

# Supercomputing Frontiers and Innovations

2019, Vol. 6, No. 1

## Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

## Editorial Board

### Editors-in-Chief

- **Jack Dongarra**, University of Tennessee, Knoxville, USA
- **Vladimir Voevodin**, Moscow State University, Russia

### Editorial Director

- **Leonid Sokolinsky**, South Ural State University, Chelyabinsk, Russia

### Associate Editors

- **Pete Beckman**, Argonne National Laboratory, USA
- **Arndt Bode**, Leibniz Supercomputing Centre, Germany
- **Boris Chetverushkin**, Keldysh Institute of Applied Mathematics, RAS, Russia
- **Alok Choudhary**, Northwestern University, Evanston, USA

- **Alexei Khokhlov**, Moscow State University, Russia
- **Thomas Lippert**, Jülich Supercomputing Center, Germany
- **Satoshi Matsuoka**, Tokyo Institute of Technology, Japan
- **Mark Parsons**, EPCC, United Kingdom
- **Thomas Sterling**, CREST, Indiana University, USA
- **Mateo Valero**, Barcelona Supercomputing Center, Spain

## Subject Area Editors

- **Artur Andrzejak**, Heidelberg University, Germany
- **Rosa M. Badia**, Barcelona Supercomputing Center, Spain
- **Franck Cappello**, Argonne National Laboratory, USA
- **Barbara Chapman**, University of Houston, USA
- **Yuefan Deng**, Stony Brook University, USA
- **Ian Foster**, Argonne National Laboratory and University of Chicago, USA
- **Geoffrey Fox**, Indiana University, USA
- **Victor Gergel**, University of Nizhni Novgorod, Russia
- **William Gropp**, University of Illinois at Urbana-Champaign, USA
- **Erik Hagersten**, Uppsala University, Sweden
- **Michael Heroux**, Sandia National Laboratories, USA
- **Torsten Hoefler**, Swiss Federal Institute of Technology, Switzerland
- **Yutaka Ishikawa**, AICS RIKEN, Japan
- **David Keyes**, King Abdullah University of Science and Technology, Saudi Arabia
- **William Kramer**, University of Illinois at Urbana-Champaign, USA
- **Jesus Labarta**, Barcelona Supercomputing Center, Spain
- **Alexey Lastovetsky**, University College Dublin, Ireland
- **Yutong Lu**, National University of Defense Technology, China
- **Bob Lucas**, University of Southern California, USA
- **Thomas Ludwig**, German Climate Computing Center, Germany
- **Daniel Mallmann**, Jülich Supercomputing Centre, Germany
- **Bernd Mohr**, Jülich Supercomputing Centre, Germany
- **Onur Mutlu**, Carnegie Mellon University, USA
- **Wolfgang Nagel**, TU Dresden ZIH, Germany
- **Alexander Nemukhin**, Moscow State University, Russia
- **Edward Seidel**, National Center for Supercomputing Applications, USA
- **John Shalf**, Lawrence Berkeley National Laboratory, USA
- **Rick Stevens**, Argonne National Laboratory, USA
- **Vladimir Sulimov**, Moscow State University, Russia
- **William Tang**, Princeton University, USA
- **Michela Taufer**, University of Delaware, USA
- **Andrei Tchernykh**, CICESE Research Center, Mexico
- **Alexander Tikhonravov**, Moscow State University, Russia
- **Eugene Tyrtyshnikov**, Institute of Numerical Mathematics, RAS, Russia
- **Roman Wyrzykowski**, Czestochowa University of Technology, Poland
- **Mikhail Yakobovskiy**, Keldysh Institute of Applied Mathematics, RAS, Russia

## Technical Editors

- **Yana Kraeva**, South Ural State University, Chelyabinsk, Russia
- **Mikhail Zymbler**, South Ural State University, Chelyabinsk, Russia
- **Dmitry Nikitenko**, Moscow State University, Moscow, Russia

# Contents

|  |    |
|--|----|
| <b>Efficient Parallel Implementation of Multi-Arrival 3D Prestack Seismic Depth Migration</b><br>A.L. Pleshkevich, A.V. Ivanov, V.D. Levchenko, S.A. Khilkov, B.P. Moroz .....   | 4  |
| <b>LAMMPS Code Simulation of the Defect Formation Induced by Ion Incidence in Carbon Nanotubes</b><br>A.A. Shemukhin, A.V. Nazarov, A.V. Stepanov .....  | 9  |
| <b>A Fully Conservative Parallel Numerical Algorithm with Adaptive Spatial Grid for Solving Nonlinear Diffusion Equations in Image Processing</b><br>A.D. Bulygin, D.A. Vrazhnov .....   | 14 |
| <b>Parametrization of the Elastic Network Model Using High-Throughput Parallel Molecular Dynamics Simulations</b><br>P.S. Orekhov, I.V. Kirillov, V.A. Fedorov, I.B. Kovalenko, N.B. Gudimchuk, A. Zhmurov .....                           | 19 |
| <b>Facilitating HPC Operation and Administration via Cloud</b><br>C. Sha, J. Zhang, L. An, Y. Zhang, Z. Wang, I. Tomi, B. Nejc, V. Miha, Q. Ji .....   | 23 |
| <b>Performance Evaluation of Different Implementation Schemes of an Iterative Flow Solver on Modern Vector Machines</b><br>K. Yamaguchi, T. Soga, Y. Shimomura, T. Reimann, K. Komatsu, R. Egawa, A. Musa, H. Takizawa, H. Kobayashi ..... | 36 |
| <b>Comparative Analysis of Virtualization Methods in Big Data Processing</b><br>G.I. Radchenko, A.B.A. Alaasam, A.N. Tchernykh .....   | 48 |



This issue is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.

# Efficient Parallel Implementation of Multi-Arrival 3D Prestack Seismic Depth Migration

*Alexander L. Pleshkevich<sup>1</sup>, Anton V. Ivanov<sup>2</sup>, Vadim D. Levchenko<sup>2</sup>, Sergey A. Khilkov<sup>3</sup>, Boris P. Moroz<sup>1</sup>*

© The Authors 2019. This paper is published with open access at SuperFri.org

The goal of seismic migration is to reconstruct the image of Earth's depth inhomogeneities on the base of seismic data. Seismic data is obtained using shots in shallow wells that are located in a dense grid points. Those shots could be considered as special point sources. A reflected and scattered seismic waves from the depth inhomogeneities are received by geophones located also in a dense grid points on a surface. A seismic image of depth inhomogeneities can be constructed based on these waves. The implementation of 3-D seismic migration implies the solution of about  $10^{4-5}$  3-D direct problems of wave propagation. Hence efficient asymptotic methods are of a great practical importance. The multi-arrival 3-D seismic migration program is implemented based on a new asymptotic method. It takes into account multi-pass wave propagation and caustics. The program uses parallel calculations in an MPI environment on hundreds and thousands of processor cores. The program was successfully tested on an international synthetic "SEG salt" data set and on real data. A seismic image cube for Timan-Pechora region is given as an example.

*Keywords: seismic imaging, multi-arrival seismic migration, HPC, aiwlib.*

## Introduction

Among all the tools applied for oil and gas exploration at the moment seismic prospecting is the most accurate and widely used method to study the structure of the inhomogeneous Earth's medium. Seismic imaging implementation based on reflected and scattered seismic waves from the depth geological structures is known as a seismic migration. Special seismic surveys are performed in order to collect seismic data. Geophones on the surface register scattered and reflected waves from the depth inhomogeneities. These waves are produced by special shots in shallow wells as sources. For now the usual area of the survey is  $10^{2-3}$  km<sup>2</sup> and corresponding dataset size is around  $10^{2-3}$  GB. The seismic migration process demands to solve numerous problems of wave propagation in heterogeneous medium. Thus asymptotic ray methods in the context of the acoustic approximation overrule the field. It is common practice to use a single-beam assumption. It states that there is only one ray from the source to an arbitrary point in the medium, the one which passes through the point at the shortest time. In complex media, if caustic occurs, the assumption breaks down and the resulting image can degrade. Therefore, to improve the image, one should discard the single-beam assumption and take multipath propagation into account. The seismic migration based on this approach and its efficient implementation are the main topics of the authors' research.

## 1. Formulation of Seismic Migration

Two asymptotic methods to solve the direct scattering problem taking the multipath propagation and caustics into account are widely known. They are Maslov's method of the canonical operator [5] and the Gaussian beam summation method [4]. Applications of Maslov's method are constrained by the lack of suitable numerical scheme.

<sup>1</sup>JSC Central Geophysical Expedition/Rosgeo, Moscow, Russian Federation

<sup>2</sup>Keldysh Institute of Applied Mathematics, Moscow, Russian Federation

<sup>3</sup>HIPERCONE, Moscow, Russian Federation

Research described in [2, 3] allowed us to introduce an alternative convenient way to solve the problem. It was implemented as a program for multi-arrival 3D depth seismic migration of multi-fold seismic data. For the numerical implementation we used the integral asymptotic solution of the direct scattering problem in the source-receiver Cartesian coordinates:

$$f(\vec{r}) = \frac{1}{16\pi^2} \sum_{m,n} \int_{S_m} \int_{S_n} A_s^m A_g^n \widehat{H}^{I_m+I_n} \dot{u}(\vec{s}, \vec{g}, \tau_s^m(\vec{s}) + \tau_g^n(\vec{g})) d\vec{s} d\vec{g}, \quad (1)$$

where  $f(\vec{r})$  stands for image value at the point  $\vec{r}$ ,  $S_m$  and  $S_n$  are surface areas touched by the wave front with KMAH indices  $I_m$  and  $I_n$  correspondingly coming from the source  $\vec{s}$  and receiver  $\vec{g}$  at the point  $\vec{r}$ .  $A$  is the integrable ray amplitude,  $\widehat{H}$  means Hilbert transform,  $\tau_s$  and  $\tau_g$  signify the time that rays take to get to the source and the receiver from the point  $\vec{r}$ ,  $\dot{u}(\vec{s}, \vec{g}, \tau)$  is the data from the dataset differentiated with respect to time.

Typical seismic migration project requires about  $10^{17}$  floating-point operations. Hence the computational cost of the procedure is exceptionally high. For the case of quasi-regular multi-fold seismic data acquisitions it is possible to reduce computational cost of ray tracings even farther to the amount of unique locations of sources and receivers. Therefore our procedure splits into two consecutive steps. On first step we obtain the ray Green's function (GF) by tracing beam fans, e.g. we calculate  $A$ ,  $I$  and  $\tau$ . And the second one is for calculation of integrals (1) with help of suitable quadratures and the previous step results. Since every GF contains up to a gigabyte of data, the overall intermediate information amount adds up to hundreds of terabytes. Efficient storage and transport of the intermediate data is the key problem arising in the program implementation of the multipath 3D depth migration.

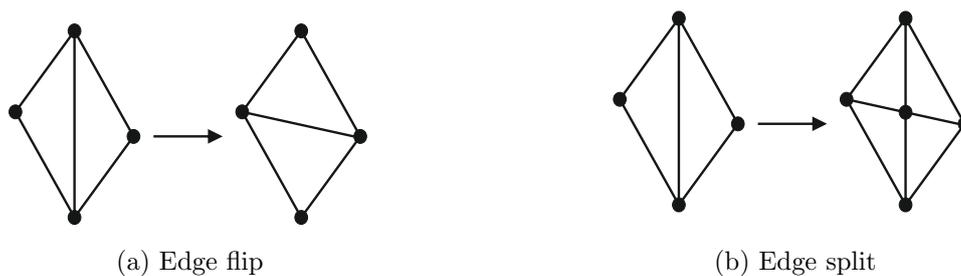
## 2. Algorithm and Implementation Specific Features

Our algorithm uses distributed RAM on a cluster to store ray GFs. Usually the intermediate data amount is greater than the available RAM size on the computational system. Thus at the scheduling stage the whole task is decomposed into smaller ones which use seismic data subsets.

Little seismic datasets require smaller GFs set, therefore, the amount of memory demanded decreases. The optimal partitioning allows us to achieve program memory consumption inversely proportional to the number of subtasks for big subtasks and square root of the number for smaller ones. In the latter case GFs recalculations also take place. That increases the total computation time, but GFs computation still takes about  $10 \div 20$  % of the runtime.

To keep additional costs less than a half of the integration time, we have to store at least several thousands GFs in the memory. Thus computational system is required to have several TB RAM to run the program fast enough. Each subtask updates the part of the seismic image (1). Since the size of the updated part is relatively small ( $\sim 10$  GB), an intermediate image may be safely stored in long-term memory, even if we take the redundancy margin into account. As a result we can ensure fault tolerance. After a fault it is necessary to restart only the failed subtasks.

The problem (1) provides plenty of opportunities to employ parallel computations. Our algorithm has three parallelization levels. It ensures the best locality of processed data and efficiency on parallel systems with distributed memory. The top parallelization level utilizes MPI protocol to make internode communications efficient. The middle level applies OpenMP interface to achieve optimal performance on multi-core systems with shared memory. The bottom level uses processor vector instructions (SSE2, AVX). On the first step of the algorithm the top level



**Figure 1.** Grid adaptation process

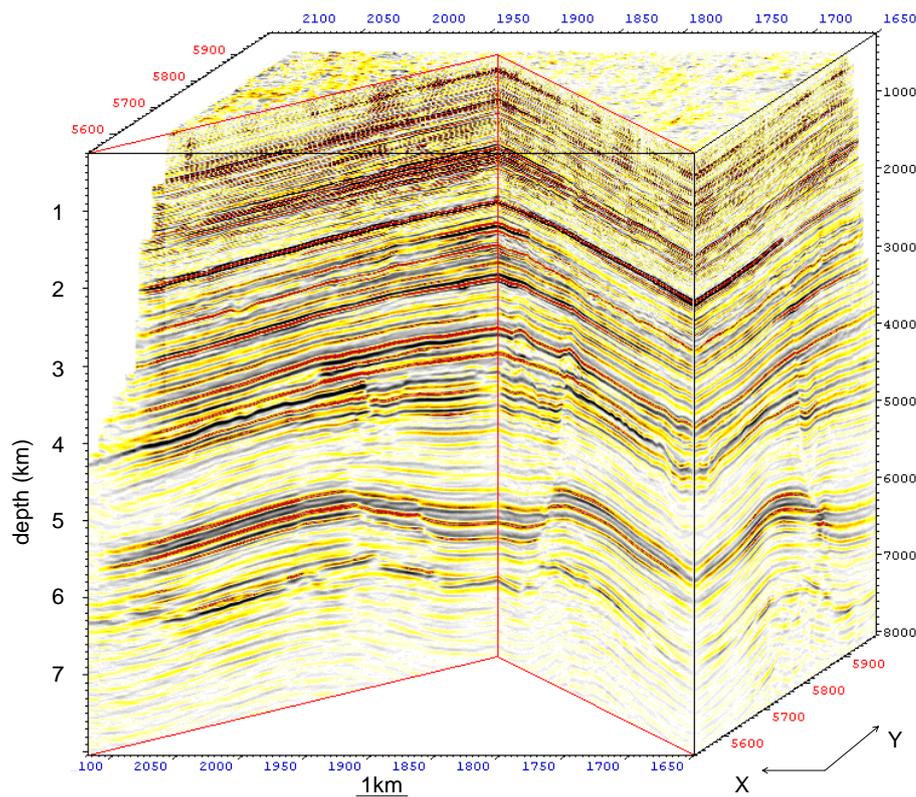
parallelise different beam fans tracings. The middle level thread processes its spatial GF tiles. The bottom level deals with the access to the medium. The second step parallelisation is done by GF tiles, seismic data subset and operations for interpolation of the medium model for levels from the top to the bottom correspondingly. The velocity model is given on a uniform Cartesian grid. VTI and TTI Tompsen anisotropy models are available in addition to the isotropic model.

The ray GF is reconstructed in heterogeneous half-space at nodes of the uniform Cartesian grid. If a caustic occurs, there may be several rays leading from the source to a node. The tracing of a ray implies solving the system of Hamilton's equations for bicharacteristics numerically with aid of Runge–Kutta 4-th order method. Hamilton's system is integrated over the travel time along the ray instead of the common parameter. The ray tracing is performed in two passes. From now on we will refer to them as  $\alpha$  and  $\beta$  tracings. The purpose of  $\alpha$ -tracing is to construct the dynamically adapting angular grid which approximates the wave front on every time step. The grid consists of triangular cells (the beam fragments corresponding to single time moment). Every grid node contains a ray. The grid of initial ray directions is a pentakis dodecahedron recursive subdivision [1]. In contrast to the spherical coordinates of the grid, this triangulation does not have a singularity on the poles, and for the same fineness it has half as many cells.

After updating all the rays positions to new time step, the grid adaptation is performed. All edges with the length above the specified threshold value are arranged in a queue. The queue is kept sorted so the longest edge is always the next to pop. The dynamic adaptation works as follows: it gets the edge from the queue, tries to flip it (Fig. 1a) and tries to split it (Fig. 1b) if the flip failed. All new edges longer than threshold value are added to the queue. If the algorithm cannot split the edge, it discards it. The adaptation stops when the queue is empty. The flip is accepted if directions of new adjacent cells normals are closer to directions normal to the wave front. It is worth mentioning that in addition to the ray position the tracing procedure provides us with time gradient which is orthogonal to the wave front.

The edge split is performed by tracing a new ray close to the middle of the line segment connecting centers of cells adjacent to the edge. Initial parameters of the new ray are deduced by means of the conjugate gradients method. The split is considered unsuccessful if the resulting ray is not close enough to the desired point.

In the presence of caustics and head waves in a complex heterogeneous medium the  $\alpha$ -tracing is a costly procedure. Thus we can avoid repeating it by saving the adaptation history to the long-term memory. It is exactly why the  $\alpha$ -tracing was detached from the  $\beta$ -tracing. During the  $\beta$ -tracing the same beam fans are formed again. The main point of the  $\beta$ -tracing is to provide the GF on a uniform grid. In view of this we have to find all the nodes each beam catches. The



**Figure 2.** The seismic image by amplitude-preserving multiarrival 3D depth migration, Timan–Pechora region

beam segment between to consecutive time moments is considered to take the space between the corresponding triangular cells. For the linear ray segment approximation the problem of finding points inside the beam reduced to finding the solution of a cubic equation. That may be easily done by means of Newton method.

In order to calculate KMAH index for a beam, one should find the number of zeros Jacobian has had to the current moment. The same linear interpolation for the beam segment leads to Jacobian in the form of a cubic polynomial. Again Newton method saves us the trouble.

Unlike the integral in the ray parametric coordinates [3], the integral solution (1) contains singularities. Those integrable singularities arise from Jacobian vanishing on caustic surfaces. Special quadrature formulas explicitly localizing them are required in order to obtain the correct solution.

The described algorithm is implemented as a program in C++11. It works on OS Linux, requires aiwlib library [1] and uses MPI protocol. The code allows us to solve the multiarrival seismic migration problem for a complex heterogeneous medium cost effectively. The produced images were verified for SEG Salt synthetic dataset and a few real-world projects. Figure 2 displays the sample of seismic image for Timan-Pechora region produced by our code.

## Conclusion

We have presented an original algorithm of multi-arrival 3-D seismic depth migration. It is based on a new asymptotic solution of Dirichlet problem for acoustic wave equation. A new asymptotic method is developed to correctly account for multipath ray propagation and caustics

in a complicated inhomogeneous Earth's medium. A corresponding computer program has been implemented. It has been successively tested on international synthetic and real seismic data sets. The program uses parallel calculations on hundreds and thousands of processor cores in an MPI environment.

## Acknowledgements

This research was initiated and sponsored by Central Geophysical Expedition JSC of Rosgeo. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University supported by the project RFMEFI62117X0011.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Ivanov, A., Khilkov, S.: Aiwlib library as the instrument for creating numerical modeling applications. *Scientific Visualization* 10(1), 110–127 (2018), DOI: 10.26583/sv.10.1.09
2. Pleshkevich, A., Ivanov, A., Khilkov, S.: Asymptotic solution of wavefield continuation problem in the ray parametric coordinates. In: *SEG Technical Program Expanded Abstracts 2017*. pp. 5551–5555 (2017), DOI: 10.1190/segam2017-17633541.1
3. Pleshkevich, A., Ivanov, A., Levchenko, V., Khilkov, S.: Multiarrival amplitude-preserving prestack 3D depth migration. *Russian Geophysics (S)* pp. 76–84 (2017)
4. Semtchenok, N., Popov, P., Verdel, A.: Depth migration by the Gaussian beam summation method. *Geophysics* (75), 81–93 (2010), DOI: 10.1190/1.3361651
5. Vainberg, B.: *Asymptotic methods in the equations of mathematical physics*. Moscow: MSU (1982)

# LAMMPS Code Simulation of the Defect Formation Induced by Ion Incidence in Carbon Nanotubes

*Andrey A. Shemukhin*<sup>1</sup>, *Anton V. Nazarov*<sup>1</sup>, *Anton V. Stepanov*<sup>2</sup>

© The Authors 2019. This paper is published with open access at SuperFri.org

A molecular dynamic calculation of the multi-walled carbon nanotube thermal sputtering induced by ion irradiation is carried out. Sputtering results comparable to experimental data are obtained. There are two models of ion and thermal sputtering discussed in the paper. The simulation tested the model of thermal amorphization and revealed that the disordering of multi-walled carbon nanotubes structure occurs as a result of their heating under ion irradiation. Classical molecular dynamic simulation was performed using LAMMPS code. Simulation cell with 14 layers multi-walled carbon nanotube  $12 \times 12 \times 30$  nm size contains 285600 atoms. Multi-walled carbon nanotube was irradiated by 80 keV energy  $Ar^+$  ions in cumulative mode. Simulation was performed on the Lomonosov-1 supercomputer. About 24600 nodes-hours were spent on one simulation as a whole. The balancing of MPI flows for a spatial grid of counting nodes occurred according to the scheme  $8 \times 8 \times 128$  MPI-stream. LAMMPS code was built with Intel 12.0 compiler. This configuration allowed to speed up the calculation in comparison with the calculation on a single-processor Xeon CPU X5570 2.93 GHz machine by 60 times.

*Keywords: ion irradiation, multiwall carbon nanotube, defects, molecular dynamics, sputtering, thermal mechanism.*

## Introduction

Studies related to the ion modification of carbon nanotubes (CNTs) [8] make it possible to create semiconductor devices (diodes, transistors) [14], sensors [13], filters [3], electrodes for electron emission [5] based on defective nanotubes, devices for controlling ionic beams [2, 17]. In studies on defects in carbon nanotubes [1, 7], little attention is paid to defects that appear as a result of ionic modification, in particular the defect formation mechanisms [4]. Among the defects induced by ion irradiation, vacancies, multi-vacancies and interstitials are most often encountered [9]. Defects formed upon irradiation or induced by ion irradiation from intrinsic defects of nanotubes have unique properties. Some types of defects, such as substitutional atoms, or intercalants can significantly change the electronic band structure of a nanotube by transferring it from one type of conductivity to another. Defects in CNTs are chemically active centers that bind molecules well from the environment, which makes the defective nanotube as a whole chemically more active. Transition of a part of the nanotube atoms from the state with  $sp^2$  hybridization to a state with  $sp^3$  hybridization increases the surface area, which affects the tribological, adsorption and capacitance properties of the nanotube. Modifications of the properties of nanotubes by chemical or plasma-chemical processing of samples, for example, for the creation of gas sensors, are what the work of collective [16] is devoted to. A directional change in the properties of carbon nanotubes by an ion beam to create elements of micro and nanoelectronic devices was carried out by Krashennnikov, Nordlund et al. [6]. Simulation was accompanied by experiments [12]. The mechanisms of sputtering the surface of multi-walled carbon nanotubes under ion irradiation are discussed in [4, 10]. At the same time, the competition of two processes, thermal and ionic sputtering, is discussed. In this paper, the first numerical calculations of the interaction of ions with a multi-walled carbon tube are presented.

<sup>1</sup>Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Moscow, Russian Federation

<sup>2</sup>Chuvash State Agricultural Academy, Cheboksary, Russian Federation

## 1. Methods

Calculations of the thermal sputtering of multi-walled nanotubes were performed using the classical molecular dynamics method using the LAMMPS code [11]. Because of the large number of atoms in the simulated system, in order to save computational time, it is advisable to solve the equations of motion by the Verlete method in velocity form. The calculation was performed for a single multiwalled carbon nanotube placed in periodic boundary conditions at all boundaries in a cell with a size of  $12 \times 12 \times 30$  nm.

The AIREBO potential [18] is used to describe the interaction between carbon atoms in a nanotube, which allows one to take into account the formation and breaking of new bonds, the reorganization of a network of bonds, and the change in the degree of hybridization. Potential energy obtaining by AIREBO potential consists of  $V^R$  – repulsive,  $V^A$  – attractive, and  $V^{LJ}$  – Lenard-Jones terms:

$$E_{REBO} = \sum_i \sum_{j(>i)} [V^R(r_{ij}) - b_{ij}V^A(r_{ij}) + V^{LJ}(r_{ij})], \quad (1)$$

responds to attraction of atoms at short distances. All coefficient details for repulsive, attractive parts and main potential function were provided in the work [18]. Repulsive and attractive parts of manybody AIREBO potential are used for short distance Feynman-Hellman forces reproducing in terms of classical molecular dynamics. The multi-walled nanotube consisted of 14 layers: the smallest in diameter was  $CNT(10, 10)$  1.4 nm in diameter, the largest in diameter was  $CNT(75, 75)$  -10 nm, the length of each single-walled nanotube was 29.2 nm. The distance between the layers is 0.34 nm, the layers are shifted relative to each other so that they form the ABAB-packing of graphite. The total number of atoms in a multi-walled nanotube is 285600. The initial temperature was set to 0.1 K. The nanotube was not thermally tested during the passage of the ion. To reduce the radiation dose and to reduce the counting time, we used heavier  $Ar^+$  atoms. The interaction of  $Ar^+C$  was modeled using the Ziegler-Biersack-Littmark (ZBL) potential [19]. The ZBL-potential is well tested for noble gas ions interaction with solids and used in SRIM or TRIM code.

The ion began to move randomly from a distance of 5.5 nm from the axis of the nanotube, with a random initial angle of 3–5 degrees and an energy of 80 keV. Over about 0.01 ps, the ion passed through, then the nanotube was cooled to 500 K for 5 ps, 5 ps, thermostating at a constant temperature, and then another cooling stage for 5 ps. The calculation was performed on Lomonosov-1 supercomputer [15] using nodes with Intel Xeon CPU X5570 2.93GHz (8 cores per node). Calculations were provided at 256 nodes. About 24600 nodes-hours were spent on one calculation as a whole. MPI technology was used for parallelization. The balancing of MPI flows for a spatial grid of counting nodes occurred according to the scheme of  $8 \times 8 \times 128$  MPI-streams. LAMMPS code was built with Intel 12.0 compiler. This configuration allowed to speed up the calculation in comparison with the calculation on a single-processor Xeon CPU X5570 2.93 GHz machine by 60 times.

It is known that when passing through a multi-walled carbon tube, the  $Ar^+$  ion can lose energy as a result of two processes: energy transfer to electrons – electronic energy losses, and energy transfer to lattice atoms of a multi-walled carbon nanotube – nuclear energy losses. Thus, as a result of motion, the ion either heats the lattice when interacting with electrons, or destroys it in an elastic collision. In this case, the tube is heated at 300 K. Heating is produced due to

ion-carbon atom scattering. As a result, this leads to an increase in the amplitude of vibrations of the carbon atoms, and as a result, the mobility of defects increases with increasing temperature.

Defect formation at the initial stages occurs due to the formation of collision cascades caused by the incident  $Ar^+$  ion, and after heating up due to rebuilding the network of bonds and changing the bond order of carbon atoms. Results of simulation have shown, that with the accumulation of the irradiation dose, the number of recoil atoms also increases, and hence the defectiveness of the nanotube. Most often mono- and multivacancies are formed at the same time, while the outer layers of the nanotube are sprayed. In addition, it can be seen that the number of atomized atoms is not significant. This indicates that the main mechanism for the modification of the relief is thermal.

## Conclusions

This calculation made it possible to test the model of thermal amorphization, according to which the disordering of the structure of multi-walled nanotubes occurs as a result of their heating under ion irradiation. The model did not take into account the heat sink, so from this point of view it is not quite adequate. Later the model will be further developed and based on this model, and it is planned to calculate the number of defects and predict electrophysical properties.

## Acknowledgements

The research was performed at the support of Grants of the President (Grant SP-4870.2018.1). The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [15].

