applied sciences

MDPI

Article

Adaptation of Parallel SaaS to Heterogeneous Co-Located Cloud Resources

Oleg Bystrov [1], Ruslan Pacevič [1] and Arnas Kačeniauskas [1,2,*]

[1] Department of Graphical Systems, Vilnius Gediminas Technical University, 10223 Vilnius, Lithuania; oleg.bystrov@vilniustech.lt (O.B.); ruslan.pacevic@vilniustech.lt (R.P.)
[2] Laboratory of Parallel Computing, Vilnius Gediminas Technical University, 10223 Vilnius, Lithuania
* Correspondence: arnas.kaceniauskas@vilniustech.lt; Tel.: +370-5-274-4913

Featured Application: The presented runtime adaptation of computational load helps efficiently run parallel SaaS on heterogeneous or co-located cloud resources.

Abstract: Cloud computing has received increasing attention due to its promise of delivering on-demand, scalable, and virtually unlimited resources. However, heterogeneity or co-location of virtual cloud resources can cause severe degradation of the efficiency of parallel computations because of a priori unknown application-specific performance metrics, load imbalance, and limitations of memory bandwidth. This paper presents the runtime adaptation of parallel discrete element method (DEM) Software as a Service (SaaS) to heterogeneous or co-located resources of the OpenStack cloud. The computational workload is adapted by using weighted repartitioning and runtime measured performance of parallel computations on Docker containers. The high improvement in performance up to 48.7% of the execution time is achieved, applying the runtime adapted repartitioning when the load imbalance is high enough. The low load imbalance leads to the close values of computational load, when small variations in the system load and performance can cause oscillations in subsets of particles. Memory stress tests cause heterogeneity of non-isolated containers, which reduces the performance of memory bandwidth bound DEM SaaS on the co-located resources. The runtime adapted repartitioning handles the constant and periodically variable performance of non-isolated containers and decreases the total execution time of DEM SaaS.

Keywords: heterogeneous co-located cloud resources; weighted partitioning; runtime measured performance; Docker containers; discrete element method; memory bandwidth bound applications

Citation: Bystrov, O.; Pacevič, R.; Kačeniauskas, A. Adaptation of Parallel SaaS to Heterogeneous Co-Located Cloud Resources. *Appl. Sci.* **2023**, *13*, 5115. https://doi.org/10.3390/app13085115

Academic Editor: Agostino Forestiero

Received: 1 March 2023
Revised: 8 April 2023
Accepted: 18 April 2023
Published: 20 April 2023

1. Introduction

Cloud computing is becoming a natural solution to the problem of expanding computational needs due to its on-demand, low-cost, and virtually unlimited resources for deploying various services [1]. The NIST SPI [2] classifies cloud services into three categories, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Private and public clouds can be developed using different implementations of cloud software, with OpenStack [3] being an open-source cloud management platform that offers a secure and reliable IaaS. To isolate workloads and regulate resource usage, cloud computing heavily employs virtual machines (VMs). Containers present a technology for improving the productivity and code portability in cloud infrastructures. Docker [4] has emerged as the standard runtime, image format, and build system for Linux containers because its layered file system requires less disk space and I/O compared to the equivalent VM disk images.

The scalable cloud resources can be dynamically allocated on demand according to consumer requirements and preferences defined in service level agreement (SLA). However, cloud resources can cause severe degradation of SaaS performance due to their heterogeneity or co-location issues [1]. Different types of heterogeneous VMs [5] present challenges to

load balancing of parallel MPI-based jobs that are in frequent communication. In resource-aware partitioning, the computational domain is divided into unequal partitions or subsets of particles according to weights of heterogeneous resources [6]. Generally, heterogeneity weights of parallel computations cannot be exactly defined a priori according to the number of instructions performed by CPU per second because of domain decomposition issues, communication overhead, and limitations of memory bandwidth. Synthetic benchmarks conducted on cloud systems typically fail to consider all the essential factors. Therefore, the application-specific tests need to be performed to define heterogeneity weights before the production runs of parallel computations on heterogeneous resources.

Virtual resources can create further issues if they are co-located on the same machine and compete for the same resources, such as memory bandwidth [7]. Common virtualization technologies do not ensure isolation of the cache usage and memory bandwidth of individual VMs accommodated by the same physical machine, leading to contention between them. If one of the VMs excessively uses memory bandwidth, the performance of memory bandwidth bound SaaS, running on the other VM, may significantly degrade. Few cloud providers [8] offers L3 cash and memory bandwidth isolation based on existing hardware technologies [9] or software solutions [10]. Unfortunately, existing solutions cannot be easily used for managing memory bandwidth with all virtualization technologies in popular cloud environments.

The use of clouds to deploy computationally demanding scientific codes for high performance computations and visualization offers better resource exploitation and greater user mobility. Consequently, cloud resources are perceived as a promising avenue for future advances in the area of scientific computations [11]. The discrete element method (DEM), originally developed by Cundall and Strack [12], describes particulate media by considering the motion and deformation behavior of individual particles. Currently, the DEM is acknowledged to be an effective method not limited to the analysis of non-cohesive granular materials but extended to cohesive powders, fluidized environments, rock cutting, and couplings with different multiphysics. However, the simulation of systems at the particle level of detail has the disadvantage of making the DEM computationally very expensive. Naturally, to solve the industrial-scale problems, parallel computing on cloud infrastructures has become an obvious way to increase computational capabilities. However, cloud computing still lacks case studies and best practices on how to efficiently run parallel memory bandwidth bound SaaS on heterogeneous co-located virtual cloud resources.

This paper presents the runtime adaptation of parallel DEM SaaS to heterogeneous co-located resources of the private OpenStack cloud. The hybrid parallelization of memory bandwidth bound DEM SaaS was developed by using OpenCL for shared-memory multi-core machines and MPI for weighted repartitioning on distributed-memory architectures. The remaining paper is organized as follows: Section 2 analyzes the related works, Section 3 describes the developed DEM SaaS based on hybrid parallelization, Section 4 discusses the results of SaaS performance attained on heterogeneous co-located cloud resources, and the conclusions are given in Section 5.

2. The Related Works

Previous studies [13–15] have already examined the performance of computation-intensive benchmarks on virtual machines and lightweight containers. However, few studies include the performance analysis of parallel MPI-based applications on heterogeneous cloud resources. Gang applications [16] are parallel jobs that frequently communicate and must execute simultaneously. Moschakis and Karatza [17] evaluated gang scheduling performance in the Amazon EC2 cloud. Hao et al. [18] proposed a 0–1 integer programming for the gang scheduling. Mohammadi et al. [19] assessed the parallel performance of the Linpack benchmark on public clouds. The obtained results demonstrated that the performance per computing core on the public clouds could be comparable to modern traditional supercomputing systems. However, the discussed studies did not investigate the impact of heterogeneous cloud resources and partitioning issues on parallel performance.

In cloud computing, the task scheduling, virtual machine load balancing, and resource provisioning problems formulated from the cloud provider's perspective have received a lot of attention. Shahid et al. [20] evaluated the performance of existing load balancing algorithms with different service broker policies for cloud computing. Particle swarm optimization, round robin, equally spread current execution, and throttled load balancing algorithms were investigated, taking into account optimized response time, data center processing time, virtual machine costs, data transfer costs, and total cost. Heidari et al. [21] introduced a deep post-decision-state learning algorithm for dynamic IoT-edge-cloud offloading scenarios. The proposed technique outperformed multiple benchmarks in terms of delay, job failure rate, cost, computational overhead, and energy consumption. Bojato et al. [22] presented a flexible cloud-based software architecture engineered to provide an efficient and robust password guessability service. A comprehensive literature review [23] addressed resilience and dependability management issues in distributed environments, namely cloud, edge, fog, internet of things, internet of drones, and internet of vehicles. However, parallel MPI-based applications with parallel frequently communicating jobs were rarely considered. Moreover, most researchers balanced the workload on virtual machines of an entire subsystem rather than the enhanced performance of a parallel frequently communicating application on heterogeneous virtual resources.

