest paging mechanism, which will lead to a decrease in performance, an imbalance in memory usage caused by a waste of resources. Hypervisors typically already support dynamic allocation of a processor pool between different VM. Therefore, many existing studies focus on exploring how to dynamically adjust memory allocation to meet the changing demand of a VM.

The polymorphic memory management proposed in [72] performs a periodic procedure of hardware-assisted page monitoring, followed by OS intervention for migration and partitioning. The key assumption here is that the page access pattern during one period is similar to the next pattern during the next period. HeteroOS [79] extends the boot allocator to initialise a NUMA node and its related data structures for each memory type. In order to achieve smart memory placement and reduce page migrations, several extensions to the OS allocators were implemented. For instance, the extension allows to use just one partition zone for FastMem nodes and to use only the HeteroOS page allocator for FastMem pages to avoid other general-purpose allocation by the default OS allocators. Reference [88] proposed three memory management policies and a combination of ballooning and swapping intended for working virtual memory. The policies are handled by using guard functions of transition in stochastic reward net (SRN) hierarchical model of memory virtualisation. Memory availability with failure related behaviour is modelled and analysed as an upper-level model. The lower-level model captures the performance of the system, especially several memory utilisation measures. Coordinate memory deduplication and partition (CMDP) approach introduced in [89] aims at reducing memory requirement and interference simultaneously. CMDP allows mapping hypervisor, VMs and applications running on VMs onto different memory banks, thus eliminating interference. Due to pages that belonged to the memory banks of VMs, especially pages with the same access behaviour, more likely have the same content, the page comparisons in the proposed behaviour-based memory deduplication approach are restricted into the same classification to reduce the overhead of futile comparison. In order to adjust memory size, a quick approximation algorithm was proposed in [90], in contrast to the brute force search. This algorithm recursively for all VMs reduces the memory allocation of one VM, increases the memory allocation of other VM respectively and calculates the total page misses until the total page misses reach a local minimum.

Balloon driver [91] is a widely used approach to enable memory resource overcommitment. However, the ballooning driver does not move memory fast enough to satisfy the memory requirement, since VM memory swapping could be triggered while the balloon driver is moving the memory across guest VMs. Another challenge is the double paging problem, which is caused by the inefficient swapping activities between the guest OSs and the host OS. The unnecessary disk I/O caused by double paging problem could seriously affect the performance of guest VMs.

F. SHARED MEMORY

Recent development has been dedicated to improving communication efficiency between VMs using shared-memory channels. Shared memory is used in cloud solutions to reduce the overhead of data access and to increase resource sharing

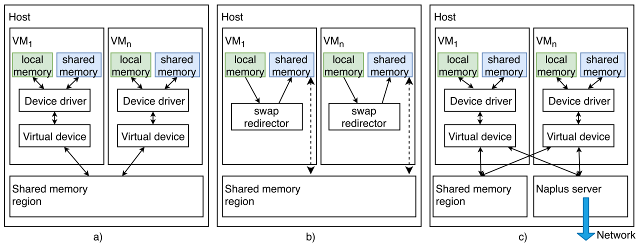

FIGURE 21. General scheme of the shared memory implementation in Cloud (a - VM communication; b - inter-domain communication; c - VM memory swapping).

between different VMs or between VMs and a host. Several techniques have been proposed that differ in terms of where and how the shared-memory channel is established. Host allocated dynamic shared memory is presented in Figure 21(a). The dynamically managed shared memory regions allow each VM to access the shared memory based on its workload requirements. In [92], a dynamic shared memory management framework was presented. The proposed approach allows multiple VMs to access shared memory resources in accordance with their demands dynamically, thereby improving VM communication efficiency and VM memory swapping efficiency. A general scheme of the VM communication via shared memory swapping is shown in Figure 21(b). A mechanism that allows VMs distributed on different physical hosts to share their local memory was presented in [93]. Naplus was implemented in a dual-host system as a software DSM between VMs by multiplexing the Nahanni [94] device in each host. A general scheme of the inter-domain communication via shared memory is shown in Figure 21(c). It manages the local and remote memory accesses via a simplified lazy consistency memory model.

The performance of the shared-memory mechanism for the VM communication depends on a number of factors, such as an implementation layer in the software stack. The implementation layer impacts on the programming transparency, hypervisor transparency and performance [95]. Implementing shared-memory for VM communication at the libraries and system calls layer leads to the need for application modifications. The shared-memory implementation above transport layer will result in missing some critical TCP/IP features, such as reliability. Finally, the shared-memory implementation below the IP layer provides high transparency and reliability but may incur high overhead.

G. CACHE

To address the storage bottleneck and to improve VM performance for cloud computing systems, the host-side flash caching was suggested to use. Recent studies have focused on solutions for reducing the capacity and endurance limitations of the flash caching systems. A general scheme of the cache organisation in Cloud is shown in Figure 22.

CloudCache [96] is an on-demand cache management solution to meet VM cache demands and minimise cache wear-out. To capture the data with a good temporal locality,

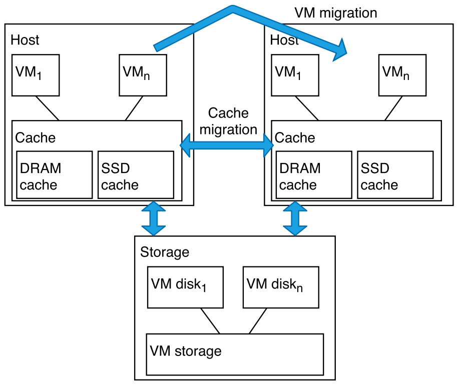

FIGURE 22. General scheme of the cache organisation in cloud.

it employs a new cache demand model, Reuse Working Set (RWS), which is defined as a set of separate (address) data blocks that the workload reused at least N times in a period of time. In [97] host-side SSD caching solution Centaur was presented. This research is mainly focused on the cache partitioning as the key mechanism that eliminates cache wastage. Centaur implements dynamically partitioned per-VM caches with per-partition local replacement to provide lower cache miss rate, better performance isolation and performance control for VM workloads. Partitioning employs online miss rate curves (MRC) construction combined with periodic partitioning and data movement to allocate cache space to individual workloads efficiently. The approach presented in [98] consists in sampling the workload and in running them in separate sample caches with different workload classifiers (thresholds) to find the optimal threshold. Based on the estimation of the latency over a lifetime vs threshold curve, the optimal threshold could be selected. A Three-level I/O Cache Architecture (TICA), which aims to improve the performance and power consumption of SSD-based I/O caching while having minimal impact on the endurance, was presented in [99]. TICA employs Read-Optimised SSD (RO-SSD), Write-Optimised SSD (WO-SSD), and DRAM as three levels of cache memory. To balance performance and endurance, a state-machine model that selects one of the existing policies, such as Endurance-Friendly (TICA-EF) for optimising endurance and Write to Endurance Disk (TICA-WED) for improving performance, was implemented.

Effective caching schemes should not focus only on estimating the VMs cache size, but also take into account other critical parameters, such as the write policy, the request type, and the workload reuse, which significantly affects the performance and endurance of the SSD cache. Intelligent data prefetching based on access patterns can hide the latencies of the disk access and data transfer by moving the data to DRAM before it is requested. However, the effectiveness of such prefetching strongly depends on the ability to recognise data access patterns and identify corresponding data to be prefetched.

H. PREDICTION

It is typical for cloud environment meeting unexpected loads, which may lead to overload and performance degradation. The accurate prediction of the future workload enables the cloud providers to provide effective resource management, including memory allocation.

