Article

Performance of Communication- and Computation-Intensive SaaS on the OpenStack Cloud

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, Institute of Applied Computer Science, Vilnius Gediminas Technical University, 10223 Vilnius, Lithuania

* Correspondence: arnas.kaceniauskas@vilniustech.lt; Tel.: +370-5-274-4913

Featured Application: The presented performance analysis helps in evaluating the infrastructure overhead and efficiently running communication- and computation-intensive MPI-based SaaS on clouds.

Abstract: The pervasive use of cloud computing has led to many concerns, such as performance challenges in communication- and computation-intensive services on virtual cloud resources. Most evaluations of the infrastructural overhead are based on standard benchmarks. Therefore, the impact of communication issues and infrastructure services on the performance of parallel MPI-based computations remains unclear. This paper presents the performance analysis of communication- and computation-intensive software based on the discrete element method, which is deployed as a service (SaaS) on the OpenStack cloud. The performance measured on KVM-based virtual machines and Docker containers of the OpenStack cloud is compared with that obtained by using native hardware. The improved mapping of computations to multicore resources reduced the internode MPI communication by 34.4% and increased the parallel efficiency from 0.67 to 0.78, which shows the importance of communication issues. Increasing the number of parallel processes, the overhead of the cloud infrastructure increased to 13.7% and 11.2% of the software execution time on native hardware in the case of the Docker containers and KVM-based virtual machines of the OpenStack cloud, respectively. The observed overhead was mainly caused by OpenStack service processes that increased the load imbalance of parallel MPI-based SaaS.

Citation: Bystrov, O.; Pacevič, R.; Kačeniauskas, A. Performance of Communication- and Computation-Intensive SaaS on the OpenStack Cloud. *Appl. Sci.* **2021**, *11*, 7379. <https://doi.org/10.3390/app11167379>

Academic Editor: Eui-Nam Huh

Received: 2 July 2021

Accepted: 9 August 2021

Published: 11 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: OpenStack cloud; KVM; Docker; MPI; performance; discrete element method software

1. Introduction

Rapid developments in computing and communication technologies have led to the emergence of a distributed computing paradigm called cloud computing, which, due to its on-demand nature, low cost, and offloaded management, has become a natural solution to the problem of expanding computational needs [1]. The term “cloud” is an acronym for common, location-independent, online utility provisioned on-demand. The capabilities of different applications are exposed as sophisticated services that can be accessed over a network. Generally, cloud providers offer different types of services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud infrastructures provide platforms and tools for building IT services at more affordable prices compared to the prices of traditional computing techniques.

Organizations can use different implementations of cloud software for deploying their own private clouds. OpenStack is an open-source cloud management platform that delivers an integrated foundation to create, deploy, and scale a secure and reliable public or private cloud [2]. The compute service Nova, object storage service Swift, and image service Glance are the main parts of OpenStack. Another open-source local cloud framework is

Eucalyptus [3], provided by Eucalyptus Systems, Inc (Santa Barbara, CA, USA). A typical Eucalyptus cloud is composed of a front-end cloud controller, a persistent storage controller, a virtual machine image repository, a cluster controller, and many compute nodes.

Cloud computing makes extensive use of virtual machines (VMs) because they allow workloads to be isolated and resource usage to be controlled. Xen is primarily a bare-metal type 1 hypervisor that can be directly installed in the computer hardware without the need for a host operating system [4]. Xen directly controls, monitors, and manages the hardware, peripheral, and I/O resources. Kernel Virtual Machine (KVM) [4] is a feature of Linux that allows Linux to act as a type 1 hypervisor, running an unmodified guest operating system inside a Linux process. KVM uses full virtualization for Linux on x86 hardware. KVM is designed and optimized to take full advantage of the hardware-assisted virtualization extensions. Furthermore, KVM is widely considered the de facto standard for OpenStack and the virtualization management API libvirt [5]. The Nova libvirt driver manages the communication between OpenStack and KVM.

Containers present an emerging technology for improving the productivity and code portability of cloud infrastructures. Rather than running a full operating system on virtual hardware, container-based virtualization uses an existing operating system and provides extra isolation. Container runtimes, such as LXC [6], Singularity [7], and Docker [8], largely abstract away the differences between the many operating systems that users run. In many cases, due to the layered file system, Docker container images require less disk space and I/O than the equivalent VM disk images. Thus, Docker has emerged as a standard runtime, image format, and build system for Linux containers. The Lawrence Berkeley National Laboratory has developed Singularity [7], a novel container technology with a format different from that of the Docker image. Singularity is designed to integrate well with resource managers of HPC systems. It also focuses on container mobility and can be run in any workload with no modification to any host [9]. HPC vendors have also begun integrating native support for containers. For example, IBM has added Docker container integration to Platform LSF to run the containers on an HPC cluster [10]. This allows the containers to be executed on an LSF-managed cluster similar to a conventional job but with a fully self-contained environment. UberCloud application software containers provide ANSYS Fluids and Structures software for execution on the Microsoft Azure cloud [11]. EDEM software has been deployed on Rescale's cloud simulation platform for high-performance computations [12]. Astyrakakis et al. [13] proposed a tool for the automatic validation of cloud-native applications, which provides insightful suggestions on how to enhance the performance and cloud-native characteristics of the validated applications. However, it is difficult to provide precise guidelines regarding the optimal cloud platform and virtualization technology for each type of research and application [1].

Deployment of scientific codes as software services for data preparation, high-performance computation, and visualization on the cloud infrastructure increases the mobility of users and achieves better exploitation because clouds feature flexible management of resources. Thus, flexible cloud infrastructures and software services are perceived as a promising avenue for future advances in the multidisciplinary area of discrete element method (DEM) applications [14]. However, the cloud SaaS might suffer from severe performance degradation due to higher latencies of networks, virtualization overheads, and other issues [15]. Cloud computing still lacks in terms of case studies and quantitative comparison of performance in the case of specific applications of granular materials and particle technology. Most evaluations of the virtualization overhead and performance of cloud services are based on standard benchmarks [1]; therefore, the impact of the virtualization and processes of the cloud infrastructure on the performance of communication- and computation-intensive MPI-based DEM computations [16] remains unclear. Moreover, performance is a critical factor in deciding whether cloud infrastructures are viable for scientific DEM software.