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Collins, P.G.: Defects and disorder in carbon nanotubes, Oxford Handbook of Nanoscience and Technology, vol. 2, chap. Materials: Structures, Properties and Characterization Techniques. Oxford University Press (2017), DOI: 10.1093/oxfordhb/9780199533053.013.2
2. Dedkov, G.: Fullerene nanotubes can be used when transporting gamma-quanta, neutrons, ion beams and radiation from relativistic particles. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 143(4), 584–590 (1998), DOI: 10.1016/S0168-583X(98)00388-7
3. Elsehly, E.M., Chechenin, N., Makunin, A., Shemukhin, A., Motaweh, H.: Enhancement of cnt-based filters efficiency by ion beam irradiation. Radiation Physics and Chemistry 146, 19–25 (2018), DOI: 10.1016/j.radphyschem.2018.01.007
4. Elsehly, E.M., Chechenin, N.G., Makunin, A.V., Shemukhin, A.A., Motaweh, H.A.: He ion irradiation effects on multiwalled carbon nanotubes structure. The European Physical Journal D 71(4), 79 (2017), DOI: 10.1140/epjd/e2017-70658-0

5. Kim, D.H., Jang, H.S., Kim, C.D., Cho, D.S., Kang, H.D., Lee, H.R.: Enhancement of the field emission of carbon nanotubes straightened by application of argon ion irradiation. *Chemical Physics Letters* 378(3), 232–237 (2003), DOI: 10.1016/S0009-2614(03)01249-1
6. Krasheninnikov, A.V., Nordlund, K., Keinonen, J.: Production of defects in supported carbon nanotubes under ion irradiation. *Phys. Rev. B* 65, 165423 (2002), DOI: 10.1103/PhysRevB.65.165423
7. Krasheninnikov, A.V., Nordlund, K., Lehtinen, P.O., Foster, A.S., Ayuela, A., Nieminen, R.M.: Adsorption and migration of carbon adatoms on carbon nanotubes: Density-functional ab initio and tight-binding studies. *Phys. Rev. B* 69, 073402 (2004), DOI: 10.1103/PhysRevB.69.073402
8. Krasheninnikov, A.V., Nordlund, K., Sirviö, M., Salonen, E., Keinonen, J.: Formation of ion-irradiation-induced atomic-scale defects on walls of carbon nanotubes. *Phys. Rev. B* 63, 245405 (2001), DOI: 10.1103/PhysRevB.63.245405
9. Krasheninnikov, A., Lehtinen, P., Foster, A., Nieminen, R.: Bending the rules: Contrasting vacancy energetics and migration in graphite and carbon nanotubes. *Chemical Physics Letters* 418(1), 132–136 (2006), DOI: 10.1016/j.cplett.2005.10.106
10. Kushkina, K., Shemukhin, A., Vorobyeva, E., Bukunov, K., Evseev, A., Tatarintsev, A., Maslakov, K., Chechenin, N., Chernysh, V.: Evolution of the multi-walled carbon nanotubes structure with increasing fluence of he ion irradiation. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 430, 11–17 (2018), DOI: 10.1016/j.nimb.2018.05.038
11. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics* 117(1), 1–19 (1995), DOI: 10.1006/jcph.1995.1039
12. Raghuvver, M.S., Ganesan, P.G., DArcy-Gall, J., Ramanath, G., Marshall, M., Petrov, I.: Nanomachining carbon nanotubes with ion beams. *Applied Physics Letters* 84(22), 4484–4486 (2004), DOI: 10.1063/1.1756191
13. Robinson, J.A., Snow, E.S., Bdescu, .C., Reinecke, T.L., Perkins, F.K.: Role of defects in single-walled carbon nanotube chemical sensors. *Nano Letters* 6(8), 1747–1751 (2006), DOI: 10.1021/nl0612289
14. Rochefort, A., Avouris, P.: Quantum size effects in carbon nanotube intramolecular junctions. *Nano Letters* 2(3), 253–256 (2002), DOI: 10.1021/nl015705t
15. Sadovnichy, V., Tikhonravov, A., Voevodin, Vl., Opanasenko, V.: “Lomonosov”: Supercomputing at Moscow State University. In: *Contemporary High Performance Computing: From Petascale toward Exascale*. pp. 283–307. Chapman & Hall/CRC Computational Science, Boca Raton, United States, Boca Raton, United States (2013)
16. Scarselli, M., Camilli, L., Castrucci, P., Nanni, F., Gobbo, S.D., Gautron, E., Lefrant, S., Crescenzi, M.D.: In situ formation of noble metal nanoparticles on multiwalled carbon nanotubes and its implication in metalnanotube interactions. *Carbon* 50(3), 875–884 (2012), DOI: 10.1016/j.carbon.2011.09.048

17. Stepanov, A., Filippov, G.: Channeling of low energy atomic particles in carbon nanotubes with heterojunctions. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 402, 263–266 (2017), DOI: 10.1016/j.nimb.2017.03.134
18. Stuart, S.J., Tutein, A.B., Harrison, J.A.: A reactive potential for hydrocarbons with intermolecular interactions. *The Journal of Chemical Physics* 112(14), 6472–6486 (2000), DOI: 10.1063/1.481208
19. Ziegler, J.F., Biersack, J.P.: The stopping and range of ions in matter. In: Bromley, D.A. (ed.) *Treatise on Heavy-Ion Science: Volume 6: Astrophysics, Chemistry, and Condensed Matter*, pp. 93–129. Springer US, Boston, MA (1985), DOI: 10.1007/978-1-4615-8103-1\_3

# A Fully Conservative Parallel Numerical Algorithm with Adaptive Spatial Grid for Solving Nonlinear Diffusion Equations in Image Processing

*Andrey D. Bulygin*<sup>1,2</sup>, *Denis A. Vrazhnov*<sup>2,3</sup>

© The Author 2019. This paper is published with open access at SuperFri.org

In this paper we present simple yet efficient parallel program implementation of grid-difference method for solving nonlinear parabolic equations, which satisfies both fully conservative property and second order of approximation on non-uniform spatial grid according to geometrical sanity of a task. The proposed algorithm was tested on Perona–Malik method for image noise filtering task based on differential equations. Also in this work we propose generalization of the Perona–Malik equation, which is a one of diffusion in complex-valued region type. This corresponds to the conversion to such types of nonlinear equations like Leontovich–Fock equation with a dependent on the gradient field according to the nonlinear law coefficient of diffraction. This is a special case of generalization of the Perona–Malik equation to the multicomponent case. This approach makes noise removal process more flexible by increasing its capabilities, which allows achieving better results for the task of image denoising.

*Keywords:* Perona–Malik method, nonlinear Schrödinger equation, fast parallel algorithm, fully conservative numerical scheme.

## Introduction

In recent years applications of mathematical physics methods in image processing have been of great interest. One of such state-of-the-art approaches is Perona–Malik method used for image denoising as numerical solution of partial differential equations (PDEs) [1]. This method was further developed in many works, e.g. [2–4]. The idea of this method is relatively simple: authors suggest to numerically solve PDE (for instance stationary diffusion equation) with image being denoised as initial conditions. This is equivalent to image blurring with Gaussian filter and has deep connection with other methods of denoising filters constructions based on application of Green function for PDEs [5]. For example, Gaussian filter is a Green function for diffusion equation. Thus, we can say that usage of diffusion equation in Perona–Malik approach for given image is equivalent to convolution of this image with Gaussian filter. It should be mentioned that application field of PDEs in image processing is not only denoising, but also broken image restoration, known as inpainting process [6]. Development of denoising methods based on solution of PDEs is troublesome because of heavy computational load. But usage of clusters and graphical processor units (GPUs) can overcome this shortage. That is why parallel implementation of algorithms based on Perona–Malik approach is very promising. In this paper we propose an algorithm for image denoising based on non-linear diffusion equation. Parallel implementation was done on FORTRAN 90 with MPI (Message Passing Interface) and Intel Fortran compiler. The solution of the given equation is based on implicit finite difference scheme of the second order.

---

<sup>1</sup>V.E. Zuev Institute of Atmospheric Optics SB RAS, Tomsk, Russian Federation

<sup>2</sup>Laboratory of Biophotonics, Tomsk State University, Tomsk, Russian Federation

<sup>3</sup>Institute of Strength Physics and Materials Science of SB RAS, Tomsk, Russian Federation

## 1. Generalized Perona–Malik Approach

Let us consider the following PDE of the given form:

$$\partial_t \psi + \nabla D(\nabla \psi) \nabla \psi = 0. \quad (1)$$

Here  $\psi(x, y)$  is a field (generally speaking - complex numbered) corresponding to processed image being denoised. Parameter  $t$  is evolutionary one. In general case coefficient  $D(\nabla \psi)$  is complex numbered function [7]. Let us analyze specific form of  $D(\nabla \psi)$ :

$$D = \exp(-i\varphi) \exp(-(\nabla \psi/q)^2). \quad (2)$$

Here  $\varphi$  is a phase, tunable parameter whose inference on efficiency of generalized Perona–Malik approach is a subject of interest. With  $\varphi = 0$  we get ordinal case of non-linear diffusion equation. With  $\varphi = \pi/2$  we get non-linear Schrödinger equation (in generalized sense, because non-linear Schrödinger equation assumes non-linearity of third order of field).

## 2. Numerical Scheme

According to the given initial conditions we construct non-uniform grid with symmetry to the origin:  $x_I = -x_{N+1-I} = x_{I-1} + h_I$  where  $h_I = h_{I-1}\epsilon(I)$ ,  $I = N/2 + 1..N$ ,  $N$  is the number of grid nodes. In the case of axially symmetric  $x_{N/2+1} = h_0/2$  and  $h_{N/2+1} = h_0$ . The symbol is entered here  $\epsilon \leq 1$  mesh heterogeneity parameter of the grid.

Similarly, we perform a partition of the orthogonal coordinate:  $y_I$ . For the simplicity let us assume, that there are equal number of points on each computational node.

We solve this task with alternative directions method [8], which allows to solve one-dimensional diffusion task on each half-step:

$$\frac{\partial}{\partial t} \psi = \frac{\partial}{\partial x} (D(\psi, \frac{\partial}{\partial x} \psi, x) \frac{\partial}{\partial x} \psi) + F(x). \quad (3)$$

The given equation is a non-linear one, and we solve it by Newton–Raphson method [8]. This leads to the necessity of solving a linear system of differential equations on each step. Condition of second order approximation of diffusion operator for scheme on non-uniform grid can be deduced from the condition of the following equations coherence:

$$\frac{\partial}{\partial x} (D(x)) \frac{\partial}{\partial x} \psi_j = \alpha_j \psi_{j-1} + \beta_j \psi_j + \gamma_j \psi_{j+1} + O(h^3). \quad (4)$$

Again, for simplicity, let each processor have the same number of points  $m_l = N/M$ , where  $M$  is the number of processors.

Global index  $J$  relates to local index  $j$  on processor with number  $q$  as follows:  $J = (q - 1) * m_l + j$ . The numerical implementation of the diffraction step will be implemented on a three-point “cross” scheme with a vory order of accuracy of approximation of the Laplace operator on a non-uniform grid. Numerical implementation of diffraction step will be performed by three point “cross” scheme with second order approximation of Laplace operator on non-uniform grid. In this case we need to solve a system of equation of the kind:

$$a_J \psi_{J-1}^{l+1} - c_J \psi_J^{l+1} b_J \psi_{J+1}^{l+1} = -f_J^l. \quad (5)$$

To solve this system of algebraic equations, we will use generalized method of fast parallelization in complex case [9].

### 3. Results of Numerical Calculations

The following are examples (Fig. 1) of the original image of a noisy image and an example of the result of a partially noise-free image:

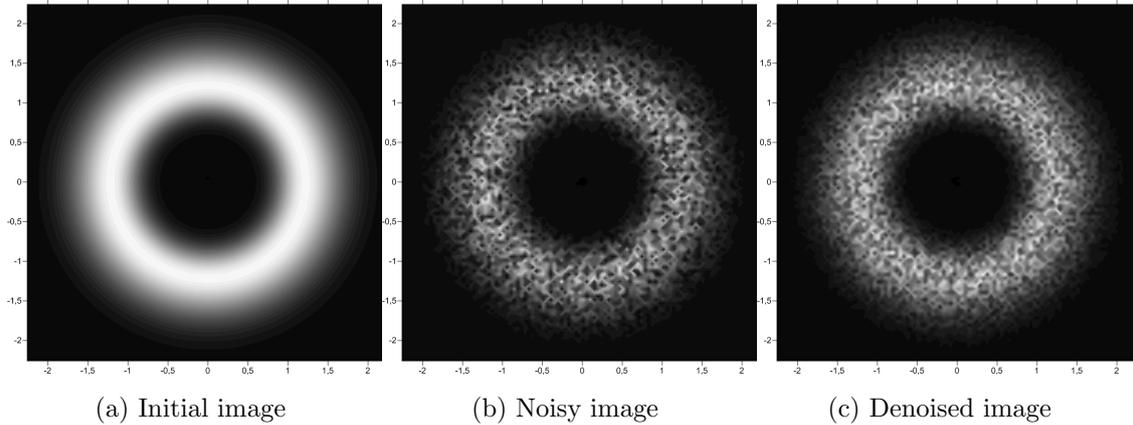


Figure 1. Examples of images

We propose to measure the quality of denoising by RMS error of difference between initial and noised image:  $m^2(t) = (|\psi(t)| - |\psi_{iso}|)^2$ . Bellow, graphics  $m^2(t)$  (Fig. 2) acquired from the solution of equations with Perona–Malik method 1 for different phase values in diffusion equation 2 are presented:

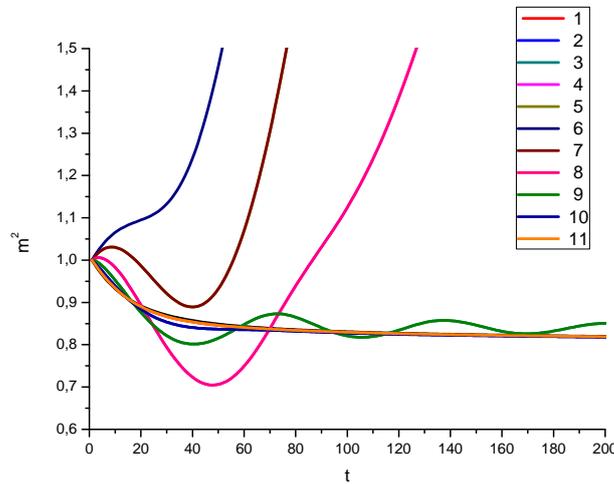


Figure 2. Dependance  $m^2(t)$  of time for different values  $k$ ,  $\varphi = k\pi/6$ , where:  $k=1,2..11$

From the presented pictures it can be seen, that efficiency of image denoising significantly depends on parameter  $\varphi$ , and for the case under study value  $\varphi = 2\pi/3$  is optimal.

## Conclusion

In this paper we present fully conservative numerical scheme (in a weak sense), which allows to control correctness of the equation solution by tracing motion integrals and using this to correct evolution variable step. This numerical scheme was parallelly implemented for complex generalization of Perona–Malik equation. It was shown, that this approach extends denoising abilities of the method, which allows to achieve better results for the image noise removal task. Thus, in particular, in the present study we show that complex parameter equals one is optimal for image denoising task when phase is equal  $\varphi = 2\pi/3$ .

## Acknowledgements

The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

The Siberian Branch of the Russian Academy of Sciences (SB RAS) Siberian Supercomputer Center is gratefully acknowledged for providing supercomputer facilities.

The reported research was funded by Russian Foundation for Basic Research and the government of the Tomsk Region of the Russian Federation, grant No. 18-41-703004 \18.

The work was carried out under partial financial support of the Russian Fund of the Fundamental Research grant No. 17-00-00186.

This work was performed within the frame of the Fundamental Research Program of the State Academies of Sciences for 2013-2020, line of research III.23.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Perona P., Malik J.: Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on pattern analysis and machine intelligence 12(7), 629–639 (1990)
2. Wei G.W. : Generalized Perona–Malik equation for image restoration. IEEE Signal Processing Letters 6(7), 165–167 (1999)
3. Wang N., et al.: A hybrid model for image denoising combining modified isotropic diffusion model and modified Perona–Malik model. IEEE Access 6, 33568–33582 (2018), DOI: 10.1109/ACCESS.2018.2844163
4. Maiseli B., et al.: Perona–Malik model with self-adjusting shape-defining constant. Information Processing Letters. 137, 26–32 (2018), DOI: 10.1016/j.ipl.2018.04.016
5. Vrazhnov D.A., Shapovalov A.V., Nikolaev V.V.: Symmetries of differential equations in computer vision applications. Computer Research and Modeling 2(4), 369–376 (2010)
6. Bertalmio M., et al.: Image inpainting. In: Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press, pp. 417–424. Addison-Wesley Publishing Co. (2000)

7. Tschumperl D., Deriche R.: Anisotropic diffusion partial differential equations in multi-channel image processing: framework and applications. *Advances in Imaging and Electron Physics (AIEP)*, pp. 145–209, AcademicPress (2007)
8. Tadmor E.: A review of numerical methods for nonlinear partial differential equations. *Bulletin of the American Mathematical Society* 49(4), 507–554 (2012), DOI: 10.1090/S0273-0979-2012-01379-4
9. Terekhov A.V.: A fast parallel algorithm for solving block-tridiagonal systems of linear equations including the domain decomposition method. *Parallel Computing* 39(6-7), 245–258 (2013), DOI: 10.1016/j.parco.2013.03.003

# Parametrization of the Elastic Network Model Using High-Throughput Parallel Molecular Dynamics Simulations

*Philipp S. Orekhov*<sup>1,2,3</sup>, *Ilya V. Kirillov*<sup>1</sup>, *Vladimir A. Fedorov*<sup>3</sup>,  
*Ilya B. Kovalenko*<sup>3,4,5,6</sup>, *Nikita B. Gudimchuk*<sup>3,7</sup>, *Artem A. Zhmurov*<sup>1,2</sup>

© The Authors 2019. This paper is published with open access at SuperFri.org

Even when modern computational platforms and parallel techniques are used, conventional all-atom simulations are limited both in terms of reachable timescale and number of atoms in the biomolecular system of interest. On the other hand, coarse-grained models, which allow to overcome this limitation, rely on proper and rigorous parametrization of the underlying force field. Here, we present a novel iterative approach for parametrization of coarse-grained models based on direct comparison of equilibrium simulations at all-atom and coarse-grained resolutions. In order to assess the accuracy of our method, we have built and parametrized an elastic network model (ENM) of the tubulin protofilament consisting of four monomers. For this system, our method shows good convergence and the parametrized ENM reproduces protein dynamics in a finer way when compared to ENMs parametrized using the conventional approach. The presented method can be extended to other coarse-grained models with a slight adjustment of the equations describing the iterative scheme.

*Keywords: molecular dynamics, coarse-grained models, tubulin, elastic network model, high-throughput simulations, parallel simulations.*

## Introduction

Despite huge advances in the computational techniques the atomistic simulations of molecular dynamics are still limited in the time and space domains to  $\mu\text{s}$  and millions of atoms, respectively, what makes various coarse-grained approaches a valuable alternative. Among the latter, the elastic network models have gained a considerable success: though relatively simple they can yield decent insights into molecular mechanisms of protein function. Particularly, the simple Gō-models have proved themselves advantageous in the studies of protein folding, allostery, etc. [2]. They consider protein as a network of  $C_{\alpha}$ -atoms connected with uniform springs when closer than a given cutoff distance. However, homogeneity of such networks, which pursues models with very few fitted parameters, imposes certain limitations on the accuracy of the obtained ENMs [4]. The latter can be overcome using a series of all-atom (AA) simulations produced by modern high-throughput MD approaches in order to bring the CG models to the highest accordance with their AA counterpart.

We suggest a methodology for accurate parametrization of ENMs based on the information derived from a batch of all-atom simulations, which accounts for inconsistency of inter-residue pair contacts, as well as the heterogeneity of their strength. We utilize the developed approach in order to build a CG model of the tubulin protofilament — a structural element of microtubule, which is a cytoskeletal structure essential for intracellular transport, cell division, cell motility,

<sup>1</sup>Moscow Institute of Physics and Technology, Dolgoprudny, Russia

<sup>2</sup>Sechenov University, Moscow, Russia

<sup>3</sup>M.V. Lomonosov Moscow State University, Moscow, Russia

<sup>4</sup>Federal Research and Clinical Center of Specialized Medical Care and Medical Technologies, Federal Medical and Biological Agency of Russia, Moscow, Russia

<sup>5</sup>Astrakhan State University, Astrakhan, Russia

<sup>6</sup>Scientific and Technological Center of Unique Instrumentation, RAS, Moscow, Russia

<sup>7</sup>Center for Theoretical Problems of Physicochemical Pharmacology, RAS, Moscow, Russia

etc. [5]. We demonstrate that our method leads to a more reliable description of protein dynamics comparing to the conventional ENM approaches. Since our approach is based on the direct comparison of the coarse-grained and all-atom simulations, it can be applied to more complex coarse-grained models. For instance, if the electrostatic energy is explicitly included to the coarse-grained potential energy function, its contribution has to be excluded from potential that is parameterized [3]. In our approach, this will be done automatically throughout iterative process.

## 1. Methods

All-atom MD simulations of tubulin tetramers were carried out in the GROMACS 5 suite [1] using the CHARMM27 force field. The PDB entry 3J6F, corresponding to a fragment of microtubule bound to GDP, was used in order to build an atomic model for simulations. The following standard protocol was used for simulations: the NPT ensemble controlled by means of Nose–Hoover thermostat ( $T_{ref} = 303$  K,  $\tau_T = 2$  ps) and Parrinello–Rahman barostat (isotropic pressure,  $p_{ref} = 1$  bar,  $\tau_p = 2$  ps, compressibility =  $4.5 \cdot 10^5$  bar<sup>-1</sup>); time step — 4 fs, allowed by the mass rescaling; the Verlet cut-off scheme and PME for the long-range electrostatics. In total, two 1  $\mu$ s-long trajectories were obtained, last 300 ns of each run was used for further analysis.

To parametrize the ENM, we first extracted mean distances and standard deviation for each interparticle distance from the AA simulations to use as reference values. At each iteration,  $4 \times 10^6$  steps of CG dynamics is performed, and the same values are extracted from the last  $3 \times 10^6$  steps. The parameters are then adjusted and procedure is repeated until convergence is reached. The update rules for the parameters are:

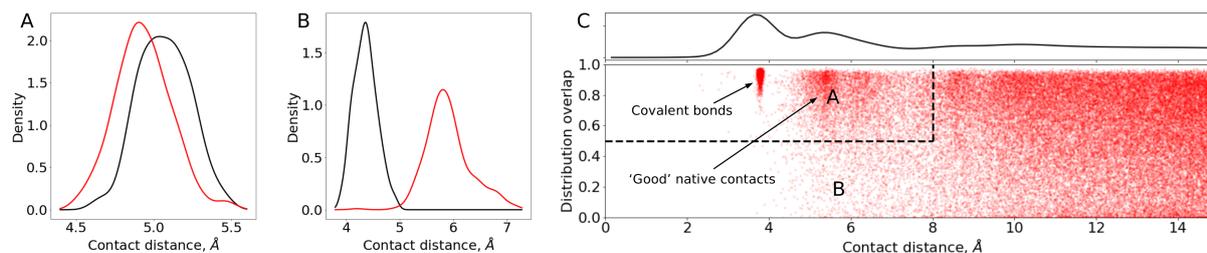
$$\frac{1}{K_{i,j}^{n+1}} = \frac{1}{K_{i,j}^n} - \alpha \frac{(\sigma_{i,j}^0)^2 - (\sigma_{i,j}^n)^2}{k_B T}, \quad b_{i,j}^{n+1} = b_{i,j}^n + \beta \frac{b_{i,j}^0 - b_{i,j}^n}{\sigma_{i,j}^0}. \quad (1)$$

Here,  $K_{i,j}^n$  is a spring constant for the contact between  $i$ -th and  $j$ -th particles at  $n$ -th iteration,  $\sigma_{i,j}^n$  is a dispersion, computed from simulations at  $n$ -th iteration,  $k_B$  is Boltzmann constant, and  $T$  is the temperature;  $b_{i,j}^n$  is an average distance between particles  $i$  and  $j$  throughout the simulation run at iteration  $n$ ; zero value for the iteration index indicates that the respective value is computed from the reference AA simulation run;  $\alpha = 0.1$  and  $\beta = 0.1$  are dimensionless iteration parameters.

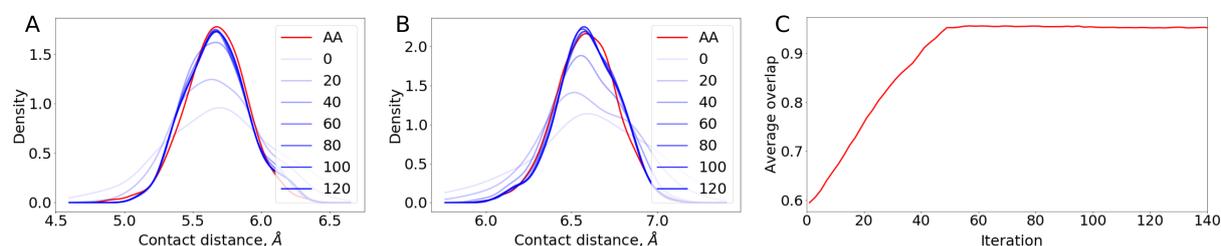
## Results and Discussion

Defining contacts in CG models solely on the distance cut-off can lead to the certain amount of false-contacts. These not only include the pairs of particles that are not interacting directly, but also those that cannot be properly described by the chosen potential. For instance, double-well potentials cannot be described by harmonic ENMs and thus have to be excluded. To assess the quality of contacts, we compared two independent AA simulation runs. We derived the distributions of interparticle (contact) distances from two runs and computed an overlap between them (Fig. 1a, b). We then excluded the contacts, for which the overlap was less then 50 % (Fig. 1b, c). The distribution of mean distances of such “good” contacts (Fig. 1c, top) allowed us to choose the optimal value for the distance cut-off (8 Å).

The iterative process for optimization of the model parameters converged in  $\sim 50$  iterations, at which point the average distribution overlap reaches the plateau at value of  $0.954 \pm 0.002$

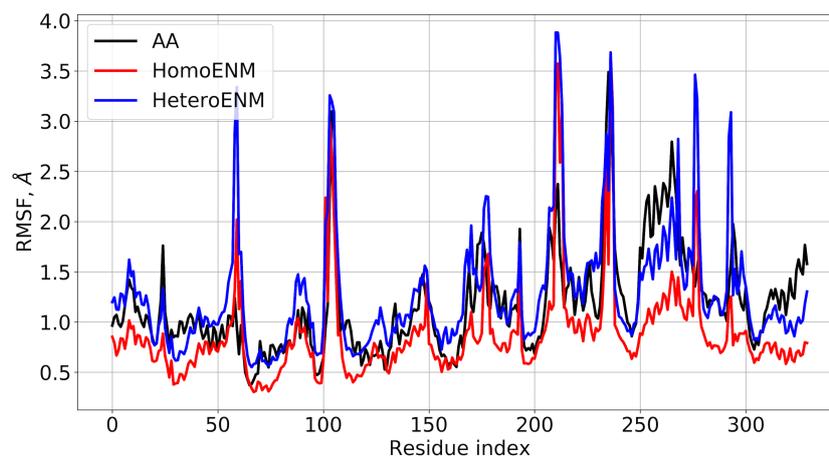


**Figure 1.** Distributions of interparticle distances for “good” (A) and “bad” (B) contacts obtained from two independent AA simulation runs (red and black curves); normalized distribution of interparticle distances over all contacts in the system with sufficient distribution overlap ( $> 0.5$ ) (C, top) and a scatter plot of the distances and distribution overlaps for all the contacts (C, bottom) with the “good” contacts embraced in the dashed rectangle



**Figure 2.** Distributions of the interparticle distances for two arbitrary chosen contacts (A, B): blue curves represent the distributions from CG MD runs at different iterations of the parametrization process, red curve shows the correspondent distribution from the AA simulation; the average overlap of the distributions from the CG simulations with the AA distributions as a function of the iteration number (C)

(Fig. 2c). Visual analysis of interparticle distributions showed a good correspondence between the CG and AA simulations both in distributions of height and dispersion. As expected, the



**Figure 3.** Root mean-square fluctuations computed per residue for one of  $\alpha$ -tubulin monomers from AA MD simulations and CG MD simulations using the homogeneous model with the uniform spring constant (homoENM) and the heterogeneous model based on AA simulations (heteroENM)

distributions obtained from CG simulations lacked certain features, observed in AA simulations, e.g. small shoulders in some cases (Fig. 2a, b).

The overall quality of the parametrized model was assessed by comparing the RMSF curves obtained from the AA MD simulations and CG simulations using either the developed ENM or the homogeneous ENM with the uniform spring constant of 10000 kJ/mol·nm<sup>2</sup>. As one can see in Fig. 3, the per-residue fluctuations calculated based on our model match those computed from the AA simulations better comparing to the conventional homogeneous ENM model. The mean-square error equals 0.24 and 0.19 Å<sup>2</sup> for the homogeneous ENM and our heterogeneous ENM, respectively.