In [65] a prediction model based on fractal mathematics was used for predicting the usage of CPU and disk I/O bandwidth load. Moreover, there are studies with memory utilisation prediction. In [67] an algorithm to predict the memory usage on the PMs to prevent overloading of PMs was used. Autoregression to predict the memory consumption per VM is used when the actual memory consumption in a PM exceeds the preset percentage. The moving average-based resource requirement forecasting algorithm presented in [70] is used for the forecast of resource requirement including bandwidth, memory and CPU. The advantage of using this prediction model is to reduce the total waiting time between the forecasting value (resource required) and the actual value (resource availability), thus improving the resource scheduling efficiency. Predictors in [90] are used to estimate the amount of memory available for reclaiming without a notable performance drop, and additional memory required for reducing the VM paging penalty. The memory growth predictors were implemented in two ways: the OS statistics based and the LRU-based ones. CloudCache proposed in [96] uses classic exponential smoothing and double exponential smoothing methods to predict the cache demand of the next time window. In [100] Gaussian process regression (GPR) is employed as a probability framework. How the prediction works depends on the monitoring statistics which reflect the historical utilisation of the cluster facilities. The memory and CPU utilisation prediction results are presented. The prediction tool proposed in [101] relies on a combination of machine learning methods, such as Support Vector Machines (SVMs) with Linear and Radial Basis Function (RBF) kernels, Random Forests, Multilayer Perceptrons (MLPs), and k Nearest Neighbors (kNN), to make predictions of job memory usage. A wide range of features collected by the proposed tool is stored in a separate embedded database. The label used in machine learning methods is the maximum memory used discretisation on bins of size 512MiB. Experiments based on job traces from two production environments show that there is no single machine learning method that produces the best predictions. The proposed tool chooses the most promising ones at a given time. The approach presented in [102] considers both statistical models and the hint extracted from the application to accurately predict when the memory will error out. Multiple fitting models (e.g. polynomial, exponential) along with applications' own traits extracted from profiling a small portion of execution are considered as inputs. They are also employed to construct a higher-level of the model that is used in migration decision making. Pacer's techniques [103] uses analytic models for progress predictions. The prediction of the migration time includes an evaluation of the total number of dirty pages/blocks during migration that is not known before the migration completes. Memory dirty rate prediction is based on the analysis of the snapshot of the dirty bitmap of memory pages. This leads to a large tracking overhead, so instead of continuous monitoring, a sample-based algorithm was proposed. Utilisation prediction-aware best fit decreasing approach described in [104] uses a k-nearest neighbour regression-based model for the prediction of the future utilisation of resources, including both CPU and memory.

Summarising the aforementioned forecasting methods, it can be noted that the memory prediction is essential in managing cloud resources and can be considered for several objectives. The taxonomy of the memory prediction objectives is presented in Figure 23. An accurate prediction of VM memory requests is still a challenging problem, especially in a changing environment. Because the hypervisor lacks the knowledge of VM memory access pattern, the virtualisation context makes the prediction even harder. Besides, access patterns can change at runtime, the workflow may include third-party libraries where the access pattern is unknown to the user, and the resources may be shared between multiple application workflows. Therefore, the current trend in the resource management is to use machine learning approaches for more accurate and long-term workload prediction with an attempt to avoid the enormous performance overhead.

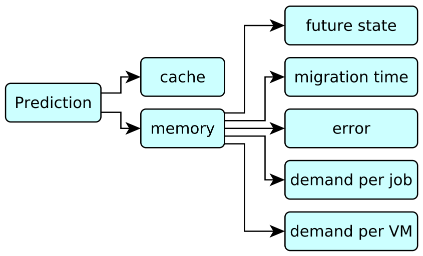

FIGURE 23. Memory prediction taxonomy in cloud.

I. VM PLACEMENT

Recent research investigates algorithms for allocating VMs to PMs in clouds by addressing distinct problems, such as energy efficiency or the tradeoff between reaching service-level agreements and costs.

The VM placement solution presented in [72] tries to guarantee a certain level of memory access performance to each VM, especially under heavy contention among VMs. To place guest VMs with different memory access latencies, the hypervisor uses a proposed heuristic equation that measures data access latency, because the hardware Performance Monitoring Units (PMU) cannot directly control the exact value. An approach that changes the VM-oriented placement to Type-oriented VM placement is presented in [105]. As it was noticed, the cloud platform provides limited types of VMs in the actual environment. Thus, the complexity of the Type-oriented VM placement problem does not increase with the size of the data centre. In order to fully utilise servers' resource, the mechanism places VMs to servers according to the patterns which result in high resource utilisation. The Krill Herd (KH) algorithm for solving optimal VM allocation problem is presented in [106]. This algorithm uses a fitness function which has two main factors: utilisation rate, including primary memory utilisation and Quality of Experience (QoE). Three major actions used to place the time-dependent position of individual VM were adapted to the KH terminology. Movement induced by other krill individuals is analogous to the effect of one VM on others; Foraging motion is analogous to the effect on the VM caused by the maximum QoE in the solution; and Random diffusion is analogous to the random VM placement. The most appropriate location for VM placement is selected based on the Lagrangian model. The number of VMs that can be accommodated on a physical host is discussed in [107]. Capacity management is responsible for the availability of the required computational resources to allow cloud providers to function efficiently. To avoid low resource utilisation, physical servers are frequently utilised at 80% or more. According to the proposed approach based on the "N+1" model, if there is an unexpected failure, then the failed host machine's load will be managed by the additional host. Several utilisation policies to ensure comprehensive capacity that is adequate to hold the dynamic workload, while maintaining the performance level, were proposed. Cloud Virtual machine Automatic Memory Procurement (CloudVAMP) introduced in [108] is a memory oversubscription framework that can be integrated into an on-premises Cloud Management Platform. This framework automatically controls the VMs and dynamically adjusts the allocated memory to adapt to the current memory requirements of running applications. CloudVAMP allows using currently unused memory from the deployed VMs that leads to the temporary oversubscription on the memory resources. The oversubscription, along with the usage of live migration, increases VM-per-host consolidation ratio with reduced impact on running applications. Reference [109] suggested combining the VM selection and VM placement in a single optimisation algorithm. The problem statement considers software components that are deployed in VMs, which in turn are deployed in PMs. The size of a component encodes its resource requirements along with multiple resource types as an n-dimensional vector, e.g., 2-dimensional if CPU and memory are considered. Utilisation prediction-aware best fit decreasing approach proposed in [104] assigns a VM to a host according to current and future resource requirements.

Available memory often limits the number of VMs supported by a host. Users tend to overestimate their VM memory requirements using the worst-case scenarios that may only be required for short periods. In addition, Cloud Providers typically offer only a few predetermined VM sizes. Consequently, the technique of over-committing VMs, where more resources are allocated than physically exist, has become common practice. While this approach may make better use of resources, it can also lead to a dramatic loss in performance and a risk of SLA violations. The efficiency of the VM placement is determined by packing efficiency and speed that have a negative correlation between them. Besides, current methods rarely take into account the stability of VM placement to improve efficiency. As shown in Figure 24, due to the variation of the memory needs and variable nature of the access patterns, current optimal VMs placement may not be suitable for future workloads. The state-of-the-art metaheuristics and greedy heuristics for solving placement problem along with workload prediction allow solving this problem.