The performance of virtual machines and lightweight containers has already received some attention in the academic literature because they are crucial components of the overall cloud performance. Walters et al. [17] compared the overheads of VMWare Server, Xen,

and OpenVZ. For OpenMP runs, the performance of Xen and OpenVZ was close to that obtained on native hardware, but VMware produced a large overhead. Macdonnell and Lu [18] measured the performance of the VMWare virtualization platform for a variety of common scientific computations. The overhead for computation-intensive tasks was around 6%. Kačeniauskas et al. [19] assessed the performance of the private cloud infrastructure and virtual machines of KVM by testing the CPU, memory, hard disk drive, network, and software services for medical engineering. The measured performance of the virtual resources was close to the performance of the native hardware when measuring only the memory bandwidth and disk I/O. Kozhircbayev et al. [20] presented an overview of the performance evaluation of virtual machines and Docker containers in terms of CPU performance, memory throughput, disk I/O, and operation speed measurement. Felter et al. [21] also looked at the performance differences of non-scientific software within virtualized and containerized environments. Estrada et al. [22] executed genomic workloads on the KVM hypervisor, the Xen para-virtualized hypervisor, and LXC containers. Xen and Linux containers exhibited near-zero overhead. Chae et al. [23] compared the performance of Xen, KVM, and Docker in three different ways: the CPU and memory usage of the host; idleness of the CPU, memory usage, and I/O performance on migrating a large file; and the performance of the web server through JMeter. In [24] the performance of software services developed for hemodynamic computations was measured on Xen hardware virtual machines, KVM-based virtual machines, and Docker containers and compared with the performance achieved by using native hardware. Kominos et al. [25] used synthetic benchmarks to empirically evaluate the overheads of bare-metal-, virtual-machine- and Docker-container-based hosts of OpenStack. Docker-container-based hosts had the fastest boot time and the best overall performance, with the exception of network bandwidth. Potdar et al. [26] evaluated the performance of Docker containers and VMs using standard benchmark tools, such as Sysbench, Phoronix, and Apache benchmark, in terms of CPU performance, memory throughput, storage read/write performance, load test, and operation speed measurement. Ventre et al. [27] investigated the performance of the instantiation process of micro-virtualized network functions for open-source virtual infrastructure managers, such as OpenStack Nova and Nomad, on the Xen virtualization platform. The source codes of virtual infrastructure managers were modified to reduce instantiation times. Shah et al. [28] evaluated the performance of VMs and containers for the HEPSCPEC06 benchmark. The results showed that hyperthreading, isolation of CPU cores, proper numbering, and allocation of vCPU cores improve the performance of VMs and containers on the OpenStack cloud. Most of these studies [18–24] have found negligible performance differences between a container and native hardware. However, none of the studies includes the performance analysis of the virtualized distributed memory architectures for communication- and computation-intensive MPI-based computations.

Han et al. [29] performed MPI-based NAS benchmarks on Xen and found that the measured overhead increases when more cores are added. Jackson et al. [30] ran MPI-based applications and observed significantly degraded performance on EC2. Commercial testbeds are naturally realistic, but they cannot provide scientists with dependable experiments and enough control. A study [31] on the use of container-based virtualization in HPC revealed that Xen is slower than LXC by roughly a factor of 2, while a native server and LXC have near-identical performances. However, the influence of cloud or other infrastructure services on the load imbalance of MPI-based computations was not investigated. Hale et al. [32] showed that the performance of Docker containers when using the system MPI library for a parallel solution of Poisson's equation carried out by FEniCS software is comparable to the native performance. However, the influence of the communication and domain decomposition issues on the speedup of parallel computations was not investigated. Mohammadi et al. [33] evaluated the High-Performance Linpack benchmark on cloud computing infrastructures managed by Amazon Web Services, Microsoft Azure, Rackspace, and IBM SoftLayer. The obtained results demonstrated that Microsoft Azure H-instances with a low-latency Infiniband interconnect network delivers

the highest speedup. Moreover, the performance per computing core on the public cloud could be comparable to modern traditional supercomputing systems. Ly et al. [34] proposed a communication-aware worst-fit decreasing heuristic algorithm for the container placement problem, but MPI-based applications were not considered. Reddy and Lastovetsky [35] formulated a bi-objective optimization problem for performance and energy for data-parallel applications on homogeneous clusters. Bystrov et al. [36] investigated a trade-off between the computing speed and the consumed energy of a real-life hemodynamic application on a heterogeneous cloud. Parallel speedups obtained by using several domain decomposition methods were compared, but load balance and communication issues were not explored. The influence of communications on the speedup of parallel computations on clouds and the influence of cloud infrastructure services on the load imbalance and overall performance of parallel MPI-based computations have not been investigated in the discussed research.

This paper describes the performance analysis of the communication- and computation-intensive discrete element method SaaS on virtual resources of the OpenStack cloud infrastructure. The research examined the influence of communication issues and infrastructure services on SaaS performance, which can be dependent on the considered software and algorithmic aspects. Information provided by the synthetic benchmarks usually performed on clouds does not include all important factors, and it is not sufficient for finding the best infrastructure setup. Therefore, application-specific tests need to be performed before production runs in order to optimize the parallel performance of the communication- and computation-intensive SaaS. The remaining paper is organized as follows: Section 2 describes discrete element method software, Section 3 presents the hosted cloud infrastructure and developed software services, Section 4 presents the parallel performance analysis of communication issues and overheads of cloud services, and the conclusions are given in Section 5.

2. Discrete Element Method Software

The discrete element or discrete particle method is considered a powerful numerical technique to understand and model granular materials [37]. However, advanced DEM models can be effectively applied to study heat transfer [38], acoustic agglomeration [39], and coupled multi-physical problems [40].

2.1. Considered Model of DEM

In this work, the employed DEM software models the non-cohesive frictional visco-elastic particle systems. The dynamic behavior of a discrete system is described by considering the motion and deformation of the interacting individual particles within the framework of Newtonian mechanics. An arbitrary particle is characterized by three translational and three rotational degrees of freedom. The forces acting on the particle may be classified into the forces induced by external fields and the contact forces between the particles in contact. This work considers the force of gravity but not the aerodynamic force [41], the electrostatic force [42], or other external forces. The normal contact force can be expressed as the sum of the elastic and viscous components. In this work, the normal elastic force is computed according to Hertz's contact model. The viscous counterpart of the contact force linearly depends on the relative velocity of the particles at the contact point. It is considered that the tangential contact force is only based on the dynamic friction force, which is directly proportional to the normal component of the contact force. The employed force model is history-independent and, therefore, requires only knowledge of the current kinematic state. It is sufficient to solve many applications of granular materials [43]. Moreover, the considered model is convenient for the investigation of communication issues because the size of the transferred data does not depend on the variable number of contacts. The details of the applied DEM model can be found in [43,44].