To conclude, we developed a novel approach to parametrize the CG models of biomolecular systems. The method outperforms the conventional approach for parametrization of ENM both in terms of the convergence and the model quality while remaining robust and simple. The proposed procedure can be extended to other CG models and the resulting models can be easily incorporated into the existing MD software, which makes it interesting to anyone using this tool in their research.

## Acknowledgments

This study was designed and conducted with support from the Russian Foundation for Basic Research, projects 17-00-00479 and 16-34-60113. Assessment of the quality of interparticle contacts is a part of Russian Science Foundation research project 17-71-10202. The research was carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University supported by the project RFMEFI62117X0011.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Abraham, M.J., Murtola, T., Schulz, R., Páll, S., Smith, J.C., Hess, B., Lindahl, E.: GRO-MACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1, 19–25 (2015), DOI: 10.1016/j.softx.2015.06.001
2. Hills, R.D., Brooks, C.L.: Insights from coarse-grained Gō models for protein folding and dynamics. *International journal of molecular sciences* 10(3), 889–905 (2009), DOI: 10.3390/ijms10030889
3. Hyeon, C., Onuchic, J.N.: Mechanical control of the directional stepping dynamics of the kinesin motor. *Proc. Natl. Acad. Sci. USA* 104(44), 17382–17387 (2007), DOI: 10.1073/pnas.0708828104
4. Lyman, E., Pfaendtner, J., Voth, G.A.: Systematic multiscale parameterization of heterogeneous elastic network models of proteins. *Biophysical journal* 95(9), 4183–4192 (2008), DOI: 10.1529/biophysj.108.139733
5. Nogales, E.: Structural insights into microtubule function. *Annual review of biochemistry* 69(1), 277–302 (2000), DOI: 10.1146/annurev.biochem.69.1.277

# Facilitating HPC Operation and Administration via Cloud

Chaoqun Sha<sup>1,2</sup>, Jingfeng Zhang<sup>1</sup>, Lei An<sup>1</sup>, Yongsheng Zhang<sup>1</sup>,  
Zhipeng Wang<sup>3</sup>, Tomi Ilijaš<sup>4</sup>, Nejc Bat<sup>4</sup>, Miha Verlič<sup>4</sup>, Qing Ji<sup>1</sup>

© The Authors 2019. This paper is published with open access at SuperFri.org

Experiencing a tremendous growth, Cloud Computing offers a number of advantages over other distributed platforms. Introducing the advantages of High Performance Computing (HPC) also brought forward the development of HPCaaS (HPC as a Service), which has mainly focused on flexible access to resources, cost-effectiveness, and the no-maintenance-needed for end-users. Besides providing and using HPCaaS, HPC centers could leverage more from Cloud Computing technology, for instance to facilitate operation and administration of deployed HPC systems, commonly faced by most supercomputer centers.

This paper reports the product, EasyOP, developed to realize the idea that one or more Cloud or HPC facilities can be run over a centralized and unified control platform. The main purpose of EasyOP is that the information of HPC systems hardware and system software, failure alarms, jobs scheduling, etc. is sent to the Wuxi cloud computing center. After a series of analysis and processing, we are able to share many valuable data, including alarm and job scheduling status, to HPC users through SMS, email, and WeChat. More importantly, with the data accumulated on the cloud computing center, EasyOP can offer several easy-to-use functions, such as user(s) management, monthly/yearly reports, one-screen monitoring and so on. By the end of 2016, EasyOP successfully served more than 50 HPC systems with almost 10000 nodes and over of 300 regular users.

*Keywords:* HPC, supercomputer, monitoring, notifications, cloud, operation, administration, EasyOP.

## Introduction

Cloud computing definitely takes top rank in recent information technology popular paradigm list, due to the many promising advantages it brings. For example, the access to resources is flexible and cost-effective, since it is neither necessary to invest a large amount of money on a computing infrastructure, nor pay salaries for maintenance [11, 22, 31]. Also, there are series of works which focus on HPC cloud, also known as HPCaaS (HPC as a Service). Some have mainly contributed to understanding the cost-benefits of using cloud over on-premise clusters [10, 23]. Other aimed at evaluating the performance gap between cloud and on-premise resources [12, 20, 28]. Additionally, a lot of effort has been invested into the HPC job scheduling, as well as application testing and tuning within the cloud environments [5, 6, 15, 19, 30]. Therefore, it could be found that most interests of combining HPC and cloud are laid on using cloud as IaaS (infrastructure as a Service).

In fact, besides buying new infrastructure and testing new technology, most HPC centers have to continuously monitor the utilization and the capacity of HPC facilities to ensure research needs are adequately met, to plan and approve operational policies, and to advise & review proposals from related departments for next step planning to ensure adequate usage statistics. HPC systems are complicated combinations of application software, system software, processors, co-processors, memory, networks and storage systems, which evolve rapidly with

<sup>1</sup>Sugon Information Industry Co, Beijing, China

<sup>2</sup>School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China

<sup>3</sup>High School Affiliated to Renmin University, Beijing, China

<sup>4</sup>Arctur računalniški inženiring d.o.o., Nova Gorica, Slovenia

technology changes. What makes it even more challenging is the fact that the HPC systems are getting larger and larger, and by consequence induce higher unreliability and complexity. Many HPC centers have several HPC systems running subsequently or together. Therefore, efficient operating of HPC systems (either several or large) efficiently conducting usage statistics and user administration are inevitable key issues. Smooth operation depends on efficient error-free server operation, stable server room conditions, and stable computing environment, which are quite important to HPC end-users [18]. Note that the qualified and experienced HPC operation engineers are not easy to recruit, because HPC operation engineers require cross-discipline knowledge accumulation and sufficient on-hand experiences.

Most HPC vendors provide operation management and administration software, such as Open Manage Essentials from Dell [13], HP's OperiView [14], Tivoli from IBM [16], and Sugon's Gridview [24] and so on. Some software solutions can even support grid or cloud environment operation. Cacti [17] and MRTG [25] are solutions to create RRD-Tool graphs that are usually used to show bandwidth consumption in network links. Nagios' main features is the support for plugins that are used to collect monitoring information from the monitored objects [2]. PARMON [3] and Rvision [8] can monitor specific systems, while GridEye [9] and Ganglia [21] are able to support grid environments. Zenoss [1], and other operation centric software, have gained a pretty substantial market share, even being adopted in Bank of America and US Air Force. All these operation software and monitoring tool-kits compose a scenario where traditional HPC system operation software is relatively mature.

As for the usage statistics and reporting side of operation, only a few software options have implemented sufficient solutions. TACC Stats is a continuous monitoring tool for HPC systems that collects data at the core and processor level for every job executing on a monitored system [7]. This data can be aggregated at the system, group, user, application, job, node or core level. TACC Stats also includes capabilities for generating several different reports including a report giving a resource use profile. Open XDMoD portal provides a rich set of analysis and charting tools that let users quickly display a wide variety of job accounting metrics over any desired timeframe [26]. Two additional tools, which provide quality-of-service metrics and job-level performance data, have been developed and integrated with Open XDMoD to extend its functionality. On the other hand, some niche software companies have been working on report-generating service, such as Zenoss [1] and Rizhiyi [27].

Leveraging the advanced characteristics of cloud computing makes it practical to send the HPC systems' information, including data regarding CPU, network, memory, storage, and job etc., to a cloud computing center, and then spread the information (after certain data-mining) to the persons in charge of reorganization. This is the main idea of EasyOP, a product recently developed. The benefits of EasyOP include:

- operate several HPC systems at the same time;
- satisfy both remote operating and security demands by only allowing the operating information to go out;
- generate live data of computing resource and jobs, triggering notices for hardware alarms, job starts, job ends, abnormal job exit, and so on;
- automatically generate reports not only for a certain HPC System, but also all the connected systems;
- spread the analyzed data through email, SMS, or Wechat;
- realize the remote monitoring of HPC systems, specially on smart portable devices.

This article is organized as follows. In Section 1, we explain the used methods and challenges behind the EasyOP. We also share the principle functions designed for HPC system operation and administration in Section 2 and 3, respectively. Section 4 exhibits the practical observation we mined through the data already collected from 50+ HPC systems. Finally, the last Section contains conclusions and our plans for future work.

## 1. Method

### 1.1. Sugon and HPC

Sugon was born from the Institute of Computing Technology of the Chinese Academy of Sciences (ICT), and was the first (and now largest) supercomputer vendor in China as shown in Fig. 1. Since 1990, Sugon has been working on supercomputing, producing seven generations of HPC systems, such as Dawning I and Dawning 1000 to 6000. It has successfully supported more than 10,000 HPC projects. In 2014, Sugon was successfully listed on the Shanghai Stock Exchange (Stock code: 603019). Besides the Silicon Cube, Sugon has successfully developed the Sugon 4000A (#10, Top500 2004) and Nebulae (#2, Top500 2010). Sugon has continuously dominated the China HPC Top100 for the past 9 times in 10 years and hit the 3rd of Top500 (Nov, 2015) per vendor market share. In 2016, Sugon was appointed as the only company to develop the prototype of Chinese Exascale HPC by the Ministry of Science and Technology in China.

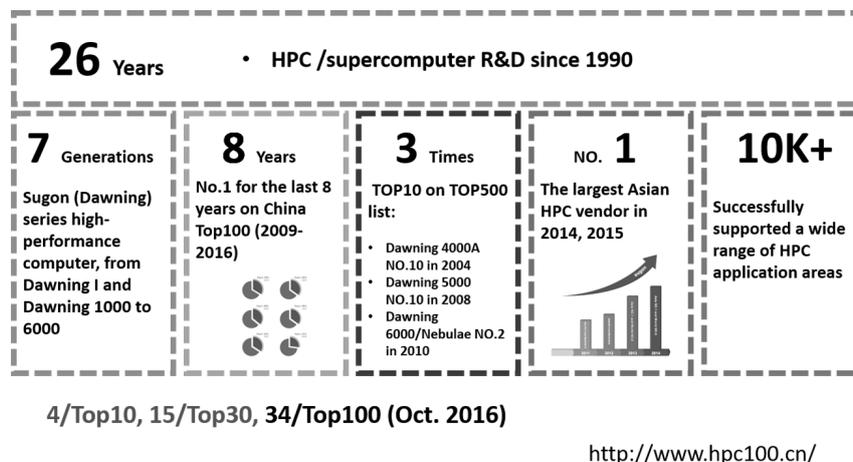


Figure 1. Sugon and HPC

### 1.2. Methodology and Challenges for EasyOP Development

EasyOP cloud service platform collects the operation and administration information of HPC systems without geographical limitations and sends the information to the Wuxi cloud computing center. It analyzes the status and performance data of HPC systems, generates live graphs and reports graphically, and then spreads the reports to end-users through Email, SMS, and WeChat. Fig. 2 shows the topology strategy of EasyOP. From the perspective of data flow, the whole process of EasyOP is composed by data collection and acquisition, data transmission, data storage, data processing, data analysis, data display etc. The architecture design of EasyOP is shown in Fig. 3.

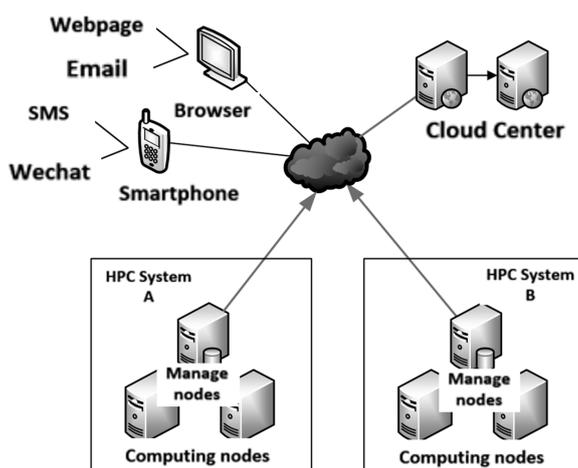


Figure 2. The network topology of EasyOP

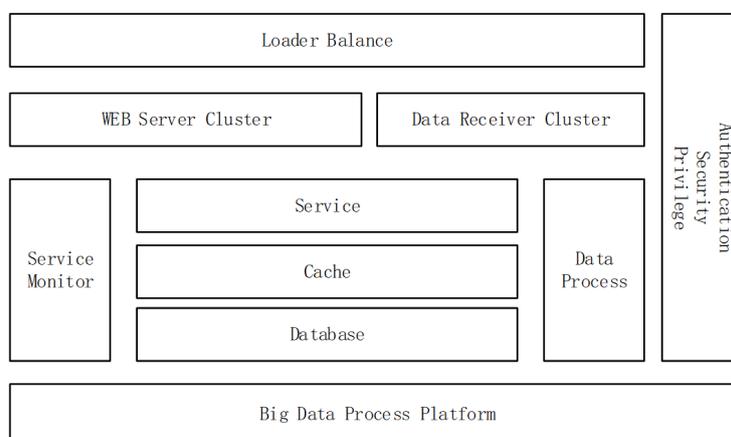


Figure 3. Architecture design of EasyOP

To develop EasyOP, we had to overcome a series of challenges. The first challenge of EasyOP R&D comes from data collection and acquisition. EasyOP relies on Gridview, a mature HPC system management product used for more than ten years. Gridview collects many data and indicators of HPC system by series of add-ins. The collected data are verified to be correct, reliable, and in a reasonable format. In order to send this data to the cloud center, we need to select the appropriate communications and transfer protocol, set up the service platform, identify the suitable carrier, choose the cloud center, assign a domain name, and so on. The HTTPS protocol has been adopted to ensure the data security is transferred. A private protocol was also developed based on HTTPS considering appropriate commonality, which shares the compatibility of various data format and data package size. The Wuxi cloud computing center has been chosen as our partner, because the center operates for 5+ years with good users assessment and can support the internet service from China Mobile, China Unicom, and China Telecom. The three companies provide 95 % of internet service in China. In testing, we found that some cities only connect with a certain internet service company. Therefore, we set up two IP address (China Unicom and China Telecom each) to resolve the EasyOP’s domain. All HPC systems linked to EasyOP transmit the system information and job information to the Wuxi cloud computing center automatically. By default, the sending time interval is 1 minute, which

could be customized for each HPC center. After the data analysis at EasyOP, the results will be sent to the EasyOP's public account on Wechat and the public webpage of EasyOP.

Data storage and analysis are the next challenges observed right after data collection and transfer. In the cloud center, a highly available and high reliable data injection and storage solution is indispensable for completely receiving and almost real-time calculating mass data (currently 300G per day). We have set up the load balancer and realized multi-point concurrent data processing. The data used in EasyOP can be classified into static configuration data, real-time status data, and history data, etc. We took different storage solutions for different data. For instance, the big history data were stored and processed by a Hadoop cluster with good scalability. In contrast, the real-time status data are stored both in the Hadoop cluster and cache. As the number of served HPC systems increased, real-time status data grew quickly. To solve this problem, we designed the storage alarms and conducted data compression for repeated data, such as indicators' name and time.

We also paid much attention to function planning and user interface design, especially to data presentation. This is the core competitive advantage of EasyOP. From user authentication, rights management, contents composition, workflow planning, to letter style selection, many technologies and experiences of web development and design have been implemented. To balance the cost and stability, we borrowed the advantages of web service SOA and EasySOC [4], and the suggestion from Sheng's review [29], and then set up the operation platform for EasyOP. The design of the interface and the inner business process are independent of each other. The browser compatibility has been included in our design by adopting jQuery and Bootstrap.

EasyOP receives regular updates every two weeks. We have implemented automatic development technologies to support code writing, testing, and deploying. Specifications of demands analysis, prototype design, solution design, coding, code debug, test, and new version release have been confirmed and put into the practice with the automatic development tools.

## 2. Operation

EasyOP offers the functions of single-page monitoring, specific cluster monitoring, alarming, and setting change and control. Using the single-page monitoring, operation engineers are able to take a quick glance across the live working status for all systems under their command. It is especially suitable for an HPC center with multiple systems. On the initial page the system size distribution, system nodes/cores, jobs, major application, busy system ranking can be quickly summarized. The specific monitoring furthermore displays information of one chosen HPC system. In addition to the information similar from the single-page monitoring, the specific monitoring also shows the live updates of job scheduling for each queue. Examples for one-page monitoring and specific monitoring are shown in Fig. 4.

"Nip in the bud" minimizes the damage on the HPC system and helps to minimize downtime and ultimately save money. Alarms from EasyOP make this possible. Some alarm examples are shown in Fig. 5. Various alarms can come from switches, servers (e.g. computing network, CPU temperature, disk usage, file system, GPU temperature, IPMI network, MIC temperature, MIC availability, management network, disk availability, server power off, fan abnormal, etc.), RACK (e.g. management network, node abnormal, fan abnormal), storage, blade cabin (e.g. abnormal blade communication, abnormal power, management network). Who and how is supposed to receive this alarms can be set through the setting page. If one wants to change the threshold value of the alarm, he/she needs to set the value at the level of the Gridview system, meaning



Figure 4. Examples for one-page monitoring and specific monitoring

one level below with higher privileges. Because the SMS might be charged by telecommunication companies, users are suggested to take email or smart phone notifications as the primary choice.

| Live |          | Past           |                            | Level | Name                |              |
|------|----------|----------------|----------------------------|-------|---------------------|--------------|
| #    | Type     | Node           | Alarm                      | Level | Time                | Latest value |
| 1    | LINUX    | test           | Network Not Available      | High  | 2017-01-05 16:14:58 | 0            |
| 2    | TC6600   | TC6600         | Network Not Available      | High  | 2016-05-10 10:31:11 | 0            |
| 3    | TC5600-H | TC5600H        | Network Not Available      | High  | 2016-05-05 14:59:42 | 0            |
| 4    | LINUX    | dcv20          | IPMI Network Not Available | High  | 2016-04-26 11:27:27 | 0            |
| 5    | LINUX    | 123456         | Network Not Available      | High  | 2016-04-08 20:40:02 | 0            |
| 6    | LINUX    | test-linux     | Network Not Available      | High  | 2016-04-08 20:19:21 | 0            |
| 7    | LINUX    | dcv20_SYS_Fan6 | System Fan Rate Low        | Low   | 2016-03-02 15:22:16 | 0.000rpm     |
| 8    | LINUX    | dcv20_SYS_Fan5 | System Fan Rate Low        | Low   | 2016-03-02 15:22:16 | 0.000rpm     |

Figure 5. Alarm examples in EasyOP

No doubt, Smartphone is the most efficient way to convey the alarms to users. Therefore, EasyOP developed the alarm function based on WeChat. Users can also watch the basic monitoring page and job page via EasyOP official account on WeChat. In the meantime, the EasyOP service group can directly contact and communicate with users by using WeChat. Snapshots of using EasyOP on Smartphone are shown in Fig. 6.

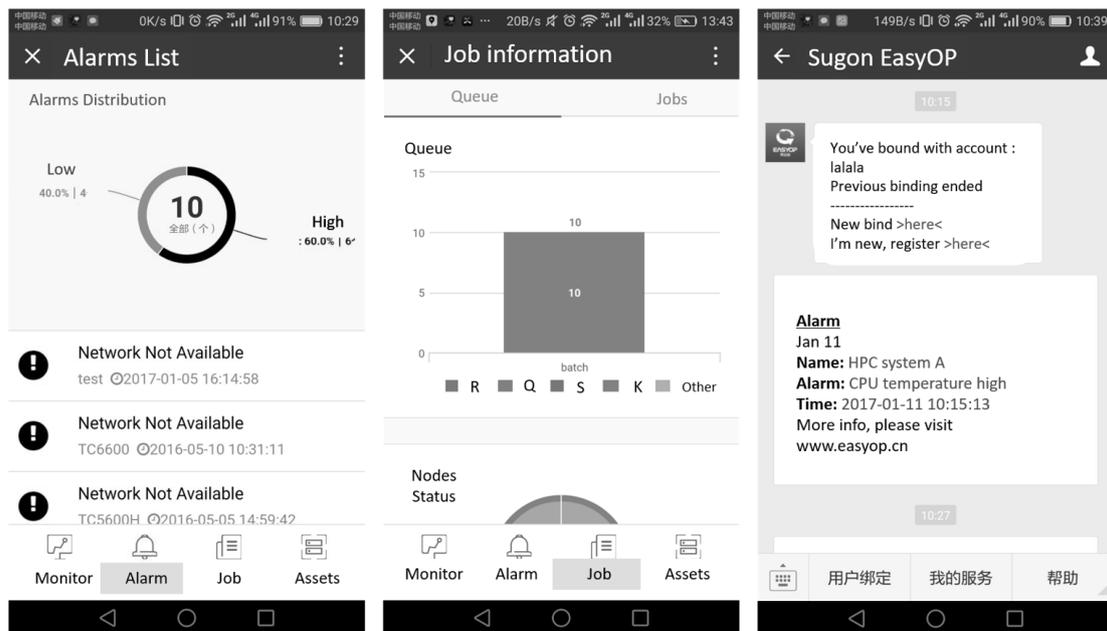


Figure 6. Examples of EasyOP service on a Smartphone

### 3. Administration

Through EasyOP, we can conduct asset management, job tracing, user management, customized statistical analysis, monthly/yearly report, and setting via browser or smart phone. For the asset management, EasyOP displays the general information, including name, joining time, node number, node type, switch SN, city address, version of Gridview, key contact person, etc. within a webpage. Part of the information can be read also on the smart phone.

| # | Job ID         | Job Name         | Owner      | Queue | Status | Start Time          | Run Time       |
|---|----------------|------------------|------------|-------|--------|---------------------|----------------|
| 1 | 145211.gvadmin | granular         | ywzhang    | n_cpu | Q      | 2016-08-14 02:15:06 |                |
| 2 | 157294.gvadmin | l0_2_c1000_d0_k3 | Tiantianli | n_cpu | R      | 2016-10-27 12:43:24 | 26days3h34m21s |
| 3 | 162717.gvadmin | vortex           | ljia       | n_cpu | R      | 2016-11-03 18:59:36 | 18days21h18m9s |
| 4 | 169073.gvadmin | granular         | ywzhang    | n_cpu | Q      |                     |                |
| 5 | 169076.gvadmin | gamma            | weicc      | n_cpu | Q      |                     |                |
| 6 | 169102.gvadmin | granular         | ywzhang    | n_cpu | Q      |                     |                |
| 7 | 170041.gvadmin | ground           | ljia       | n_cpu | R      | 2016-12-02 20:15:36 | 1days20h35m25s |
| 8 | 170301.gvadmin | ground           | JQPam      | n_cpu | R      | 2016-12-03 11:17:21 | 1days5h33m40s  |

Figure 7. Tracing jobs at webpage of EasyOP

Jobs on the HPC system can be traced by EasyOP in terms of live job, history job, abnormal job, and the specification & working status of each node as shown in Fig. 7. When the job quits out with error, alarms are automatically sent out to users through email, SMS, or smartphone according to the users' preference setting.

With EasyOP, administrators are able to evaluate user applications in addition to managing the users account database. Super administrators can assign the permission on certain HPC system(s) to certain ordinary administrator(s), if needed.

Reports can cause a lot of problems for many HPC centers. Here, EasyOP offers two types of reports to free administrators from the tedious work. First, customized report can be created for



Figure 8. An example of a customized report

monitoring indicators and jobs (29 in total), such as system performance trend, system status, performance Top10, performance trends, job status, job number, and so on. The performance is described in terms of CPU loads, memory loads, disk capacity, alarm, energy consumption, etc. All these reports for job can be classified for each queue or for all queues. An example of a customized report is shown in Fig. 8.

It is found that more HPC centers want to have monthly or yearly reports. A rich analysis of job status is always welcomed. Therefore, besides all the information included in the customized report, EasyOP also delivers reports for the number of jobs, running time, the variation within months, proportion, top jobs, top submitters, queue distribution, abnormal job analysis, and the comparison to the year before. Part of the snapshots of the yearly report is shown in Fig. 9. This monthly or yearly reports are the most popular functions in EasyOP ranked by users.

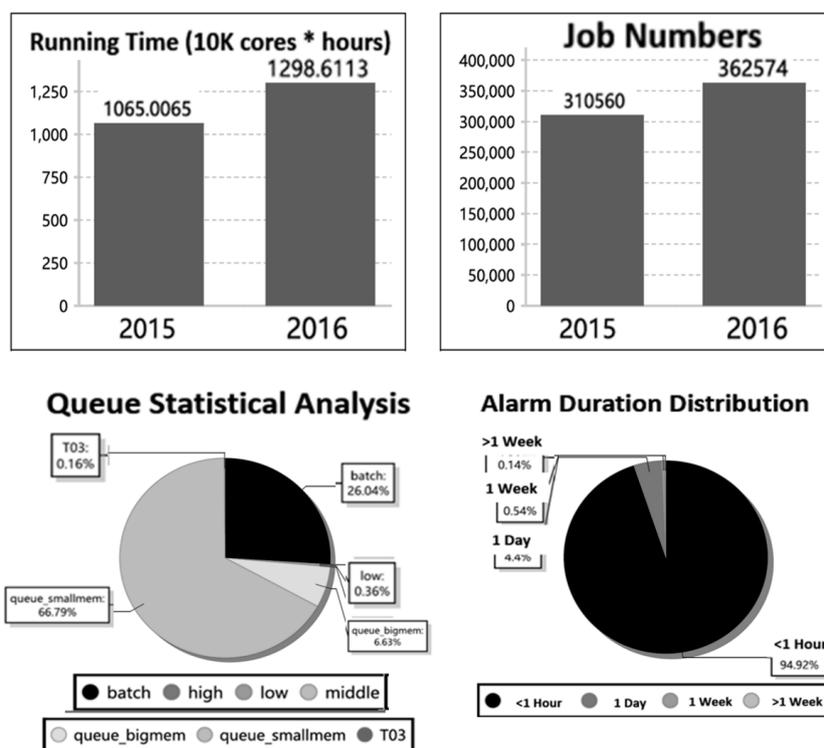


Figure 9. Part snapshots of the yearly report generated by EasyOP

## 4. In Practice

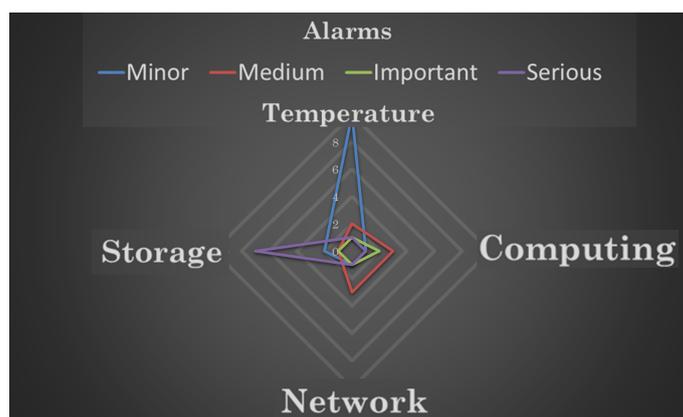
By the end of 2016, EasyOP has successfully served more than 50 HPC systems (seen in Fig. 10) with almost 10000 nodes and more than 300 regular users and many more random users. These systems are located in the most HPC active 18 cities in China. Within those systems, 2 % of the systems have more than 1000 nodes and 86 % systems have less than 100 nodes.



**Figure 10.** 18 cities have HPC systems serving by EasyOP

By the end of 2016, EasyOP has already generated 5000 alarms (as shown in Fig. 11), about 4000 of the alarms were minor, i.e. quickly self-repaired, such as “temperature is higher than the set threshold”. The number of serious alarms was 500, which mainly lay on three aspects, unstable IPMI network, CPU overloaded, and storage module broke.

Most of the 50+ systems, EasyOP is serving, are quite busy. The average CPU utilization ratio is 75.69 % with the peak at 85 %. The average memory utilization ratio is 32.31 % with the peak at 45 %. As for storage use these are 24.7 % and 38 %, respectively as shown in Fig. 12.



**Figure 11.** Alarms distribution from Oct. 2015 to Nov. 2016

The job running status shows an interesting drop at Feb. 2016, which is just during the Spring Festival of China as shown in Fig. 13. This indicates that during the holiday season, less scientists ran their calculations. Another drop is around Aug. 2016. This is the summer holiday for most universities and institutions. The waiting jobs always exit and take 15 % of the total jobs number. As for the jobs running time periods, we found all periods are quite even to each

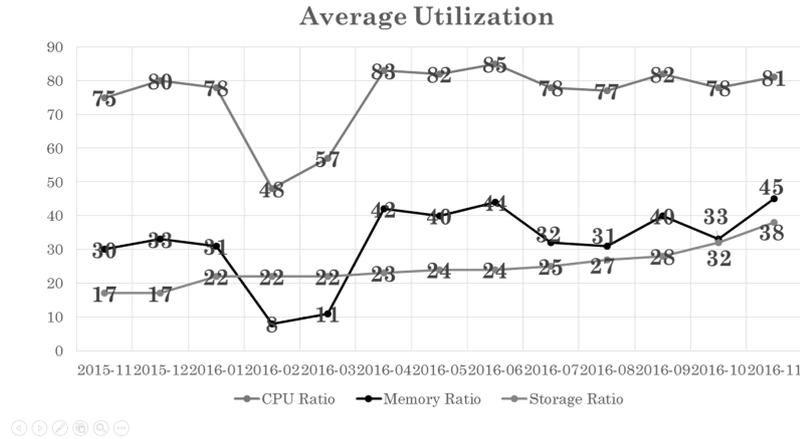


Figure 12. The average utilization of HPC systems serving by EasyOP

other, if we classify the period as > 500 hours, > 100 hours, 50–100 hours, 10–50 hours, and < 10 hours.