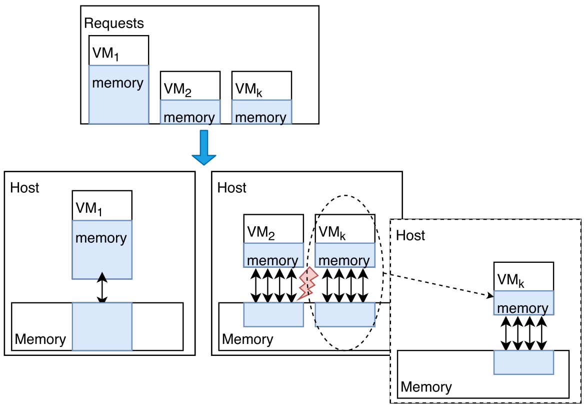

FIGURE 24. General scheme of the VM placement.

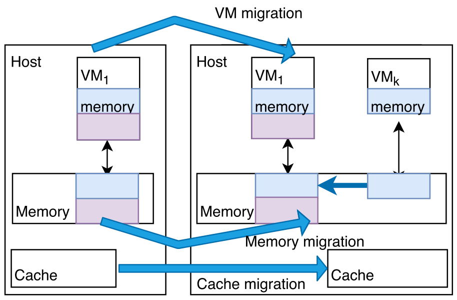

FIGURE 25. General scheme of the VM migration.

J. VM MIGRATION

Emphasis on memory is also present in works related to VM migration. A general scheme of the VM migration is shown in Figure 25. Along with the problems with overloading and underloading PM detection where the memory utilisation rate can be considered, some approaches for the migrating VM selection analyse the memory similarity. By using the content similarity among the memory of VMs, the time and the network traffic during VM migration can be reduced.

To minimise the amount of transferred memory data, research presented in [110] is based on Memory Buddies, a memory sharing-aware placement system for VM. Content similarity checking method with O(1) time complexity is used to evaluate the page sharing potential between VMs. In [64] algorithm for selecting new hosts for the VMs to be migrated is the same with the VM placement strategy. Multi-dimensional overload detection of nodes is used for detecting the overload nodes. It analyses the weight and current use of the CPU, memory, and network bandwidth. The approach presented in [100] selects the node to be migrated as the node with the lowest CPU and memory utilisation. The node with the highest estimation of utilisation and the blocking rate, which is evaluated based on the queuing theory, is determined by the power management component as the destination for VM migration. In the resource-aware VM migration technique proposed in [111] migration controller maintains the cluster table with all servers clustered into eight clusters based on the CPU, memory utilisation, and job arrival rate. The overloaded servers are identified using preference value. The VMs from the server with high preference values are migrated to the destination server using a resource-aware virtual machine migration technique. In traditional VM migrations, the whole data used to be transferred from source host to the destination host. In [112], a technique that reduces data to be sent back to the source host was proposed. According to the reusable memory approach, the memory image on the source host is not deleted after migration. Once the VM migrates back to the original host, only processed data and not the whole data are sent back to the source host. In order to reduce the size of memory image that is stored on the source host, unnecessary data are deleted according to a probability factor. To handle cache overload situations, CloudCache [96] provides a live VM migration with its cached data to meet the cache demands of the VMs. The approach presented in [102] assumes the application is migrated between different VMs without physical data movements aiming at reducing the overall monetary cost in face of memory depletion. Introducing swap for highly memory-intensive application on a memory-restrained instance was recognised as the cause of the significant application slowdown and, consequently, of the absence of lower per-hour prices. Based on the lightweight prediction methodology of the memory usage, application migrates only when the predicted memory usage exceeds the physical allocation. It migrates to the least expensive instance type that has a memory capacity larger than the current running instance. This protocol is optimistic and it shows that doubling the memory capacity could satisfy the application's memory requirement. Pacer's techniques [103] are based on lightweight runtime system and workload measurements, analytic models for progress predictions, and online adaptation to achieve user-defined migration objectives. The progress prediction is used to realise the best-effort migration time control and to coordinated migration of VMs. Utilisation prediction-aware best fit decreasing approach presented in [104] allows to migrate VMs from physical hosts that are currently overloaded or predicted to become overloaded in the near future.

The live migration advantage is that the system is not turned off; consequently, there is no downtime. Nevertheless, live migration is time-consuming and challenging to perform. On the other hand, offline migration has an advantage in speed, but the host has to be turned off during migration. Whenever a VM is prepared to move from one host to another, its current state is stored in the source host for future transfer and use on the target host. In order to reduce migration time and network overhead, resent studies are focused on reducing the memory image size and cache migration policies. However, the memory migration cost, in many cases, exceeds the gain received from the migration.

Surveyed research on MM in Cloud was summarised with respect to the evaluated outcome in Table 4 and the previously mentioned techniques in Table 5.

TABLE 4. Evaluated outcome.

makespan/ waiting time	[60] [61] [62] [63] [64] [65] [66] [68] [70] [71] [72] [73] [75] [76] [79] [80] [84] [86] [89] [90] [92] [93] [99] [102] [103] [105] [111]
resource utilisation	[61] [63] [64] [65] [66] [67] [69] [71] [73] [75] [77] [81] [82] [88] [90] [92] [96] [97] [100] [105] [106] [107] [108] [112]
energy/ power	[62] [64] [65] [72] [81] [99] [100] [104] [109] [110]
cost	[61] [63] [66] [75] [76] [82] [83] [102] [106] [109]

VI. OPEN PROBLEMS

Memory management systems have been a topic of interest in the HPC and Cloud Computing domain for the last decade. The variety of served workloads causes the difference in memory optimisation techniques in HPC and Clouds environments. While for HPC the most inherent target scenario is a batch execution of a specific high-performance application with input data of the same size but different content, for Cloud Computing the workload has a higher level of variability and performance goes along with economic efficiency. In this regard, it is quite reasonable that memory optimisation techniques are different in:

Attention: HPC pays attention particularly to acceleration due to the hardware modernisation of memory device and cache. In Clouds, on the other hand, the emphasis is placed on optimising resource provisioning and scheduling.

Using statistical information: Research related to the HPC domain is aimed at investigating the memory access patterns of the application. For Cloud Computing, instead, analysis and prediction of the workload or resource utilisation are typical.

Memory migration: Memory migration in HPC is becoming increasingly important due to memory heterogeneity and memory locality. Memory migration in Clouds, in additional to the memory locality, takes into account the memory similarity in order to reduce the amount of transfer. However, since memory migration is a relatively expensive operation, it always requires a cost estimate.

Cache: In addition to improving traditional cache management in HPC, a hybrid memory, that is, a memory that can be configured as a cache or as a fat memory depending on the requirements, acquire value. Cloud cache research mostly regards the per-VM caches partitioning.