2.2. Parallel Implementation

The employed DEM software was developed using the C++ programming language. The GNU compiler collection (GCC) was used with the second-level optimization option for compiling the code. In this study, CPU-time-consuming computational procedures, such as contact detection, contact force computation, and time integration, were implemented using standard algorithms, widely available in open-source codes [45], to increase the usability of obtained results. Contact detection was based on the simple and fast implementation of a cell-based algorithm [46]. The explicit velocity Verlet algorithm [46] was used for time integration.

The long computational time of DEM simulations limits the analysis of industrial-scale applications. The selection of an efficient parallel solution algorithm depends on the specific characteristics of the considered problem and the numerical method used [44–47]. The parallel DEM algorithms differ from the analogous parallel processing in the continuum approach. Moving particles dynamically change the workload configuration, making parallelization of DEM software much more difficult and challenging. Domain decomposition is considered one of the most efficient coarse-grain strategies for scientific and engineering computations; therefore, it was implemented in the developed DEM code [16,44]. The recursive coordinate bisection (RCB) method from the Zoltan library [48] was used for domain partitioning because it is highly effective for particle simulations [16,48]. The RCB method recursively divides the computational domain into nearly equal subdomains by cutting planes orthogonal to the coordinate axes, according to particle coordinates and workload weights. This method is attractive as a dynamic load-balancing algorithm because it implicitly produces incremental partitions and reduces data transfer between processors caused by repartitioning. Interprocessor communication was implemented in the DEM code by subroutines of the message passing library MPI.

The main CPU-time-consuming computational procedures of the DEM code are contact detection, contact force computation, and time integration. Each processor computes the forces and updates the positions of particles only in its subdomain. To perform their computations, the processors need to share information about particles that are near the division boundaries in ghost layers. A small portion of communications is performed when processors exchange particles as the particles move from one subdomain to another. This communication is optional and is performed only in the case of a non-zero number of exchanging particles. The main portion of communications is performed prior to performing contact detection and contact force computation. In the present implementation, particle data from the ghost layers are exchanged between neighboring subdomains. The exchange of positions and velocities of particles between MPI processes is a common strategy often used in DEM codes [45], but an alternative based on transferring computed forces also exists. Despite its local character, interprocessor particle data transfer requires a significant amount of time and reduces the parallel efficiency of computations. The size of ghost layers can depend on the particle size, particle flow, and implemented algorithms. Therefore, in this study, ghost layers of different sizes were considered in order to study communication issues.

3. OpenStack Cloud Infrastructure and Services

The university's private cloud infrastructure based on OpenStack is hosted in the Vilnius Gediminas Technical University. The cloud system architecture consists of several layers of cloud services deployed on the virtualized hardware. The NIST SPI model [49] represents a layered, high-level abstraction of cloud services classified into three main categories (Figure 1): Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). Higher-level services for developers and users are deployed on top of the IaaS layer managed by the OpenStack Train 2019 version [2]. The deployed capabilities of the OpenStack cloud include the compute service Nova version 20.0.0, compute service Zun version 3.0.1, networking service Neutron version 15.0.0, container network plug-in Kuryr version 3.0.1, image service Glance version 19.0.0, identity service Keystone version 16.0.0, object storage service Swift version 2.22.0, and block storage service Cinder

version 15.0.0. Nova automatically deploys the provisioned virtual compute instances (VMs), Zun launches and manages containers, Swift provides redundant storage of static objects, Neutron manages virtual network resources, Kuryr connects containers to Neutron, Keystone is responsible for authentication and authorization, and Glance provides service discovery, registration, and delivery for virtual disk images.

Figure 1. Layers of cloud services.

In the present cloud infrastructure, two alternatives of the virtualization layer are implemented to gain more flexibility and efficiency in resource configuration. Version 2.11.1 of QEMU-KVM is used for virtual machines (VMs) deployed and managed by Nova. Alternatively, Docker version 19.03.6 containers launched and managed by Zun create an abstraction layer between computing resources and the services using them. The containers and VMs have the following characteristics: 4 CPUs, 31.2 GB RAM, 80 GB HDD, and Ubuntu 18.04 LTS (Bionic Beaver).

In terms of architecture, the cloud testbed is composed of nodes hosting OpenStack services and compute nodes hosting the virtual machines and containers connected to a 1 Gbps Ethernet LAN via 3COM Baseline Switch 2928-SFP Plus. The OpenStack services are installed on four dedicated nodes free of another load. Ubuntu 18.04 LTS (Bionic Beaver) is installed in the compute nodes. The hardware characteristics of nodes hosting the virtual machines and containers are as follows: Intel® Core i7-6700 3.40 GHz CPU, 32 GB DDR4 2133 MHz RAM, and 1 TB HDD.

The layers of deployed cloud services are shown in Figure 1. The OpenStack cloud IaaS provides platforms (PaaS) to develop and deploy software services called SaaS (Figure 1). The cloud infrastructure is managed by the OpenStack API, which provides access to infrastructure services. The PaaS layer supplies engineering application developers with programming-language-level environments and compilers, such as GNU compiler collection (GCC), for the development of DEM software using the C++ programming language. Parallel software for distributed memory systems is developed using the Open MPI platform, which includes the open-source implementation of the MPI standard for message passing. The development platform as a service for domain decomposition and dynamic load balancing is provided based on the Zoltan library [48]. It simplifies the load-balancing and data movement difficulties that arise in dynamic simulations. The Visualization Toolkit (VTK) [50] is deployed as the platform for developing visualization software. VTK applications are platform-independent, which is attractive for heterogeneous cloud architectures.

The SaaS layer contains software services deployed on top of the provided platforms (Figure 1). The DEM SaaS was developed using the C++ programming language (GNU GCC PaaS), the message passing library Open MPI, and the Zoltan library. The communication- and computation-intensive DEM SaaS is used to solve applications of

granular materials and particle technology, such as hopper discharge, avalanche flow, and powder compaction. Computational results are visualized using the cloud visualization service VisLT [51]. The visualization SaaS is developed using the VTK platform. VisLT is supplemented with the developed middleware component, which can reduce the communication between different parts of the cloud infrastructure. The environment launchers are designed for users to configure the SaaS and define custom settings. After successful authorization, the user can define configuration parameters and run the SaaS on ordered virtual resources.

4. Results and Discussion

This study aimed to investigate the performance of the developed DEM SaaS for discrete element method computations of granular materials on KVM-based VMs and Docker containers managed by the OpenStack cloud infrastructure. The parallel performance of the developed DEM SaaS was evaluated by measuring the speedup S_p and the efficiency E_p :

$$S_p = \frac{t_1}{t_p}, E_p = \frac{S_p}{p}, \quad (1)$$

where t_1 is the program execution time for a single processor and t_p is the wall clock time for a given job to be executed on p processors.