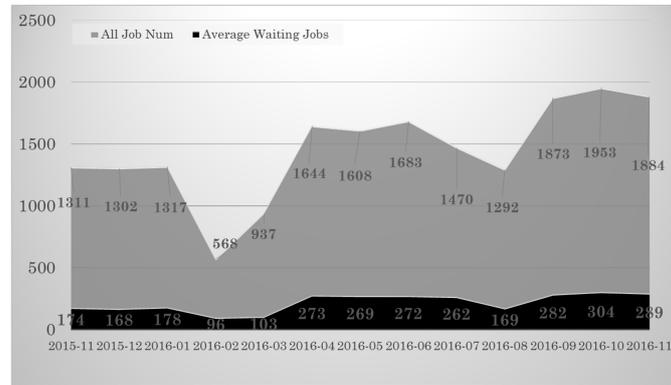


Figure 13. Job status on HPC systems serving by EasyOP

## Conclusions and Future Work

EasyOP, i.e. Easy Operation, is an on-line cloud service platform developed to facilitate HPC centers’ operation and administration. From 2015, we identified the technological method, grouped related engineers, developed the product, and provided service through Wuxi Cloud Supercomputing Center. In addition to the R&D group, we also have a service group, who can keep track of alarms and offer support to users in emergency, fix the HPC system problems per users’ requests, and also compose the trend reports on average status for all HPC centers in service. Currently, we have been serving 10000 nodes, distributed in 18 Chinese cities.

This is worth a try to grasp cloud ideas and their advantages for HPC centers’ operation and administration after system deployment. EasyOP could also be taken as a good model for Software as a Service (SaaS) to expand the internet+ applications. Probably, EasyOP offers a different way to combine HPC with cloud compare to commonly noticed HPCaaS. However, please keep in mind that EasyOP has no intention to replace operation engineers or administration officials. It is just a convenient tool for them.

Next step, we will make efforts on monitoring HPC applications, trouble shooting tracing, multi-level service centers, and on-line sharing of HPC related resources, including but not

limited to computing resource, document resource etc. As a blueprint, we hope to emerge and support the HPC eco-system with both HPC facility enthusiasts and application scientists. The really big challenge that lays ahead in the development of EasyOP as a monitoring and support system is the long reaching plan of Sugon to start service for the European market. With its strategic partner Arctur, Sugon will now start to (re)develop and localize the EasyOP application to be better applicable to the European market. The specific challenges that are now becoming apparent are the complete translation of the application in to at least English language, but maybe also in to some other major European languages, for instance German and French. After that the subsequent step will be to have an automated deployment system of EasyOP at the customer solution since we are aware that many different European HPC centres cannot be run and operated from a central location, but rather require a on-site solution. The first successful deployment of EasyOP has already been established in Slovenia at the Arctur's location. It is now the challenge that faced by Sugon and Arctur to strengthen the foothold in Europe and develop EasyOP in a reliable and stable solution to support the systems that will serve European HPC customers.

## Acknowledgements

This research was supported by the National Key R&D Program of China (Grant No. 2017YFB1001600). Di Mai, Ruixian Liu, Yi Zhang, Wenlong Xie, Lu Yang, Yanru Bi, Haifeng Miao, Yingying Wang, Yongrui Liang, Dandan Zhang, Rong Wang, Quanbiao Hu have contributed in EasyOP's service and R&D in different aspects.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Badger, M.: Zenoss Core 3.x Network and System Monitoring. Packt Publishing (2011), DOI: 10.1109/infoteh.2018.8345528
2. Barth, W.: Nagios: System and Network Monitoring. No Starch Press (2008), DOI: 10.1016/b978-1-59749-267-6.x0001-0
3. Buyya, R.: Parmon: a portable and scalable monitoring system for clusters. *Software Practice & Experience* 30(7), 723–739 (2015), DOI: 10.1002/(sici)1097-024x(200006)
4. Crasso, M., Mateos, C., Zunino, A., Campo, M.: Easysoc: Making web service outsourcing easier. *Information Sciences An International Journal* 259(3), 452–473 (2014), DOI: 10.1016/j.ins.2010.01.013
5. Cunha, R.L.F., Rodrigues, E.R., Tizzei, L.P., Netto, M.A.S.: Job placement advisor based on turnaround predictions for HPC hybrid clouds. *Future Generation Computer Systems* 67, 35–46 (2016), DOI: 10.1016/j.future.2016.08.010
6. Duran-Limon, H.A., Flores-Contreras, J., Parlavantzas, N., Zhao, M., Meulenert-Peña, A.: Efficient execution of the WRF model and other HPC applications in the cloud. *Earth Science Informatics* 9(3), 1–18 (2016), DOI: 10.1007/s12145-016-0253-7

7. Evans, R.T., Browne, J.C., Barth, W.L.: Understanding application and system performance through system-wide monitoring pp. 1702–1710 (2016), DOI: 10.1109/ipdpsw.2016.145
8. Ferreto, T.C., De Rose, C.A.F., De Rose, L.: Rvision: An open and high configurable tool for cluster monitoring. In: IEEE/ACM International Symposium on CLUSTER Computing and the Grid. pp. 75–75 (2002), DOI: 10.1109/ccgrid.2002.1017114
9. Fu, W., Huang, Q.: Grideye: A service-oriented grid monitoring system with improved forecasting algorithm. In: International Conference on Grid and Cooperative Computing Workshops. pp. 5–12 (2006), DOI: 10.1109/gccw.2006.51
10. Gupta, A., Kale, L.V., Gioachin, F., March, V., Suen, C.H., Lee, B.S., Faraboschi, P., Kaufmann, R., Milojevic, D.: The who, what, why, and how of high performance computing in the cloud. In: IEEE International Conference on Cloud Computing Technology and Science. pp. 306–314 (2014), DOI: 10.1109/CloudCom.2013.47
11. Hashem, I.A.T., Yaqoob, I., Anuar, N.B., Mokhtar, S., Gani, A., Khan, S.U.: The rise of “big data” on cloud computing: Review and open research issues. *Information Systems* 47(C), 98–115 (2015), DOI: 10.1016/j.is.2014.07.006
12. Hassan, H.A., Mohamed, S.A., Sheta, W.M.: Scalability and communication performance of HPC on Azure cloud. *Egyptian Informatics Journal* 17(2), 175–182 (2016), DOI: 10.1016/j.eij.2015.11.001
13. Hernandez, H.M., Winter, R.L.: Sender device based pause system (2015), <https://patents.google.com/patent/US9055467B2/en>, accessed: 2019-01-24
14. Huntington-Lee, J., Terplan, K., Gibson, J.: *HP Openview: A Manager’s Guide*. McGraw-Hill, Inc. (1997), <https://dl.acm.org/citation.cfm?id=548549>, accessed: 2019-01-24
15. Kannan, J., Munday, P.: *Challenges in Using Cloud Technology for Promoting Learner Autonomy in a Spanish Language Course*. IGI Global (2017)
16. Karjoth, G.: Access control with IBM Tivoli access manager. *Acm Transactions on Information & System Security* 6(2), 232–257 (2003), DOI: 10.1145/762476.762479
17. Kundu, D., Kundu, D.: *Cacti 0.8 Network Monitoring*. Packt Publishing Ltd (2009), [https://www.packtpub.com/sites/default/files/sample\\_chapters/5968-cacti-sample-chapter-4-creating-and-using-templates.pdf](https://www.packtpub.com/sites/default/files/sample_chapters/5968-cacti-sample-chapter-4-creating-and-using-templates.pdf), accessed: 2019-01-24
18. Lin, J., Xu, W., Zhang, W., Yang, G.: Equipment management and system maintenance on HPC platform. *Experimental Technology & Management* (2013), [http://en.cnki.com.cn/Article\\_en/CJFDTOTAL-SYJL201305028.htm](http://en.cnki.com.cn/Article_en/CJFDTOTAL-SYJL201305028.htm), accessed: 2019-01-24
19. Mantripragada, K., Tizzei, L.P., Binotto, A.P.D., Netto, M.A.S.: An SLA-based advisor for placement of HPC jobs on hybrid clouds. In: International Conference on Service-Oriented Computing. pp. 324–332 (2015), DOI: 10.1016/j.eij.2015.11.001

20. Marathe, A., Harris, R., Lowenthal, D.K., Supinski, B.R.D., Rountree, B., Schulz, M., Yuan, X.: A comparative study of high-performance computing on the cloud. In: International Symposium on High-Performance Parallel and Distributed Computing. pp. 239–250 (2013), DOI: 10.1145/2462902.2462919
21. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing* 30(7), 817–840 (2004), DOI: 10.1016/j.parco.2004.04.001
22. Mell, P.M., Grance, T.: SP 800-145. The NIST Definition of Cloud Computing. National Institute of Standards & Technology (2011), DOI: 10.6028/nist.sp.800-145
23. Navaux, P.O.A., Carissimi, A., Roloff, E., Diener, M.: High performance computing in the cloud: Deployment, performance and cost efficiency. In: IEEE International Conference on Cloud Computing Technology and Science. pp. 371–378 (2012), DOI: 10.1109/Cloud-Com.2012.6427549
24. Ni, G., Jie, M., Bo, L.: Gridview: A dynamic and visual grid monitoring system. In: High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference. pp. 89–92 (2004), DOI: 10.1109/hpcasia.2004.1324020
25. Oetiker, T.: MRTG: The multi router traffic grapher. In: Conference on Systems Administration. pp. 141–148 (1998), [https://www.usenix.org/legacy/event/lisa98/full\\_papers/oetiker/oetiker.pdf](https://www.usenix.org/legacy/event/lisa98/full_papers/oetiker/oetiker.pdf), accessed: 2019-01-24
26. Palmer, J.T., Gallo, S.M., Furlani, T.R., Jones, M.D., Deleon, R.L., White, J.P., Simakov, N., Patra, A.K., Spermach, J., Yearke, T.: Open XDMoD: A tool for the comprehensive management of high-performance computing resources. *Computing in Science & Engineering* 17(4), 52–62 (2015), DOI: 10.1109/mcse.2015.68
27. Rizhiyi: IT maintenance platform won 60 M investment. *Information Technology and Informatization* 1(12), 8–8 (2015), <https://www.rizhiyi.com>, accessed: 2019-01-24
28. Sadooghi, I., Martin, J.H., Li, T., Brandstatter, K., Zhao, Y., Maheshwari, K., Ruivo, T.P.P.D.L., Timm, S., Garzoglio, G., Raicu, I.: Understanding the performance and potential of cloud computing for scientific applications. *IEEE Transactions on Cloud Computing* PP(99), 1–1 (2015), DOI: 10.1109/TCC.2015.2404821
29. Sheng, Q.Z., Qiao, X., Vasilakos, A.V., Szabo, C., Bourne, S., Xu, X.: Web services composition: A decade’s overview. *Information Sciences* 280, 218–238 (2014), DOI: 10.1016/j.ins.2014.04.054
30. Wibisono, A., Suhartanto, H.: Cloud computing model and implementation of molecular dynamics simulation using Amber and Gromacs. In: International Conference on Advanced Computer Science and Information Systems. pp. 31–36 (2012), <https://ieeexplore.ieee.org/abstract/document/6468763>, accessed: 2019-01-24
31. Yang, C., Huang, Q., Li, Z., Liu, K., Hu, F.: Big Data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth* 10(1), 13–53 (2017), DOI: 10.1080/17538947.2016.1239771

# Performance Evaluation of Different Implementation Schemes of an Iterative Flow Solver on Modern Vector Machines

*Kenta Yamaguchi*<sup>1,2</sup>, *Takashi Soga*<sup>2,4</sup>, *Yoichi Shimomura*<sup>2,4</sup>,  
*Thorsten Reimann*<sup>3</sup>, *Kazuhiko Komatsu*<sup>2</sup>, *Ryusuke Egawa*<sup>2</sup>,  
*Akihiro Musa*<sup>2,4</sup>, *Hiroyuki Takizawa*<sup>2</sup>, *Hiroaki Kobayashi*<sup>5</sup>

© The Authors 2019. This paper is published with open access at SuperFri.org

Modern supercomputers consist of multi-core processors, and these processors have recently employed vector instructions, or so-called SIMD instructions, to improve performances. Numerical simulations need to be vectorized in order to achieve higher performance on these processors. Various legacy numerical simulation codes that have been utilized for a long time often contain two versions of source codes: a non-vectorized version and a vectorized version that is optimized for old vector supercomputers. It is important to clarify which version is better for modern supercomputers in order to achieve higher performance. In this paper, we evaluate the performances of a legacy fluid dynamics simulation code called FASTEST on modern supercomputers in order to provide a guidepost for migrating such codes to modern supercomputers. The solver has a non-vectorized version and a vectorized version, and the latter uses the hyperplane ordering method for vectorization. For the evaluation, we also implement the red-black ordering method, which is another way to vectorize the solver. Then, we examine the performance on NEC SX-ACE, SX-Aurora TSUBASA, Intel Xeon Gold, and Xeon Phi. The results show that the shortest execution times are with the red-black ordering method on SX-ACE and SX-Aurora TSUBASA, and with the non-vectorized version on Xeon Gold and Xeon Phi. Therefore, achieving a higher performance on multiple modern supercomputers potentially requires maintenance of multiple code versions. We also show that the red-black ordering method is more promising to achieve high performance on modern supercomputers.

*Keywords:* performance evaluation, legacy code, numerical fluid dynamics simulation, vectorization, hyperplane method, red-black method.

## Introduction

Numerical simulations of fluid dynamics can solve many of the important problems the scientific and engineering field faces today. Simulations for analyzing and understanding more complex problems require higher performance supercomputers. Modern supercomputers consist of multi-core processors such as Intel Xeon and Xeon Phi processors to increase the degree of parallelism. Moreover, modern supercomputers have employed vector instructions to exploit the loop-level parallelism. Thus, in order to achieve a higher performance on these processors, it is necessary to vectorize the simulation codes. In this paper, the supercomputers composed of processors employing vector instructions are referred to as *modern vector machines*.

Many numerical simulations of fluid dynamics utilize implicit methods for stably solving large-scale models. However, the implicit methods generally have loop-carried dependencies on stencil calculations, and these simulations cannot be vectorized in a straightforward way. The hyperplane ordering method [1] was devised to vectorize them, and until recently this method was widely used for vectorizing the implicit methods on vector supercomputers. Accordingly,

---

<sup>1</sup>NEC Solution Innovators, Tokyo, Japan

<sup>2</sup>Cyberscience Center, Tohoku University, Sendai, Japan

<sup>3</sup>Technische Universität Darmstadt, Darmstadt, Germany

<sup>4</sup>NEC Corporation, Tokyo, Japan

<sup>5</sup>Graduate School of Information Science, Tohoku University, Sendai, Japan

some legacy numerical simulation codes (called old vectorized codes) often have two versions of source codes: non-vectorized version (called naive version) and vectorized version.

As the hyperplane ordering method generally increases memory loads, Soga et al. [2] utilized the red-black ordering method to vectorize the successive over-relaxation method, and then demonstrated that both of the vector supercomputer SX-9 system and Intel Xeon processor (Nehalem-EX) can achieve higher performance by using the red-black ordering method in comparison with the hyperplane ordering method. Therefore, old vectorized codes may require another vector optimization (i.e., red-black ordering) to achieve a higher performance on modern vector machines.

In this paper, we evaluate the performances of the naive version, hyperplane, and red-black ordering methods on modern vector machines, NEC SX-ACE, SX-Aurora TSUBASA (hereafter: SX-Aurora), Intel Xeon Gold, and Xeon Phi (Knights Landing), so as to clarify which method achieves higher performance. Here, the finite-volume solver FASTEST [3] serves as basis for our evaluation.

In Section 1, we present related work regarding optimization and performance evaluations of FASTEST. Section 2 explains the system architecture of four modern vector machines. Section 3 provides an overview of the FASTEST code. Section 4 presents the results of performance evaluations. We conclude in Summary section with a brief summary and mention of future work.

## 1. Related Work

FASTEST is a legacy fluid dynamics code and was originally developed in the 1990s. Scheit and Becker [4] optimized the FASTEST-3D code for multi-core processors and evaluated its performance on an Intel Xeon eight-cores processor (*Sandy Bridge*) and six-cores processor (*Westmere*). They found that the code was mostly memory-bound and that the solver of Stone's strongly implicit (SIP) method [5] was the main performance bottleneck. In order to decrease the memory load, they used single-precision floating-point data, and avoided unnecessary re-computation of the incomplete LU factorization to solve the SIP method. This improved the performance by about 40 %. They also utilized non-blocking MPI functions and showed that the scalability improved, i.e. the number of nodes of the optimized code was 2.8 times more than that of the non-optimized code when each parallel efficiency was 50 %.

Burger et al. [6] examined optimizations of memory access on the *sipsol* subroutine and found, similarly, that the solver of the SIP method was memory intensive and was the most time-consuming. In particular, the memory access on the vectorized version of the solver was inefficient due to memory accesses along a plane, called the *hyperplane*, which is a skew cutting plane across the three-dimensional space. The solver needs long-interval accesses. They packed data elements on a three-dimensional array into two two-dimensional arrays in sequential memory access and showed that the performance becomes double on an AMD Opteron twelve-cores processor.

These studies utilized an Intel Xeon processor and AMD Opteron along with the AVX instruction set, which enabled the simulation codes to be vectorized. However, there was no discussion from a view point of vectorization. Therefore, in this paper, we clarify the performance on the vectorized FASTEST codes on modern vector machines.

**Table 1.** Specifications of each machine used in the evaluation

|                   |                 | SX-ACE | SX-Aurora | Xeon Gold | Xeon Phi |
|-------------------|-----------------|--------|-----------|-----------|----------|
| Clock freq. (GHz) |                 | 1      | 1.4       | 2.6       | 1.3      |
| CPU               | No. of cores    | 4      | 8         | 16        | 64       |
|                   | Perf. (Gflop/s) | 256    | 2150      | 1331      | 2662     |
|                   | LLC (MB)        | -      | 16        | 22        | 32       |
|                   | Memory (GB)     | 64     | 48        | 192       | 96       |
|                   | (MCDRAM)        |        |           |           | 16       |
|                   | Mem. BW (GB/s)  | 256    | 1200      | 128       | 102      |
| (MCDRAM)          |                 |        |           | 409       |          |

## 2. Modern Vector Machines

Modern supercomputers consist of many processors and accelerators employing multi-core technologies. For example, the world’s fastest supercomputer in the top 500 list in June 2018 [7] is equipped with 2,397,824 cores and achieved the theoretical peak-performance of 200.8 Pflop/s. Recently, scalar processors have provided support for vector instruction sets such as Streaming SIMD Extensions (SSE), Advanced Vector Extensions (AVX), and AVX-512 instructions. Thanks to these vector instructions, numerical simulations have been accelerated not only on vector supercomputers but also on scalar supercomputers. In this section, we give overviews of modern vector machines, NEC SX-ACE, SX-Aurora Tsubasa, Intel Xeon Gold, and Xeon Phi. Table 1 lists the hardware configurations of each machine used in the evaluation.

### 2.1. SX-ACE

SX-ACE is a vector supercomputer launched by NEC in 2013 [8, 9]. Its processor consists of four vector cores being comprised of a vector processing unit (VPU), Assignable Data Buffer (ADB) and Miss Status Handling Register (MSHR). The VPU is a key component of SX-ACE. The vector length of SX-ACE is 256 vector elements, 8 B each, and VPU can execute 256 operations by a single vector instruction using 16 vector pipelines in 16 clock cycles. The VPU has two multiply units and two add units, which can be independently operated by different vector instructions. Thus the core can simultaneously execute 64 operations by four vector instructions in one clock cycle. As the clock frequency of the core is 1.0 GHz, the single core performance of SX-ACE is 64 Gflop/s. The total performance of a single processor reaches 256 Gflop/s with four cores. Each core is connected to a memory control unit (MCU) through the memory crossbar network at the memory bandwidth of 256 GB/s, and the bandwidth is shared by the four cores. Thanks to this architecture, a single core can use the entire bandwidth of 256 GB/s if the other three cores do not access the memory. Each core is equipped with 1 MB ADB, which is a software controllable data buffer with a directory that can be accessed by the VPU at the rate of 256 GB/s. Unlike caches on general scalar processors, ADB is controlled by manually inserting directives into the source program. Consequently, ADB can retain only reusable data, which will not be evicted by non-reusable data. MSHR is used to handle outstanding vector loads on cache misses by eliminating unnecessary vector load accesses. It can reduce redundant load requests between ADB and the main memory.

## 2.2. SX-Aurora TSUBASA

SX-Aurora TSUBASA was released in 2017 [10, 11]. An SX-Aurora TSUBASA system is comprised of a vector host (VH) and a vector engine (VE). A VE is implemented as a PCI Express (PCIe) card equipped with a vector processor, i.e., the card is connected to the VH via PCIe. One VH can control eight VEs. The vector processor consists of eight vector cores, a 16 MB last-level cache (LLC), and six High Bandwidth Memory 2 (HBM2) memory modules. The SX-Aurora TSUBASA's VPU has three fused multiply add (VFMA) units and each VFMA unit has 32 vector pipelines. The vector length of SX-Aurora TSUBASA is 256, which is the same as that of SX-ACE. Thus, a VPU can execute 256 operations by a single vector instruction in eight clock cycles. As the clock frequency of the VE is 1.4 GHz, a single core provides 268.8 Gflop/s (32 operations in one clock cycle  $\times$  two floating-point operations (add and multiply) by VFMA  $\times$  3 VFMA units  $\times$  1.4 GHz). Hence, a vector processor, which consists of eight cores, achieves 2.15 Tflop/s. The LLC is directly connected to the vector registers of each core, and shared by eight cores. The six HBM2 memory modules deliver the high memory bandwidth of 1.288 TB/s in total. This memory architecture enables a high sustained performance especially in executing memory-intensive applications.

## 2.3. Intel Xeon Gold

Intel Xeon Gold is marketed as a 6th-generation Intel core. In this paper, we evaluate the performance of the application using a Xeon Gold 6142 processor, which consists of 16 cores. On the previous generations of Intel Xeon processor families, cores, last-level cache (LLC), memory controller, IO controller, and inter-socket Intel QuickPath Interconnect (Intel QPI) ports are connected together using a ring architecture. In contrast, the 6-th generation Xeon Gold processor introduces a mesh architecture that encompasses an array of vertical and horizontal communication paths. This architecture allows traversal from one core to another through the shortest path. The processor interconnect is Intel Ultra Path Interconnect (Intel UPI) which replaced the Intel QPI. Modern scalar processors such as the Xeon Gold processor introduce SIMD instructions. For example, AVX-512 instructions, which are supported by Xeon Gold, provide 512-bit-wide vector instructions, 32 logical registers, eight mask registers, and indirect vector access via gathers and scatters. A single AVX-512 instruction can execute 32 single-precision or 16 double-precision floating-point operations per cycle. AVX-512 instructions are classified into five categories: foundation instructions (AVX-512F), which are the base 512-bit extensions; conflict-detection instructions (AVX-512CD); doubleword and quadword instructions (AVX-512DQ); byte and word instructions (AVX-512BW); and vector length extensions (AVX-512VL) [12].

## 2.4. Intel Xeon Phi

We use the second-generation Intel Xeon Phi 7210, so called Knights Landing (KNL), for performance evaluation. The processor consists of 32 active physical tiles, and each tile is comprised of two cores and two vector processing units (VPU) per core. A 1 MB level-2 (L2) cache which is shared by two cores is also included in each tile. Multi-channel DRAM (MCDRAM) and double data rate (DDR) memory are used in the Xeon Phi. The total memory capacities of MCDRAM and DDR are 16 GB and 96 GB, respectively. MCDRAM, which is 3D-stacked DRAM integrated on-package, provides high bandwidth of 409 GB/s. The two types of memory

```

do k=2,nkblk-1
  lkk=(k-1)*nijblk
  do i=2,niblk-1
    lki=ijksblk+lkk+(i-1)*njblk
    do jk=lki+2,lki+njblk-1
      res(jk)=ae(jk)*fi(jk+njblk)+aw(jk)*fi(jk-njblk)+an(jk)*fi(jk+1)+as(jk)*fi(jk-1) &
        +at(jk)*fi(jk+nijblk)+ab(jk)*fi(jk-nijblk)+su(jk)-ap(jk)*fi(jk)
      rhelp(m)=rhel(m)+abs(res(jk))
      res(jk)=(res(jk)-bb(jk)*res(jk-nijblk)-bw(jk)*res(jk-njblk)-bs(jk)*res(jk-1))*bp(jk)
    end do; end do; end do

```

Figure 1. Source code of the naive version

provide three memory modes, which are selectable through the BIOS setting at boot time. One mode is the cache mode in which MCDRAM is used as a cache for the DDR memory. Another mode is the flat mode, which handles both MCDRAM and DDR in the same way to organize one memory space. The other is the hybrid mode, in which either 25 % or 50 % of MCDRAM is used as cache and the rest is used as memory. In this work, the flat mode is selected for the evaluation. As with Xeon Gold, the Xeon Phi's VPU also provides support for AVX-512 instructions. Xeon Phi's AVX-512 instructions fall into four categories. AVX-512F and AVX-512CD are the same as those in Xeon Gold. Two additional categories – exponential and reciprocal instructions (AVX-512ER) and prefetch instructions (AVX-PF) – are only provided by Xeon Phi [13].

### 3. Incompressible Flow Solver, FASTEST

FASTEST is a three-dimensional incompressible flow solver [6]. Several academic developers have independently enhanced it to simulate various flow phenomena and multi-physics couplings. In this paper, we use the code developed at Technische Universität Darmstadt, whose codes are parallelized by using MPI and OpenMP.

The solver is based on the Semi-Implicit method for the Pressure Linked Equations (SIMPLE) method [14], which iteratively solves the momentum and pressure-correction equations. These equations are discretized using a second-order finite-volume scheme on a block-structured grid. The resulting linear equation system is solved using the SIP method, which is based on an incomplete  $LU$  factorization. The subroutine containing the SIP method (called *sipsol* subroutine) is the main performance bottleneck and holds two different implementations [6]. The first, called the naive version, is shown in Fig. 1. This code calculates the residual calculations after the  $LU$  decomposition in the *sipsol* subroutine. Each data element in the figure is saved in a linearized one-dimensional array. All elements are calculated along the three coordinate axes via a nested loop using variables  $lkk$  and  $lki$ . The variable  $lkk$  indicates the start points of  $k$ . Similarly, the variable  $lki$  also indicates the start points of  $i$ . Here, a loop-carried dependency exists between the left-hand array  $res(ijk)$  and the right-hand arrays  $res(ijk-nijblk)$ ,  $res(ijk-njblk)$ , and  $res(ijk-1)$  in the second line from the bottom in Fig. 1. Hence, this *do* loop cannot be vectorized.

The second one, called the HP version, utilizes the hyperplane ordering method. The first *do* loop in Fig. 2 converts a triple-nested *do* loop in Fig. 1 into a single *do* loop. The second nested *do* loop is a loop of the hyperplane. Array  $ijkdia$  constructs planes fulfilling the condition  $i + j +$

```

do ijk=ijksblk+njblk+njblk+2,ijksblk+(nkblk-2)*njblk+njblk*(niblk-2)+njblk-1
  res(ijk)=ae(ijk)*fi(ijk+njblk)+aw(ijk)*fi(ijk-njblk)+an(ijk)*fi(ijk+1)+as(ijk)*fi(ijk-1) &
    +at(ijk)*fi(ijk+njblk)+ab(ijk)*fi(ijk-njblk)+su(ijk)-ap(ijk)*fi(ijk)
  rhelp(m)=rhelp(m)+abs(res(ijk))
end do
do ndia=1,numdia(ngr,m)
  do npoi=npsta(ndia,ngr,m)+1,npsta(ndia+1,ngr,m)
    ijk=ijkdia(npoi)
    res(ijk)=(res(ijk)-bb(npoi)*res(ijk-njblk)-bw(npoi)*res(ijk-njblk)-bs(npoi)*res(ijk-1))*bp(npoi)
  end do; end do

```

Figure 2. Source code of the HP version

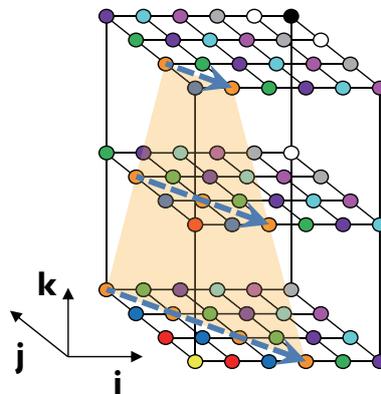


Figure 3. Diagram of a hyperplane

$k = \text{constant}$ . The plane is a skew cutting plane across the three-dimensional space. The orange plane in Fig. 3 is one of these hyperplanes that consist of the same color across the grid in three-dimensional space. Then, the inner *do* loop calculates the data elements on the planes. Since this calculation does not have any loop-carried dependency, the code of the *sipsol* subroutine is then vectorized. However, the hyperplane ordering method generally increases the number of memory loads. This is because the memory accesses of array  $res(ijk)$  are stride accesses due to access along the planes, and they have a long access latency. Moreover, the length of loop *npoi* is a variable and the efficiency of vectorized calculation becomes low when *npoi* is small.