TABLE 5. Survey of memory management in cloud.

scheduling	[60]	hybrid evolutionary algorithm
	[61]	greedy heuristic
	[62]	b-level based algorithm, task and data co-scheduling, b-level task stealing
	[63]	kNN as the model learning algorithm for profiling
	[64]	multi-dimensional power-aware
	[65]	improved ant colony algorithm
	[66]	multi-objective optimisation based on the ant colony
	[67]	ant colony optimisation & VM dynamic forecast
	[68]	adaptations of diversification and reinforcement strategies with slave ants
	[69]	model-based allocation
	[70]	max min fuzzy inference
	[71]	agglomeration
	[72]	task scheduling policy based on the memory accesses diffusing through time
	[73]	genetic-based algorithm
resource and user allocation	[64]	multi-dimensional vector bin packing problem
	[65]	improved ant colony algorithm
	[69]	slot aware mapping
	[70]	markov decision model
	[75]	weighted version of the max-min fairness allocation policy
	[76]	fully connected Neural Network and dynamic programming to solve a bounded knapsack problem
	[77]	fuzzy control
	[78]	profile-oriented clustering algorithm
memory allocation	[79]	extended Dominant resource fairness for addressing max-min fairness across multiple resources
	[80]	genetic programming
	[81]	reducing the number of ranks occupied by VM & minimising the global rank set for all VMs
	[82]	sequential fits and zoning
	[83]	continuous (any) memory request size
aggregated and disaggregated memory	[84]	monitoring based distributing global memory capacity
	[86]	heuristic algorithms based on the approximation algorithm
memory pages management	[72]	page access pattern
	[79]	OS extension
	[88]	stochastic reward net
	[89]	coordinate memory deduplication
	[90]	automatic memory resizing
shared memory	[92]	host side dynamic shared memory mechanism
	[93]	lock-based protocol to enable the DSM model between VMs on different PMs
cache	[96]	reuse working set demand model
	[97]	online miss rate curves construction combined with periodic partitioning and data movement
	[98]	adaptive method based on the workload sampling
	[99]	adaptive tree-level I/O cache architecture
prediction	[65]	fractal mathematics
	[67]	auto regression
	[70]	moving average
	[90]	OS statistics based and the LRU-based
	[96]	classic and double exponential smoothing methods
	[100]	gaussian process regression method
	[101]	SVM, random forests, multilayer perceptrons, k nearest neighbours
	[102]	multiple fitting models (polynomial, exponential)
	[103]	sampling-based analysis of the dirty bitmap of memory pages
	[104]	k-nearest neighbour regression based model
VM placement	[72]	VM placement policy considering NUMA architecture of target memory hierarchy
	[104]	utilisation prediction-aware best fit decreasing
	[105]	pattern based placement
	[106]	krill herd optimisation
	[107]	capacity management based on N+1 model
	[108]	memory oversubscription
	[109]	combining VM selection and VM placement in a single optimisation algorithm
VM migration	[64]	multi-dimensional host overloading detection
	[96]	live VM migration with cache balancing
	[100]	queuing theory
	[102]	forthcoming out of memory determines based on meta-models
	[103]	best-effort migration time control
	[104]	utilisation prediction-aware best fit decreasing
	[110]	memory buddies
	[111]	cluster table
	[112]	advanced memory reusing

Memory hardware. At the hardware level, the traditional memory enhancement methods focus on increasing the memory clock frequency or storage bus bandwidth. These techniques are approaching their physical limits [113]. The emerging 2.5D/3D stack memory technology gives new opportunities to break the memory wall. In particular, it is expected that the hybrid memory cube and the high bandwidth memory will be widely used in future HPC systems. Along with the new hardware, it is necessary to develop modern software support to address memory management challenges and complexities.

Resources provisioning and scheduling. Nowadays, most schedulers do not consider either NUMA architecture or general memory binding. The degree of memory participation in the provisioning and scheduling techniques can be divided in the following groups: **memory utilisation:** memory, CPU, etc. are considered as separate resources, the solver of the bin packing problem is applied for all resources independently; **memory power model:** a set of power models for CPU, memory, etc. takes into account resource specifications, then all power models are integrated to provide optimal or near-optimal solutions. **memory-processor interaction:** memory is described not only by the hardware characteristics but also by the interaction with the processor or set of processors or accelerators to execute a specific application. Most of the surveyed approaches use the first or second method.

Memory access patterns. MM analyses the memory access behaviour and uses the information obtained for the thread- and memory mapping to increase performance. It can be either offline (precompiled) or online (runtime). This approach is suggested for HPC because of the prevailing workload uniformity. However, this approach can be used to analyse the memory-intensive scientific workload in Clouds.

Workload prediction. Instead of analysing each application in Clouds, it is common to predict the workload or future resource utilisation. This approach helps to arrange VMs on physical nodes in an effective and/or energy-aware way. As in the case of memory access patterns, it can be extended to the HPC domain by taking into account the trade-off between prediction accuracy and computational complexity.

Memory migration. Despite the fact that migration is widely used in both HPC and Cloud environments, due to the high cost of migration, research in this area pays great attention to assessing the need for migration, choosing a destination for the transferred data and increasing the migration speed.

VII. CONCLUSION

The world is becoming aware of the need to reduce energy consumption. Power consumption and energy efficiency are crucial aspects in the design of new HPC solutions. Nowadays, it is obvious that power consumption will strongly limit future exascale supercomputers. To this end, in recent years, special attention has been paid to Green Computing, which generally aims at using energy in a beneficially and efficiently. Since the memory system is one of the most energy-hungry parts in the modern computing environment, in both HPC traditional systems and in Cloud solutions, it is necessary to pay attention to energy-efficient memory management. A naive reduction in energy consumption does not always lead to energy efficiency due to a corresponding increase in the execution time. Therefore, research studies on energy efficiency often focus more on achieving competitive performance. Increased performance, especially for memory-intensive HPC workload, is associated with efficient usage of the different memory types, as well as with a reduction of data transfers within the memory hierarchy and nodes.

Memory management software requires additional support for new hardware technologies, such as HBM, NVM, RRAM, SSD. In addition to hardware improvements, new policy features can be applied to memory. The implementation of new policy features will provide access to a wide range of future hardware capabilities, such as bandwidth and latency explicit control, sharing for inter-process, inter-node and accelerator communication, persistence and encryption. Platforms with heterogeneous memory hardware with a wide number of memory system policies applied to that hardware generate an enormous solution search space for resource management techniques. Each level of the application software stack may have different memory properties requirements. To deal with the increasing complexity, it is crucial to find a way to consolidate some of the properties common for clients, leaving only some unique properties to the client.

This paper provides an in-depth survey of the most recent state-of-art memory management techniques for HPC and Cloud Computing. Here, we put together resource management techniques that take into account memory optimisation problem. Even though recent memory optimisation research covers both hardware and software support aspects, most studies evaluate the results in terms of power consumption and/or speed.

ACKNOWLEDGMENT

The proofreading given by Dr. F. M. Rossi is gratefully acknowledged. Authors are also thankful to two anonymous reviewers for their comments that greatly improved the manuscript.

REFERENCES

[1] B. Koller, N. Struckmann, J. Buchholz, and M. Gienger, *Towards an Environment to Deliver High Performance Computing to Small and Medium Enterprises*. Cham, Switzerland: Springer, 2015, pp. 41–50, doi: 10.1007/978-3-319-20340-9_4.

[2] B. Koller and M. Gienger, "Enhancing high performance computing with cloud concepts and technologies," in *Sustained Simulation Performance 2014*. Cham, Switzerland: Springer, 2015, pp. 47–56.

[3] M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *J. Netw. Comput. Appl.*, vol. 66, pp. 106–127, May 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516000291

[4] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surveys*, vol. 51, no. 1, p. 8, Apr. 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3177787.3150224

[5] N. S. Kim, D. Chen, J. Xiong, and W.-M. W. Hwu, "Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era," *IEEE Micro*, vol. 37, no. 4, pp. 10–18, Aug. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/8013455/

[6] D. Zivanovic, M. Pavlovic, M. Radulovic, H. Shin, J. Son, S. A. Mckee, P. M. Carpenter, P. Radojković, and E. Ayguadé, "Main memory in HPC: Do we need more or could we live with less?" *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, pp. 1–26, Mar. 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3058793.3023362

[7] J. Dongarra *et al.*, "The international exascale software project roadmap," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 1, pp. 3–60, 2011, doi: 10.1177/1094342010391989.