4.1. Description of the Benchmark

The gravity packing problem of granular material, falling, under the influence of gravity, into a container, was considered in order to investigate the performance of the developed DEM SaaS because it often serves as a benchmark for performance measurements. The geometry and physical data of the problem are described for research reproducibility. The solution domain was assumed to be a cubic container with 1.0-m-long edges. Half of the domain was filled with monosized particles distributed by using a cubic structure. The granular material was represented by an assembly of 1,000,188 particles with a radius $R = 0.004$ m. The initial velocities of the particles were defined randomly with a uniform distribution, with their magnitudes being in the range of 0.0 to 0.1 m/s. The physical data of the particles of the artificially assumed material were as follows: density = 7000 kg/m³, Poisson's ratio = 0.2, elasticity modulus = 1.0×10^7 Pa, friction coefficient = 0.4, and coefficient of restitution = 0.5.

The representative computational experiments for performance analysis were repeated 10 times, and the averaged values were examined. Performing the benchmark, the computation time of 5000 time steps was measured to investigate the computational performance of the developed DEM SaaS. The short time interval was considered in order to avoid domain repartitioning and reduce particle exchange between subdomains, which helps to focus on the main interprocess communication due to ghost particles. Ghost layers of different sizes were considered in order to study the influence of interprocess communication on the performance of the DEM SaaS. The ghost layers GL1, GL2, and GL3 had thicknesses = $2R$, $4R$, and $6R$, respectively, where $2R$ was the most common because it contained one layer of particles. Thicker layers might decrease the frequency of communication due to particle migration and domain repartitioning but increase the amount of data transferred between processes.

4.2. Computational Load

In general, the computational load can be estimated by the number of particles or contacts between neighboring particles. Contacts can rapidly change during computation, while the number of particles remains constant. Thus, the number of particles is a slightly less accurate but more convenient preliminary measure of computational load. Figure 2 shows the number of particles processed by a varying number of parallel processes p . Figure 2a presents the total number of particles, including ghost particles, owned by all processes in particular parallel runs. The dotted columns represent the local particles,

always equal to 1,000,188. The red (GL1), blue (GL2), and green (GL3) columns without dots represent ghost particles in ghost layers of thickness = $2R$, $4R$, and $6R$, respectively. Each MPI process handles local particles in its subdomain and ghost particles in relevant ghost layers. Therefore, the total number of processed particles depends on the total number of ghost particles, which increases with the number of parallel processes. In the implemented parallel algorithm, only contact detection and a part of contact force computations are performed on ghost particles, which reduces the computational load of ghost layers. However, the computations performed on ghost particles increase the load and cannot be neglected. The ghost particles of the thinnest layer, GL1, made up 3.1% and 10.0% of the total number of processed particles in the case of 4 and 16 processes, respectively. In the cases of the thicker ghost layers, GL2 and GL3, this percentage increased up to 16.6% and 26.1%, respectively. Moreover, the number of ghost particles defines the amount of data transferred among MPI processes. Thus, the ratio of the number of ghost particles to the total number of particles represents the ratio of communication to computation.

Figure 2. The number of particles owned by a varying number of parallel processes: (a) total number of particles and (b) varying number of particles per process.

Figure 2b shows the variation in the maximum (max) and mean numbers of particles per process, which indicates load imbalance. The RCB method divides particles into nearly equal subsets according to particle coordinates. The maximum number of local particles owned by a processor differed from the mean number of local particles by 3.2% of the mean in the case of the GL1 layer and 16 processes. The number of ghost particles owned by processors can be different due to domain boundaries defined by implicit planes, where ghost layers are not necessary. Thus, the difference between the maximum number of all particles owned by a processor and the mean varied by up to 6.2% (16 processes) of the mean in the case of the GL1 layer. For thicker ghost layers, GL2 and GL3, the difference increased by up to 8.9% and 11.4% of the mean, respectively, indicating the growing load imbalance.

Load balance, minimizing the idle time of processes, can be critical to the parallel performance of the computationally intensive SaaS. Load imbalance can be estimated by using the percentage imbalance measure, which evaluates how unevenly the computational load is distributed. The percentage imbalance λ was computed using the following formula:

$$\lambda = \left(\frac{L_{\max}}{L_{\text{avg}}} - 1 \right) \cdot 100\% \quad (2)$$

where L_{avg} is the averaged load over all processes and $L_{\max}$ is the load of the process that has the largest computational load.

The time consumed by computational procedures is almost the exact measure of the computational load. Therefore, it was considered the load in this study. The computing time is measured by timers in the computational procedures of the DEM code. The communication time is not included in the computing time. Figure 3 shows the computing time and load imbalance for benchmarks with ghost layers of different thicknesses solved by a varying number of MPI processes. The computing time averaged over all processes (mean) and the computing time of the process that had the longest computing time (max) are presented. In the case of GL1, the percentage imbalance varied from 0.2% to 4.8%. The measured imbalance was even smaller than the variation in the number of particles owned by processes because not all computations were performed on ghost particles. In the case of GL2 and GL3, the percentage imbalance increased up to 5.8% and 7.1%, respectively.

Figure 3. Load imbalance expressed by the maximum and mean computing times.

4.3. Communication

Interprocess communication highly influences the parallel performance of DEM software. Generally, computations are performed significantly faster than MPI communications between nodes, especially over high-latency and low-bandwidth Ethernet networks. The DEM SaaS is communication-intensive software because of the necessary exchange of ghost particle data at each time step. Thus, intensive interprocess communication can drastically decrease the parallel performance if the communication-to-computation ratio is not low enough.

Figure 4 shows DEM SaaS data transfer between the cores and nodes of the cloud infrastructure. The data transfer between processes gradually increased with the number of processes. In the case of GL1, data transfer between 16 processes was 3.3 times more than that between 4 processes. We observed the nearly linear dependency of data transfer on the number of particles. However, a sudden increase in data transfer between nodes was obtained in the case of benchmarks solved by three nodes (12 cores), which was not observed in communication between processes (cores).

Figure 4. Communication: data transferred between (a) cores and (b) nodes.

Figure 5a shows the MPI communication time measured solving benchmarks with ghost layers of different thicknesses. The GL1, GL2, and GL3 solid curves represent usual benchmarks with ghost layers of thickness = $2R$, $4R$, and $6R$, respectively. The GL1, GL2, and GL3 remapped curves represent communication times obtained by using the improved mapping of subdomains to cloud resources based on the multicore architecture. It is worth noting that a twofold increase in data transfer between nodes (Figure 4b) caused up to a threefold increase in the communication time for a larger number of processes. For benchmarks solved by three nodes (12 cores), a sudden increase in communication time was observed, which is relevant to data transfer between nodes presented in Figure 4b. In this particular case, the communication time made up 9.1% of the computing time, which significantly reduced the parallel performance. In the case of four nodes (16 cores), the communication time made up only 3.9% of the computing time.