We implement the red-black ordering method (called the RB version), the memory loads of which are lower than those of the hyperplane ordering method, as shown in Fig. 4. However, the number of iterations required for convergence is generally increased on the red-black ordering method. Thus there is a performance trade-off with increasing the number of iterations. The *do* loop in the second line from the top in Fig. 4 is a switch of *Red* and *Black* executions using the *if* statement in the fourth line from the top in Fig. 4. Array *itbl* is a mask table to skip elements with the color that are not calculated at each iteration, and then the loop is vectorized. This implementation also increases the number of operations compared to that of the naive version because the number of iterations generally increases and inserting loop *n2* doubles the total iteration count within the loop nest. Although the *if* statement is used to skip the computation in the loop body, SX-ACE executes the computation of both *true* and *false* and then the results are discarded by the so-called masking capability [2]. The benefit of this implementation is to

```

do m2=ijkgit(ngr,m)+1,ijkgit(ngr,m)+njbk*nbk*nbk,ibk
do n2=1,2
do ijk=m2,min(m2+ibk-1,ijkgit(ngr,m)+njbk*nbk*nbk)
if(itbl(ijk-ijkgit(ngr,m),1,1,n2,m).ne.1) cycle
res(ijk)=ae(ijk)*fi(ijk+njbk)+aw(ijk)*fi(ijk-njbk)+an(ijk)*fi(ijk+1)+as(ijk)*fi(ijk-1) &
+at(ijk)*fi(ijk+njbk)+ab(ijk)*fi(ijk-njbk)+su(ijk)-ap(ijk)*fi(ijk)
rhelp(m)=rhelp(m)+abs(res(ijk))
res(ijk)=(res(ijk)-bb(ijk)*res(ijk-njbk)-bw(ijk)*res(ijk-njbk)-bs(ijk)*res(ijk+rb_jm1))*bp(ijk)
end do; end do; end do

```

**Figure 4.** Source code of the RB version

increase the length of the innermost loop. This code also uses cache blocking to improve the performance of memory accesses. This is possible because the data loaded at iteration  $n2=1$  (*Red*) can be reused at iteration  $n2=2$  (*Black*). Here, the variable *ibk* is the size of a cache block.

## 4. Performance Evaluation

### 4.1. Experimental Setup

The performance of FASTEST is evaluated by using four modern vector machines: NEC SX-ACE, SX-Aurora TSUBASA, Intel Xeon Gold, and Xeon Phi. The evaluated codes are the naive version, the HP version, and the RB version, which are parallelized with OpenMP. Here, we evaluate performances of a single processor on each machine because the performance per processor is a key factor to achieve high performance on a large-scale parallel simulation. Then, since modern HPC applications are usually written with considering the cache memory, we use the model size of  $(32 \times 32 \times 53)$  per processor so that the data potentially fit in the cache memory of each processor. Here, the maximum degree of parallelism using OpenMP is 16 determined from the model size. Thus, the code is executed on four cores of SX-ACE, eight cores of SX-Aurora, 16 cores of Xeon Gold, and 16 cores of Xeon Phi. The memory mode of Xeon Phi is set to the flat mode.

Table 2 shows the versions and options of each machine's Fortran compiler. The options are high-level optimizations and inlining functions and subroutines. The codes are parallelized by using OpenMP. Table 3 lists the compiler directives for each code on each machine. The *sipsol* subroutine is vectorized by using the compiler directives: *nodep*, *ivdep*, and *simd*. SX-ACE and SX-Aurora use the optimization directives: *vovertake*, *gthreorder*, and *gather\_reorder*, which optimize the operation sequences of memory accesses. Here, the codes are executed with double precision floating-point operations.

### 4.2. Experimental Results and Discussion

FASTEST iteratively computes the momentum and pressure-correction equations from initial values of pressure and velocity at each point to their converged values. The number of iterations varies with optimized and vectorized methods. We measure the execution time of the *sipsol* subroutine per iteration, and Figure 5 shows the performance ratios of HP and RB ver-

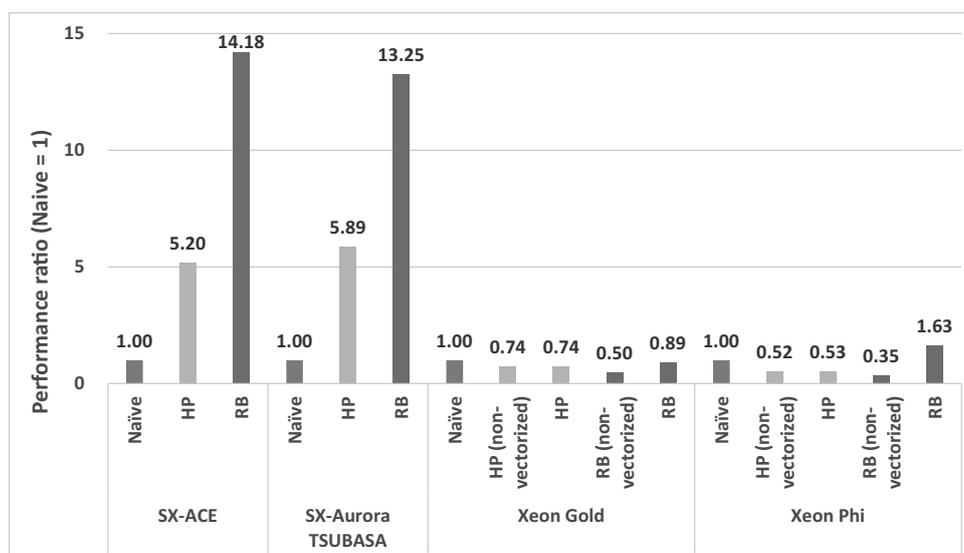
**Table 2.** Versions and options of each compiler

| Machine   | Versions                 | Options  |
|-----------|--------------------------|--|
| SX-ACE    | FORTRAN90/SX Rev.534     | -Popenmp -Chopt -pi  |
| SX-Aurora | NEC Fortran 1.5.1        | -fopenmp -O3 -msched-block<br>-finline-functions                 |
| Xeon Gold | Intel Fortran 18.0.2.199 | -qopenmp -O3 -xCORE-AVX512<br>-ipo -no-prec-div -fp-model fast=2 |
| Xeon Phi  | Intel Fortran 18.0.2.199 | -qopenmp -O3 -xMIC-AVX512<br>-ipo -no-prec-div -fp-model fast=2  |

**Table 3.** Compiler directives of each code

| Machine   | Naive | HP                               | RB               |
|-----------|-------|----------------------------------|------------------|
| SX-ACE    | -     | nodep, vovertake, gthreorder     | nodep, vovertake |
| SX-Aurora | -     | ivdep, vovertake, gather_reorder | ivdep, vovertake |
| Xeon Gold | -     | ivdep, simd                      | ivdep, simd      |
| Xeon Phi  | -     | ivdep, simd                      | ivdep, simd      |

sions to the naive version (the performance ratio = the execution time of naive version  $\div$  the execution time of HP (RB) version). Here, the execution times of the naive version are 0.156 seconds on SX-ACE, 0.053 seconds on SX-Aurora TSUBASA, 0.017 seconds on Xeon Gold, and 0.039 seconds on Xeon Phi. HP(non-vectorized) and RB(non-vectorized) indicate that the codes are not vectorized, and hence the performance differences between HP/RB and HP(non-vectorized)/RB(non-vectorized) indicate the performance gain by vectorization in Xeon Gold and Xeon Phi.

**Figure 5.** Performance ratio of HP and RB versions to naive version of the *sipsol* subroutine with an iteration on each machine

SX-ACE and SX-Aurora have the same performance characteristics. The vectorized codes, HP version and RB version, can achieve a higher performance than the naive version. In the case of SX-Aurora, the HP and RB versions are 5.89 and 13.25 times faster than the naive version, respectively. In other words, SX-ACE and SX-Aurora cannot achieve high performance unless codes are vectorized. Meanwhile, although both the HP and RB versions are vectorized, the performances of the HP version on SX-ACE and SX-Aurora are lower than that of the RB version, respectively. This is because the HP version needs indirect memory access and its memory access latency is longer than that of 2-stride memory access used in the RB version. Moreover, the efficiency of vectorized calculations becomes lower as the length of *npoi* is small.

Table 4 lists the ratios of stall time of the processor to total execution time on SX-ACE, SX-Aurora and Xeon Gold. Here, the execution statistics of all the versions on Xeon Phi and the naive version on SX-ACE and SX-Aurora cannot be measured by performance tools. This table shows that the stall time of the HP version is larger than that of the others, and then the HP version is inefficient on these machines.

**Table 4.** Ratio of stall time of processor on SX-ACE and Xeon Gold

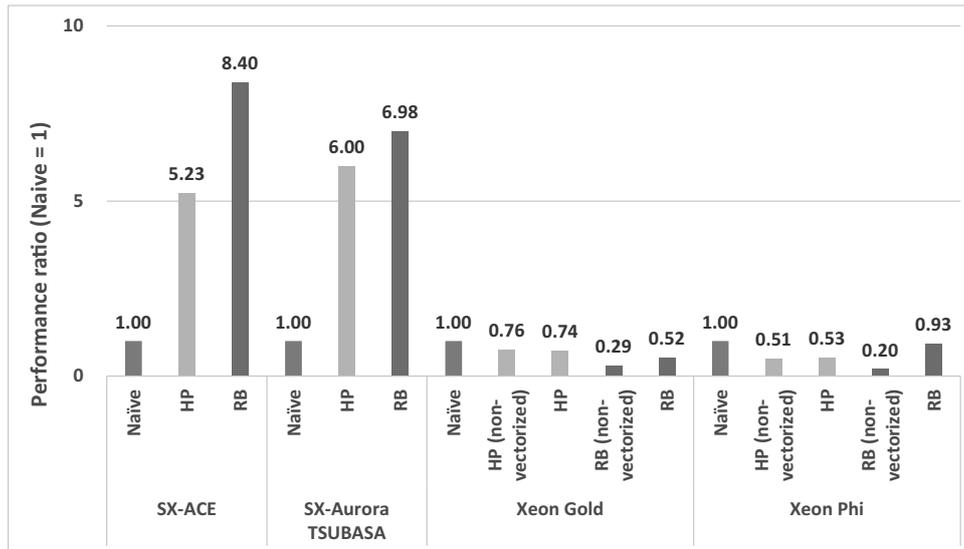
|           | Naive  | HP     | RB     |
|-----------|--------|--------|--------|
| SX-ACE    | -      | 76.6 % | 56.5 % |
| SX-Aurora | -      | 72.7 % | 67.9 % |
| Xeon Gold | 11.7 % | 63.3 % | 31.4 % |

In Xeon Gold and Xeon Phi, vectorization does not improve the performance of the HP version. The execution time is almost unchanged by enabling vectorization. In contrast, the vectorized RB version is 1.78 and 4.66 times faster than its non-vectorized version in Xeon Gold and Xeon Phi, respectively. Especially, the vectorized RB version on Xeon Phi is faster than the naive version. Meanwhile, as the clock frequency of Xeon Gold decreases, the performance gain of Xeon Gold by the vectorization is not high. However, this demonstrates that vectorization is important to achieve high performance even on Xeon Gold and Xeon Phi processors.

**Table 5.** Number of iterations on each method

|                      | Naive  | HP     | RB     |
|----------------------|--------|--------|--------|
| Number of iterations | 10,004 | 10,004 | 17,516 |

Table 5 shows the number of iterations required by each method for convergence. The RB version requires 1.8 times more iterations than the naive and HP versions. The total execution times of the *sipsol* subroutine increase. Figure 6 shows the performance ratio of HP and RB versions to naive version (the performance ratio = the execution time of naive version ÷ the execution time of HP (RB) version). Here, the execution times of the naive version are 1565.5 seconds on SX-ACE, 530.8 seconds on SX-Aurora TSUBASA, 171.4 seconds on Xeon Gold, and 386.0 seconds on Xeon Phi.



**Figure 6.** Performance ratio of HP and RB versions to naive version of the *sipsol* subroutine with the total execution times on each machine

In SX-ACE and SX-Aurora, the RB version can achieve the shortest execution time, in spite of an increase in the number of iterations. These results indicate that the red-black method is suitable to vectorize FASTET for SX-ACE and SX-Aurora. On the other hand, the naive version achieves the shortest execution time on Xeon Gold and Xeon Phi. These results show that, in case of the RB version, there is a trade-off between the performance degradation by needing more iterations and the performance improvement by using vectorization. Therefore, achieving a higher performance on multiple modern vector machines will require maintenance of multiple code versions. As a result, it will be more important to consider how to manage the code complexity in a systematic way in the future. We have been working on this problem [15, 16], and will further discuss such a methodology in our future work.

## Summary

Processors of modern supercomputers have employed vector instructions to exploit loop-level parallelism efficiently, and the vector lengths are increasing, resulting in increasing the importance of vectorization to fully exploit the system performance. Various legacy numerical simulation codes have been vectorized considering old vector machines in mind, and hence the simulations have two different code versions of their kernel codes: non-vectorized version and vectorized version. In this paper, we evaluate the performances of such a legacy code called FASTEST, which is vectorized by the hyperplane and red-black ordering methods, using four modern vector machines (SX-ACE, SX-Aurora, Xeon Gold, and Xeon Phi) in order to provide which version is suitable for the modern vector machines.

Experimental results show that the red-black ordering method can achieve the shortest execution time on SX-ACE and SX-Aurora, while the naive version achieves the shortest execution time on Xeon Gold and Xeon Phi. This demonstrates that achieving higher performance on multiple modern vector machines will require maintenance of multiple code versions, namely, the naive version and the RB version. Overall, these results indicate that the red-black ordering method has the potential to achieve high performance on the modern vector machine, and that vectorization is a key optimization method of the modern vector machine.

In this work, we showed that the maintenance of multiple code versions is required to achieve higher performance on multiple modern vector machines. We will pursue our methodology of maintenance of multiple code versions. Moreover, since the number of cores on modern machines has increased year by year, our future work will investigate performance characteristics of thread-level parallelism such as OpenMP and OpenACC in order to achieve high performance on legacy numerical simulation codes.

## Acknowledgements

This research was conducted as a joint project of Tohoku University and NEC Corporation. This research is partially supported by JST CREST “An Evolutionary Approach to Construction of a Software Development Environment for Massively-Parallel Heterogeneous Systems”, DFG SPPEXA ExaFSA project, and Grant-in-Aid for Scientific Research(B) #16H02822. We are grateful for the use of NEC SX-ACE at Cyberscience Center at Tohoku University.

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. Fujino, S., Mori, M., Takeuchi, T.: Performance of Hyperplane ordering on vector computers, *Journal of Computational and Applied Mathematics* 38, 125–136 (1991), DOI: 10.1016/0377-0427(91)90165-G
2. Soga, T., Musa, A., Okabe, K., Komatsu, K., Egawa, R., Takizawa, H., Kobayashi, H., Takahashi, S., Sasaki, D., Nakahashi, K.: Performance of SOR Methods on Modern Vector and Scalar Processors, *Journal of the Computers & Fluids* 45(1), 215–221 (2011), DOI: 10.1016/j.compfluid.2010.12.024
3. Project Site for Fastest at Technische Universität Darmstadt. [https://www.fnb.tu-darmstadt.de/forschung\\_fnb/software\\_fnb/software\\_fnb.en.jsp](https://www.fnb.tu-darmstadt.de/forschung_fnb/software_fnb/software_fnb.en.jsp), accessed: 2019-01-21
4. Scheit, C., Becker, S., Hager, G., Treibig, J., Wellein, G.: Optimization of FASTEST-3D for Modern Multicore System. <https://arxiv.org/pdf/1303.4538>, accessed: 2019-01-21
5. Stone, H.L.: Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM Journal on Numerical Analysis* 5(3), 530–558 (1968), DOI: 10.1137/0705044
6. Burger, M., Bischof, C.: Optimizing the memory access performance of FASTEST’s sipsol routine. In: 6th European Conference on Computational Fluid Dynamics, ECFD VI, July 2014, Barcelona, Spain. DOI: 10.13140/RG.2.2.32568.14089
7. Top 500 supercomputers sites. <https://www.top500.org/>, accessed: 2019-01-21
8. Momose, S.: Next generation vector supercomputer for providing higher sustained performance. In: Proceedings of IEEE Symposium on Low-Power and High-Speed Chips and Systems XVI, COOLChips 19, Yokohama, Japan, April 17–19, 2013

9. Egawa, R., Komatsu, K., Momose, S., Isobe, Y., Musa, A., Takizawa, H., Kobayashi, H.: Potential of a Modern Vector Supercomputer for Practical Applications - Performance Evaluation of SX-ACE , *The Journal of Supercomputing* 73(9), 3948–3976 (2017), DOI: 10.1007/s11227-017-1993-y
10. Yamada Y., Momose, S.: Vector Engine Processor of NEC’s Brand-New Supercomputer SX-Aurora TSUBASA. In: *Proceedings of A Symposium on High Performance Chips, Hot Chips 30*, Cupertino, California, USA, August 19–21, 2018
11. Komatsu, K., Momose, S., Isobe, Y., Sato, M., Musa, A., Kobayashi, H.: Early Evaluation of a New Vector Processor SX-Aurora TSUBASA. In: *Research Poster of ISC Higher Performance 2018, ISC 18*, Frankfurt, Germany, June 24–28, 2018
12. Mulnix, D.: Intel Xeon Processor Scalable Family Technical Overview. <https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview>, accessed: 2019-01-21
13. Sodani, A., Gramunt, R., Corbal, J., Kim, H.-S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.-C.: Knight Landing: Second-Generation Intel Xeon Phi Product, *IEEE Micro* 36(2), 34–46 (2016), DOI: 10.1109/MM.2016.25
14. Patankar, S.V.: *Numerical heat transfer and fluid flow*, Hemisphere Publishing Corporation (1980)
15. Takizawa, H., Hirasawa, S., Hayashi, Y., Egawa, R., Kobayashi, H.: Xevolver: An XML-based code translation framework for supporting HPC application migration. In: *Proceedings of IEEE International Conference on High Performance Computing, HiPC 2014*, Goa, India, December 17–20, 2014, vol. 1, pp. 1–11 (2014)
16. Suda, R., Takizawa, H., Hirasawa, S.: Xevtgen: Fortran code transformer generator for high performance scientific codes, *International Journal of Networking and Computing* 6(2), 263–289 (2016)

# Comparative Analysis of Virtualization Methods in Big Data Processing

Gleb I. Radchenko<sup>1</sup> , Ameer B. A. Alaasam<sup>1</sup>, Andrei N. Tchernykh<sup>1,2</sup> 

© The Authors 2019. This paper is published with open access at SuperFri.org

Cloud computing systems have become widely used for Big Data processing, providing access to a wide variety of computing resources and a greater distribution between multi-clouds. This trend has been strengthened by the rapid development of the Internet of Things (IoT) concept. Virtualization via virtual machines and containers is a traditional way of organization of cloud computing infrastructure. Containerization technology provides a lightweight virtual runtime environment. In addition to the advantages of traditional virtual machines in terms of size and flexibility, containers are particularly important for integration tasks for PaaS solutions, such as application packaging and service orchestration. In this paper, we overview the current state-of-the-art of virtualization and containerization approaches and technologies in the context of Big Data tasks solution. We present the results of studies which compare the efficiency of containerization and virtualization technologies to solve Big Data problems. We also analyze containerized and virtualized services collaboration solutions to support automation of the deployment and execution of Big Data applications in the cloud infrastructure.

*Keywords: Big Data, visualization, containerization, cloud computing, Xen, KVM, Docker, orchestration.*

## Introduction

Cloud computing systems have become widely used for implementation of Big Data processing tasks. Clouds rely on virtualization technology to achieve the elasticity of shared computing resources.

Virtual Machines (VMs) underlie the cloud computing infrastructure layer, providing virtual operating systems. Virtualization is a combination of hardware and software solutions that support the creation and operation of virtual versions of devices or resources, such as servers, storage devices, networks, or operating systems. The virtualization platform allows one to divide the physically unified hardware system into a logical set of independent computing resources [55]. Virtualization of computing resources allowed to solve the problem of increasing the efficiency of scheduling in cluster computing systems, by presenting their resources in the format of independent virtual machines [6]. Virtual machines provide isolation of the file system, memory, network connections, and system information [92]. But the use of virtual machines is associated with large overheads [38, 101], which can significantly limit the performance of I/O systems and efficiency of the computational resources.

The containerization technology has significantly advanced recently. It is based on the concept of limiting the amount of resources that are provided to an application by the computational node. The container provides a runtime environment for the application at the operating system level [13], reducing the overhead compared to a virtual machine.

Virtual machines and containers are virtualization technologies, but they differ in goals and functions. Containers can be viewed as a flexible tool for packaging, delivering and orchestrating both applications and software infrastructure services. They allow you to focus on a portable way to increase compatibility [73], while still using the principles of operating system virtualization.

<sup>1</sup>South Ural State University, Chelyabinsk, Russian Federation

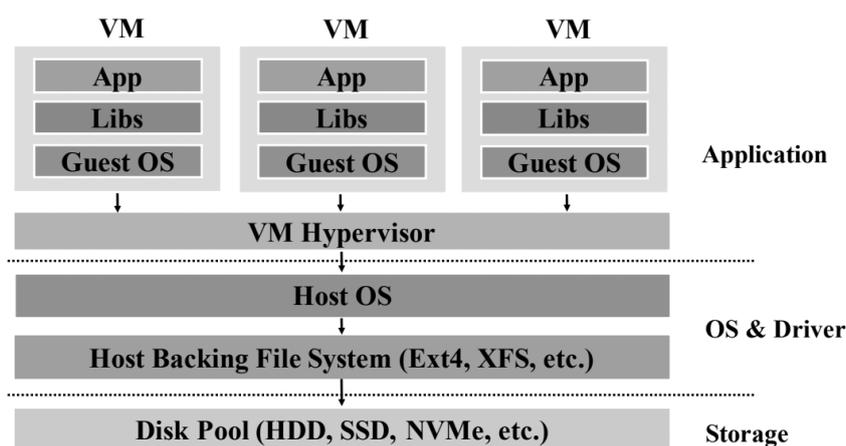
<sup>2</sup>CICESE Research Center Ensenada, Mexico

On the other hand, virtual machines are associated with the distribution and management of computational infrastructure.

In this paper we would provide an overview of the current state of virtualization and containerization approaches and technologies in the context of Big Data tasks solution. The rest of the paper is organized as follows. In Section 1 we would analyze the basics of virtualization technologies, together with the brief overview of most popular open-source virtualization solutions: Xen and KVM hypervisors. Section 2 is devoted to the overview of containerization technologies. We would analyze the key features of containerization approach and take a look at the architecture of the most popular containerization solutions and frameworks, such as Docker. In the Section 3 we would overview the main results on the comparison of containerization and virtualization solutions. Section 4 would be devoted to the overview of container orchestration technologies. In the last Section, we would provide conclusions on the performed analysis.

## 1. Virtualization Technologies

Virtualization was developed for abstracting the hardware and system resources to provide simultaneous execution of several operating systems on a single hardware platform [6].



**Figure 1.** The architecture of the virtual machine hypervisor (based on [9])

Virtual Machine Hypervisor technology, also called Virtual Machine Monitor, has a long history since the 1960s and was widely used before the era of cloud computing. As shown in Fig. 1, a virtual machine hypervisor (for example, Xen, KVM, VMware, etc.) is software that provides a virtual platform, so several guest operating systems can run on one system server. The hypervisor runs as a middleware between the virtual machine and the OS. Each virtual machine has its own guest OS.

There are several approaches to implementing a virtual machine hypervisor. So, *full virtualization* [98] is aimed at hardware emulation. In this case, a non-modified OS is used inside a virtual machine, and the hypervisor controls the execution of privileged operations of the guest OS. *Paravirtualization* requires modifications of the virtualized OS and coordination of operations between the virtual OS and the hypervisor [98]. Usage of paravirtualization improves the performance with respect to full virtualization by performing most of the operations in the host OS.

Virtual instances use isolated, large files on their host as guest images that store the file system and run the virtual machine as one large process on the host. This approach leads to some performance degradation. A complete image of a guest OS, together with the binary files and libraries needed for applications, are required for each virtual machine. This leads to significant overhead on the required disk space, and also leads to additional RAM requirements when executing virtual machines on the server. It also causes performance issues, such as slow image startup. In addition, multiple owner clouds require sharing of disk space and processor cycles. In a virtualized environment, this should be managed in such a way that the underlying platform and infrastructure can be shared in a safe but compatible way [72].

### 1.1. Xen Hypervisor

Created by researchers at Cambridge University [6], Xen is a VM hypervisor that is operating on a physical machine in the most privileged processor mode compared to any other software. The Xen hypervisor is responsible for memory management, processor resources scheduling and running a privileged domain of a physical machine (referred to as Dom0 in Xen terminology), which has direct access to hardware devices.

Dom0 domain starts at the launch of a physical machine and is usually implemented as a modified Linux, NetBSD, or Solaris OS [102]. One can manage the hypervisor and run other non-privileged guest domains (DomU) from this domain. Xen implements paravirtualization approach, so the guest OS in the DomU domain should be modified to access hardware through the Dom0 using paravirtualized drivers and an interdomain channel [69]. Figure 2 shows the operation of the Xen hypervisor and its domains.

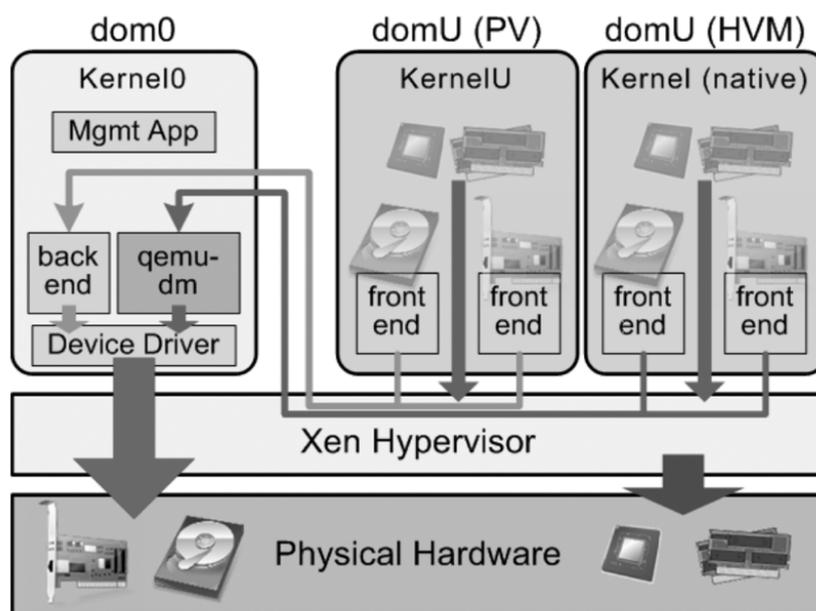


Figure 2. Xen hypervisor architecture (based on [69])

## 1.2. KVM Hypervisor

Kernel-based Virtual Machine (KVM) is a virtualization infrastructure integrated into the Linux kernel. This hypervisor was first developed by Qumranet company in 2006 [42]. Instead of creating a hypervisor from scratch, the Linux kernel was used as the KVM basis. It relies on hardware-assisted virtualization support by the host processors. KVM includes a loadable kernel module (`kvm.ko`), providing the basic virtualization infrastructure and a processor module (either `kvm-intel.ko`, or `kvm-amd.ko`).

In KVM virtualization model, virtual machines are created by opening a `/dev/kvm` device node. KVM uses a slightly modified QEMU emulator (called `qemu-kvm`) to create virtual machine instances. Each guest virtual machine is implemented as a standard Linux process, managed by a standard Linux scheduler. In addition to the two standard execution modes (kernel mode and user mode), KVM adds a third mode: guest mode, which has its own kernel and user modes that the hypervisor does not control. The modified QEMU emulator also handles I/O hardware emulation, as shown in Fig. 3.