Early attempts to parallelize DEM computations were based on force decomposition techniques or simple variants of the domain decomposition methods. Washington and Meegoda [24] divided the interparticle force computation among the processors. The domain decomposition methods partition the computational domain into particle subsets, each being assigned to a processor. Kačeniauskas et al. [25] used the static domain decomposition with regular partitions for parallel DEM computations on gLite grid infrastructure. Wang et al. [26] applied domain decomposition with ghost layers of particles for the DEM and large eddy simulation. However, the dynamically changing workload configuration may lead to load imbalance and low parallel efficiency. More flexible but more complicated dynamic domain decomposition is one of the solutions to this load balance problem that allows higher scalability in parallel computing performance. In the case of the DEM, Owen et al. [27] used a topological dynamic domain decomposition method based on a dynamic graph repartitioning. Walther and Sbalzarini [28] presented large-scale parallel simulations of granular flows, employing adaptive domain decomposition based on the multilevel k-way graph partitioning method [29]. Markauskas and Kačeniauskas [30] applied k-way graph partitioning and the recursive coordinate bisection (RCB) [31] to solve the hopper discharge problem. Higher parallel speedup was achieved by using RCB, which confirmed the efficiency of conceptually simpler geometric methods for particle simulations [6]. The parallel efficiency of 0.87 was achieved on 2048 cores, modeling the hopper filled with 5.1×10^6 particles.

Distributed hardware systems based on common multicore nodes have evolved to heterogeneous hybrid architectures, embodying both shared and distributed memories. Hybrid parallelization of DEM codes can lead to improved load balance, decreased communication overhead, and reduced memory consumption on contemporary shared- and distributed-memory systems. Liu et al. [32] described a hybrid MPI/OpenMP parallelization of their MFIX-DEM solver, emphasizing the importance of data locality and thread placement policies in scaling OpenMP implementation to large core counts. The speed increased 185 times on 256 cores of their hybrid parallelization compared to 138 times of a standalone MPI computation with 5.12 million particles. The LIGGGHTS DEM software [33] employed a recursive multi-sectioning algorithm for global domain decomposition and an RCB method for defining subsets of particles assigned to threads. Experiments of the load balancing with different MPI/OpenMP configurations with up to 128 threads were presented. Cintra et al. [34] demonstrated a hybrid MPI/OpenMP parallelization of DEMOOP software, employing the RCB method for domain partitioning and various shared-memory implementations for particle sorting and distribution. However, the parallel efficiency of the software rapidly drops, increasing the number of cores up to 64. Incardona et al. [35] presented the open-source framework OpenFPM for shared-memory and distributed-memory implementations of particle and

particle-mesh codes. This scalable framework provides methods for domain decomposition, dynamic load balancing, and internode communication. Yan and Regueiro [36] examined the hybrid MPI/OpenMP mapping schemes and influences of the memory/cache hierarchy for 3D DEM simulations of ellipsoidal and poly-ellipsoidal particles. However, the pure MPI implementation achieved a higher efficiency than the hybrid MPI/OpenMP software. In the discussed research, only OpenMP was used for shared-memory programming.

Compared to the CPU-based parallelization, GPU has a higher parallel structure, which makes it very efficient for particle-based algorithms, where large blocks of data can be processed in parallel. Software environments, such as CUDA or OpenCL [37], are targeted at general-purpose GPU (GPGPU) programming. Govender et al. [38] designed the modular Blaze-DEMGPU framework for the GPU architecture. Kelly et al. [39] adopted a dimensionalization process combined with mixed-precision data to simulate 3D scenarios with up to 710 million spherical frictionless particles. To achieve a higher speedup ratio for a larger number of particles, a few efforts have been made to use the combined GPU and MPI technology. Xu et al. [40] achieved the quasi-real-time simulation of an industrial rotating drum, when about 9.6×10^6 particles were treated with 270 GPUs. The one-dimensional domain decomposition with multiple GPUs was applied to the simulation of 128 million particles by Tian et al. [41]. The GPU-based DEM combined with MPI has been applied by Gan et al. [42] to study granular flows in the ironmaking industry. However, the communication overhead among GPUs significantly reduces the parallel performance because of the costly data transfer to the CPU memory and the MPI message passing among different nodes. It is worth noting that only CUDA was employed for shared-memory programming on the GPU together with MPI technology for distributed-memory communications in the case of DEM software.

Very few attempts to exploit low-cost flexible cloud resources for computationally demanding memory bandwidth bound DEM software [43] have been reported in the academic literature [44–46]. Rescale's cloud platform offers the commercial EDEM software for computations of particle systems by the DEM [44]. The open-source DEM code MercuryDPM [45] was also deployed on a cloud computing platform. Bystrov et al. [46] developed parallel MPI-based DEM software for distributed-memory architectures and deployed SaaS on the OpenStack cloud. The presented performance analysis revealed MPI communication issues and overhead caused by processes of the OpenStack services Nova and Zun. However, the heterogeneity of cloud resources and co-location issues were not considered.

DEM computations were rarely performed on heterogeneous resources, but heterogeneous architectures were often employed for computationally intensive applications in other research areas [47]. Danovaro et al. [47] analyzed the performance of widely used applications, such as FFT, convolution, and N-body simulation on a multicore cluster node with or without GPUs. In plasma plume simulations with the particle-in-cell model, Araki et al. [48] proposed a patch-based dynamic load balancing method with over-decomposition and a Hilbert space-filling curve. For microscopy image analysis, Barreiros et al. [49] developed a cost-aware data partitioning strategy, minimizing load imbalance on hybrid CPU-GPU machines. Zhong et al. [50] optimally distributed the workload of data-parallel scientific applications between heterogeneous computing resources by using functional performance models of processing elements and relevant data partitioning algorithms. Meanwhile, Bystrov et al. [51] explored the trade-off between execution time and consumed energy for aortic valve computations on a heterogeneous OpenStack cloud, comparing parallel speedups obtained through several partitioning techniques, but did not consider load imbalance and co-located resources.

The related works are listed and compared in Table 1. The parallel applications that frequently communicate between nodes and must execute simultaneously are indicated in the column "Parallel (internode)". The parallelization software is provided in the next column. The performance studies on heterogeneous and co-located resources are indicated in the columns named "Heterogeneous resources" and "Co-located resources", respectively. It can be observed that parallel DEM computations are rarely performed on heterogeneous resources because of increased load imbalance or communication. In other research areas, the parallel frequently communicating applications are more often solved on heterogeneous

cloud resources. In distributed cloud environments, parallel frequently communicating applications are rare because of their inherent limitations to achieve high parallel performance and scalability on heterogeneous or co-located resources. To the best of our knowledge, there are no articles on the adaptation of parallel frequently communicating applications to co-located cloud resources in the literature. Excessive usage of memory bandwidth by one virtual resource can substantially reduce the performance of parallel memory bandwidth bound DEM computations on the other resource co-located on the same physical machine.

Table 1. Comparison of the related works.

Authors	Parallel (Internode)	Parallelization Software	Heterogeneous Resources	Co-Located Resources
Devine et al. [6]	Yes	MPI	Yes	No
Kačeniauskas et al. [13]	Yes	MPI	No	No
Chae et al. [14]	No	-	No	No
Potdar et al. [15]	No	-	No	No
Papazachos and Karatza [16]	Yes	MPI	Yes	No
Moschakis and Karatza [17]	Yes	MPI	No	No
Hao et al. [18]	Yes	MPI	No	No
Mohammadi and Bazhirov [19]	Yes	MPI	No	No
Shahid et al. [20]	No	-	No	Yes
Heidari at al. [21]	No	-	Yes	No
Bojato et al. [22]	No	-	No	No
Amiri et al. [23]	No	-	No	No
Washington and Meegoda [24]	Yes	MPI	No	No
Kačeniauskas et al. [25]	Yes	MPI	No	No
Wang et al. [26]	Yes	MPI	No	No
Owen and Feng [27]	Yes	MPI	No	No
Walther and Sbalzarini [28]	Yes	MPI	Yes	No
Markauskas and Kačeniauskas [30]	Yes	MPI	No	No
Liu et al. [32]	Yes	MPI/OpenMP	No	No
Berger et al. [33]	Yes	MPI/OpenMP	No	No
Cintra et al. [34]	Yes	MPI/OpenMP	No	No
Incardona et al. [35]	Yes	MPI	No	No
Yan and Regueiro [36]	Yes	MPI/OpenMP	No	No
Govender et al. [38]	No	CUDA	No	No
Kelly et al. [39]	No	CUDA	No	No
Xu et al. [40]	Yes	MPI/CUDA	No	No
Tian et al. [41]	Yes	MPI/CUDA	No	No
Gan et al. [42]	Yes	MPI/CUDA	No	No
Pacevič and Kačeniauskas [43]	No	OpenCL	No	No
Weinhart et al. [45]	Yes	MPI	No	No
Bystrov et al. [46]	Yes	MPI	No	No
Danovaro et al. [47]	Yes	MPI/OpenMP, CUDA, OpenACC, and OpenCL	Yes	No
Araki and Martin [48]	Yes	MPI	Yes	No
Barreiros et al. [49]	No	CUDA	Yes	No
Zhong et al. [50]	No	CUDA	Yes	No
Bystrov et al. [51]	Yes	MPI	Yes	No
Current study	Yes	MPI/OpenCL	Yes	Yes

3. Cloud Infrastructure and Developed SaaS

The parallel MPI/OpenCL-based software was developed and deployed as SaaS on the OpenStack cloud infrastructure to perform time-consuming computations of the discrete element method.