(a)

(b)

Figure 5. Communication time (a) and default particle distribution among three nodes (b).

Figure 5b shows the default particle distribution among three nodes, which illustrates unsuccessful subdomain mapping to cloud resources based on the multicore architecture. The RCB method of the Zoltan library divides particles into nearly equal subsets but does not optimize internode communication or perform relevant mapping of particle subsets to multicore nodes. As a result, four spatially scattered subdomains were mapped to one node, which had a large number of ghost particles requiring data exchange with other nodes. Spatially connected subdomains were correctly remapped to nodes, reducing the MPI data transfer. In Figure 4a, dotted lines show the significantly reduced communication time due to improved mapping. However, the communication time measured on 12 cores (three nodes) was still larger than that obtained on 16 cores (four nodes), which could be easily observed in the cases of benchmarks with thicker ghost layers. It is natural that the best performance of the method based on recursive bisection can be observed by dividing particles into 2^P subsets.

Figure 6 shows the contribution of computation, communication, and waiting to the total benchmark time in the case of the GL1 benchmark solved by 12 processes on three nodes. Performing code profiling, two MPI barriers were placed before and after communication routines to measure wait times caused by computational load imbalance and communication imbalance, respectively. The Calc, Comm, Wait1, and Wait2 columns represent the computing time, communication time, wait time due to computational load imbalance, and wait time due to communication imbalance, respectively.

Figure 6. Contribution of computation, communication, and waiting to the benchmark time ($p = 12$): (a) default and (b) improved mapping of subdomains to processes.

The computing load was balanced well enough in both cases (Figure 6a,b). Evaluating the wait time, the percentage imbalance was 2.5% of the mean computing time. In contrast, the wait time due to communication imbalance can be unexpectedly long when the communication time is long (Figure 6a). The mean wait time due to communication imbalance was 188% of the mean communication time. The default mapping of subdomains to processes led to large data transfer and imbalanced communication when processes on one node needed to send and receive approximately twice the amount of data sent and received by processes running on other nodes. Even processes with the highest mean communication load sometimes needed to wait for others due to perturbations on the network switch, which further increased the wait time. The improved mapping took into account the spatial location and communication pattern of neighboring processes by distributing them among nodes. Thus, the data transferred between nodes reduced because the local communications between the processes within nodes increased. Figure 6b shows that improved mapping significantly reduced the communication time, which also decreased the mean wait time by up to 108% of the mean communication time. Thus, improved mapping reduced the communication-to-computation ratio from 0.26 to 0.08.

4.4. Parallel Performance

Figure 7 shows the speedup of parallel computations, solving the benchmarks with ghost layers of different thicknesses. The special curve called Ideal illustrates the ideal speedup. The GL1, GL2, and GL3 curves represent the speedup obtained solving the benchmarks with ghost layers of thickness = $2R$, $4R$, and $6R$, respectively. In the case of four processes running on one node, benchmarks with different numbers of ghost particles demonstrated nearly equal speedup because of the absence of internode communication. A reduction in the speedup owing to communication overhead and computation on ghost particles was obtained for a larger number of processes, leading to a larger number of ghost particles. Thus, benchmarks with thicker ghost layers and more data to exchange between nodes revealed lower parallel speedup values. Speedups equal to 12.3, 10.5, and 9.4 were measured solving benchmarks with ghost layers of thickness = $2R$, $4R$, and $6R$, respectively, which gave parallel efficiency = 0.77, 0.66, and 0.58, respectively, for 16 processes. In the case of 12 processes working on three nodes, a significant reduction in the speedup was not observed, because we applied improved mapping of subdomains to processes. In the case of the GL1 layer, the measured speedup values were close to those obtained for relevant numbers of processes in other parallel performance studies of DEM software [45]. However, the speedup curves of GL2 and GL3 showed lower parallel performance because of an increased communication-to-computation ratio.

Figure 7. Speedup of parallel computations.

Figure 8 shows the contribution of computation (Calc), communication (Comm), wait time due to computational load imbalance (Wait1), and wait time due to communication imbalance (Wait2) to the execution times of benchmarks with ghost layers of different thicknesses. Wait times were measured by using two MPI barriers that increased execution times but helped profile the code and evaluate the influence of communication and load imbalance on parallel performance. In the case of the GL1 benchmark with the thinnest ghost layer, communication consumed a reasonable amount of time, which increased from 0.1% (4 processes) to 3.9% (16 processes) of the computing time. However, solving benchmarks with thicker ghost layers, the communication time increased up to 9.8% and 13.2% of the computing time, which notably reduced parallel performance. Moreover, increased data transfer led to a growing communication imbalance. Therefore, the wait time increased up to 108%, 148%, and 169% of the communication time for GL1, GL2, and GL3 benchmarks, respectively. The GL1 benchmark demonstrated a satisfactory communication-to-computation ratio, which was equal to 0.06 for 16 processes. In the case of GL2 and GL3 benchmarks with two- and three-fold thicker ghost layers, the communication-to-computation ratio increased to 0.20 and 0.29, respectively, which significantly reduced the parallel efficiency to 0.66 and 0.58, respectively. It should be noted that an increase in transferred data can severely limit the number of used virtual resources, running on nodes connected by high-latency and low-bandwidth Ethernet networks.

Figure 8. Contribution of computation, communication, and waiting to the benchmark execution times.

4.5. Overhead of Cloud Infrastructure

The overhead of the cloud infrastructure can be important for the performance of any communication- and computation-intensive SaaS. In this study, the percentage difference in the execution time or the overhead was computed as

$$O_{cloud} = \left(\frac{t_{cloud} - t_{native}}{t_{native}} \right) \cdot 100\%, \quad (3)$$

where t_{cloud} is the SaaS execution time measured on the cloud infrastructure and t_{native} is the SaaS execution time attained on the native hardware. Figure 9 presents the percentage difference in execution time between Docker containers without OpenStack services (Docker), KVM-based VMs without OpenStack services (KVM), Docker containers with OpenStack services (ZunDocker), KVM-based VMs with OpenStack services (NovaDocker), and the native hardware. Figure 9a shows the overhead in computing time without waiting and communication, while Figure 9b presents the overhead in the total execution time of the GL1 benchmark.

Figure 9. Overhead of the cloud infrastructure in (a) computing time and (b) total execution time.