[8] M. A. Khaleel, "Scientific grand challenges: Crosscutting technologies for computing at the exascale," U.S. Dept. Energy Office Sci., Washington, DC, USA, Tech. Rep. PNNL-20168, 2011. [Online]. Available: https://Digital.library.unt.edu/ark:/67531/metadc841613/

[9] A. Geist and D. A. Reed, "A survey of high-performance computing scaling challenges," *Int. J. High Perform. Comput. Appl.*, vol. 31, no. 1, pp. 104–113, Aug. 2017, doi: 10.1177/1094342015597083.

[10] R. Gerber, J. Hack, K. Riley, K. Antypas, R. Coffey, E. Dart, T. Straatsma, J. Wells, D. Bard, S. Dosanjh, I. Monga, M. E. Papka, and L. Rotman, "Crosscut report: Exascale requirements reviews, March 9–10, 2017-tysons corner, virginia. an office of science review sponsored by: Advanced scientific computing research, basic energy sciences, biological and environmental research, fusion energy sciences, high energy physics, nuclear physics," U.S. Dept. Energy Office Sci., USA, Tech. Rep. 1417653, Jan. 2018. [Online]. Available: http://www.osti.gov/servlets/purl/1417653/

[11] M. Parashar, M. AbdelBaky, I. Rodero, and A. Devarakonda, "Cloud paradigms and practices for computational and data-enabled science and engineering," *Comput. Sci. Eng.*, vol. 15, no. 4, pp. 10–18, Jul./Aug. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6530588/

[12] M. Maiterth, G. Koenig, K. Pedretti, S. Jana, N. Bates, A. Borghesi, D. Montoya, A. Bartolini, and M. Puzovic, "Energy and power aware job scheduling and resource management: Global survey—initial analysis," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 685–693. [Online]. Available: https://ieeexplore.ieee.org/document/8425478/

[13] M. Bielski, C. Pinto, D. Raho, and R. Pacalet, "Survey on memory and devices disaggregation solutions for HPC systems," in *Proc. IEEE Intl Conf. Comput. Sci. Eng. (CSE) IEEE Intl Conf. Embedded Ubiquitous Comput. (EUC) 15th Intl Symp. Distrib. Comput. Appl. Bus. Eng. (DCABES)*, Aug. 2016, pp. 197–204. [Online]. Available: https://ieeexplore.ieee.org/document/7982247/

[14] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *J. Netw. Comput. Appl.*, vol. 74, pp. 66–85, Oct. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516301746

[15] J. Zhang, H. Huang, and X. Wang, "Resource provision algorithms in cloud computing: A survey," *J. Netw. Comput. Appl.*, vol. 64, pp. 23–42, Apr. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804516000643

[16] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *J. Netw. Comput. Appl.*, vol. 82, pp. 93–113, Mar. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517300231

[17] C. Ababei and M. G. Moghaddam, "A survey of prediction and classification techniques in multicore processor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1184–1200, May 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8514815/

[18] D. Mishra and P. Kulkarni, "A survey of memory management techniques in virtualized systems," *Comput. Sci. Rev.*, vol. 29, pp. 56–73, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013716301186

[19] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak, "A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 24:1–24:40, Jun. 2017, doi: 10.1145/3057267.

[20] T. Hussain, "A novel hardware support for heterogeneous multi-core memory system," *J. Parallel Distrib. Comput.*, vol. 106, pp. 31–49, Aug. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074373151730076X

[21] E. H. M. Cruz, M. Diener, L. L. Pilla, and P. O. A. Navaux, "A sharing-aware memory management unit for online mapping in multi-core architectures," in *Proc. 22nd Int. Conf. Euro-Par, Parallel Process.*, vol. 9833. New York, NY, USA: Springer-Verlag, 2016, pp. 490–501, doi: 10.1007/978-3-319-43659-3_36.

[22] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based hybrid memory management," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2017, pp. 152–165. [Online]. Available: http://ieeexplore.ieee.org/document/8048927/

[23] S. Williams, L. Ionkov, M. Lang, and J. Lee, "Heterogeneous memory and arena-based heap allocation," in *Proc. Workshop Memory Centric High Perform. Comput. (MCHPC)*. New York, NY, USA: ACM, 2018, pp. 67–71. [Online]. Available: http://doi.acm.org/10.1145/3286475.3286568

[24] S. Perarnau, J. A. Zounmevo, B. Gerofi, K. Iskra, and P. Beckman, "Exploring data migration for future deep-memory many-core systems," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Sep. 2016, pp. 289–297. [Online]. Available: http://ieeexplore.ieee.org/document/7776521/

[25] J.-H. Ding, Y.-T. Chang, Z.-D. Guo, K.-C. Li, and Y.-C. Chung, "An efficient and comprehensive scheduler on asymmetric multicore architecture systems," *J. Syst. Archit.*, vol. 60, no. 3, pp. 305–314, Mar. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762113000738

[26] B. Goglin, "Exposing the locality of heterogeneous memory architectures to HPC applications," in *Proc. 2nd Int. Symp. Memory Syst. (MEMSYS)*. New York, NY, USA: ACM, 2016, pp. 30–39, doi: 10.1145/2989081.2989115.

[27] L. Oden and P. Balaji, "Hexe: A toolkit for heterogeneous memory management," in *Proc. IEEE 23rd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2017, pp. 656–663. [Online]. Available: https://ieeexplore.ieee.org/document/8368419/

[28] C. Cantalupo, V. Venkatesan, J. Hammond, K. Czurlyo, and S. D. Hammond, "memkind: An extensible heap memory manager for heterogeneous memory platforms and mixed memory policies," Memkind Library Team Intel Corp., Santa Clara, CA, USA, Tech. Rep., 2015.

[29] S. Williams, L. Ionkov, and M. Lang, "NUMA distance for heterogeneous memory," in *Proc. Workshop Memory Centric Program. HPC (MCHPC)*. New York, NY, USA: ACM, 2017, pp. 30–34, doi: 10.1145/3145617.3145620.

[30] S. Perarnau, J. A. Zounmevo, M. Dreher, B. C. Van Essen, R. Gioiosa, K. Iskra, M. B. Gokhale, K. Yoshii, and P. Beckman, "Argo NodeOS: Toward unified resource management for Exascale," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2017, pp. 153–162. [Online]. Available: http://ieeexplore.ieee.org/document/7967105/

[31] B. Kocoloski and J. Lange, "Lightweight memory management for high performance applications in consolidated environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 468–480, Feb. 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7029138/

[32] M. Diener, E. H. M. Cruz, and P. O. A. Navaux, "Locality vs. Balance: Exploring data mapping policies on NUMA systems," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 9–16. [Online]. Available: http://ieeexplore.ieee.org/document/7092693/

[33] M. Gupta, V. Sridharan, D. Roberts, A. Prodromou, A. Venkat, D. Tullsen, and R. Gupta, "Reliability-aware data placement for heterogeneous memory architecture," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 583–595. [Online]. Available: http://ieeexplore.ieee.org/document/8327039/

[34] C. Su, D. Roberts, E. A. León, K. W. Cameron, B. R. de Supinski, G. H. Loh, and D. S. Nikolopoulos, "HpMC: An energy-aware management system of multi-level memory architectures," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*. New York, New York, USA: ACM, 2015, pp. 167–178. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2818950.2818974

[35] S. Li, D. Reddy, and B. Jacob, "A performance & power comparison of modern high-speed DRAM architectures," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*. New York, NY, USA: ACM, 2018, pp. 341–353, doi: 10.1145/3240302.3240315.