3.1. Hosted Cloud Infrastructure

The university private cloud infrastructure based on the OpenStack Train 2019 version [3] is hosted by Vilnius Gediminas Technical University. The deployed capabilities of the OpenStack

cloud infrastructure include the compute service Nova, compute service Zun for containers, networking service Neutron, container network plugin Kuryr, image service Glance, identity service Keystone, object storage service Swift, and block storage service Cinder.

The cloud infrastructure is composed of OpenStack service nodes and compute nodes connected to 1 Gbps Ethernet LAN. Hardware characteristics of faster compute nodes hosting the containers are listed as follows: Intel®Core i7-6700 3.40 GHz CPU, 32 GB DDR4 2133 MHz RAM, 34.13 GB/s memory bandwidth, and 1 TB HDD. Hardware characteristics of slower compute nodes are listed as follows: Intel®Core i7-4790 3.60 GHz CPU, 32 GB DDR3 1866 MHz RAM, 29.87 GB/s memory bandwidth, and 1 TB HDD. Docker version 20.10.7 containers managed by Zun are used in the cloud infrastructure. Ubuntu 20.04.3 LTS (Focal Fossa) is installed in the containers. Characteristics of containers are provided in Table 2. The container CN-GPU is equipped with the NVIDIA® Tesla™ P100 GPU (1792 FP64 CUDA Cores, 12 GB HBM2, 549 GB/s memory bandwidth).

Table 2. Characteristics of containers.

Containers	Cores	Architecture	RAM, GB	HDD, TB
CN-6700-1	1	i7-6700	8	0.5
CN-6700-2	2	i7-6700	16	0.5
CN-6700-3	3	i7-6700	24	0.5
CN-6700-4	4	i7-6700	32	0.5
CN-4790-4	4	i7-4790	32	0.5
CN-GPU	1792 (CUDA)	Tesla™ P100	12	0.5

The architecture of a cloud system comprises multiple layers of deployed services. The OpenStack API manages the IaaS and grants access to infrastructure services. The PaaS layer is provided for developing and deploying software services called SaaS. The Open MPI platform is used to create parallel software for distributed-memory architectures. The Zoltan library [52] serves as a development platform for dynamic load balancing and partitioning particles into subsets. OpenCL (Open Computing Language) [37] is deployed as the platform for parallel programming on shared-memory architectures, such as accelerators, multicore CPUs, and GPUs of various vendors. OpenCL greatly improves the speed and responsiveness of a wide spectrum of applications, including scientific codes. The parallel DEM SaaS is deployed on top of the provided platforms, such as GNU compilers, OpenCL for shared-memory programming, Open MPI for message passing, and the Zoltan library for partitioning of particles into subsets.

3.2. Parallel DEM SaaS

The DEM model for granular flows of the non-cohesive frictional visco-elastic spherical particles is implemented in the parallel SaaS. The particle system consists of a finite number of deformable particles with the specified size distribution and material properties. An arbitrary particle undergoes translational and rotational motion, involving the forces and torques originated in the process of particles' interaction. In this study, the force of gravity is considered, but not the electrostatic force [53] or other external forces [54]. The normal contact force comprises elastic and viscous components. The normal elastic force is calculated using Hertz's contact model in this research. The tangential contact force is divided into the parts of static friction and dynamic friction. The dynamic friction force is directly proportional to the normal component of the contact force. The static friction force is calculated by summing up the elastic counterpart and the viscous damping counterpart. A contact search was performed by using the infinite grid method [43]. Time integration with small time steps was performed by the explicit velocity Verlet method [55]. The details of the governing DEM relations for granular flows are provided in [25,30].

Hybrid Parallelization and Runtime Load Adaptation to Heterogeneous Resources

The simulations at the particle level of detail make the DEM computationally very expensive. The specific characteristics of the solved problem and employed numerical method

highly influence the choice of a parallel solution algorithm [30,38]. Hybrid parallelization of the DEM SaaS was developed to exploit the potential of different types of memory and to simplify mapping of partitions to heterogeneous multicore cloud resources. The dynamic partitioning of particles was performed, implementing the required internode communications by the MPI library for distributed-memory architectures. The RCB method from the Zoltan library [52] was utilized to partition particles into subsets, as it is particularly effective for particle simulations [6]. The main computational procedures of the DEM were implemented by using OpenCL [37] for shared-memory multicore machines or GPUs. Thus, each OpenCL device performs computations only on its subset of particles. However, it needs to share data of ghost particles with OpenCL devices, working on other nodes. Therefore, internode communication, required after partitioning, was implemented by using MPI.

The algorithm of the developed hybrid MPI/OpenCL parallelization is outlined in the pseudocode (Algorithm 1). All computations are performed in the time loop. In line 5 of the pseudocode, the load $\boldsymbol{L}$ is internally monitored for resource-aware partitioning. The computational load L_i of each parallel process i is measured by timers in computational procedures of the DEM software. Heterogeneous resources, such us containers of highly different computing performance, can cause substantial load imbalance of the parallel software. The percentage load imbalance measure λ quantifies the uneven distribution of computational load by using the following formula:

$$\lambda = \left(\frac{L_{\max}}{L_{\text{avg}}} - 1 \right) \cdot 100\%, \tag{1}$$

where L_{avg} denotes the averaged load over all processes and L_{max} represents the largest load.

Algorithm 1 The pseudocode of hybrid MPI/OpenCL parallelization

```
1:  Input
2:      W Weights of partitions, initial value 1.0
3:  while time_loop do
4:      W^old ← W
5:      L ← Monitor_load()
6:      if Condition(L) then
7:              W ← Weight_calculation(W^old,L)
8:              Transfer_Particles_To_Host()
9:              Zoltan_Partitioning(W)
10:             Redistribute_Particles()
11:             Registration_Ghost_Particles()
12:             Exchange_Ghost_Information()
13:             Transfer_Particles_To_Device()
14:     end if
15:     Transfer_Ghost_Particles_To_Host()
16:     Exchange_Ghost_Particles()
17:     Transfer_Ghost_Particles_To_Device()
18:     Contact_search()
19:     Contact_history()
20:     Calculation_forces()
21:     Boundary_conditions()
22:     External_forces()
23:     Integrator()
24:     if write_result then
25:             Transfer_Particles_To_Host()
26:             Write_results()
27:     end if
28: end while
```

In lines 6–14 of the pseudocode, domain decomposition or partitioning of particles into subsets is outlined. In line 6, the subroutine Condition() makes the decision if the repartitioning procedure should be performed. Commonly, repartitioning is necessary if load imbalance measure (1) exceeds the predefined value. In the performed research, a more sophisticated approach is developed according to the specific needs of DEM computations. The dynamic nature of granular flows leads to rapid particle movement through the computational domain, large changes in contact topology, and notable variations in the load configuration. When particles move from one partition to another, there is a need for some communication between MPI processes. The particle data transfer is optional, but information exchange is necessary in each time step. Despite its local character, information exchange and internode particle data transfer consume a significant amount of time, which decreases the performance of parallel computations. Therefore, this particle exchange is performed during the repartitioning procedure. The skinning technique is employed to avoid frequent repartitioning. In the considered time step, particles of one subset can contact particles of another subset only if they are in the ghost layer. In order to make contact with a particle from another subset, the internal particle needs time to cross the ghost layer. This time can be estimated [55], which is often used to reduce the frequency of contact search. Thus, repartitioning should be performed with frequency defined by granular flow physics despite low load imbalance. Finally, the subroutine Condition() combines the load imbalance condition with repartitioning frequency defined by application physics. In most benchmarks of the presented research, repartitioning is regularly performed with the frequency defined by application physics to smooth load oscillations and save execution time extensively consumed by repartitioning. In the case of highly variable loads, the percentage load imbalance threshold equal to 10% is also used to trigger repartitioning, which helps accurately capture the load variations. Lower threshold values approach the accuracy limit of the repartitioning method and can lead to insignificant improvements sensitive to load oscillations. Higher threshold values are close to the load imbalance of the lowest heterogeneity cases considered in the research.