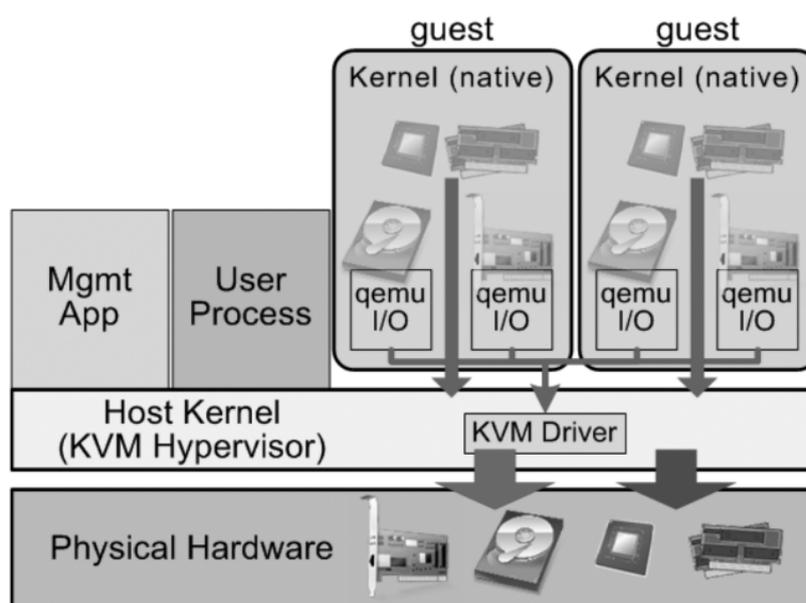


Figure 3. KVM hypervisor architecture (based on [69])

## 2. Containerization Technologies

Unlike full virtualization and paravirtualization, the OS-level virtualization approach does not require a hypervisor. It implies that the OS is changed to ensure that several OS copies are able to be executed on the same machine [98]. Linux OS-based virtualization is called container-based virtualization [101].

A container is a packaged, standalone, deployable set of application components, which may also include middleware and business logic as binary files and libraries for running applications [85] (see Fig. 4). Containers are the building blocks of OS-level virtualization that allows isolated virtual environments without the need of hypervisors. These virtual structures are in-

dependent of each other, but share the same underlying operating system (i.e., the kernel and device drivers) [17].

Docker today is one of the most well-known platforms for organizing solutions based on container technologies [2]. Such platforms turn containers into a convenient way to package and deliver applications [13].

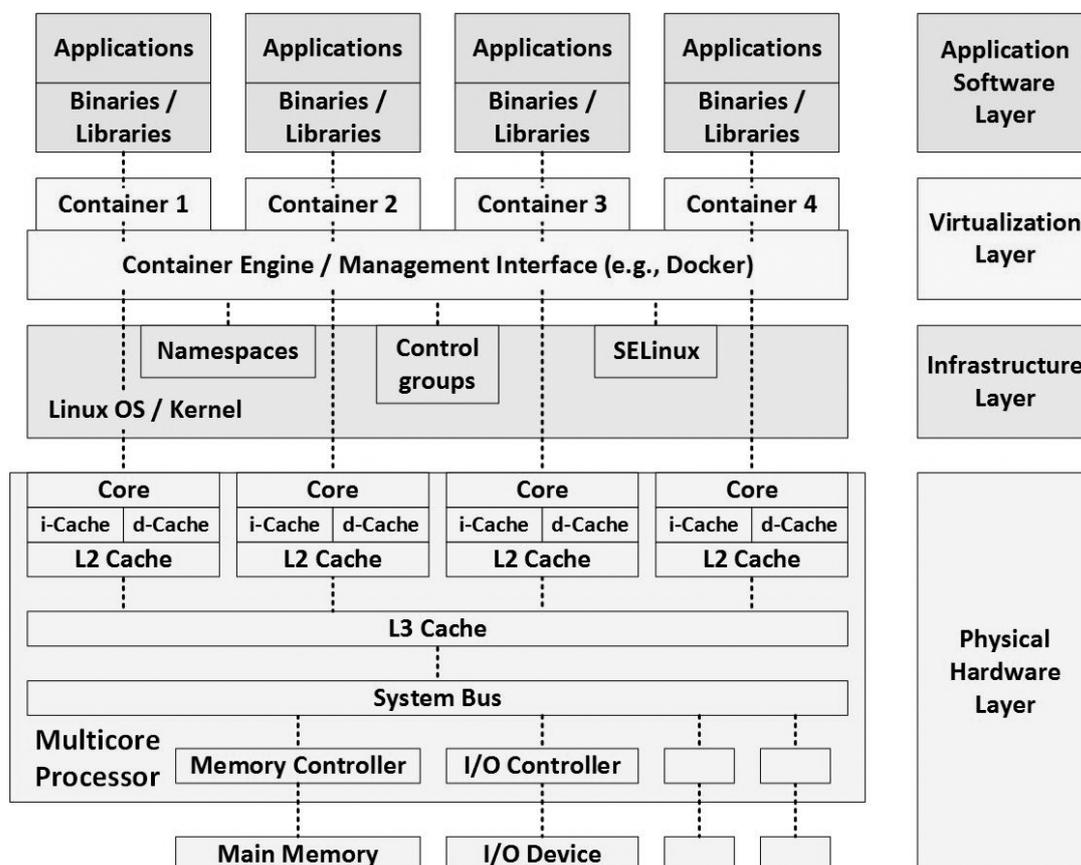


Figure 4. Container-based virtualization architecture (based on [31])

## 2.1. Namespaces and Cgroups

In modern Linux distributions, the LXC virtualization project (Linux containers) implements kernel mechanisms such as *namespaces* and *control groups* (*cgroups*) to isolate processes on a shared operating system [85].

Namespace isolation allows you to separate process groups. This ensures that they cannot see resources in other groups. Different namespaces are used to isolate processes, network interfaces, access interposes communication, mount points, or to isolate kernel identifiers and version identifiers.

On the other hand, cgroups control and restrict access to resources for groups of processes by enforcing CPU resource limits, accounting and isolation, for example, limiting the memory available to a specific container. This provides isolation between applications on the host. It also limits containers in multi-tenant host environments. Control groups allow containers to share available hardware resources and establish restrictions as needed.

At the same time, the authors of [104] explore the performance of the cgroups technology using the example of LinkedIn, where this technology is used as the basis for management of the distributed computing resources. They indicate the following key technology limitations:

- 1) memory is not reserved for cgroups (as opposed to virtual machines);
- 2) both the shared memory and the cache pages are in the common memory pool, while the former can displace the latter;
- 3) the OS can take over a page cache from any of the cgroups.

## 2.2. Key Containerization Technologies

The simplicity of tools and ease of creation and management of containerized environment made *Docker* a popular open source project. Docker containers can run only Linux processes, but one can use Linux, Windows or MacOS machines as a host. Docker containers provided greater efficiency in software development, but orchestration tools such as Docker Swarm [55] or Kubernetes [90] are required for enterprise use.

*Java containers* such as Jetty [88], Tomcat [4], Wildfly [76], and Spring Boot [71] are examples of container technologies that provide usage of standalone Java applications. The result of such systems are containerized Java applications that can run without the need for an external Java environment.

*LXD* is a container platform from Canonical [16]. LXD containers are created and operated using the same tools as traditional virtual machines, but they can provide high performance at runtime that matches the performance of container solutions. Unlike the Docker, LXD container management does not require additional orchestration systems, such as Swarm or Kubernetes. LXD is much closer to the full operating environment of the virtual machine hypervisor, including the virtualization of network interfaces and data storage interfaces. LXD containers in this case are much closer to full-featured virtual machines. For example, it is possible to run multiple Docker containers within LXD [84].

*OpenVZ (Open Virtuozzo)* [64] is one of the oldest Linux container platforms still in use today, with roots dating back to 2005. Before OpenVZ the Linux kernel had no means to create any sort of containerization, apart from the `chroot()` functionality that allowed a process to be run using a different view of the filesystem. LXC itself is a spiritual successor of OpenVZ. While OpenVZ is still around, today LXC is the tool of choice for many who wish to run a full operating system in a container [70].

*Rkt* [75] is a container technology introduced in the CoreOS platform to address security vulnerabilities in earlier versions of Docker. In 2014, CoreOS published the App Container (appc) specification to stimulate innovation in container space, which spawned a number of open source projects. It is necessary to clarify that Docker's early vulnerabilities have already been resolved, and Docker v1.11 implements the Open Container Initiative (OCI) standard [89] supported by RedHat, Google, AWS, VMware and CoreOS, thus ensuring compatibility with rkt.

Another consequence of the implementation of the Open Container Initiative standard was the *CRI-O* project [21], launched in 2016 with the participation of such companies as Red Hat, Intel, SUSE, Hyper and IBM. CRI-O allows users to run and manage any containers that are compatible with OCI directly from the Kubernetes platform without additional code or tools. From the end user's point of view, both the Docker and CRI-O implement the Kubernetes Container Runtime Interface (CRI) and implement the same functionality (loading and launching containers). CRI was built mainly to ensure that the Kubernetes platform was not heavily

dependent on Docker. Before this standard was adopted, Kubernetes was developed based on assumptions specific to the docker architecture, including such variables as paths to volumes, the internal architecture of the container, containers images storage specification, etc.

*Windows Containers* [54] technology was introduced along with the launch of Windows Server 2016. Microsoft has made significant improvements to the architecture of the Windows operating system to ensure the operation of container technologies, working closely with Docker to ensure the seamless operation of Docker container management tools in the Microsoft infrastructure. Currently, a number of works are underway to optimize the size of container images. Their work is provided in Windows 10, Server 2019 and Microsoft Azure cloud platform.

While discussing virtualization and containerization technologies we should mention the Unikernels approach. *Unikernels* are single-purpose appliances that are compile-time specialised into standalone kernels, sealed against modification when deployed to a cloud platform and act as separate software components [48]. The final application consists of a set of executable unikernels working together as a distributed system [49]. Unikernels provide optimization of the resources required by the container. One can identify the dependencies of the runtime of the application and package them into a single image, providing only the functionality of the OS that is necessary at the application runtime. Unlike Docker containers, unikernels can load and run completely independently, without a host operating system or external libraries, while Docker relies on external resources and the host environment to run. Unikernels can reduce complexity, improve portability, and reduce the attack surface of applications, but they require new development and deployment tools that are still not well developed.

### 2.3. Docker

Docker is a container solution based on LXC approach [22]. Docker images are made up of file systems arranged in layers, one above the other, like a Linux virtualization stack using LXC mechanisms. A container daemon, called *dockerd*, starts containers as application processes. It plays a key role as the root of the user process tree.

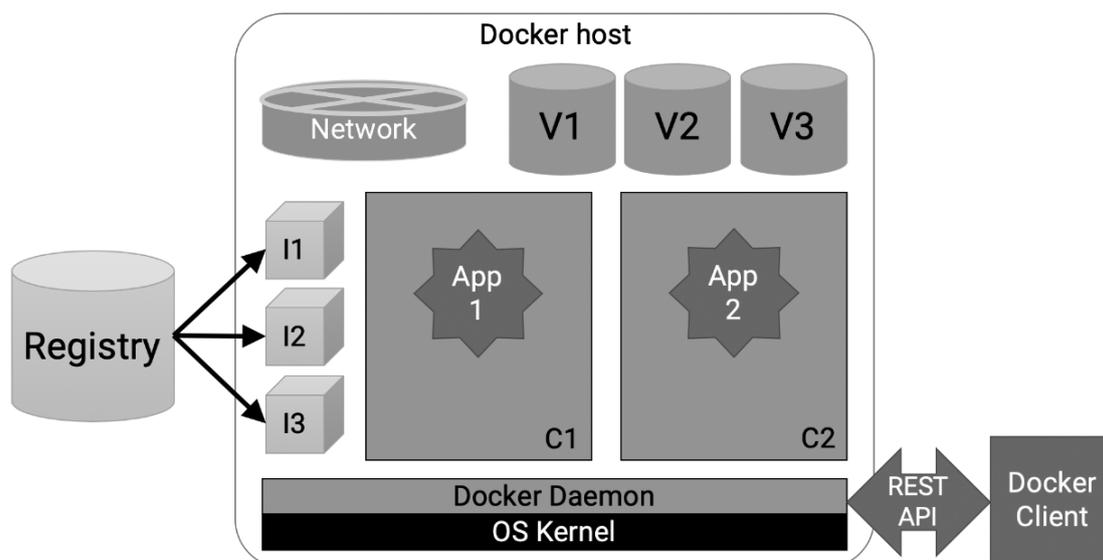


Figure 5. Docker deployment architecture on a node

Docker provides a complete infrastructure for containerization, including:

1. *Docker Engine* (kernel) consisting of a daemon, a REST API and a client.
2. *Orchestration mechanisms*: Docker Machine, Docker Swarm, Docker Compose.
3. *Installation packages* for desktop and cloud platforms: Linux, MacOS, Windows, Microsoft Azure, Amazon Web Services.
4. *Online services*: Docker HUB, CS Engine, Trusted Registry, Docker Cloud, Docker Store.

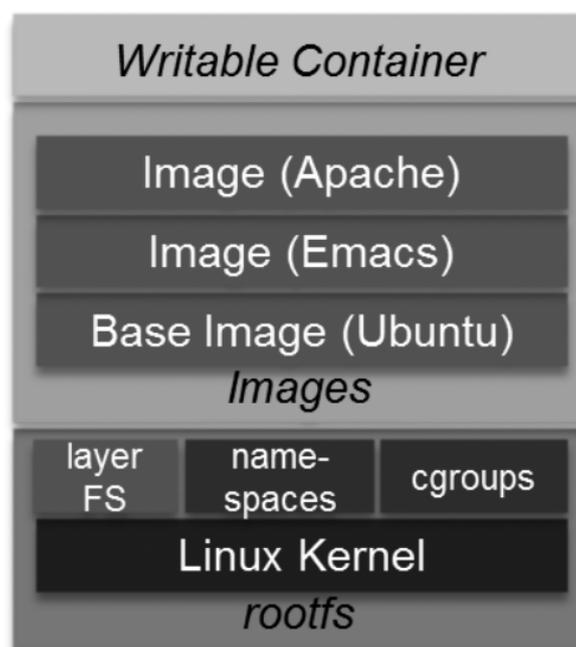
The Docker system deployed on the node is shown in Fig. 5, where:

- *Docker Daemon* is a management system, that manages containers, images, volumes, virtual networks.
- *C1..C2* are executable Docker containers, representing one or several processes running in an isolated environment. This environment provides no access to external processes, has its own root file system, network configuration (including hostname, IP-address, etc.). A container is executed within the framework of limitations on computational, memory, network and other I/O resources.
- *I1..I3* are images of Docker containers: read-only file system presets, containing the OS files, applications, all the necessary libraries and dependencies, except for the OS kernel itself. A container image is a way to distribute applications. Container images have a layered structure consisting of several images built on top of the base image. A typical division of an image of a container into layers may include from top to bottom: a modifiable container image for applications, basic images (for example, Apache and Emacs), a Linux image (for example, Ubuntu), and a *rootfs* kernel *image* (see Fig. 6).
- *V1..V3* are virtual volumes that provide long-term storage of data outside the container.
- *Docker Network* — a virtual network infrastructure that provides communication between containers, containers and the host, containers and the outside world.
- *REST API* is an API for Docker Daemon management.
- *Docker Client* is a Command Line Interface that provides management for the Docker infrastructure.

The Docker platform is gradually gaining popularity in the field of scientific problems associated with the Big Data processing. This is due to the fact that the Docker platform provides a single mechanism for containerized applications execution on the basis of a distributed computational environment, while simultaneously providing the necessary interfaces for network ports, volumes, etc., allowing different system users to work within the standardized computing environment [30].

## 2.4. NVIDIA Container Runtime

Big Data processing tasks often involve GPU resources in their solution for implementing parallel computing. In this regard, a number of attempts have been made to introduce virtualized graphics processors into virtual machines, including such approaches as GViM [35], gVirtuS [34] and vCUDA [81]. These approaches are based on creating copies of the CUDA API for virtualizing graphics processors and providing them to virtual machines. As a part of the rCUDA solution [26], the technology of remote graphics processors usage was proposed. However, these methods have their drawbacks, as they degrade the performance of the graphics processor during the virtualization process [41]. Moreover, these methods provide only a part of the CUDA API [40].



**Figure 6.** Container image structure in the Docker system (based on [68])

In 2016 NVIDIA Corporation proposed a solution called NVIDIA Docker [57], which differs from the approaches described above. NVIDIA Docker is a utility that makes it easy to use an NVIDIA GPU inside a Docker container. NVIDIA Docker contains two executable programs: `nvidia-docker` and `nvidia-docker-plugin`.

`nvidia-docker` is a shell on top of the Docker [59], which intercepts user commands to use the `nvidia-docker` command instead of the original `docker` command. The role of this command is to interpret and modify user commands with their subsequent transfer to the Docker command interface. `nvidia-docker` captures only the `run` and `create` commands, the rest of the commands are translated directly to the Docker. `nvidia-docker` checks whether the launched image uses the CUDA API using `com.nvidia.volumes.needed` and `com.nvidia.cuda.version` Docker labels, which specify the required CUDA versions. `nvidia-docker` uses data from these labels to determine the number and location of installed graphics devices and links them using the `--device` option. In addition, it links the correct versions of the GPU drivers using the `--volume` option, which is directly related to the `nvidia-docker-plugin`.

The `nvidia-docker-plugin` is an add-on designed to facilitate the deployment of containers that support GPUs. It acts as a daemon, identifies driver files, GPU devices, and responds to volume mount requests from the Docker daemon [60]. The purpose of the `nvidia-docker-plugin` is to check the existence of the NVIDIA GPU and CUDA API, as well as to provide the necessary versions of the binaries and libraries to the running container. The version of the CUDA API requested by the `nvidia-docker` command is provided via the `nvidia-docker-plugin` in the volume with the corresponding name. When the container completes its work, the driver volume shuts down.

The evaluation of the NVIDIA Docker approach shows that the performance of containerized GPU-accelerated applications is no different from the performance of the same applications

using GPUs outside the container [33]. NVIDIA Docker is successfully used to solve problems in machine learning, implemented on top of containerized infrastructures [46].

This approach is developed further into the NVIDIA Container Runtime [58] solution, which removes some of the limitations of the `nvidia-docker` project, including:

- support for the most common container technologies, such as LXC and CRI-O [21];
- compatibility with docker ecosystem tools, such as compose, for management of applications that use GPUs and consist of several containers;
- support of GPUs as resources in Kubernetes and Swarm container orchestration systems.

### 3. Comparison of Containerization and Virtualization Solutions

The virtual machine hypervisor emulates the hardware on top of which guest operating systems are running. Containers provide virtualization solutions at the level of the operating system: each guest OS uses the same kernel (and in some cases other parts of the OS) as the host. Such difference gives an advantage to containers approach: they are smaller and more compact than hypervisor guest environments since they have much more in common with the host.

In this section, we study the opportunities offered by containerization and virtualization solutions and present the results which compare the efficiency of containerization and virtualization technologies to solve the practical problems.

#### 3.1. Virtual Machines and Containers Comparison

Authors of [9, 25, 68] provide various methodologies for comparing containerization and virtualization technologies. General characteristics and approaches of comparing these technologies are presented in Tab. 1.

The following advantages have led to the widespread use of virtualization via containers [31]:

1. **Hardware costs.** Virtualization via containers decreases hardware costs by enabling consolidation. It enables concurrent software to take advantage of the true concurrency provided by a multicore hardware architecture.
2. **Reliability and robustness.** The modularity and isolation provided by VMs improve reliability, robustness, and operational availability by localizing the impact of defects to a single VM and enabling software failover and recovery.
3. **Scalability.** A single container engine can efficiently manage large numbers of containers, enabling additional containers to be created as needed.
4. **Spatial isolation.** Containers support lightweight spatial isolation by providing each container with its own resources (e.g., core processing unit, memory, and network access) and container-specific namespaces.
5. **Storage.** Compared with virtual machines, containers are lightweight with regard to storage size. The applications within containers share both binaries and libraries.
6. **Performance.** Compared with virtual machines, containers increase performance (throughput) because they do not emulate the underlying hardware. Note that this advantage is lost if containers are hosted on virtual machines (i.e., when using a hybrid virtualization architecture).
7. **Real-time applications.** Containers provide more consistent timing than virtual machines, although this advantage is lost when using hybrid virtualization.

Table 1. Comparison of virtual machines and containers (based on [9, 25, 68])

| Characteristic                          | Virtual machines   | Containers  |
|---|--|---|
| <b>Products</b>                         | VMware, Xen, KVM, ...  | Docker, rkt, LCX, OpenVZ, ...   |
| <b>Guest OS</b>                         | Each virtual machine is executed on the basis of its own OS loaded into its own block of RAM within the framework of virtual hardware. | All containers are executed on the basis of the OS kernel of the host machine.  |
| <b>Process management</b>               | Virtual machine hypervisor.  | Namespaces, cgroups.  |
| <b>Isolation</b>                        | Direct sharing of files or system libraries among guest and host OS is impossible.   | Catalogues can be transparently shared across multiple containers.  |
| <b>Image size</b>                       | Large image, because it includes the entire image of the base OS and all related libraries.  | Smaller image size, since a common OS kernel image is used.   |
| <b>Start-up time</b>                    | Starting a virtual machine takes a few minutes.  | Start-up time can be a few seconds.   |
| <b>Process of loading and execution</b> | After the standard boot process on the host, each virtual machine is represented as a separate process.                                | Applications can be started in containers directly or via an initial daemon known to the container, for example <i>dockerd</i> . They appear as normal processes on the host. |

8. **Continuous integration.** Containers support continuous development processes by enabling the integration of increments of container-hosted functionality.

9. **Portability.** Containers support portability from development to production environments, especially for cloud-based applications.

Although there are some disadvantages of containerization that should be addressed, especially in such cases as shared resources usage, weaker isolation, and security issues, comparing to virtual machines. Researchers and developers must address challenges and associated risks in the following areas when using containers:

1. **Shared resources.** Applications within containers share many resources including container engine, OS kernel, the host operating system, processor-internal resources (L3 cache, system bus, memory controller, I/O controllers, and interconnects) and processor-external resources (main memory, I/O devices, and networks). Such shared resources imply that software running in one container can impact software running in another container [31].

2. **Security.** Containers are not by default secure and require significant work to make them secure. One must ensure that no data is stored inside the container, container processes

are forced to write to container-specific file systems, the container's network namespace is connected to a specific, private intranet, and the privileges of container services are minimized (e.g., non-root if possible). There were several well-known flaws, those allowed attackers to create an application that would be able to escape the container and attack the host system. One of such vulnerabilities (recently patched) allowed a malicious container to (with minimal user interaction) overwrite the host `runc` binary and thus gain root-level code execution on the host [79].

In view of the above many cloud administrators and software architects suggest running a container on top of a VM to enhance security. Moreover this technic is implemented for easier management and upgrade of the system, as well as to overcome hardware and software incompatibility with the physical server. But all the aforementioned benefits come with a performance cost [51].

### 3.2. Comparative Effectiveness of Virtualization Technologies

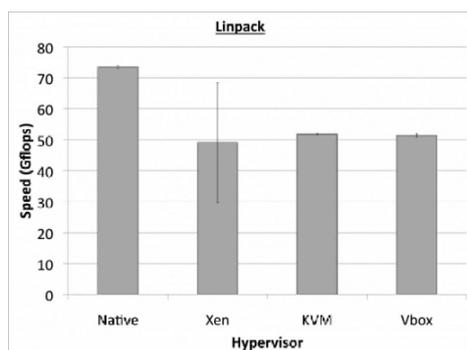
Authors of [103] compare hypervisor models with full virtualization and paravirtualization, such as Xen [6], KVM [44], VirtualBox [66] and VMWare ESX [95]. A comparison of characteristics of analyzed virtualization platforms is provided in Tab. 2. According to the test results, which included a comparison of the performance of the Linpack test, performing a fast Fourier transform, evaluating bandwidth and latency of network connections, as well as running an OpenMP program, the KVM platform (followed by VirtualBox) was recognized as the leader in performance (see Fig. 7). Unfortunately, the results of the effectiveness of VMWare ESX are not published in the article due to the limitations of the license agreement.

**Table 2.** Comparison of characteristics of virtualization platforms (based on [65, 87, 91, 94])

|                            | Xen 4.11             | KVM                    | VirtualBox 4.1       | VMWare ESXi 6.7      |
|----------------------------|----------------------|------------------------|----------------------|----------------------|
| <b>Paravirtualization</b>  | Yes                  | No                     | No                   | Yes                  |
| <b>Supported CPU</b>       | x86, x86-64, IA64    | x86, x86-64, IA64, PPC | x86, x86-64          | x86, x86-64          |
| <b>Host OS</b>             | Linux, UNIX          | Linux, UNIX            | Windows, Linux, UNIX | Proprietary UNIX     |
| <b>Guest OS</b>            | Windows, Linux, UNIX | Windows, Linux, UNIX   | Windows, Linux, UNIX | Windows, Linux, UNIX |
| <b>CPUs per host (x86)</b> | 4095                 | 4095                   | No Limit             | 768                  |
| <b>Memory per host</b>     | 16 TB                | 16 TB                  | 1 TB                 | 16 TB                |
| <b>License</b>             | GPL                  | GPL                    | GPL/proprietary      | Proprietary          |

The mechanisms of Xen's complete and paravirtualization models were investigated in [28]. The research results show that when using the full virtualization approach, the overhead is at least 35 % more compared to the paravirtualization model.

Many researchers are focusing on the efficient use of virtualized resources for solving Big Data processing problems. So, the authors of [47] compare an unnamed, but widely used, commercial hypervisor with open solutions: Xen and KVM. The main analysis is a series of typical data processing tasks on the Hadoop platform [83]. The greatest differences in performance were observed in tasks related to high I/O usage, while tests related to high CPU performance demand showed smaller differences between hypervisors. It was discovered that a commercial hypervisor



(a) Linpack test results

|              | Xen | KVM | VirtualBox |
|--------------|-----|-----|------------|
| Linpack      | 3   | 1   | 2          |
| FFT          | 3   | 1   | 2          |
| Bandwidth    | 2   | 3   | 1          |
| Latency      | 3   | 2   | 1          |
| OpenMP       | 2   | 1   | 3          |
| Total Rating | 13  | 8   | 9          |

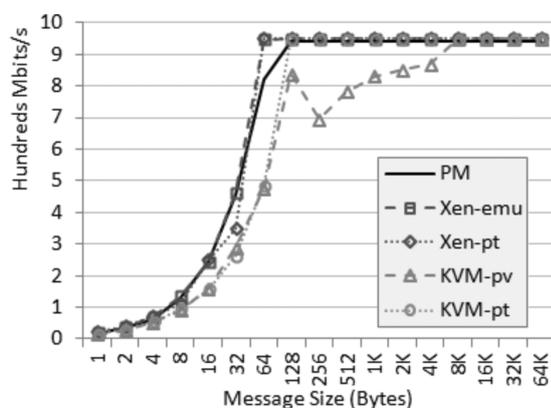
(b) The resulting rating (the smaller the better)

**Figure 7.** Testing of virtualization platforms (based on [103])

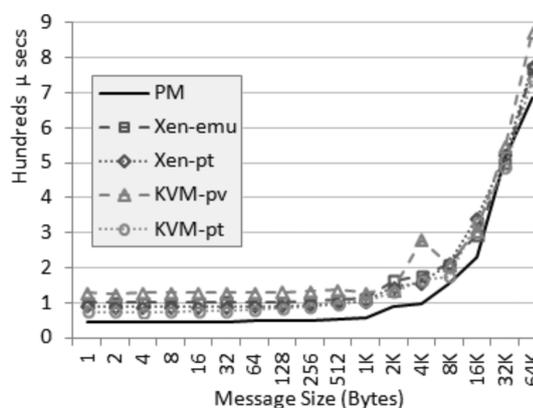
works better with disk write tasks, KVM is better for reading from disk, and Xen was better when the task required a combination of reading and writing a disk with intensive CPU calculations.

Virtualization with hardware support was tested on the basis of Xen and KVM hypervisors in the article [69] (see Fig. 8 and 9). In particular, to provide direct access to hardware devices by virtual machines, PCI transit technologies were used. The results showed that the use of hardware support technologies for virtualization can achieve low overhead costs both for performing I/O operations and for tasks that require large CPU resources.