[36] I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, and S. Markidis, "Characterizing the performance benefit of hybrid memory system for HPC applications," *Parallel Comput.*, vol. 76, pp. 57–69, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167819118301224

[37] J.-Y. Cho, H.-W. Jin, and D. Nam, "Enhanced memory management for scalable MPI intra-node communication on many-core processor," in *Proc. 24th Eur. MPI Users Group Meeting (EuroMPI)*. New York, NY, USA: ACM, 2017, pp. 1–9, doi: 10.1145/3127024.3127035.

[38] K. Chandrasekar, X. Ni, and L. V. Kale, "A memory heterogeneity-aware runtime system for bandwidth-sensitive HPC applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May/Jun. 2017, pp. 1293–1300. [Online]. Available: http://ieeexplore.ieee.org/document/7965187/

[39] G. Park, S. Rho, J.-S. Kim, and D. Nam, "Towards optimal scheduling policy for heterogeneous memory architecture in many-core system," *Cluster Comput.*, vol. 22, no. 1, pp. 121–133, Mar. 2019, doi: 10.1007/s10586-018-2825-4.

[40] A. Rigo, C. Pinto, K. Pouget, D. Raho, D. Dutoit, P.-Y. Martinez, C. Doran, L. Benini, I. Mavroidis, M. Marazakis, V. Bartsch, G. Lonsdale, A. Pop, J. Goodacre, A. Colliot, P. Carpenter, P. Radojkovic, D. Pleiter, D. Drouin, and B. D. de Dinechin, "Paving the way towards a highly energy-efficient and highly integrated compute node for the exascale revolution: The ExaNoDe approach," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, Aug./Sep. 2017, pp. 486–493. [Online]. Available: http://ieeexplore.ieee.org/document/8049829/

[41] M. Katevenis *et al.*, "Next generation of Exascale-class systems: ExaNeSt project and the status of its interconnect and storage development," *Microprocessors Microsyst.*, vol. 61, pp. 58–71, Sep. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933118300188

[42] A. Drebes, A. Pop, K. Heydemann, A. Cohen, and N. Drach, "Scalable task parallelism for NUMA: A uniform abstraction for coordinated scheduling and memory management," in *Proc. Int. Conf. Parallel Architectures Compilation (PACT)*. New York, NY, USA: ACM, 2016, pp. 125–137, doi: 10.1145/2967938.2967946.

[43] K. Wu, Y. Huang, and D. Li, "Unimem: Runtime data managementon non-volatile memory-based heterogeneous main memory," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*. New York, NY, USA: ACM, 2017, pp. 1–14, doi: 10.1145/3126908.3126923.

[44] E. Kissel and M. Swany, "Photon: Remote memory access middleware for high-performance runtime systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2016, pp. 1736–1743. [Online]. Available: http://ieeexplore.ieee.org/document/7530077/

[45] T. Dysart, P. Kogge, M. Deneroff, E. Bovell, P. Briggs, J. Brockman, K. Jacobsen, Y. Juan, S. Kuntz, R. Lethin, J. McMahon, C. Pawar, M. Perrigo, S. Rucker, J. Ruttenberg, M. Ruttenberg, and S. Stein, "Highly scalable near memory processing with migrating threads on the Emu system architecture," in *Proc. 6th Workshop Irregular Appl., Archit. Algorithms (IA)*, Nov. 2016, pp. 2–9. [Online]. Available: http://ieeexplore.ieee.org/document/7833297/

[46] A. Kulkarni, L. Dalessandro, E. Kissel, A. Lumsdaine, T. Sterling, and M. Swany, "Network-managed virtual global address space for message-driven runtimes," in *Proc. 25th ACM Int. Symp. High-Perform. Parallel Distrib. Comput. (HPDC)*. New York, NY, USA: ACM, 2016, pp. 15–18, doi: 10.1145/2907294.2907320.

[47] M. K. Bhatti, I. Oz, S. Amin, M. Mushtaq, U. Farooq, K. Popov, and M. Brorsson, "Locality-aware task scheduling for homogeneous parallel computing systems," *Computing*, vol. 100, no. 6, pp. 557–595, 2018, doi: 10.1007/s00607-017-0581-6.

[48] P. Ramesh and U. Ramachandraiah, "Energy aware proportionate slack management scheduling for multiprocessor systems," *Procedia Comput. Sci.*, vol. 133, pp. 855–863, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050918310627

[49] H. Arabnejad and J. G. Barbosa, "Multi-QoS constrained and Profit-aware scheduling approach for concurrent workflows on heterogeneous systems," *Future Gener. Comput. Syst.*, vol. 68, pp. 211–221, Mar. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16303740

[50] D. W. Dauwe, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Resilience-aware resource management for exascale computing systems," *IEEE Trans. Sustain. Comput.*, vol. 3, no. 4, pp. 332–345, Oct./Dec. 2018.

[51] S. S. Jha, W. Heirman, A. Falcón, J. Tubella, A. González, and L. Eeckhout, "Shared resource aware scheduling on power-constrained tiled many-core processors," *J. Parallel Distrib. Comput.*, vol. 100, pp. 30–41, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731516301186

[52] M. Aigner, C. M. Kirsch, M. Lippautz, and A. Sokolova, "Fast, multicore-scalable, low-fragmentation memory allocation through large virtual memory and global data structures," in *Proc. ACM SIGPLAN Int. Conf. Object-Oriented Program., Syst., Lang., Appl. (OOPSLA)*. New York, NY, USA: ACM, 2015, pp. 451–469, doi: 10.1145/2814270.2814294.

[53] D. Diamantopoulos, S. Xydis, K. Siozios, and D. Soudris, *Dynamic Memory Management in Vivado-HLS for Scalable Many-Accelerator Architectures*. Cham, Switzerland: Springer, 2015, pp. 117–128, doi: 10.1007/978-3-319-16214-0_10.

[54] W. Wei, D. Jiang, S. A. McKee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *Proc. Int. Conf. Parallel Archit. Compilation (PACT)*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 163–173, doi: 10.1109/PACT.2015.10.