In line 7, the heterogeneity of resources is evaluated by using different values of weights that result in subsets of different sizes after repartitioning. The new weight W_i^{new} is computed according to the runtime measured computational load L_i of the parallel process i:

$$W_i^{\text{new}} = W_i^{\text{old}} \cdot \left(2 - \frac{L_i}{L_{\text{avg}}}\right), \tag{2}$$

where W_i^{old} is the previous weight of process i. It is worth noting that these weights also consider variations in application load and system load on virtualized hardware. Sometimes load oscillations cause high variations in weight values, which can lead to drastic changes in topology and the size of subsets of particles after repartitioning. Thus, the previous weights help reduce significant oscillations that often occur in dynamic simulations on co-located resources. Usually, several applications of the repartitioning procedure are required for the computational load and weights to stabilize. In the case of over-subscribed nodes, when several containers are co-located on the same node and compete for the same resources, the algorithm evaluates the runtime measured computational load and reduces the values of relevant weights. Finally, RCB-based repartitioning adapts subsets of particles to heterogeneous resources according to the runtime measured load of application-specific computations.

In line 8, data of particles are transferred from the OpenCL device memory to the host memory for following repartitioning. In line 9, the RCB method from the Zoltan library is used to perform parallel repartitioning for distributed-memory architectures. In line 10, some particles migrate from the old partition to a new one, which requires internode communication handled by MPI. The number of redistributed particles depends on changes in load configuration. However, incremental partitions produced by RCB limit particle transfer between nodes. In line 11, the subroutine Registration_Ghost_Particles() defines ghost layers and registers ghost particles. In line 12, MPI processes exchange

information of ghost particles, which is necessary for internode communications performed after repartitioning. At the end of the repartitioning procedure, data of internal particles are transferred from the host memory to the OpenCL device memory for the main shared-memory computations.

In lines 15–17, the main communications are made before the main DEM computations. Ghost particle data are copied from the OpenCL device memory to the host memory and exchanged between neighboring partitions by MPI. The internode exchange of positions and velocities of particles is a popular approach in parallel DEM software [35] because it allows nodes to independently perform a contact search and computation of forces. It is worth noting that data of ghost particles, transferred by MPI from other nodes at each time step, also should be copied to the OpenCL device memory, which makes internode data exchange very expensive.

The main computational procedures of the DEM are implemented by using OpenCL kernels to run the same software on all shared-memory architectures, including CPUs and GPUs of various vendors. Main kernels are performed on the thread-per-particle basis, which takes advantage of the massive parallel computation capabilities of modern GPUs and can be considered to be the most suitable parallelism in the case of DEM computations. In lines 18–23 of the pseudocode, the main computational procedures of the DEM performed by the OpenCL device are outlined. The contact search, the contact history, the computation of contact forces and moments, the boundary conditions, the external gravitational force, and the time integrator are implemented as separate OpenCL kernels to avoid overfill of the private memory. In lines 24–27, at the end of the time step, the particle data can be copied from the OpenCL device memory to the host memory and stored on the hard disk drive in HDF5 format. It is recommended to transfer the data to the host memory as seldom as possible because it is a time-consuming process.

4. Results and Discussion

To measure the performance of the parallel DEM SaaS, a gravity-packing problem was solved on heterogeneous co-located resources of the OpenStack cloud. The gravity-packing problem was considered because it is commonly employed as a performance benchmark [33,46]. The solution domain was assumed to be a cubic container with 1.0 m long edges. Half of the domain was filled with 500,000 monosized particles, using a cubic structure. Then, 10,000 time steps equal to 1.0×10^{-8} s were performed, which resulted in the physical time interval of 0.0001 s. Physics of the considered problem with the skinning technique required performing repartitioning at each 1000 time steps. The size of the input data file with DEM parameters was negligibly small; therefore, the containers did not transfer data from the object storage in the performed benchmarks.

Six cases of heterogeneous or co-located resources were considered in the performed research. In Case 1, four containers, CN-6700-1, CN-6700-2, CN-6700-3, and CN-6700-4, with different numbers of cores were employed to make the benchmark. CN-6700-4 could perform four times more floating-point operations per second than CN-6700-1 in the case of CPU-bound applications. Thus, the computational performance of the fastest resource was significantly higher than that of the lowest resource. In Case 2, five faster CN-6700-4 containers were supplemented by two slower containers CN-4790-4. Thus, a higher number of faster resources worked together with a lower number of slower resources, but the difference in performance was not high. In Case 3, the powerful container CN-GPU with GPU was used with five CN-6700-4 containers. Thus, one resource, of which the computational performance was significantly higher than that of the others, worked together with a larger number of slower resources.

In the next three cases, each pair of containers was co-located on the same physical node. Parallel computations were carried out on several nodes, but only one of the containers co-located on each node was used by the MPI process. Synthetic stress tests were executed on another container co-located on the same node to investigate the performance decrease in parallel memory bandwidth bound computations on co-located containers. In

Case 4, five CN-6700-2 containers solved the considered problem on five nodes of the cloud infrastructure. Five other CN-6700-2 containers were co-located on the same nodes to run stress tests and to verify isolation of containers. The CPU stress test [56]

stress-ng –cpu 2 –cpu-method fft

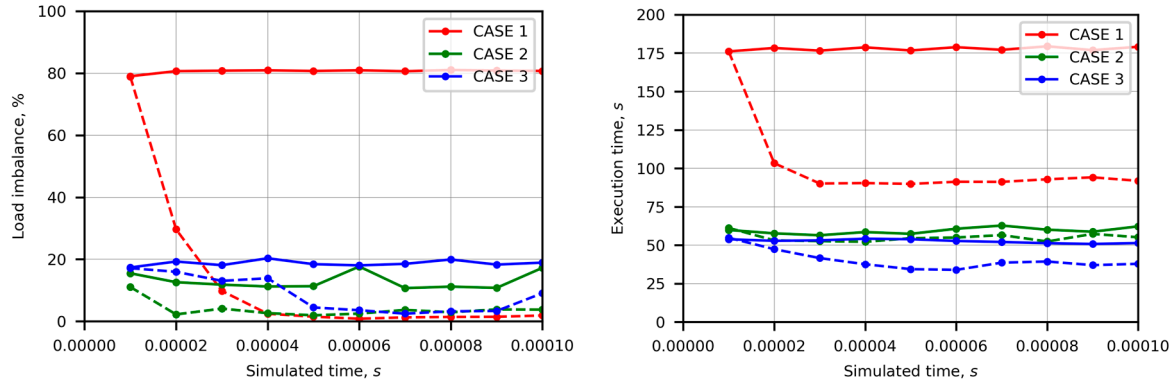
based on fast Fourier transform was performed on co-located containers. It was executed on a different number of containers to introduce different heterogeneities and to produce different load imbalances. In Case 5, the same configuration of containers was considered, but the synthetic memory stress test [56] was carried out on co-located containers. The memory stress test

stress-ng –vm 2 –vm-bytes 2G –vm-keep –m 2

started two workers, continuously calling mmap/munmap and writing 2 Gbytes to the allocated memory. In Case 6, the same configuration of containers was also considered, but the memory stress test was periodically executed on one co-located container. The periodically variable load was generated, interrupting the fixed periods of the memory stress test by the sleep command lasting 50 s. The periodically executed memory stress test produces the periodically peaking load on co-located containers, which corresponds to a real-world scenario. The ratio of the memory stress time to the whole simulation time, including sleep periods, expresses load frequency and indirectly represents the heterogeneity of co-located resources.

In the performed research, the benchmark execution time was considered as the main metric to evaluate the performance of the runtime adaptive repartitioning. A comparison of the execution time obtained by using the runtime adapted repartitioning with variable weights with that attained by using unweighted repartitioning revealed the gain of runtime adapted repartitioning. The percentage load imbalance measure (1) was used to determine load imbalance and indicate the heterogeneity of resources. The time evolution of the percentage load imbalance measure also showed how efficiently the runtime adapted repartitioning worked. Computational load, particle count, weights, wait time, and communication time can be examined to understand various issues of the repartitioning procedure.