The carried out tests showed that the network delay increases on average by 30–60 microseconds when using a Gigabit Ethernet type network and by 20–30 microseconds in InfiniBand systems. Performing tests on pure InfiniBand hardware in TCP mode gave a network delay of 20 to 130 microseconds depending on the packet size (about 25 microseconds with packets up to 2 kilobytes on average). Using RDMA reduced the delay to about 10–50 microseconds, depending on the packet size (about 10 microseconds with a packet size of up to 2 kilobytes on average). The use of virtualization, in this case, results in an increase in the response delay of 1.5–2.5 times: in TCP mode from 50 to 170 microseconds (about 65 microseconds with a packet size up to 2 kilobytes on average), in RDMA mode both for Xen and KVM the delay is from 25 to 80 microseconds, depending on the packet size (about 30 microseconds with packet sizes up to 2 kilobytes on average). Performance evaluation of an MPI application showed performance degradation of virtualized systems by 20–50 % with KVM virtualization versus Xen.

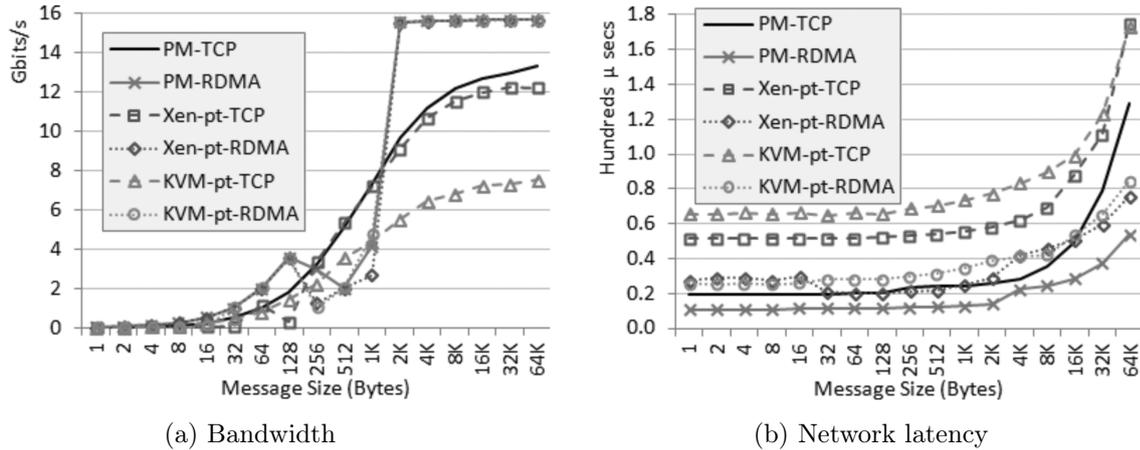


(a) Bandwidth



(b) Network latency

**Figure 8.** Comparison of Gigabit Ethernet type network performance on bare metal (PM), and different types of Xen and KVM virtualization (based on [69])



**Figure 9.** Comparison of an InfiniBand network performance on bare metal (PM), and different types of Xen and KVM virtualization (based on [69])

### 3.3. Comparison of Container and Virtual Machine Performance

The authors of [98] discuss the comparison of VMware, Xen and OpenVZ virtualization and containerization technologies from the point of view of different workloads. OpenVZ as the solution for virtualization at the OS level showed the least amount of overhead and the highest performance. Che et al in [18] also evaluated the performance of OpenVZ, Xen and KVM. Test results showed that OpenVZ has better performance, and KVM (full virtualization) has lower performance than Xen (paravirtualization).

The authors of [67] compares Xen and OpenVZ in different configurations. The results showed that Xen had a large overhead, which led to an increase in OpenVZ performance. Various solutions for virtualization at the operating system level were compared to Xen in [101]. Overall, Xen achieved lower performance than container-based virtualization solutions. LXC achieved the best results among container-based solutions. However, performance isolation between virtual machines was best on Xen, which may be a drawback to OS-based virtualization solutions.

Authors of [29] present the results of a comparison of various aspects of the performance of virtualization and containerization technologies, such as I/O performance, network connection latency and computational performance. For performance testing, the authors use the following types of workload (see Tab. 3):

- PXZ, a parallel lossless data compression utility that uses LZMA algorithm;
- LINPAC, a package that provides a solution to linear equation systems using a LU factoring algorithm with partial rotation.

The authors also tested the performance of data processing on virtualized/containerized systems (see Fig. 10 to 12).

**Table 3.** Comparison of the performance of bare metal, docker and KVM solutions (based on [29])

| Load                 | Bare metal               | Docker                          | KVM without fine tuning         | KVM (fine tuning)           |
|----------------------|--------------------------|---------------------------------|---------------------------------|-----------------------------|
| PXZ (MB/s)           | 76.2 [ $\pm 0.93$ ]      | 73.5 (-4 %) [ $\pm 0.64$ ]      | 59.2 (-22 %) [ $\pm 1.88$ ]     | 62.2 (-18 %) [ $\pm 1.33$ ] |
| LINPAC (Gigaflop)    | 290.8 [ $\pm 1.13$ ]     | 290.9 (-0 %) [ $\pm 0.98$ ]     | 241.3 (-17 %) [ $\pm 1.18$ ]    | 284.2 (-2 %) [ $\pm 1.45$ ] |
| Random Access (GUPS) | 0.0126 [ $\pm 0.00029$ ] | 0.0124 (-2 %) [ $\pm 0.00044$ ] | 0.0125 (-1 %) [ $\pm 0.00032$ ] | Not performed               |

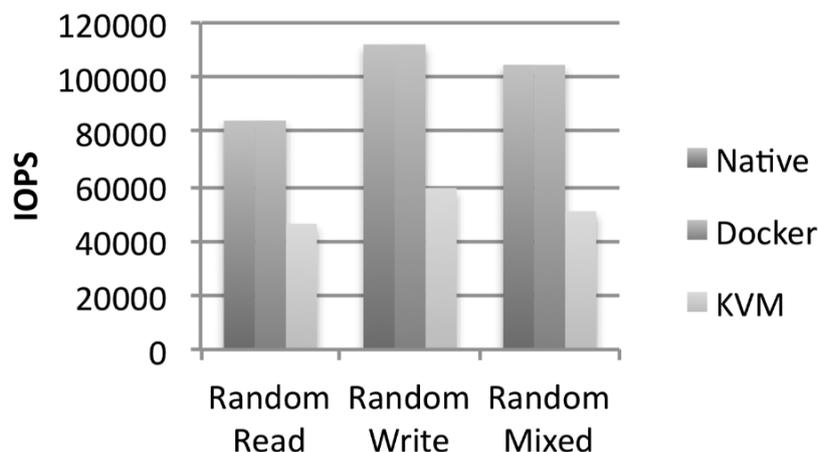


Figure 10. Testing the performance of random I/O (IOPS) (based on [29])

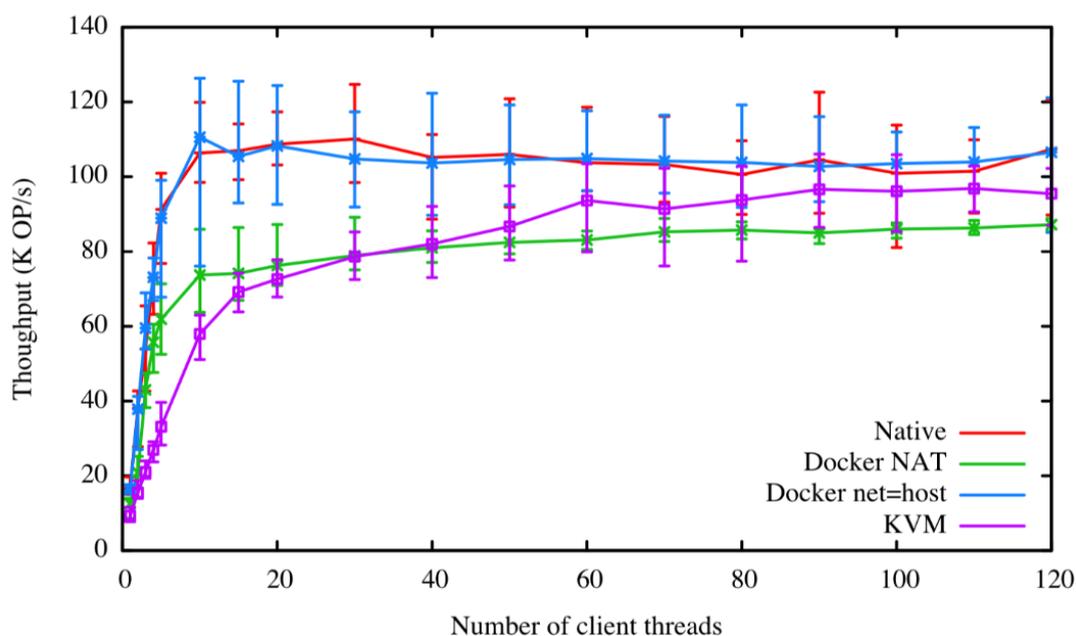
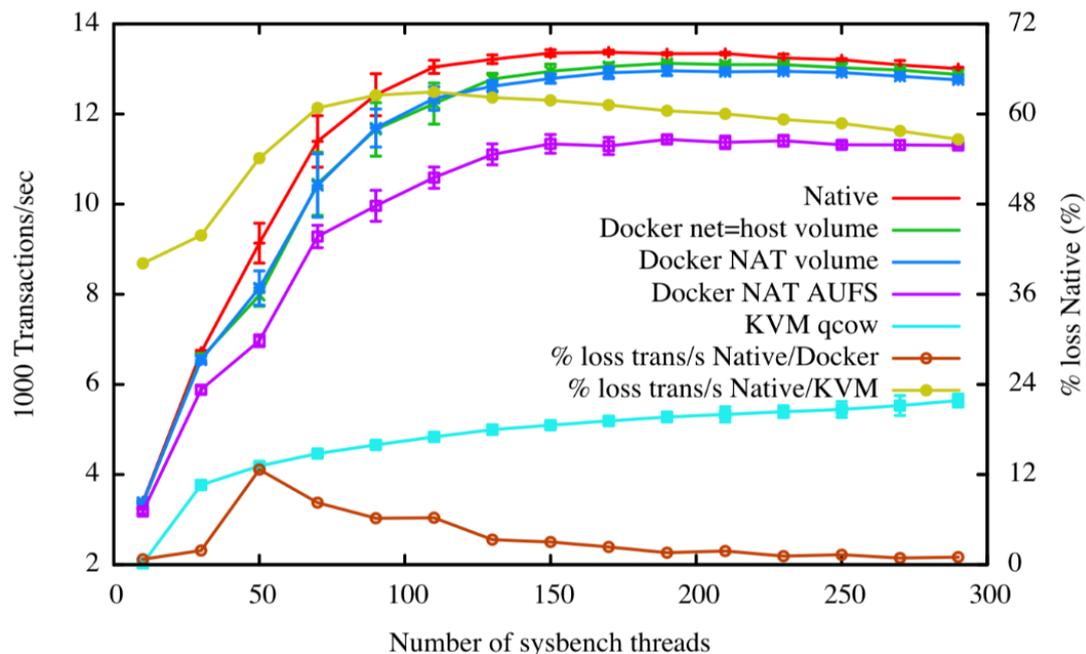


Figure 11. Performance evaluation (queries/s) of NoSQL Redis database for several deployment scenarios. Each data point is an arithmetic average obtained from 10 runs (based on [29])

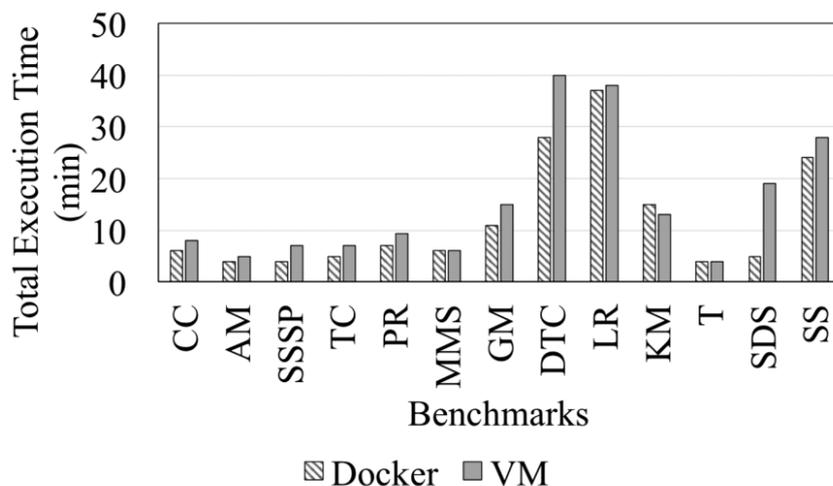
The authors of [29] make the following conclusion: “In general, Docker equals or exceeds KVM performance in every case we tested... Although containers themselves have almost no overhead, Docker is not without performance gotchas. Docker volumes have noticeably better performance than files stored in AUFS. Docker’s NAT also introduces overhead for workloads with high packet rates. These features represent a tradeoff between ease of management and performance and should be considered on a case-by-case basis”.

The authors of [9] provide an analysis of the effectiveness of using virtualization and containerization technologies to solve problems in machine learning, large graphs processing and SQL queries execution (see Tab. 4) based on the Apache Spark platform. Test results show that using Spark based on Docker allows you to get acceleration more than 10 times compared to using virtual machines. However, these results vary considerably by the type of executable ap-



**Figure 12.** MySQL database performance (transactions/s) depending on parallelism (based on [29])

plications due to different load patterns and different resource management schemes (Fig. 13). In particular, it turned out that the implementation of the problem of data clustering based on the k-means method in a containerized environment is slower than inside a virtual machine. The analysis showed that this is due to a large number of mixing operations required for the implementation of the k-means algorithm, each of which causes I/O operations. The specifics of the implementation of the AUFS file system (copy-on-write technology) used in Docker, leads to the fact that a large number of I/O operations can lead to inhibition of other processes.



**Figure 13.** Comparison of the execution time of typical Spark tasks in the virtual machine environment and Docker containers (based on [9])

**Table 4.** List of typical data processing tasks (based on [9])

| The area                 | Typical tasks  |
|--------------------------|--|
| <b>Machine learning</b>  | K-Means (KM)<br>MinMaxScaler (MMS)<br>GaussianMixture (GM)<br>Logistic Regression (LR)<br>DecisionTreeClassification (DTC)<br>Tokenizer (T)<br>Alternating Least Squares (ALS) |
| <b>Graphs processing</b> | PageRank (PR)<br>ConnectedComponents (CC)<br>TriangleCounting (TC)<br>AggregateMessages (AM)<br>Single-Source Shortest Paths (SSSP)  |
| <b>SQL queries</b>       | SQLDataSource (SDS)<br>SparkSQL (SS)   |

---

As another example of usage of containerization technologies for Big Data problems, Sogand Shirinbab et al. [82] provide a performance comparison between VMware virtual machine and Docker container while running Apache Cassandra NoSQL distributed database as workload. Authors conclude that Docker had lower overhead compared to the VMware when running Cassandra.

The solution of Big Data problems often requires an effective way to utilize GPU resources. Authors of [99] conducted a comparative study of the performance of KVM, Xen and VMWare ESXi hypervisors together with LXC container management solution for execution of CUDA and OpenCL Applications. The experiments show, that LXC solution consistently performed closest to the native case. Authors also show that GPU passthrough to KVM achieves 98–100 % of the base system’s performance, while Xen and VMWare achieve 96–99 % of the base systems performance, respectively.

### 3.4. Conclusion

The analysis shows that virtualization and containerization technologies can be used to solve the tasks of Big Data processing as a mean of deploying specialized software platforms. Analysis of the performance of these solutions shows that, with rare exceptions, the overhead for container virtualization when deploying and executing software solutions are substantially less (from 10 % to 90 %) than for virtualization using the hypervisor. Exceptional cases are associated with the tasks that are focused on a large amount of I/O. This is due to the fact that, at the present stage of development, containers are not capable of leveling the mutual influence on each other of tasks implemented within the framework of the same computing module. In this case, containerization performance becomes comparable to the performance of virtual machine-based solutions.

## 4. Container Orchestration Technologies

Microservice architecture approach is aimed at solving the problem of partitioning of monolithic applications into SOA-style independently deployable services that are well supported by container architectures. Such services should be loosely coupled, supporting rapid deployment and termination.

The architectural style of microservices is an approach to developing a single application as a set of small services, each of which works in its own process and interacts with other services through standardized network protocols [83]. The life cycle of microservices should be supported by a fully automated deployment and orchestration platform. They require independent deployment and scalability depending on the current load and need, as opposed to synchronous deployment at specific times, typical for monolithic applications.

Also, the use of containerized computing services for solving real-world problems requires solving the problem of organizing their joint work. So, it is necessary to provide dependency management between containers. Orchestration plan describes components, their dependencies, and their life cycle. The PaaS cloud can then trigger workflows from the orchestration plan through agents. Software platform services can support packaging applications and their deployment from containers [56].

In this section, we discuss solutions that enable the deployment and collaboration of containerized applications within cloud infrastructures.

### 4.1. Private Cloud IaaS Platforms

Cloud computing is characterized by the dynamic provision of computing resources based on service level agreements between the service provider and the consumer [14]. To implement cloud computing, many open source solutions have been developed [96]. Authors of [27] present a taxonomy and architecture for cloud solutions, together with a comparison between open source cloud solutions.

The architectural and philosophical differences and similarities between the Eucalyptus [5], OpenNebula [62] cloud platforms and Nimbus [93] cloud solutions were compared in [80]. Authors of [100] compare the OpenNebula and OpenStack [63] platforms in terms of architecture, support for virtualization hypervisors, cloud APIs, and security aspects. Authors of [45] provide a comparison of the OpenStack, OpenNebula, Nimbus and Eucalyptus platforms in terms of interfaces, hypervisors, network capabilities, deployment and storage procedures, to assess the suitability of their use for the FutureGrid testing environment. The scalability of physical and virtual machine provisioning has also been verified. As a result of testing, the OpenStack platform showed the best results.

Q. Huang et al [36] compared CloudStack, Eucalyptus and OpenNebula platforms with the performance of classic server solutions. The results show that the use of cloud solutions provide about 10 % degradation in application performance due to the use of virtualization technologies. Performance deteriorates as the number of virtual machines in use increases. OpenNebula achieved better performance than other cloud solutions (CloudStack, Eucalyptus). Authors presented a comparison of the capabilities of the same cloud solutions for developing applications in the field of geophysical research in their next paper [37]. In particular, they compared their performance characteristics, including computational performance, I/O performance, memory performance, network data transfer speeds, and applications for geophysical research. The dif-

ference was observed in web application support, where OpenNebula showed slightly better performance since this system traffic is not routed through the cloud controller. OpenNebula also achieved better results in geophysical applications, albeit by a small margin as compared with the alternatives.

## 4.2. PaaS Clouds and Containerization

Currently, virtual machines create a fabric of the IaaS level clouds. Containers, however, look like a very suitable technology for packaging and managing applications in PaaS clouds. The PaaS model provides mechanisms for designing and deploying cloud applications, providing software infrastructure and libraries for transparently launching web services, routing requests and distributing workload between cloud resources [43].

Container platforms eliminate application deployment problems with compatible, lightweight, and virtualized packaging. Those containers, that are developed outside the PaaS platform, can be transferred to other computing environments, because the container provides encapsulation of applications. Some PaaS models are now compatible with containerization and standardized packaging of applications, for example, based on Docker. This development is part of the evolution of PaaS approach, moving towards a compatible PaaS based on containers. The first generation of PaaS solutions includes classic proprietary PaaS platforms, such as Microsoft Azure, Heroku, and Google App Engine [97].

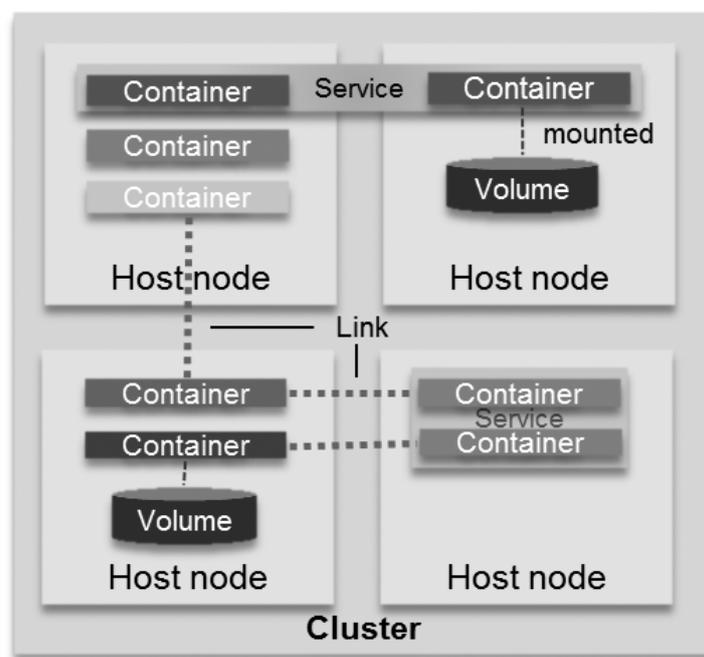
The second generation of PaaS is built on open source solutions [7], such as Cloud Foundry [8] or OpenShift [74], which allow users to run their own PaaS (both locally and in the cloud), with integrated container support. In 2017, OpenShift switched from its own container model to the Docker model. The Cloud Foundry platform implements containerization based on its internal Diego solution [19]. Cloud Foundry and OpenShift handle containers differently. Cloud Foundry supports stateless applications through containers, but services that require state preservation are deployed in virtual machines. On the other hand, OpenShift does not distinguish them [68].

## 4.3. Container Clustering and Orchestration

The next problem is to facilitate the transition from one container to clusters of containers that allow running containerized applications on several clusters or in several clouds [50].

Authors of [43] proposed a general solution to ensure the formation of container clusters (Fig. 14). Within this model, each *computing cluster* consists of several nodes — virtual servers on hypervisors or, possibly, on physical servers with a single tenant. Each node contains several containers with common services that provide planning, load balancing, and application execution. Further, *application services* are logical groups of containers from the same image. Services allow you to scale applications through sites. *Volumes* are used for applications requiring persistent data. Containers can mount volumes. The data stored in these volumes is retained even after the container has been terminated. Finally, *links* allow you to create a connection and provide connectivity for two or more containers.

Deployment of distributed applications through containers is supported by a virtual scalable *head node (or head cluster)* that provides scaling, load balancing and fault tolerance of the entire system. At the same time, the API allows you to manage the life cycle of clusters. The head node of the cluster accepts commands from the outside and sends them to the container hosts. This



**Figure 14.** Containers cluster architecture (based on [43])

allows designing multi-container cloud applications without regard to the basic network topology and avoids manual configuration [32].

As part of the cluster architecture, the basic cloud infrastructure provides solutions of service discovery (for example, through shared distributed “key-value” storages), orchestration and deployment, including load balancing, monitoring, scaling, and data transfer management.

It should be noted that the OASIS organization is developing a “*Topology and Orchestration Specification for Cloud Applications*” (TOSCA) standard [11, 12, 61]. TOSCA supports several functions:

- compatible description of application cloud services and infrastructure;
- communication between parts of the service;
- operational behavior of these services (deployment, updating or shutdown) in terms of orchestration.

The implementation of such a standard provides the advantage of independence from the service supplier, as well as any particular cloud service or hosting provider. TOSCA templates can be used to define container clusters, abstract nodes and relation types, as well as application stack templates [10].

TOSCA standard defines cloud services orchestration plan on the basis of the YAML language. This orchestration plan is used to organize the deployment of applications, as well as automation processes that are implemented after deployment. The orchestration plan describes the applications and their life cycle, as well as the relationships between the components. This includes connections between applications and their locations. Using TOSCA, one can describe the service infrastructure, the intermediate computational layer of the platform, and the application layer located on the top (see Fig. 15).

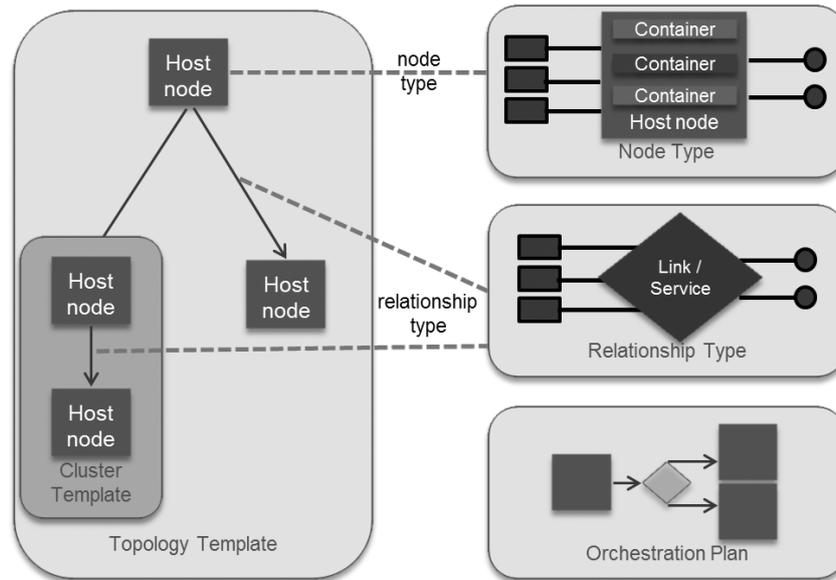


Figure 15. Reference structure for orchestrating cluster topology.

There are a number of PaaS platforms that support the TOSCA standard. For example, the Cloudify platform [20] will be able to adopt the TOSCA orchestration plan and then apply it by creating the necessary infrastructure and running the application.

#### 4.4. Review of Container Orchestration Solutions

The goal for a container-based virtual clusters is to provide the users with computer clusters to be used as if they were physical computing clusters, with the added value of using containers instead of VMs. Therefore, the requirements for the container-based virtual cluster is to preserve the very same environment and usage patterns that are commonly used in this computing platforms, i.e. the software stack: the OS, the cluster middleware, the parallel environments and the applications, as shown in Fig. 16 [1].

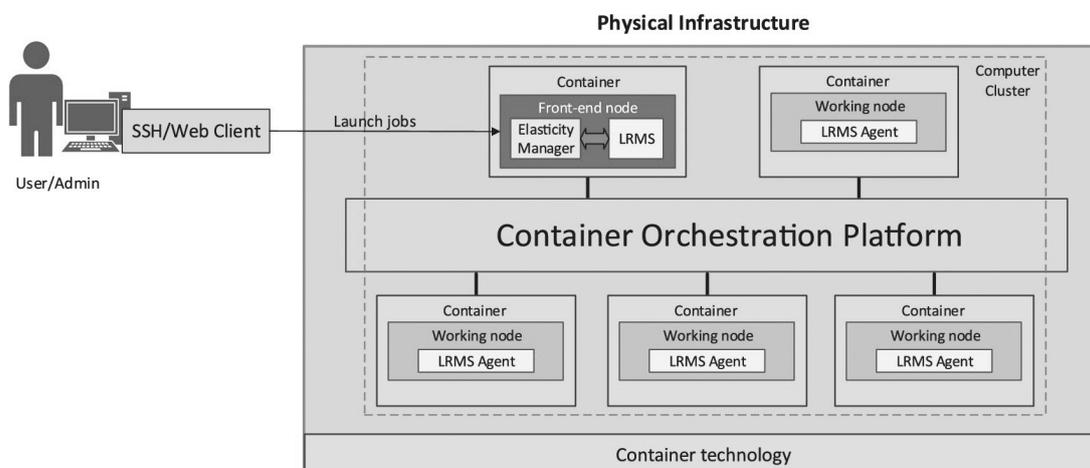


Figure 16. Generic architecture to deliver container-based virtual computer clusters deployed on a computing infrastructure managed by a Container Orchestration Platform (based on [1])

In this section, we would consider the most used systems that provide an orchestration of containerized applications. We would overview the following systems: Docker Compose, Docker Swarm, Apache Mesos, and Kubernetes.

*Docker Compose* is a Docker solution that provides the creation of multi-container Docker-based applications [23]. This solution is based on the YAML configuration files, which define the relationships between containers and the details of their interaction (such as images, volumes, service configurations). We can highlight the following main advantages of this project: comparative ease of implementation, convenience and ease of setup, as well as the possibility of easy distribution of cluster configurations in the format of YAML files [30]. The disadvantages include slightly less functionality compared to other projects under consideration (including, supporting high availability, etc.). Docker Compose is a suitable solution for application developers, but not functional enough to support large infrastructures.

*Docker Swarm* was originally a supplement to the Docker platform, but since Docker version 1.12 it has been added to the main package [24]. The Docker Swarm system provides a gradual update of services, the organization of inter-node network connections, load balancing in multi-container applications. In a Swarm, each task maps to one container. When using Docker you can control container placement decisions by using labels (tags) either user defined or system defined. The scheduler also takes into consideration CPU and memory constraints when scheduling containers (see Fig. 17) [39].