The difference in computing time (Figure 9a), representing the overhead of computer hardware virtualization increased up to 1.2% and 0.5% of the computing time on the native hardware in the case of Docker containers and KVM-based VMs without OpenStack services, respectively. The performance overhead of the Docker containers was consistent with previous results [24,25,32]. The observed overhead of KVM-based VMs was even smaller than that measured in related works [19,22–25]. However, the obtained difference was rather small, while the highest values of the standard deviation were of the same order (up to 0.3%). The overhead in terms of computing time on KVM-based VMs with OpenStack services was larger than that on KVM-based VMs without OpenStack services by only 1.4% of the computing time on the native hardware. For Docker containers with OpenStack services, this difference increased from 1.7% to 2.6% of the computing time on the native hardware. It is worth noting that processes of the OpenStack infrastructure for the Docker containers had more influence on the overhead in terms of computing time than those of the OpenStack infrastructure for KVM-based VMs.

The total execution overhead (Figure 9b), measured on Docker containers and KVM-based VMs without OpenStack services, grew, increasing the number of parallel processes, in contrast to the overhead in computing time (Figure 9a). The overhead due to virtualization of the network interface card was evaluated by examining the difference in the total execution overhead (Figure 9b) and the overhead in computing time (Figure 9a) on Docker containers and KVM-based VMs without OpenStack services. This difference was rather small and increased up to 1.2% and 1.8% for Docker containers and KVM-based VMs.

However, a large increase in the overhead, increasing the number of parallel processes for the solution of the fixed-size problem, was observed in the cases of Docker containers and KVM-based VMs with OpenStack services. For Docker containers and KVM-based VMs on the OpenStack cloud, the overhead increased up to 13.7% and 11.2% of the execution time on the native hardware, respectively. On average, Nova with KVM-based VMs outperformed Zun with Docker containers by 2.5% of the execution time on the native hardware. Growth in the infrastructure overhead, increasing the number of processes used to solve the aortic valve problem by a parallel SaaS based on the finite volume method and commercial ANSYS Fluent software, was also observed for Docker containers of the OpenStack cloud [36]. The observed overhead values were less than 5% of the total execution time. However, the commercial software used as a black box did not allow extending the investigation and finding the reason for the observed overhead increase.

The observed overhead of the cloud infrastructure can be caused by the virtual network overhead. Therefore, synthetic network benchmarks were performed. The bandwidth of the native 1 Gbps Ethernet network measured using Iperf [52] was 941 Mbit/s. The virtualization of the Ethernet network reduced the network bandwidth by 2.8% of the bandwidth measured on the native hardware. Nearly the same results were observed on KVM-based VMs and Docker containers connected by the OpenStack network service Neutron. The Docker containers were connected to Neutron by Kuryr, but this did not significantly influence the network bandwidth. It is well known that the transfer of small messages is highly influenced by network latency. The increasing number of parallel processes leads to a larger number of messages of a smaller size for fixed-size problems. Thus, the latency of network communication becomes more important, especially for smaller-size problems. On average, the round-trip time on the native Ethernet network measured by the ping command was 17.9 μ s. For KVM-based VMs without and with OpenStack services, the round-trip time increased by 4.5 and 4.7 times, respectively. A lower increase in latency was measured on the virtual network connecting Docker containers. For Docker containers without and with OpenStack services, the round-trip time increased by 1.5 and 2.0 times, respectively.

The results of benchmarks with MPI barriers were examined to determine how network virtualization influences communication time. The communication time increased by up to 2.4% and 2.2% of the benchmark time on the native hardware in the case of KVM-based VMs and Docker containers with OpenStack services, respectively. MPI communication between Docker containers was faster than that between KVM-based VMs connected by the virtual OpenStack network. However, the obtained difference was small because the communication-to-computation ratio was only up to 0.08 in the case of the GL1 benchmark on native hardware. In contrast, the wait time due to computational load imbalance increased by up to 8.2% and 8.6% of the benchmark time on the native hardware in the case of KVM-based VMs and Docker containers with OpenStack services, respectively. Thus, the increase in the wait time was significantly larger than that in the communication time. Moreover, the overhead on KVM-based VMs was smaller than that on Docker containers, which was relevant to the total execution overhead presented in Figure 9b.

Figure 10 shows the load imbalance measured on the native hardware and the OpenStack cloud in the case of the GL1 benchmark. The Native, Docker, KVM, ZunDocker, and NovaKVM columns represent the percentage imbalance, including wait times, obtained on the native hardware, Docker containers, KVM-based VMs, Docker containers with OpenStack services, and KVM-based VMs with OpenStack services, respectively. It is obvious that the percentage imbalance measured on the native hardware was the lowest. The percentage imbalance measured on Docker containers and KVM-based VMs without OpenStack services was only slightly higher. The observed differences can be treated as negligible because they do not exceed 0.9% and 0.5% in the cases of the Docker containers and KVM-based VMs, respectively. However, the percentage imbalance increased up to 13.8% and 12.5% on Docker containers and KVM-based VMs with OpenStack services, respectively. It is worth noting that the wait time of the process with the largest computational load and the longest computing time did not exceed 0.56% of the computing time on

the native hardware, which was almost negligible (Figure 6). Times, when processes with average computational load waited for the process with the largest computational load to complete, were almost the same on the native hardware and the cloud. However, the wait times of the process with the largest computational load increased up to 9.1% and 8.5% on Docker containers and KVM with OpenStack services, respectively. This means that the process with the largest application load waited while other threads completed additional tasks of the cloud infrastructure. The background processes of the OpenStack service Zun for Docker containers required more CPU time and produced a larger load imbalance than those of the OpenStack service Nova for KVM-based VMs, probably because Zun processes run on nodes together with Nova processes, using part of their functionality. The CPU time required by Zun and Nova background processes can be short, but the processes have a significant influence on the load balance of the communication- and computation-intensive SaaS based on MPI. Moreover, the load imbalance grows with the number of employed MPI processes and multicore nodes.

Figure 10. Load imbalance measured on the native hardware and the OpenStack cloud.