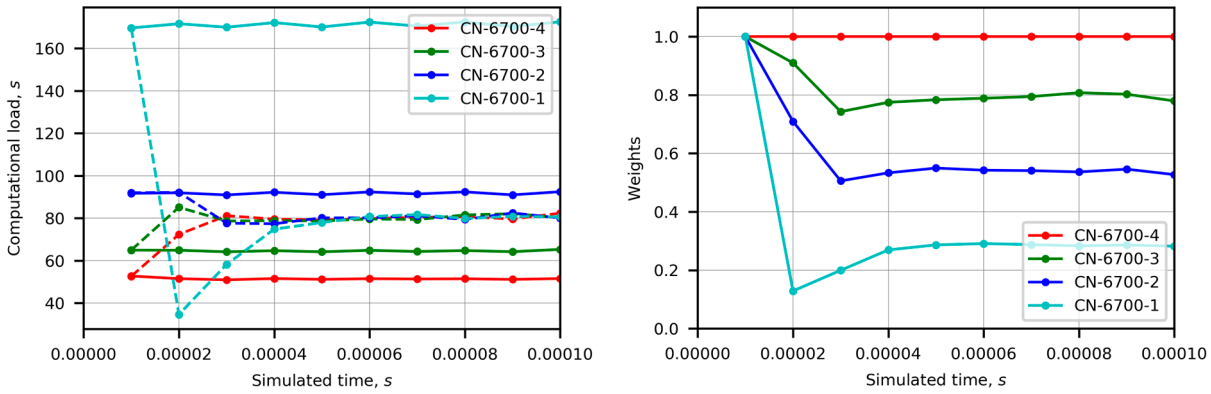
Figure 1 shows the time evolution of the load imbalance (Figure 1a) and execution time (Figure 1b) in the three first cases of heterogeneous cloud resources (Case 1, Case 2, and Case 3). The dashed curves represent the results of runtime adapted repartitioning with variable weights, while the solid lines represent that of unweighted repartitioning, which results in partitions of nearly equal size. In Case 1, load imbalance was very high because the theoretical performance of the fastest container CN-6700-4 was four times higher than that of the slowest container CN-6700-1. Runtime adapted repartitioning decreased the execution time up to 48.7% of the execution time obtained without using weights. In Case 2, the load imbalance of computations without weighting varied from 10.7% to 17.6%. The execution time was reduced from 4.9% to 6.7% of the execution time obtained without using weights. Thus, the low heterogeneity of resources indicated by low load imbalance percentage limited the gain of runtime adapted repartitioning. In Case 3, including the container CN-GPU, the load imbalance of computations without weighting was only slightly higher than that observed in Case 2. It varied from 18.0% to 20.3%, which was considerably lower than the load imbalance measured in Case 1. The decrease in the execution time up to 36.1% of the execution time obtained without weighting was observed, which was higher than the decrease obtained in Case 2 but lower than that attained in Case 1. Generally, the employed GPU performs DEM computations significantly faster than the CPU [43]; therefore, a higher load imbalance percentage as well as a gain in execution time can be expected. Thus, a detailed investigation of other measures and communication issues is required.



(**a**) (**b**)

Figure 1. Time evolution of the load imbalance (**a**) and execution time (**b**).

Figures 2 and 3 present the time evolution of additional measures, such as runtime measured computational load, weights, and particle count. In Figure 2a, the time evolution of the runtime measured computational load perfectly illustrates runtime adapted repartitioning in comparison with the unweighted repartitioning. Nearly straight solid lines, representing unweighted repartitioning, show four different computational loads. It is worth noting that the load of the slowest container CN-6700-1 was only 3.3 times larger than that of the fastest container CN-6700-4, which was specific to memory bandwidth bound applications. The sudden decrease in one curve indicates severe changes in subsets of particles after repartitioning, which is also confirmed by the variation in weights in Figure 2b. Three applications of the repartitioning procedure were required for the computational load of four containers to become nearly equal. In Figure 3, the different heterogeneity cases can be observed. Figure 3a shows the time evolution of particle count, which is directly related to the computational load, in Case 2. The lowest load imbalance (Figure 1a) led to the close values of computational load and execution time (Figure 1b). The observed oscillations of particle count uncovered the undesirable influence of chaotic variations in the system load and performance on the results of runtime adapted repartitioning.

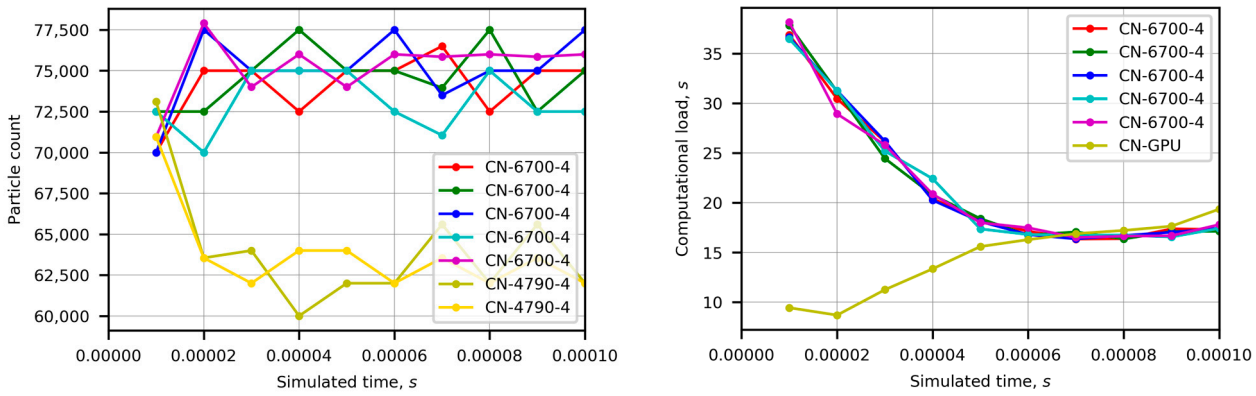


(**a**) (**b**)

Figure 2. Time evolution of the computational load (**a**) and weights (**b**) in Case 1.

Figure 3b shows the time evolution of the runtime measured computational load in Case 3. Initially, the load of any CN-6700-4 container is approximately 4 times larger than that of the powerful container CN-GPU. The difference in computational loads in Case 1 is lower than that in Case 3 despite the higher load imbalance of Case 1. This can be explained by the fact that the percentage load imbalance Formula (1) compares the maximal load with the averaged load. In Case 3, the load of any slower container serves as the maximal load, while the averaged load is highly influenced by five slower CN-6700-4 containers and

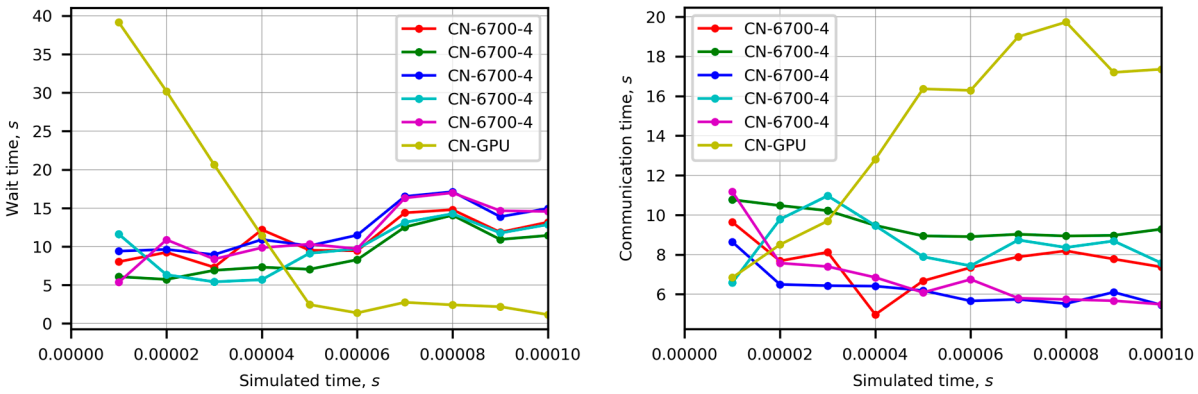
reduced by only one powerful container CN-GPU. Anyway, all curves of the computational load approached the average, which was closer to the initial load of the container CN-GPU than the initial loads of the CN-6700-4 containers. It is worth noting that Case 3 required more applications of the repartitioning procedure than Case 1 to make the computational load of all containers nearly equal.