The main advantage of Docker Swarm is the built-in support for the Docker Compose configuration files. Compared to Docker Compose, this is a more advanced solution, similar to other orchestrators, although its capabilities are still limited compared to the Kubernetes ecosystem. The Docker Swarm solution is suitable for small clusters that require simple management, deployment and use in small production environments. One of the limitations of Swarm Orchestrator is its scalability, as well as the fact that it only works with Docker containers [52]. The *Apache*

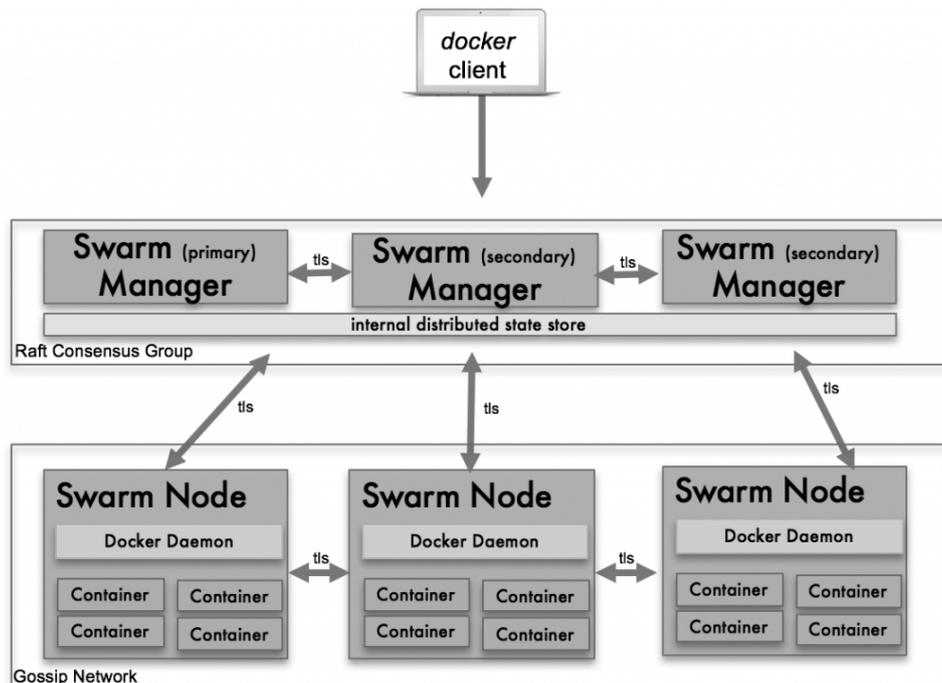


Figure 17. Docker Swarm Architecture (based on [39])

*Mesos* [3] project is a platform that links distributed hardware resources into a single pool of resources. These resources can be used by application frameworks to distribute the load between them. The project is based on the core of a distributed system, which follows the same principles as the Linux kernel, but at a different level of abstraction. The Mesos kernel runs on all machines in the cluster. This makes it easy to use applications with APIs for resource management and scheduling in cloud environments. The difference between Apache Mesos and other solutions is that Mesos is suitable not only for containers. This is an open source solution composed of Apache Foundation projects (such as Hadoop and Zookeeper). Container orchestration is provided by the Marathon platform [53], which is part of the Apache Mesos project. As for compatibility, it initially supports LXC, as well as Docker. This orchestration system is often used to organize the solution of issues in the field of Big Data processing [15, 52, 77, 86].

Another solution for managing clusters, albeit at a higher level than Mesos, is the *Kubernetes* system [90]. Kubernetes is an open source project developed by Google, that automates the deployment, scaling and management of container applications. This platform was built on the basis of their experience with containers over the past decade. This project is an orchestrator, which, unlike other technologies, supports several container solutions including, Docker, rkt, CRI-O, Windows containers. Kubernetes consists of two main roles: the master and the node. The master is the main component that controls each group of nodes. Nodes accept the requirements of the master and perform the actions defined by the master. Three main components are used to launch and manage containers:

- The pod is the basic unit of planning and deployment, which is a group of related containers.
- A replication controller that is a supervisor for pods and controls the status of the cluster.
- Kubernetes Services and kubelets, which run on each container and manage containers.

The Kubernetes platform is also gaining popularity in such fields as machine learning and Big Data processing. For example, authors of [46] present the BAIPAS system — a distributed platform supporting Big Data processing, artificial intelligence, and machine learning. BAIPAS uses Kubernetes and Docker to provide simple platform installation and monitoring, as well as NVIDIA Docker to provide GPU resources for deploying TensorFlow within containers.

The authors of [78] describe the computing system architecture to support the EO4Wildlife project ([www.eo4wildlife.eu](http://www.eo4wildlife.eu)). In order to ensure the interaction of a wide range of researchers with a set of large geo-information data, the EO4wildlife platform was developed, providing the implementation of the Spark system based on containerized infrastructure supported by Kubernetes. In this project, Kubernetes is responsible for application deployment and management, including automatic scaling, load balancing, and monitoring of tasks.

## 4.5. Summary

The analysis shows that orchestration and containerization technologies are increasingly used to implement Big Data processing solutions. They can be used either explicitly, by using container orchestration systems such as Docker Compose or Kubernetes, or indirectly, by deploying applications in PaaS environments that support automatic scaling and lifecycle management of computing services, such as Cloudify or OpenShift.

The use of such systems allows one to automate the process of application deployment. It also makes it easier to repeat experiments, because files describing the computing infrastructure and the deployment of computing services can be easily distributed among researchers. Also, these technologies are well integrated with other solutions and extensions of container computing

systems, such as NVIDIA Docker, which allows them to be effectively used for solving problems related to data analysis and machine learning.

## Conclusion

Solving the issues of Big Data processing is impossible without the use of distributed computing infrastructures. To date, the key technologies that support the fabric of distributed computing systems are technologies of virtualization and containerization of resources.

We have analyzed the key technologies of virtualization and containerization of computing resources used today. Virtualization has been designed to abstract hardware and system resources in order to ensure that multiple operating systems work together on the basis of one physical node. There are several approaches to the implementation of virtualization. Full virtualization is aimed at hardware emulation. Paravirtualization requires modification of the virtualized OS and coordination of operations between the virtual OS and the hypervisor. The OS-level virtualization approach (or containerization) does not require a hypervisor. Instead, the base OS is modified to ensure that several instances of the OS are able to be executed on the same machine. Although there are some disadvantages of containerization that should be addressed, especially in such cases as shared resources usage, weaker isolation, and security issues, comparing to virtual machines.

The analysis shows that virtualization and containerization technologies can be used in Big Data processing as a means of deploying of specialized software platforms. Analysis of the performance of these solutions shows that, with rare exceptions, the storage costs for container virtualization when deploying and executing software solutions are substantially less (from 10 % to 90 %) than for virtualization using the hypervisor.

Also, orchestration technologies are increasingly being used to implement Big Data processing solutions. They can be used either explicitly, by using container orchestration systems, such as Docker Compose or Kubernetes, or indirectly, by deploying applications in PaaS environments that support automatic scaling and lifecycle management of computing services, such as Cloudify or OpenShift.

The use of such systems allows one to automate the process of deployment of applications in a cloud environment. Such approach also makes it easier to repeat experiments, because files describing the computing infrastructure and the deployment of computing services can be easily distributed among researchers. Also, these technologies are well integrated with other solutions and extensions of container computing systems, such as NVIDIA Docker, which allows them to be effectively used for solving problems related to data analysis and machine learning.

It can be noted that the transition from virtualization to containerization has reduced the overhead costs associated with managing computing resources by orders of magnitude. This made it possible to efficiently use containerization technologies for solving a large class of problems requiring high performance from computing resources, including Big Data processing tasks.

## Acknowledgements

This article contains the results of a project carried out as part of the implementation of the Program of the Center for Competence of the National Technology Initiative “Center for Storage and Analysis of Big Data”, supported by the Ministry of Science and Higher Education of the Russian Federation under the Contract of Moscow State University with the Fund for Support of Projects of the National Technology Initiative No. 13/1251/2018 (December 11, 2017).

*This paper is distributed under the terms of the Creative Commons Attribution-Non Commercial 3.0 License which permits non-commercial use, reproduction and distribution of the work without further permission provided the original work is properly cited.*

## References

1. de Alfonso, C., Calatrava, A., Moltó, G.: Container-based virtual elastic clusters. *Journal of Systems and Software* 127, 1–11 (2017), DOI: 10.1016/j.jss.2017.01.007
2. Anderson, C.: Docker. *IEEE Software* 32(3), 102–c3 (2015), DOI: 10.1109/MS.2015.62
3. Apache Software Foundation: Apache Mesos. <http://mesos.apache.org/>, accessed: 2018-12-04
4. Apache Software Foundation: Apache Tomcat. <http://tomcat.apache.org/>, accessed: 2018-11-29
5. Appscale Systems: Eucalyptus. <https://www.eucalyptus.cloud/>, accessed: 2018-11-30
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: *Proceedings of the nineteenth ACM symposium on Operating systems principles - SOSP '03*. pp. 164–177. ACM Press, New York, New York, USA (2003), DOI: 10.1145/945445.945462
7. Baset, S.A.: Open source cloud technologies. In: *Proceedings of the Third ACM Symposium on Cloud Computing - SoCC '12*. pp. 1–2. ACM Press, New York, New York, USA (2012), DOI: 10.1145/2391229.2391257
8. Bernstein, D.: Cloud Foundry Aims to Become the OpenStack of PaaS. *IEEE Cloud Computing* 1(2), 57–60 (2014), DOI: 10.1109/MCC.2014.32
9. Bhimani, J., Yang, Z., Leiser, M., Mi, N.: Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In: *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1–7. IEEE (2017), DOI: 10.1109/HPEC.2017.8091086
10. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A runtime for TOSCA-based cloud applications. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2013), DOI: 10.1007/978-3-642-45005-1\_62
11. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications. In: *Advanced Web Services*, pp. 527–549. Springer New York, New York, NY (2014), DOI: 10.1007/978-1-4614-7535-4\_22
12. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable Cloud Services Using TOSCA. *IEEE Internet Computing* 16(3), 80–85 (2012), DOI: 10.1109/MIC.2012.43
13. Boettiger, C.: An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49(1), 71–79 (2015), DOI: 10.1145/2723872.2723882

14. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25(6), 599–616 (2009), DOI: 10.1016/j.future.2008.12.001
15. Canizo, M., Onieva, E., Conde, A., Charramendieta, S., Trujillo, S.: Real-time predictive maintenance for wind turbines using Big Data frameworks. In: 2017 IEEE International Conference on Prognostics and Health Management (ICPHM). pp. 70–77. IEEE (2017), DOI: 10.1109/ICPHM.2017.7998308
16. Canonical Ltd.: Linux Containers - LXD- Introduction Accessed: 2018-11-29
17. Canosa, R., Tchernykh, A., Cortés-Mendoza, J.M., Rivera-Rodriguez, R., Rizk, J.E.L., Avetisyan, A., Du, Z., Radchenko, G., Morales, E.C.: Energy consumption and quality of service optimization in containerized cloud computing. In: The Proceedings of the 2018 Ivannikov ISPRAS Open Conference (ISPRAS 2018). pp. 47–55. IEEE, Mocsow (2018), DOI: 10.1109/ISPRAS.2018.00014
18. Che, J., Shi, C., Yu, Y., Lin, W.: A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. In: 2010 IEEE Asia-Pacific Services Computing Conference. pp. 587–594. IEEE (2010), DOI: 10.1109/APSCC.2010.83
19. Cloud Foundry Foundation: Diego Components and Architecture | Cloud Foundry Docs. <https://docs.cloudfoundry.org/concepts/diego/diego-architecture.html>, accessed: 2018-12-04
20. Cloudify Platform: Cloud NFV Orchestration Based on TOSCA | Cloudify. <https://cloudify.co/>, accessed: 2018-12-05
21. CRI-O author: Cri-o. <http://cri-o.io/>, accessed: 2018-11-30
22. Docker Inc.: Docker - Build, Ship, and Run Any App, Anywhere. <https://www.docker.com/>, accessed: 2019-02-28
23. Docker Inc.: Docker Compose | Docker Documentation. <https://docs.docker.com/compose/>, accessed: 2018-12-04
24. Docker Inc.: Swarm mode key concepts | Docker Documentation. <https://docs.docker.com/engine/swarm/key-concepts/>, accessed: 2018-12-04
25. Dua, R., Raja, A.R., Kakadia, D.: Virtualization vs Containerization to Support PaaS. In: 2014 IEEE International Conference on Cloud Engineering. pp. 610–614. IEEE (2014), DOI: 10.1109/IC2E.2014.41
26. Duato, J., Pena, A.J., Silla, F., Mayo, R., Quintana-Orti, E.S.: rCUDA: Reducing the number of GPU-based accelerators in high performance clusters. In: 2010 International Conference on High Performance Computing & Simulation. pp. 224–231. IEEE (2010), DOI: 10.1109/HPCS.2010.5547126
27. Dukaric, R., Juric, M.B.: Towards a unified taxonomy and architecture of cloud frameworks. *Future Generation Computer Systems* 29(5), 1196–1210 (2013), DOI: 10.1016/j.future.2012.09.006

28. Fayyad-Kazan, H., Perneel, L., Timmerman, M.: Full and Para-Virtualization with Xen: A Performance Comparison. *Journal of Emerging Trends in Computing and Information Sciences* 4(9), 719–727 (2013)
29. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and Linux containers. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 171–172. IEEE (2015), DOI: 10.1109/ISPASS.2015.7095802
30. Filgueira, R., Da Silva, R.F., Deelman, E., Christodoulou, V., Krause, A.: IoT-Hub: New IoT data-platform for Virtual Research Environments. In: 10th International Workshop on Science Gateways (IWSG 2018). pp. 13–15 (2018)
31. Firesmith, D.: Virtualization via Containers. [https://insights.sei.cmu.edu/sei\\_blog/2017/09/virtualization-via-containers.html](https://insights.sei.cmu.edu/sei_blog/2017/09/virtualization-via-containers.html) (2017), accessed: 2018-02-28
32. Gass, O., Meth, H., Maedche, A.: PaaS Characteristics for Productive Software Development: An Evaluation Framework. *IEEE Internet Computing* 18(1), 56–64 (2014), DOI: 10.1109/MIC.2014.12
33. Gerdau, B.L., Weier, M., Hinkenjann, A.: Containerized Distributed Rendering for Interactive Environments. In: EuroVR 2017: Virtual Reality and Augmented Reality. pp. 69–86 (2017), DOI: 10.1007/978-3-319-72323-5\_5
34. Giunta, G., Montella, R., Agrillo, G., Coviello, G.: A GPGPU Transparent Virtualization Component for High Performance Computing Clouds. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 379–391 (2010), DOI: 10.1007/978-3-642-15277-1\_37
35. Gupta, V., Gavrilovska, A., Schwan, K., Kharche, H., Tolia, N., Talwar, V., Ranganathan, P.: GViM: GPU-accelerated virtual machines. In: *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing - HPCVirt '09*. pp. 17–24. ACM Press, New York, New York, USA (2009), DOI: 10.1145/1519138.1519141
36. Huang, Q., Xia, J., Yang, C., Liu, K., Li, J., Gui, Z., Hassan, M., Chen, S.: An experimental study of open-source cloud platforms for dust storm forecasting. In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12*. p. 534. ACM Press, New York, New York, USA (2012), DOI: 10.1145/2424321.2424408
37. Huang, Q., Yang, C., Liu, K., Xia, J., Xu, C., Li, J., Gui, Z., Sun, M., Li, Z.: Evaluating open-source cloud computing solutions for geosciences. *Computers & Geosciences* 59, 41–52 (2013), DOI: 10.1016/j.cageo.2013.05.001
38. Huber, N., von Quast, M., Hauck, M., Kounev, S.: Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In: *Proceedings of the 1st International Conference on Cloud Computing and Services Science*. pp. 563–573. SciTePress - Science and Technology Publications (2011), DOI: 10.5220/0003388905630573

39. IT Solution Architects: Containers 102: Continuing the Journey from OS Virtualization to Workload Virtualization. <https://medium.com/@ITsolutions/containers-102-continuing-the-journey-from-os-virtualization-to-workload-virtualization-54fe5576969d> (2017), accessed: 2019-02-27
40. Kang, D., Jun, T.J., Kim, D., Kim, J., Kim, D.: ConVGPU: GPU Management Middleware in Container Based Virtualized Environment. In: 2017 IEEE International Conference on Cluster Computing (CLUSTER). pp. 301–309. IEEE (2017), DOI: 10.1109/CLUSTER.2017.17
41. Kim, J., Jun, T.J., Kang, D., Kim, D., Kim, D.: GPU Enabled Serverless Computing Framework. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). pp. 533–540. IEEE (2018), DOI: 10.1109/PDP2018.2018.00090
42. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux Virtual Machine Monitor. In: Ottawa Linux Symposium (2007), DOI: 10.1186/gb-2008-9-1-r8
43. Koukis, V., Venetsanopoulos, C., Koziris, N.: ~okeanos: Building a Cloud, Cluster by Cluster. IEEE Internet Computing 17(3), 67–71 (2013), DOI: 10.1109/MIC.2013.43
44. KVM contributors: Kernel Virtual Machine. [https://www.linux-kvm.org/page/Main\\_Page](https://www.linux-kvm.org/page/Main_Page), accessed: 2018-11-29
45. von Laszewski, G., Diaz, J., Wang, F., Fox, G.C.: Comparison of Multiple Cloud Frameworks. In: 2012 IEEE Fifth International Conference on Cloud Computing. pp. 734–741. IEEE (2012), DOI: 10.1109/CLOUD.2012.104
46. Lee, M., Shin, S., Hong, S., Song, S.k.: BAIPAS: Distributed Deep Learning Platform with Data Locality and Shuffling. In: 2017 European Conference on Electrical Engineering and Computer Science (EECS). pp. 5–8. IEEE (2017), DOI: 10.1109/EECS.2017.10
47. Li, J., Wang, Q., Jayasinghe, D., Park, J., Zhu, T., Pu, C.: Performance Overhead among Three Hypervisors: An Experimental Study Using Hadoop Benchmarks. In: 2013 IEEE International Congress on Big Data. pp. 9–16. IEEE (2013), DOI: 10.1109/Big-Data.Congress.2013.11
48. Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., Crowcroft, J.: Unikernels. ACM SIGPLAN Notices 48(4), 461–472 (2013), DOI: 10.1145/2499368.2451167
49. Madhavapeddy, A., Scott, D.J.: Unikernels. Communications of the ACM 57(1), 61–69 (2014), DOI: 10.1145/2541883.2541895
50. Martin-Flatin, J.: Challenges in Cloud Management. IEEE Cloud Computing 1(1), 66–70 (2014), DOI: 10.1109/MCC.2014.4
51. Mavridis, I., Karatza, H.: Performance and Overhead Study of Containers Running on Top of Virtual Machines. In: 2017 IEEE 19th Conference on Business Informatics (CBI). pp. 32–38. IEEE (2017), DOI: 10.1109/CBI.2017.69

52. Mercl, L., Pavlik, J.: The Comparison of Container Orchestrators. In: Third International Congress on Information and Communication Technology, pp. 677–685. Springer (2019), DOI: 10.1007/978-981-13-1165-9\_62
53. Mesosphere Inc.: Marathon: A container orchestration platform for Mesos and DC/OS. <https://mesosphere.github.io/marathon/>, accessed: 2018-12-05
54. Microsoft: Windows Containers on Windows Server | Microsoft Docs. <https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-server>, accessed: 2018-11-29
55. Naik, N.: Migrating from Virtualization to Dockerization in the Cloud: Simulation and Evaluation of Distributed Systems. In: 2016 IEEE 10th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA). pp. 1–8. IEEE (2016), DOI: 10.1109/MESOCA.2016.9
56. Noor, T.H., Sheng, Q.Z., Ngu, A.H., Dustdar, S.: Analysis of Web-Scale Cloud Services. *IEEE Internet Computing* 18(4), 55–61 (2014), DOI: 10.1109/MIC.2014.64
57. NVIDIA: GPU-Enabled Docker Container | NVIDIA. <https://www.nvidia.com/object/docker-container.html>, accessed: 2018-12-01
58. NVIDIA: NVIDIA Container Runtime | NVIDIA Developer. <https://developer.nvidia.com/nvidia-container-runtime>, accessed: 2018-11-30
59. NVIDIA: nvidia-docker. <https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker>, accessed: 2018-11-29
60. NVIDIA: nvidia docker plugin. <https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker-plugin>, accessed: 2018-11-30
61. OASIS: OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC. [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=tosca](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca), accessed: 2018-11-28
62. OpenNebula Project: Home - OpenNebula. <https://openebula.org/>, accessed: 2018-12-04
63. OpenStack Foundation: OpenStack: Build the future of Open Infrastructure. <https://www.openstack.org/>, accessed: 2018-12-04
64. OpenVZ community: Open source container-based virtualization for Linux. <https://openvz.org/>, accessed: 2018-11-29
65. Oracle: Changelog for VirtualBox 4.1. <https://www.virtualbox.org/wiki/Changelog-4.1>, accessed: 2019-06-03
66. Oracle: VirtualBox. <https://www.virtualbox.org/>, accessed: 2018-11-29
67. Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K.G.: Performance Evaluation of Virtualization Technologies for Server Consolidation. HP Technical Reports (2007), DOI: 10.1.1.70.4605

68. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures-A technology review. In: Proceedings - 2015 International Conference on Future Internet of Things and Cloud, FiCloud 2015 and 2015 International Conference on Open and Big Data, OBD 2015 (2015), DOI: 10.1109/FiCloud.2015.35
69. Palit, H.N., Li, X., Lu, S., Larsen, L.C., Setia, J.A.: Evaluating hardware-assisted virtualization for deploying HPC-as-a-service. In: Proceedings of the 7th international workshop on Virtualization technologies in distributed computing - VTDC '13. p. 11. ACM Press, New York, New York, USA (2013), DOI: 10.1145/2465829.2465833
70. Pasztor, J.: LXC vs Docker. <https://pasztor.at/blog/lxc-vs-docker> (2018), accessed: 2018-02-28
71. Pivotal Software Inc.: GETTING STARTED: Building an Application with Spring Boot. <https://spring.io/guides/gs/spring-boot/>, accessed: 2018-11-29
72. Qanbari, S., Li, F., Dustdar, S.: Toward Portable Cloud Manufacturing Services. IEEE Internet Computing 18(6), 77–80 (2014), DOI: 10.1109/MIC.2014.125
73. Ranjan, R.: The Cloud Interoperability Challenge. IEEE Cloud Computing 1(2), 20–24 (2014), DOI: 10.1109/MCC.2014.41
74. Red Hat Inc.: OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes. <https://www.openshift.com/>, accessed: 2018-12-04
75. Red Hat Inc.: rkt, a security-minded, standards-based container engine. <https://coreos.com/rkt/>, accessed: 2018-11-30
76. Red Hat Inc.: WildFly. <http://wildfly.org/>, accessed: 2018-11-29
77. Reuther, A., Byun, C., Arcand, W., Bestor, D., Bergeron, B., Hubbell, M., Jones, M., Michaleas, P., Prout, A., Rosa, A., Kepner, J.: Scheduler technologies in support of high performance data analysis. In: 2016 IEEE High Performance Extreme Computing Conference (HPEC). pp. 1–6. IEEE (2016), DOI: 10.1109/HPEC.2016.7761604
78. Sabeur, Z.A., Correndo, G., Veres, G., Arbab-Zavar, B., Lorenzo, J., Habib, T., Haugomard, A., Martin, F., Zigna, J.M., Weller, G.: EO Big Data Connectors and Analytics for Understanding the Effects of Climate Change on Migratory Trends of Marine Wildlife. In: ISESS 2017: Environmental Software Systems. Computer Science for Environmental Protection. pp. 85–94. Zadar (2017), DOI: 10.1007/978-3-319-89935-0\_8
79. Sarai, A.: CVE-2019-5736: runc container breakout (all versions). <https://seclists.org/oss-sec/2019/q1/119>, accessed: 2019-03-07
80. Sempolinski, P., Thain, D.: A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science. pp. 417–426. IEEE (2010), DOI: 10.1109/CloudCom.2010.42
81. Shi, L., Chen, H., Sun, J., Li, K.: vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines. IEEE Transactions on Computers 61(6), 804–816 (2012), DOI: 10.1109/TC.2011.112

82. Shirinbab, S., Lundberg, L., Casalicchio, E.: Performance evaluation of container and virtual machine running cassandra workload. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). pp. 1–8. IEEE (2017), DOI: 10.1109/CloudTech.2017.8284700
83. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST2010 (2010), DOI: 10.1109/MSST.2010.5496972
84. Singh, R.: LXD vs Docker. <https://linuxhint.com/lxd-vs-docker/>, accessed: 2018-11-29
85. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L.: Container-based operating system virtualization. ACM SIGOPS Operating Systems Review 41(3), 275 (2007), DOI: 10.1145/1272998.1273025
86. Soualhia, M., Khomh, F., Tahar, S.: Task Scheduling in Big Data Platforms: A Systematic Literature Review. Journal of Systems and Software 134, 170–189 (2017), DOI: 10.1016/j.jss.2017.09.001
87. SUSE LLC: SUSE Linux Enterprise Server 11 SP4 Virtualization with KVM. [https://www.suse.com/documentation/sles11/singlehtml/book\\_kvm/book\\_kvm.html#cha.kvm.limits](https://www.suse.com/documentation/sles11/singlehtml/book_kvm/book_kvm.html#cha.kvm.limits), accessed: 2019-03-06
88. The Eclipse Foundation: Eclipse Jetty. <https://www.eclipse.org/jetty/>, accessed: 2018-11-29
89. The Linux Foundation: Home - Open Containers Initiative. <https://www.opencontainers.org/>, accessed: 2018-11-29
90. The Linux Foundation: Production-Grade Container Orchestration - Kubernetes. <https://kubernetes.io/>, accessed: 2018-12-04
91. The Linux Foundation: Xen Project Release Features. [https://wiki.xen.org/wiki/Xen\\_Project\\_Release\\_Features#Limits](https://wiki.xen.org/wiki/Xen_Project_Release_Features#Limits), accessed: 2019-03-06
92. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A., Martins, F., Anderson, A., Bennett, S., Kagi, A., Leung, F., Smith, L.: Intel virtualization technology. Computer 38(5), 48–56 (2005), DOI: 10.1109/MC.2005.163
93. University of Chicago: Nimbus. <http://www.nimbusproject.org/>, accessed: 2018-12-04
94. VMware: VMware Configuration Maximus. <https://configmax.vmware.com/>, accessed: 2019-03-06
95. VMware: VMware ESXi: The Purpose-Built Bare Metal Hypervisor. <https://www.vmware.com/products/esxi-and-esx.html>, accessed: 2018-11-29
96. Voras, I., Mihaljević, B., Orlić, M., Pletikosa, M., Žagar, M., Pavić, T., Zimmer, K., Čavrak, I., Paunović, V., Bosnić, I., Tomić, S.: Evaluating open-source cloud computing solutions. In: Proceedings of the 34th International Convention for Information and Communication Technology, Electronics and Microelectronics (MIPRO 2011). pp. 209–214. Opatija (2011)

97. Walraven, S., Truyen, E., Joosen, W.: Comparing PaaS offerings in light of SaaS development. *Computing* 96(8), 669–724 (2014), DOI: 10.1007/s00607-013-0346-9
98. Walters, J.P., Chaudhary, V., Cha, M., Jr., S.G., Gallo, S.: A Comparison of Virtualization Technologies for HPC. In: 22nd International Conference on Advanced Information Networking and Applications (aina 2008). pp. 861–868. IEEE (2008), DOI: 10.1109/AINA.2008.45
99. Walters, J.P., Younge, A.J., Kang, D.I., Yao, K.T., Kang, M., Crago, S.P., Fox, G.C.: GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications. In: 2014 IEEE 7th International Conference on Cloud Computing. pp. 636–643. IEEE (2014), DOI: 10.1109/CLOUD.2014.90
100. Wen, X., Gu, G., Li, Q., Gao, Y., Zhang, X.: Comparison of open-source cloud management platforms: OpenStack and OpenNebula. In: 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery. pp. 2457–2461. IEEE (2012), DOI: 10.1109/FSKD.2012.6234218
101. 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: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 233–240. IEEE (2013), DOI: 10.1109/PDP.2013.41
102. Xen Project community: Xen Project wiki: Dom0. <https://wiki.xenproject.org/wiki/Dom0>, accessed: 2018-11-29
103. Younge, A.J., Henschel, R., Brown, J.T., von Laszewski, G., Qiu, J., Fox, G.C.: Analysis of Virtualization Technologies for High Performance Computing Environments. In: 2011 IEEE 4th International Conference on Cloud Computing. pp. 9–16. IEEE (2011), DOI: 10.1109/CLOUD.2011.29
104. Zhenyun Zhuang, Cuong Tran, Weng, J., Ramachandra, H., Sridharan, B.: Taming memory related performance pitfalls in linux Cgroups. In: 2017 International Conference on Computing, Networking and Communications (ICNC). pp. 531–535. IEEE (2017), DOI: 10.1109/IC-CNC.2017.7876184