[55] Y. Kwon and M. Rhu, "Beyond the memory wall: A case for memory-centric HPC system for deep learning," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2018, pp. 148–161. [Online]. Available: https://ieeexplore.ieee.org/document/8574538/

[56] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 27–39. [Online]. Available: http://ieeexplore.ieee.org/document/7551380/

[57] S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," *Mach. Learn. Knowl. Extraction*, vol. 1, no. 1, pp. 75–114, 2018. [Online]. Available: http://www.mdpi.com/2504-4990/1/1/5

[58] I. Grasso, P. Radojkovic, N. Rajovic, I. Gelado, and A. Ramirez, "Energy efficient HPC on embedded SoCs: Optimization techniques for mali GPU," in *Proc. IEEE 28th Int. Parallel Distrib. Process. Symp.*, May 2014, pp. 123–132. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6877248

[59] M. R. Meswani, G. H. Loh, S. Blagodurov, D. Roberts, J. Slice, and M. Ignatowski, "Toward efficient programmer-managed two-level memory hierarchies in exascale computers," in *Proc. Hardw.-Softw. Co-Des. High Perform. Comput.*, Nov. 2014, pp. 9–16. [Online]. Available: http://ieeexplore.ieee.org/document/7017958/

[60] L. Teylo, U. de Paula, Y. Frota, D. de Oliveira, and L. M. A. Drummond, "A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds," *Future Gener. Comput. Syst.*, vol. 76, pp. 1–17, Nov. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17309883

[61] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "An algorithm for network and data-aware placement of multi-tier applications in cloud data centers," *J. Netw. Comput. Appl.*, vol. 98, pp. 65–83, Nov. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804517302989

[62] T. Maqsood, N. Tziritas, T. Loukopoulos, S. A. Madani, S. U. Khan, and C.-Z. Xu, "Leveraging on deep memory hierarchies to minimize energy consumption and data access latency on single-chip cloud computers," *IEEE Trans. Sustain. Comput.*, vol. 2, no. 2, pp. 154–166, Apr./Jun. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7932118/

[63] I. F. Senturk, P. Balakrishnan, A. Abu-Doleh, K. Kaya, Q. Malluhi, and Ü. V. Çatalyürek, "A resource provisioning framework for bioinformatics applications in multi-cloud environments," *Future Gener. Comput. Syst.*, vol. 78, pp. 379–391, Jan. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16301911

[64] W. Zhu, Y. Zhuang, and L. Zhang, "A three-dimensional virtual resource scheduling method for energy saving in cloud computing," *Future Gener. Comput. Syst.*, vol. 69, pp. 66–74, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16306586

[65] H. Duan, C. Chen, G. Min, and Y. Wu, "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems," *Future Gener. Comput. Syst.*, vol. 74, pp. 142–150, Sep. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X16300292

[66] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7355287/

[67] M. Seddigh, H. Taheri, and S. Sharifian, "Dynamic prediction scheduling for virtual machine placement via ant colony optimization," in *Proc. Signal Process. Intell. Syst. Conf. (SPIS)*, Dec. 2015, pp. 104–108. [Online]. Available: http://ieeexplore.ieee.org/document/7422321/

[68] Y. Moon, H. Yu, J.-M. Gil, and J. Lim, "A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments," *Hum.-Centric Comput. Inf. Sci.*, vol. 7, no. 1, p. 28, Oct. 2017, doi: 10.1186/s13673-017-0109-2.

[69] A. Shukla and Y. Simmhan, "Model-driven scheduling for distributed stream processing systems," *J. Parallel Distrib. Comput.*, vol. 117, pp. 98–114, Jul. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731518300686

[70] P. V. and C. N. K. Babu, "Moving average fuzzy resource scheduling for virtualized cloud data services," *Comput. Standards Interfaces*, vol. 50, pp. 251–257, Feb. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0920548916301349

[71] B. Balis, K. Figiela, K. Jopek, M. Malawski, and M. Pawlik, "Porting HPC applications to the cloud: A multi-frontal solver case study," *J. Comput. Sci.*, vol. 18, pp. 106–116, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187775031630148X

[72] K. H. Park, W. Hwang, H. Seok, C. Kim, D.-J. Shin, D. J. Kim, M. K. Maeng, and S. M. Kim, "MN-MATE: Elastic resource management of manycores and a hybrid memory hierarchy for a cloud node," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 12, no. 1, p. 5, Aug. 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2810396.2701429

[73] M. Qiu, Z. Ming, J. Li, K. Gai, and Z. Zong, "Phase-change memory optimization for green cloud with genetic algorithm," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3528–3540, Dec. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7054465/

[74] L. Wei, C. H. Foh, B. He, and J. Cai, "Towards efficient resource allocation for heterogeneous workloads in IaaS clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 264–275, Jan./Mar. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/7277025/

[75] K. Almi'ani, Y. C. Lee, and B. Mans, "On efficient resource use for scientific workflows in clouds," *Comput. Netw.*, vol. 146, pp. 232–242, Dec. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128618303384

[76] H. M. Makrani and H. Homayoun, "MeNa: A memory navigator for modern hardware in a scale-out environment," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2017, pp. 2–11. [Online]. Available: http://ieeexplore.ieee.org/document/8167751/

[77] S. Farokhi, E. B. Lakew, C. Klein, I. Brandic, and E. Elmroth, "Coordinating CPU and memory elasticity controllers to meet service response time constraints," in *Proc. IEEE Int. Conf. Cloud Autonomic Comput.*, Sep. 2015, pp. 69–80. [Online]. Available: http://ieeexplore.ieee.org/document/7312142/

[78] P.-Y. Hsu, S.-T. Hsieh, and Y.-C. Chuang, "Effective memory reusability based on user distributions in a cloud architecture to support manufacturing ubiquitous computing," *Int. J. Comput. Integr. Manuf.*, vol. 30, nos. 4–5, pp. 459–471, 2017, doi: 10.1080/0951192X.2015.1067912.

[79] S. Kannan, A. Gavrilovska, V. Gupta, and K. Schwan, "HeteroOS—OS design for heterogeneous memory management in datacenter," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 521–534.

[80] K. Gai, M. Qiu, and H. Zhao, "Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing," *IEEE Trans. Cloud Comput.*, to be published. [Online]. Available: http://ieeexplore.ieee.org/document/7523230/

[81] X. Liao, H. Jin, S. Yu, and Y. Zhang, "A novel memory allocation scheme for memory energy reduction in virtualization environment," *J. Comput. Syst. Sci.*, vol. 81, no. 1, pp. 3–15, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000014001056

[82] A. Al-Yatama, I. Ahmad, and N. Al-Dabbous, "Memory allocation algorithm for cloud services," *J. Supercomput.*, vol. 73, no. 11, pp. 5006–5033, Nov. 2017, doi: 10.1007/s11227-017-2069-8.

[83] A. Alyatama, A. Alsumait, and M. Alotaibi, "Continuous memory allocation model for cloud services," *J. Supercomput.*, vol. 74, no. 10, pp. 5513–5538, Oct. 2018, doi: 10.1007/s11227-018-2455-x.

[84] L. A. Garrido and P. Carpenter, "Aggregating and managing memory across computing nodes in cloud environments," in *High Performance Computing*, J. M. Kunkel, R. Yokota, M. Taufer, and J. Shalf, Eds. Cham, Switzerland: Springer, 2017, pp. 642–652.

[85] D. Magenheimer, C. Mason, D. McCracken, and K. Hackel, "Transcendent memory and linux," in *Proc. Linux Symp.*, 2009, pp. 191–200.

[86] K. Koh, K. Kim, S. Jeon, and J. Huh, ''Disaggregated cloud memory with elastic block management,'' *IEEE Trans. Comput.*, vol. 68, no. 1, pp. 39–52, Jan. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8400405/

[87] T. L. Nguyen and A. Lebre, ''Virtual machine boot time model,'' in *Proc. 25th Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process. (PDP)*, Mar. 2017, pp. 430–437. [Online]. Available: http://ieeexplore.ieee.org/document/7912684/

[88] C. Ro, ''Modeling and analysis of memory virtualization in cloud computing,'' *Cluster Comput.*, vol. 18, no. 1, pp. 177–185, Mar. 2015, doi: 10.1007/s10586-014-0353-4.