5. Conclusions

The paper presents a performance analysis of the communication- and computation-intensive DEM SaaS on the OpenStack cloud. The following observations and conclusions may be drawn:

- The performance of the communication- and computation-intensive DEM SaaS highly depends on MPI communication issues, load mapping to virtual resources based on the multicore architecture, and the overhead of the cloud infrastructure.
- Casual mapping of particle subsets to multicore hardware resources can increase MPI communication time and decrease the parallel speedup. In the case of the benchmark with the thinnest ghost layer, improved mapping based on spatially connected subsets reduced the internode data transfer by 34.4% of the data transfer required by the casual mapping, decreased the communication time by 2.47 times, and raised the parallel efficiency from 0.67 to 0.78 for 12 processes.
- The performance analysis revealed that interprocess MPI communication highly influences the parallel performance of the DEM SaaS. A three-fold increase in the ghost layer thickness and the subsequent increase in transferred data decreased the parallel speedup from 12.3 to 9.4 for 16 processes. Significantly, the communication-to-computation ratio increased from 0.08 to 0.29.
- The virtualization layer reduced the computational performance of the developed parallel DEM SaaS by 2.4% and 2.0% in the case of Docker containers and KVM-based VMs without OpenStack services, respectively.
- The overall overhead of the cloud infrastructure increased significantly when the number of parallel processes increased. The software execution time increased by up

to 13.7% and 11.2% of the execution time on the native hardware in the case of Docker containers and KVM-based VMs with the OpenStack cloud, respectively.

- The large overhead was mainly caused by OpenStack processes that increased the load imbalance of the parallel DEM SaaS based on MPI communication. The processes of the OpenStack service Zun for Docker containers consumed more CPU time and produced a larger load imbalance than those of the OpenStack service Nova for KVM-based VMs, which resulted in a larger overall overhead of the cloud infrastructure. On average, the difference in overhead was about 2.5% of the execution time on the native hardware.
- The study revealed that standard benchmarks can hardly provide the comprehensive information required for efficient scheduling of parallel DEM computations. Preliminary specific benchmarks are required to evaluate the parallel performance of the developed SaaS and the overhead of the cloud infrastructure.

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 study is part of project no. 09.3.3-LMT-K-712-02-0131, funded under the European Social Fund measure “Strengthening the Skills and Capacities of Public Sector Researchers for Engaging in High Level R&D Activities,” administered by the Research Council of Lithuania.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

References

1. Sakellari, G.; Loukas, G. A Survey of Mathematical Models, Simulation Approaches and Testbeds Used for Research in Cloud Computing. *Simul. Model. Pract. Theory* **2013**, *39*, 92–103. [CrossRef]
2. OpenStack. Available online: <https://www.openstack.org/> (accessed on 9 May 2021).
3. Nurmi, D.; Wolski, R.; Grzegorzczak, C.; Obertelli, G.; Soman, S.; Youseff, L.; Zagorodnov, D. The Eucalyptus Open-Source Cloud-Computing System. In Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, Shanghai, China, 18–21 May 2009; pp. 124–131.
4. Chierici, A.; Veraldi, R. A Quantitative Comparison between Xen and Kvm. *J. Phys. Conf. Ser.* **2010**, *219*, 042005. [CrossRef]
5. Libvirt. Available online: <https://libvirt.org/> (accessed on 9 May 2021).
6. LXC. Available online: <https://linuxcontainers.org/> (accessed on 9 May 2021).
7. Kurtzer, G.M.; Sochat, V.; Bauer, M.W. Singularity: Scientific Containers for Mobility of Compute. *PLoS ONE* **2017**, *12*, e0177459. [CrossRef]
8. Docker. Available online: <https://www.docker.com/> (accessed on 9 May 2021).
9. Li, G.; Woo, J.; Lim, S.B. HPC Cloud Architecture to Reduce HPC Workflow Complexity in Containerized Environments. *Appl. Sci.* **2021**, *11*, 923. [CrossRef]
10. McMillan, B.; Chen, C. *High Performance Docking*; Technical White Paper; IBM: Armonk, NY, USA, 2014.
11. UberCloud. ANSYS Fluids and Structures on Cloud. Available online: <https://www.theubercloud.com/ansys-cloud> (accessed on 9 May 2021).
12. EDEM Now Available on Rescale’s Cloud Simulation Platform. Available online: <https://www.edemsimulation.com/blog-and-news/news/edem-now-available-rescales-cloud-simulation-platform/> (accessed on 9 May 2021).
13. Astyrakakis, N.; Nikoloudakis, Y.; Kefaloukos, I.; Skianis, C.; Pallis, E.; Markakis, E.K. Cloud-Native Application Validation Amp; Stress Testing through a Framework for Auto-Cluster Deployment. In Proceedings of the 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Limassol, Cyprus, 11–13 September 2019; pp. 1–5.
14. Zhu, H.P.; Zhou, Z.Y.; Yang, R.Y.; Yu, A.B. Discrete Particle Simulation of Particulate Systems: A Review of Major Applications and Findings. *Chem. Eng. Sci.* **2008**, *63*, 5728–5770. [CrossRef]
15. Khan, A.A.; Zakarya, M. Energy, Performance and Cost Efficient Cloud Datacentres: A Survey. *Comput. Sci. Rev.* **2021**, *40*, 100390. [CrossRef]
16. 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]