(**a**) (**b**)

Figure 3. Time evolution of particle count in Case 2 (**a**) and computational load in Case 3 (**b**).

Figure 4 shows time evolution of wait time and communication time in Case 3. At the beginning of the simulation interval, the fastest container CN-GPU completed its computations and waited for the data. The runtime adapted weights increased the computational load of the container, which gradually decreased its wait time (Figure 4a). However, the increased number of processed particles caused the rise in communication time (Figure 4b). Communication times of slower containers differed from each other, which revealed partitioning challenges. In the time interval [0.00005, 0.00010] s, the container CN-GPU had so much data of ghost particles to send that other containers must wait for data longer than CN-GPU. In the time interval [0.00006, 0.00009] s, the most intensive communication of the container CN-GPU increased the wait time of other containers even more (Figure 4a), which caused the rise in the execution time observable in Figure 1b. Performing repartitioning, the RCB method does not directly optimize internode communication; therefore, communication issues can influence the load balance and overall performance.



(**a**) (**b**)

Figure 4. Time evolution of wait time (**a**) and communication time (**b**) in Case 3.

Figure 5a,b show total execution times of DEM SaaS, when the CPU stress test and memory stress test were performed on co-located containers in Case 4 and Case 5, respec-

tively. Stress tests were executed on one, two, three, four, or five co-located containers, which were represented by five different cases on the horizontal axes of charts (the abbreviations "Stress" with relevant numbering). In the legend of columns, the abbreviations "NOW", "RW", and "RW5" represent unweighted repartitioning, runtime adapted repartitioning, and runtime adapted repartitioning when weights were updated only if the load imbalance was higher than 5%, respectively. The condition of 5% was used to damp oscillations of repartitioning results caused by small variations in the system load and performance. No significant difference in the results of unweighted and runtime adapted repartitioning can be observed in Figure 5a. CPU cores of containers were well isolated; therefore, CPU stress tests did not influence the performance of benchmarks performed on co-located containers. In Figure 5b, the total execution time of benchmarks with runtime adapted repartitioning is substantially shorter than that of benchmarks with unweighted repartitioning. The memory bandwidth of co-located containers was not isolated; therefore, memory stress tests reduced the performance of the co-located containers, causing heterogeneity of resources. It can be observed that the largest gain in runtime adapted repartitioning was achieved in the case of one stressed container. No substantial gain can be observed in the case of five stressed containers, because the performance of all containers was nearly homogeneous in the case of the same memory stress test executed on all five co-located containers. It is worth noting that the condition of 5% reduced the total execution time in the most cases, but the obtained improvement did not exceed 2.5% of the total execution time measured without the condition of 5%.

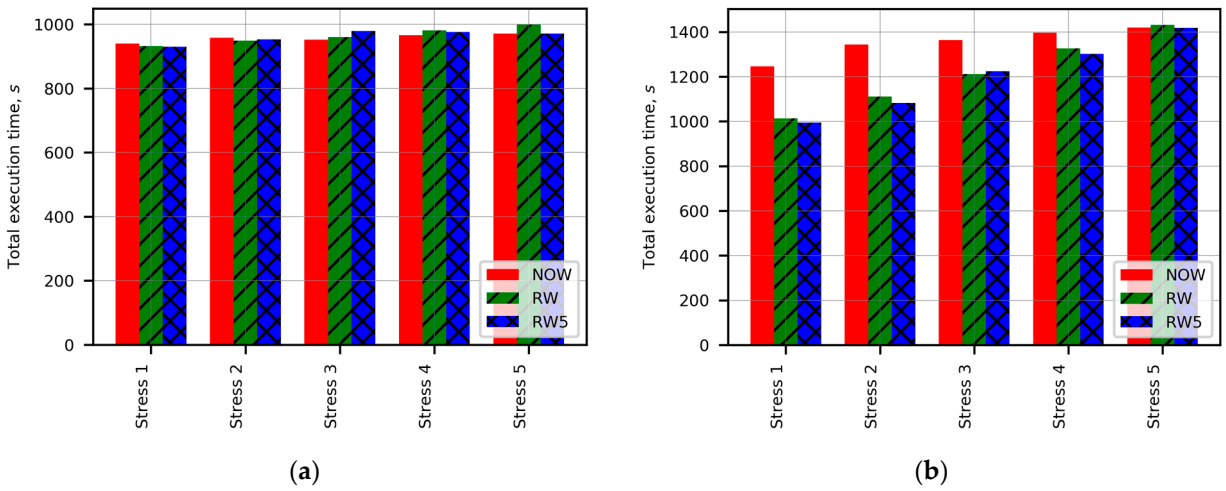


Figure 5. Total execution times of DEM SaaS on co-located resources: (**a**) benchmark co-located with the CPU stress test in Case 4; (**b**) benchmark co-located with the memory stress test in Case 5.

Figure 6 shows the time evolution of load imbalance when the memory stress test was performed on one (the solid curves) and four (the dashed curves) co-located containers in Case 5. In the legend, the abbreviations "NOW", "RW", and "RW5" again represent unweighted repartitioning, runtime adapted repartitioning, and runtime adapted repartitioning when weights were updated only if the load imbalance was higher than 5%, respectively. The load imbalance caused by one stressed co-located container was almost 4 times higher than that caused by four stressed co-located containers. Runtime adapted repartitioning required only one repartition to decrease the load imbalance up to 5%. In both stress cases, the additional condition of 5% perfectly reduced oscillations of the load imbalance and even slightly decreased the total execution time (Figure 5b).

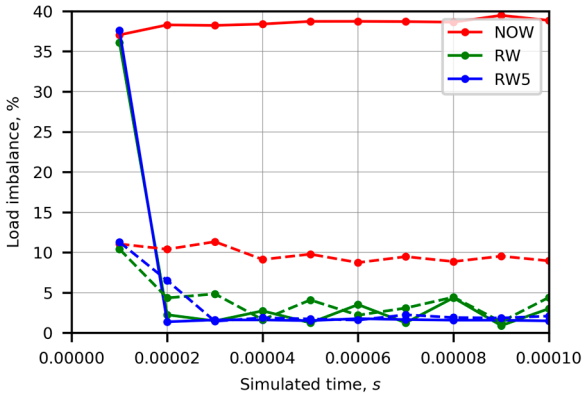


Figure 6. Time evolution of load imbalance when memory stress test was performed on one (the solid curves) and four (the dashed curves) co-located containers in Case 5.

Figure 7 presents the dependency of the total execution time on the load frequency ratio in Case 6. The markers show the load frequency ratios of performed experiments that are connected by curves to improve visuality. The black curve represents the unweighted repartitioning and serves as the reference. It shows how the growing heterogeneity of co-located resources increases the total execution time of benchmarks when runtime computed weights are not applied. The red curve represents the runtime adapted repartitioning. A comparison of these curves reveals how much the total execution time can be reduced, applying the runtime adapted repartitioning. In the case of the load frequency ratio equal to zero, the memory stress test was not executed, which led to homogeneous containers. When the load frequency ratio varied from 0 to 30% of the whole simulation time, the memory stress test caused low load imbalance. Therefore, the gain in the runtime adapted repartitioning was smaller than the computational costs introduced by weighting and less regular partitions. When the load frequency ratio was larger than 30%, the load imbalance was high enough, and the substantial gain in the runtime adapted repartitioning could be observed. In the case of the load frequency ratio equal to 100%, the sleep command was not performed, and the total execution time of Case 5 (Stress 1) was obtained. Thus, the runtime adapted repartitioning handled the periodically variable performance of non-isolated resources and decreased the total execution time of benchmarks on co-located containers.

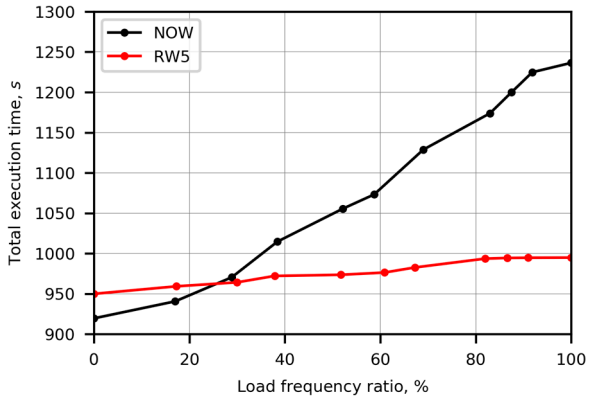


Figure 7. The dependency of the total execution time on the load frequency ratio in Case 6.