[89] G. Jia, G. Han, J. J. P. C. Rodrigues, J. Lloret, and W. Li, ''Coordinate memory deduplication and partition for improving performance in cloud computing,'' *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 357–368, Jun. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/7364233/

[90] W. Zhao and Z. Wang, ''Dynamic memory balancing for virtual machines,'' in *Proc. ACM SIGPLAN/SIGOPS Int. Conf. Virtual Execution Environ. (VEE)*. New York, New York, USA: ACM, 2009, p. 21, doi: 10.1145/1508293.1508297.

[91] C. A. Waldspurger, ''Memory resource management in VMware ESX server,'' *ACM SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, 2002, doi: 10.1145/844128.844146.

[92] Q. Zhang and L. Liu, ''Shared memory optimization in virtualized cloud,'' in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun. 2015, pp. 261–268. [Online]. Available: http://ieeexplore.ieee.org/document/7214053/

[93] L. Zeng, Y. Wang, K. B. Kent, and Z. Xiao, ''Naplus: A software distributed shared memory for virtual clusters in the cloud,'' *Softw., Pract. Exper.*, vol. 47, no. 9, pp. 1201–1220, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2486

[94] A. C. Macdonell, ''Shared-memory optimizations for virtual machines,'' Ph.D. dissertation, Dept. Comput. Sci., Univ. Alberta, Edmonton, Canada, 2011.

[95] Y. Ren, L. Liu, Q. Zhang, Q. Wu, J. Guan, J. Kong, H. Dai, and L. Shao, ''Shared-memory optimizations for inter-virtual-machine communication,'' *ACM Comput. Surv.*, vol. 48, no. 4, pp. 49:1–49:42, Feb. 2016, doi: 10.1145/2847562.

[96] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao, ''CloudCache: On-demand flash cache management for cloud computing,'' in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*. Santa Clara, CA, USA: USENIX Association, 2016, pp. 355–369. [Online]. Available: https://www.usenix.org/conference/fast16/technical-sessions/presentation/arteaga

[97] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, ''Centaur: Host-side SSD caching for storage performance control,'' in *Proc. IEEE Int. Conf. Autonomic Comput.*, Jul. 2015, pp. 51–60. [Online]. Available: http://ieeexplore.ieee.org/document/7266934/

[98] S. Moon and A. L. N. Reddy, ''Adaptive policies for balancing performance and lifetime of mixed SSD arrays through workload sampling,'' in *Proc. 32nd Symp. Mass Storage Syst. Technol. (MSST)*, May 2016, pp. 1–13. [Online]. Available: https://ieeexplore.ieee.org/document/7897084/

[99] R. Salkhordeh, M. Hadizadeh, and H. Asadi, ''An efficient hybrid I/O caching architecture using heterogeneous SSDs,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1238–1250, Jun. 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8550676/

[100] D.-M. Bui, Y. Yoon, E.-N. Huh, S. Jun, and S. Lee, ''Energy efficiency for cloud computing system based on predictive optimization,'' *J. Parallel Distrib. Comput.*, vol. 102, pp. 103–114, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731516301708

[101] E. R. Rodrigues, R. L. F. Cunha, M. A. S. Netto, and M. Spriggs, ''Helping HPC users specify job memory requirements via machine learning,'' in *Proc. 3rd Int. Workshop HPC User Support Tools (HUST)*, Nov. 2016, pp. 6–13. [Online]. Available: http://ieeexplore.ieee.org/document/7830451/

[102] X. Wang, C. Xu, K. Wang, F. Yan, and D. Zhao, ''Toward cost-effective memory scaling in clouds: Symbiosis of virtual and physical memory,'' in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, Jul. 2018, pp. 33–40, doi: 10.1109/CLOUD.2018.00012.

[103] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, and Z. Liu, ''Pacer: A progress management system for live virtual machine migration in cloud computing,'' *IEEE Trans. Netw. Service Manage.*, vol. 10, no. 4, pp. 369–382, Dec. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6662353/

[104] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, and H. Tenhunen, ''Utilization prediction aware VM consolidation approach for green cloud computing,'' in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, Jun./Jul. 2015, pp. 381–388. [Online]. Available: http://ieeexplore.ieee.org/document/7214068/

[105] J. Shi, J. Luo, F. Dong, J. Jin, and J. Shen, ''Fast multi-resource allocation with patterns in large scale cloud data center,'' *J. Comput. Sci.*, vol. 26, pp. 389–401, May 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877750317305185

[106] D. Kesavaraja and A. Shenbagavalli, ''QoE enhancement in cloud virtual machine allocation using Eagle strategy of hybrid krill herd optimization,'' *J. Parallel Distrib. Comput.*, vol. 118, pp. 267–279, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731517302459

[107] T. R. Philomine, C. J. M. Tauro, and M. Miranda, ''An approach for efficient capacity management in a cloud,'' in *Emerging Research in Computing, Information, Communication and Applications*, N. R. Shetty, L. M. Patnaik, N. H. Prasad, and N. Nalini, Eds. Singapore: Springer, 2018, pp. 365–375.

[108] G. Moltó, M. Caballer, and C. de Alfonso, ''Automatic memory-based vertical elasticity and oversubscription on cloud platforms,'' *Future Gener. Comput. Syst.*, vol. 56, pp. 1–10, Mar. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X15003155

[109] Z. Á. Mann, ''Resource optimization across the cloud stack,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 169–182, Jan. 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8016586/

[110] H. Li, W. Li, H. Wang, and J. Wang, ''An optimization of virtual machine selection and placement by using memory content similarity for server consolidation in cloud,'' *Future Gener. Comput. Syst.*, vol. 84, pp. 98–107, Jul. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17323063

[111] G. J. L. Paulraj, S. A. J. Francis, J. D. Peter, and I. J. Jebadurai, ''Resource-aware virtual machine migration in IoT cloud,'' *Future Gener. Comput. Syst.*, vol. 85, pp. 173–183, Aug. 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17322471

[112] G. Singh, S. Behal, and M. Taneja, ''Advanced memory reusing mechanism for virtual machines in cloud computing,'' *Procedia Comput. Sci.*, vol. 57, pp. 91–103, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091501902X

[113] X.-K. Liao, K. Lu, C.-Q. Yang, J.-W. Li, Y. Yuan, M.-C. Lai, L.-B. Huang, P.-J. Lu, J.-B. Fang, J. Ren, and J. Shen, ''Moving from exascale to zettascale computing: Challenges and techniques,'' *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 10, pp. 1236–1244, Oct. 2018, doi: 10.1631/FITEE.1800494.

ANNA PUPYKINA received the higher education degree in data processing and control systems engineering from Samara State Aerospace University, Russia, in 2009. She is currently pursuing the Ph.D. degree with the Politecnico di Milano, under the supervision of Prof. G. Agosta.

She was with Russian State Humanitarian University (RSUH), Togliatti, and also with the Department of Applied Mathematics and Informatics, Togliatti State University (TSU). Her main research interests include runtime management algorithms and programming model extensions for deeply heterogeneous systems.

GIOVANNI AGOSTA received the Laurea degree in computer engineering and the Ph.D. degree in information technology from the Politecnico di Milano, Italy, in 2000 and 2004, respectively. He is currently a tenured Researcher. He has authored more than 80 articles in international journals, conferences, and workshops. His main research interest includes compiler technology. He was a recipient of four HiPEAC Awards and three Best Paper Awards. He has been involved in eight European funded projects, since 2010, holding workpackage and task leader responsibilities. He is a Full Member of the HiPEAC Network and the Organizer of two workshops colocated with the HiPEAC Conference.

• • •