17. Walters, J.P.; Chaudhary, V.; Cha, M.; Guercio, S.; Gallo, S. A Comparison of Virtualization Technologies for HPC. In Proceedings of the 22nd International Conference on Advanced Information Networking and Applications, Gino-wan, Japan, 25–28 March 2008; pp. 861–868.
18. Macdonell, C.; Lu, P. Pragmatics of Virtual Machines for High-Performance Computing: A Quantitative Study of Basic Overheads. In Proceedings of the 2007 High Performance Computing and Simulation Conference, Prague, Czech, 4–6 June 2007; pp. 1–7.
19. Kačeniauskas, A.; Pacevič, R.; Staškūnienė, M.; Šešok, D.; Rusakevičius, D.; Aidietis, A.; Davidavičius, G. Private Cloud Infrastructure for Applications of Mechanical and Medical Engineering. *Inf. Technol. Control* **2015**, *44*, 254–261. [\[CrossRef\]](#)
20. Kozhribayev, Z.; Sinnott, R.O. A Performance Comparison of Container-Based Technologies for the Cloud. *Future Gener. Comput. Syst.* **2017**, *68*, 175–182. [\[CrossRef\]](#)
21. Felter, W.; Ferreira, A.; Rajamony, R.; Rubio, J. An Updated Performance Comparison of Virtual Machines and Linux Containers. In Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Philadelphia, PA, USA, 29–31 March 2015; pp. 171–172.
22. Estrada, Z.J.; Deng, F.; Stephens, Z.; Pham, C.; Kalbarczyk, Z.; Iyer, R. Performance Comparison and Tuning of Virtual Machines for Sequence Alignment Software. *Scalable Comput. Pract. Exp.* **2015**, *16*, 71–84. [\[CrossRef\]](#)
23. 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. [\[CrossRef\]](#)
24. 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. [\[CrossRef\]](#)
25. Kominos, C.G.; Seyvet, N.; Vandikas, K. Bare-Metal, Virtual Machines and Containers in OpenStack. In Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), Paris, France, 7–9 March 2017; pp. 36–43.
26. 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. [\[CrossRef\]](#)
27. Ventre, P.L.; Pisa, C.; Salsano, S.; Siracusano, G.; Schmidt, F.; Lungaroni, P.; Blefari-Melazzi, N. Performance Evaluation and Tuning of Virtual Infrastructure Managers for (Micro) Virtual Network Functions. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016; pp. 141–147.
28. Shah, S.A.R.; Waqas, A.; Kim, M.-H.; Kim, T.-H.; Yoon, H.; Noh, S.-Y. Benchmarking and Performance Evaluations on Various Configurations of Virtual Machine and Containers for Cloud-Based Scientific Workloads. *Appl. Sci.* **2021**, *11*, 993. [\[CrossRef\]](#)
29. Han, J.; Ahn, J.; Kim, C.; Kwon, Y.; Choi, Y.; Huh, J. The Effect of Multi-Core on HPC Applications in Virtualized Systems. In *European Conference on Parallel Processing, Proceedings of the Euro-Par 2010 Parallel Processing Workshops, Ischia, Italy, 31 August 2010*; Guarracino, M.R., Vivien, F., Träff, J.L., Cannatoro, M., Danelutto, M., Hast, A., Perla, F., Knüpfer, A., Di Martino, B., Alexander, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 615–623.
30. Jackson, K.R.; Ramakrishnan, L.; Muriki, K.; Canon, S.; Cholia, S.; Shalf, J.; Wasserman, H.J.; Wright, N.J. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, USA, 30 November 2010; pp. 159–168.
31. Xavier, M.G.; Neves, M.V.; Rossi, F.D.; Ferreto, T.C.; Lange, T.; De Rose, C.A.F. Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments. In Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, Belfast, UK, 27 February 2013; pp. 233–240.
32. Hale, J.S.; Li, L.; Richardson, C.N.; Wells, G.N. Containers for Portable, Productive, and Performant Scientific Computing. *Comput. Sci. Eng.* **2017**, *19*, 40–50. [\[CrossRef\]](#)
33. 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, New York, NY, USA, 15 March 2018; pp. 1–5.
34. Lv, L.; Zhang, Y.; Li, Y.; Xu, K.; Wang, D.; Wang, W.; Li, M.; Cao, X.; Liang, Q. Communication-Aware Container Placement and Reassignment in Large-Scale Internet Data Centers. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 540–555. [\[CrossRef\]](#)
35. Manumachu, R.R.; Lastovetsky, A. Bi-Objective Optimization of Data-Parallel Applications on Homogeneous Multicore Clusters for Performance and Energy. *IEEE Trans. Comput.* **2018**, *67*, 160–177. [\[CrossRef\]](#)
36. 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. Spec. Issue Provid. Comput. Solut. Exascale Chall.* **2020**, *39*, 695–723. [\[CrossRef\]](#)
37. Cundall, P.A.; Strack, O.D.L. A Discrete Numerical Model for Granular Assemblies. *Géotechnique* **1979**, *29*, 47–65. [\[CrossRef\]](#)
38. Chen, L.; Wang, C.; Moscardini, M.; Kamlah, M.; Liu, S. A DEM-Based Heat Transfer Model for the Evaluation of Effective Thermal Conductivity of Packed Beds Filled with Stagnant Fluid: Thermal Contact Theory and Numerical Simulation. *Int. J. Heat Mass Transf.* **2019**, *132*, 331–346. [\[CrossRef\]](#)
39. Kačianauskas, R.; Rimša, V.; Kačeniauskas, A.; Maknickas, A.; Vainorius, D.; Pacevič, R. Comparative DEM-CFD Study of Binary Interaction and Acoustic Agglomeration of Aerosol Microparticles at Low Frequencies. *Chem. Eng. Res. Des.* **2018**, *136*, 548–563. [\[CrossRef\]](#)
40. Lu, C.; Ma, L.; Li, Z.; Huang, F.; Huang, C.; Yuan, H.; Tang, Z.; Guo, J. A Novel Hydraulic Fracturing Method Based on the Coupled CFD-DEM Numerical Simulation Study. *Appl. Sci.* **2020**, *10*, 3027. [\[CrossRef\]](#)

41. Stupak, E.; Kačianauskas, R.; Kačeniauskas, A.; Starikovičius, V.; Maknickas, A.; Pacevič, R.; Staškūnienė, M.; Davidavičius, G.; Aidietis, A. The Geometric Model-Based Patient-Specific Simulations of Turbulent Aortic Valve Flows. *Arch. Mech.* **2017**, *69*, 317–345. [[CrossRef](#)]
42. 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](#)]
43. Govender, N.; Cleary, P.W.; Kiani-Oshtorjani, M.; Wilke, D.N.; Wu, C.-Y.; Kureck, H. The Effect of Particle Shape on the Packed Bed Effective Thermal Conductivity Based on DEM with Polyhedral Particles on the GPU. *Chem. Eng. Sci.* **2020**, *219*, 115584. [[CrossRef](#)]
44. 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](#)]
45. Berger, R.; Kloss, C.; Kohlmeyer, A.; Pirker, S. Hybrid Parallelization of the LIGGGHTS Open-Source DEM Code. *Powder Technol.* **2015**, *278*, 234–247. [[CrossRef](#)]
46. Norouzi, H.R.; Zarghami, R.; Sotudeh-Gharebagh, R.; Mostoufi, N. *Coupled CFD-DEM Modeling: Formulation, Implementation and Application to Multiphase Flows*; Wiley: Chichester, UK, 2016; ISBN 978-1-119-00513-1.
47. Kačeniauskas, A.; Rutschmann, P. Parallel FEM Software for CFD Problems. *Informatica* **2004**, *15*, 363–378. [[CrossRef](#)]
48. 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](#)]
49. Mell, P.; Grance, T. The NIST Definition of Cloud Computing. Available online: <https://csrc.nist.gov/publications/detail/sp/800-145/final#> (accessed on 11 August 2021).
50. Schroeder, W.; Martin, K.; Lorensen, B. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th ed.; Kitware: Clifton Park, NY, USA, 2006; ISBN 978-1-930934-19-1.
51. Pacevič, R.; Kačeniauskas, A. The Development of VisLT Visualization Service in Openstack Cloud Infrastructure. *Adv. Eng. Softw.* **2017**, *103*, 46–56. [[CrossRef](#)]
52. Iperf. Available online: <http://sourceforge.net/projects/iperf/> (accessed on 9 May 2021).