Because of the heterogeneity of non-isolated resources, the memory stress tests executed on one container substantially reduced the performance of parallel memory bandwidth bound DEM computations on the other container co-located on the same physical machine. To the best of our knowledge, there is no published research on the adaptation of parallel memory bandwidth bound applications to co-located cloud resources and relevant performance studies in the literature. In the present research, the developed

runtime adapted repartitioning handled the constant and periodically variable performance of non-isolated resources and decreased the total execution time of benchmarks on co-located containers.

Chaotic variation in the system load and performance can be treated as the main limitation of runtime adapted repartitioning in the case of low load imbalance. The application of updated weights only if load imbalance is higher than 5% damps some oscillations of repartitioning results, but it is still difficult to substantially reduce the execution time. The frequently variable spiky load on co-located containers can also lead to a similar effect, when the unresolved trade-off between the spike capturing accuracy and oscillation damping limits the gain in runtime adapted repartitioning. Highly heterogeneous resources result in partitions of highly different sizes, which can significantly increase communication time. The RCB method does not directly optimize internode communication; therefore, performance analysis of a graph partitioning method might be beneficial to overcome communication issues in the future research.

5. Conclusions

The paper presents the runtime adaptation of DEM SaaS based on the hybrid MPI/OpenCL parallelization to heterogeneous co-located resources of the OpenStack cloud. In three considered cases of heterogeneous cloud resources, including the container equipped by GPU, the load imbalance varied from 11% to 81%. The highest improvement in performance equal to 48.7% of the execution time obtained without weights was achieved by using the runtime adapted repartitioning in the case of the highest load imbalance, which indicated the highest heterogeneity of resources. The lowest load imbalance led to the close values of computational load, when small variations in the system load and performance caused oscillations in partitions of particles. Nevertheless, the runtime adapted repartitioning decreased the execution time up to 6.7% of the execution time obtained without weighting. Memory stress tests caused heterogeneity of non-isolated containers, which reduced the performance of memory bandwidth bound DEM SaaS on the co-located cloud resources. The decrease in the total execution time achieved by applying runtime adapted repartitioning varied from 6.7% to 20.1% of the total execution time obtained without using weights, which depended on load imbalance. The runtime adapted repartitioning handled the periodically variable performance of non-isolated resources and decreased the total execution time of DEM SaaS on co-located containers when the stress load frequency ratio was larger than 30% of the whole simulation time. In future research, the application of a graph partitioning method might help in overcoming communication issues and improving performance of the runtime adapted repartitioning on highly heterogeneous resources.

Author Contributions: Conceptualization, methodology, formal analysis, and investigation, O.B., R.P. and A.K.; software, O.B. and R.P.; writing—original draft preparation, A.K.; writing—review and editing, O.B. and R.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Khan, A.A.; Zakarya, M. Energy, Performance and Cost Efficient Cloud Datacentres: A Survey. *Comput. Sci. Rev.* **2021**, *40*, 100390. [CrossRef]
2. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing*; Technical Report SP 800-145; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2011. [CrossRef]
3. OpenStack. Available online: https://www.openstack.org/ (accessed on 1 December 2022).
4. Docker. Available online: https://www.docker.com/ (accessed on 1 December 2022).
5. Shirvani, M.H.; Talouki, R.N. A Novel Hybrid Heuristic-Based List Scheduling Algorithm in Heterogeneous Cloud Computing Environment for Makespan Optimization. *Parallel Comput.* **2021**, *108*, 102828. [CrossRef]

6. Devine, K.D.; Boman, E.G.; Karypis, G. Partitioning and Load Balancing for Emerging Parallel Applications and Architectures. In *Parallel Processing for Scientific Computing*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2006; Chapter 6, pp. 99–126. [CrossRef]
7. Mann, Z.Á. Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms. *ACM Comput. Surv.* **2015**, *48*, 1–34. [CrossRef]
8. Resource Isolation Based on the L3 Cache and MBA. Available online: https://www.intel.com/content/www/us/en/developer/articles/technical/use-intel-resource-director-technology-to-allocate-memory-bandwidth.html. (accessed on 1 December 2022).
9. Use Intel®Resource Director Technology to Allocate Memory Bandwidth. Available online: https://www.alibabacloud.com/help/en/container-service-for-kubernetes/latest/use-the-l3-cache-and-mba-to-improve-the-resource-isolation-of-tasks-with-different-priorities. (accessed on 1 December 2022).
10. Yun, H.; Yao, G.; Pellizzoni, R.; Caccamo, M.; Sha, L. Memory Bandwidth Management for Efficient Performance Isolation in Multi-Core Platforms. *IEEE Trans. Comput.* **2016**, *65*, 562–576. [CrossRef]
11. Sakellari, G.; Loukas, G. A Survey of Mathematical Models, Simulation Approaches and Testbeds Used for Research in Cloud Computing. *Simul. Modell. Pract. Theory* **2013**, *39*, 92–103. [CrossRef]
12. Cundall, P.A.; Strack, O.D.L. A Discrete Numerical Model for Granular Assemblies. *Géotechnique* **1979**, *29*, 47–65. [CrossRef]
13. Kačeniauskas, A.; Pacevič, R.; Starikovičius, V.; Maknickas, A.; Staškūnienė, M.; Davidavičius, G. Development of Cloud Services for Patient-Specific Simulations of Blood Flows through Aortic Valves. *Adv. Eng. Softw.* **2017**, *103*, 57–64. [CrossRef]
14. Chae, M.; Lee, H.; Lee, K. A Performance Comparison of Linux Containers and Virtual Machines Using Docker and KVM. *Clust. Comput.* **2019**, *22*, 1765–1775. [CrossRef]
15. Potdar, A.M.; Narayan, D.G.; Kengond, S.; Mulla, M.M. Performance Evaluation of Docker Container and Virtual Machine. *Procedia Comput. Sci.* **2020**, *171*, 1419–1428. [CrossRef]
16. Papazachos, Z.C.; Karatza, H.D. Performance Evaluation of Bag of Gangs Scheduling in a Heterogeneous Distributed System. *J. Syst. Softw.* **2010**, *83*, 1346–1354. [CrossRef]
17. Moschakis, I.A.; Karatza, H.D. Evaluation of Gang Scheduling Performance and Cost in a Cloud Computing System. *J. Supercomput.* **2010**, *59*, 975–992. [CrossRef]
18. Hao, Y.; Liu, G.; Hou, R.; Zhu, Y.; Lu, J. Performance Analysis of Gang Scheduling in a Grid. *J. Netw. Syst. Manag.* **2014**, *23*, 650–672. [CrossRef]
19. Mohammadi, M.; Bazhirov, T. Comparative Benchmarking of Cloud Computing Vendors with High Performance Linpack. In Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications, Hongkong, China, 15–17 March 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 1–5. [CrossRef]
20. Shahid, M.A.; Alam, M.M.; Su'ud, M.M. Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing. *Appl. Sci.* **2023**, *13*, 1586. [CrossRef]
21. Heidari, A.; Navimipour, N.J.; Jamali, M.A.J.; Akbarpour, S. A green, secure, and deep intelligent method for dynamic IoT-edge-cloud offloading scenarios. *Sustain. Comput.: Inform. Syst.* **2023**, *38*, 100859. [CrossRef]
22. Bojato, J.; Donado, D.; Jimeno, M.; Moreno, G.; Villanueva-Polanco, R. Password Guessability as a Service (PGaaS). *Appl. Sci.* **2022**, *12*, 1562. [CrossRef]
23. Amiri, Z.; Heidari, A.; Navimipour, N.J.; Mehmet, U. Resilient and dependability management in distributed environments: A systematic and comprehensive literature review. *Cluster. Comput.* **2023**, *26*, 1565–1600. [CrossRef]
24. Washington, D.W.; Meegoda, J.N. Micro-Mechanical Simulation of Geotechnical Problems Using Massively Parallel Computers. *Int. J. Numer. Anal. Methods Geomech.* **2003**, *27*, 1227–1234. [CrossRef]
25. Kačeniauskas, A.; Kačianauskas, R.; Maknickas, A.; Markauskas, D. Computation and Visualization of Discrete Particle Systems on gLite-Based Grid. *Adv. Eng. Softw.* **2011**, *42*, 237–246. [CrossRef]
26. Wang, S.; Luo, K.; Yang, S.; Hu, C.; Fan, J. Parallel LES-DEM Simulation of Dense Flows in Fluidized Beds. *Appl. Therm. Eng.* **2017**, *111*, 1523–1535. [CrossRef]
27. Owen, D.; Feng, Y. Parallelised Finite/Discrete Element Simulation of Multi-Fracturing Solids and Discrete Systems. *Eng. Comput.* **2001**, *18*, 557–576. [CrossRef]
28. Walther, J.H.; Sbalzarini, I.F. Large-Scale Parallel Discrete Element Simulations of Granular Flow. *Eng. Comput.* **2009**, *26*, 688–697. [CrossRef]
29. Karypis, G.; Kumar, V. Parallel Multilevel Series K-Way Partitioning Scheme for Irregular Graphs. *SIAM Rev.* **1999**, *41*, 278–300. [CrossRef]
30. Markauskas, D.; Kačeniauskas, A. The Comparison of Two Domain Repartitioning Methods Used for Parallel Discrete Element Computations of the Hopper Discharge. *Adv. Eng. Softw.* **2015**, *84*, 68–76. [CrossRef]
31. Berger, M.; Bokhari, S. A Partitioning Strategy for Nonuniform Problems on Multiprocessors. *IEEE Trans. Comput.* **1987**, *36*, 570–580. [CrossRef]
32. Liu, H.; Tafti, D.K.; Li, T. Hybrid Parallelism in MFIX CFD-DEM Using OpenMP. *Powder Technol.* **2014**, *259*, 22–29. [CrossRef]
33. Berger, R.; Kloss, C.; Kohlmeyer, A.; Pirker, S. Hybrid Parallelization of the LIGGGHTS Open-Source DEM Code. *Powder Technol.* **2015**, *278*, 234–247. [CrossRef]
34. Cintra, D.T.; Willmersdorf, R.B.; Lyra, P.R.M.; Lira, W.W.M. A Hybrid Parallel DEM Approach with Workload Balancing Based on HSFC. *Eng. Comput.* **2016**, *33*, 2264–2287. [CrossRef]

35. Incardona, P.; Leo, A.; Zaluzhnyi, Y.; Ramaswamy, R.; Sbalzarini, I.F. OpenFPM: A Scalable Open Framework for Particle and Particle-Mesh Codes on Parallel Computers. *Comput. Phys. Commun.* **2019**, *241*, 155–177. [CrossRef]
36. Yan, B.; Regueiro, R.A. Comparison between Pure MPI and Hybrid MPI-OpenMP Parallelism for Discrete Element Method (DEM) of Ellipsoidal and Poly-Ellipsoidal Particles. *Comput. Part. Mech.* **2018**, *6*, 271–295. [CrossRef]
37. Du, P.; Weber, R.; Luszczek, P.; Tomov, S.; Peterson, G.; Dongarra, J. From CUDA to OpenCL: Towards a Performance-Portable Solution for Multi-Platform GPU Programming. *Parallel Comput.* **2012**, *38*, 391–407. [CrossRef]
38. Govender, N. Study on the Effect of Grain Morphology on Shear Strength in Granular Materials via GPU Based Discrete Element Method Simulations. *Powder Technol.* **2021**, *387*, 336–347. [CrossRef]
39. Kelly, C.; Olsen, N.; Negrut, D. Billion Degree of Freedom Granular Dynamics Simulation on Commodity Hardware via Heterogeneous Data-Type Representation. *Multibody Sys. Dyn.* **2020**, *50*, 355–379. [CrossRef]
40. Xu, J.; Qi, H.; Fang, X.; Lu, L.; Ge, W.; Wang, X.; Xu, M.; Chen, F.; He, X.; Li, J. Quasi-Real-Time Simulation of Rotating Drum Using Discrete Element Method with Parallel GPU Computing. *Particuology* **2011**, *9*, 446–450. [CrossRef]
41. Tian, Y.; Zhang, S.; Lin, P.; Yang, Q.; Yang, G.; Yang, L. Implementing Discrete Element Method for Large-Scale Simulation of Particles on Multiple GPUs. *Comput. Chem. Eng.* **2017**, *104*, 231–240. [CrossRef]
42. Gan, J.; Evans, T.; Yu, A. Application of GPU-DEM Simulation on Large-Scale Granular Handling and Processing in Ironmaking Related Industries. *Powder Technol.* **2020**, *361*, 258–273. [CrossRef]
43. Pacevič, R.; Kačeniauskas, A. The Performance Analysis of the Thermal Discrete Element Method Computations on the GPU. *Comput. Inform.* **2022**, *41*, 931–956. [CrossRef]
44. Combier, R. EDEM Now Available on Rescale's Cloud Simulation Platform. Available online: https://rescale.com/blog/edem-now-available-on-rescales-cloud-simulation-platform/ (accessed on 7 April 2023).
45. Weinhart, T.; Orefice, L.; Post, M.; van Schrojenstein Lantman, M.P.; Denissen, I.F.; Tunuguntla, D.R.; Tsang, J.; Cheng, H.; Shaheen, M.Y.; Shi, H.; et al. Fast, Flexible Particle Simulations—An Introduction to MercuryDPM. *Comput. Phys. Commun.* **2020**, *249*, 107129. [CrossRef]
46. Bystrov, O.; Pacevič, R.; Kačeniauskas, A. Performance of Communication- and Computation-Intensive SaaS on the OpenStack Cloud. *Appl. Sci.* **2021**, *11*, 7379. [CrossRef]
47. Danovaro, E.; Clematis, A.; Galizia, A.; Ripepi, G.; Quarati, A.; D'Agostino, D. Heterogeneous architectures for computational intensive applications: A cost-effectiveness analysis. *J. Comput. Appl. Math.* **2014**, *270*, 63–77. [CrossRef]
48. Araki, S.J.; Martin, R.S. Dynamic Load Balancing with over Decomposition in Plasma Plume Simulations. *J. Parallel Distrib. Comput.* **2022**, *163*, 136–146. [CrossRef]
49. Barreiros, W.; Melo, A.C.; Kong, J.; Ferreira, R.; Kurc, T.M.; Saltz, J.H.; Teodoro, G. Efficient Microscopy Image Analysis on CPU-GPU Systems with Cost-Aware Irregular Data Partitioning. *J. Parallel Distrib. Comput.* **2022**, *164*, 40–54. [CrossRef]
50. Zhong, Z.; Rychkov, V.; Lastovetsky, A. Data Partitioning on Multicore and Multi-GPU Platforms Using Functional Performance Models. *IEEE Trans. Comput.* **2015**, *64*, 2506–2518. [CrossRef]
51. Bystrov, O.; Kačeniauskas, A.; Pacevič, R.; Starikovičius, V.; Maknickas, A.; Stupak, E.; Igumenov, A. Performance Evaluation of Parallel Haemodynamic Computations on Heterogeneous Clouds. *Comput. Inform.* **2020**, *39*, 695–723. [CrossRef]
52. Devine, K.; Boman, E.; Heaphy, R.; Hendrickson, B.; Vaughan, C. Zoltan Data Management Services for Parallel Dynamic Applications. *Comput. Sci. Eng.* **2002**, *4*, 90–96. [CrossRef]
53. Tumonis, L.; Schneider, M.; Kačianauskas, R.; Kačeniauskas, A. Comparison of Dynamic Behaviour of EMA-3 Railgun under Differently Induced Loadings. *Mechanika* **2009**, *78*, 31–37.
54. Liu, G.; Marshall, J.S.; Li, S.Q.; Yao, Q. Discrete-Element Method for Particle Capture by a Body in an Electrostatic Field. *Int. J. Numer. Methods Eng.* **2010**, *84*, 1589–1612. [CrossRef]
55. Norouzi, H.R.; Zarghami, R.; Sotudeh-Gharebagh, R.; Mostoufi, N. *Coupled CFD-DEM Modeling: Formulation, Implementation and Application to Multiphase Flows*; John Wiley & Sons: Chichester, UK, 2016; ISBN 978-1-119-00513-1. [CrossRef]
56. Stress-ng. A Tool to Load and Stress a Computer System. Available online: https://github.com/ColinIanKing/stress-ng (accessed on 1 February 2023).