Supercomputing Frontiers and Innovations

2021, Vol. 8, No. 3

Scope

- Future generation supercomputer architectures
- Exascale computing
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Novel approaches to computing targeted to solve intractable problems
- Convergence of high performance computing, machine learning and big data technologies
- Distributed operating systems and virtualization for highly scalable computing
- Management, administration, and monitoring of supercomputer systems
- Mass storage systems, protocols, and allocation
- Power consumption minimization for supercomputing systems
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Scientific visualization in supercomputing environments
- Education in high performance computing and computational science

Editorial Board

Editors-in-Chief

Editorial Director

Associate Editors

Subject Area Editors

Technical Editors

Contents

DOI: 10.14529/jsfi210301

Evaluating Performance of Mixed Precision Linear Solvers with Iterative Refinement

Boris I. Krasnopolsky[1], *Alexey V. Medvedev*[1]

The solution of systems of linear algebraic equations is among the time-consuming problems when performing the numerical simulations. One of the possible ways of improving the corresponding solver performance is the use of reduced precision calculations, which, however, may affect the accuracy of the obtained solution. The current paper analyzes the potential of using the mixed precision iterative refinement procedure to solve the systems of equations occurring as a result of the discretization of elliptic differential equations. The paper compares several inner solver stopping criteria and proposes the one allowing to eliminate the residual deviation and minimize the number of extra iterations. The presented numerical calculation results demonstrate the efficiency of the adopted algorithm and show about the decrease in the solution time by a factor of 1.5 for the turbulent flow simulations when using the iterative refinement procedure to solve the corresponding pressure Poisson equation.

Keywords: systems of linear algebraic equations, elliptic equations, algebraic multigrid methods, iterative refinement, mixed precision calculations.

Introduction

The solution of systems of linear algebraic equations (SLAEs) is a typical task when performing most of the high performance computing-related applications. This issue motivates the researchers to both develop novel mathematical algorithms for solving SLAEs and improve the implementation aspects to provide better correspondence between the numerical methods and modern compute platforms hardware.

The use of reduced precision floating-point numbers [4, 8, 14] or lower size integer numbers [19] is a well-known way to improve the performance of the calculations, which is discussed in the literature for some time. The reduced precision floating-point calculations can be beneficial for both memory-bound and compute-bound applications [18]. Decreasing the size of the data types allows decreasing the amount of memory traffic. Besides, multiple more floating-point arithmetic operations can be performed per each CPU processor cycle for the vectorized code sections when performing the calculations.

A new wave of interest in the topic of mixed precision calculations is associated with the emergence of the compute devices bringing the hardware support for the half-precision floating-point numbers (e.g., NVIDIA Pascal and Volta GPUs). While the driving force of this innovation is related to machine learning applications, the potential of fold calculations speed up compared to the double precision calculations raised the interest in many other areas including numerical modeling. For example, the use of mixed precision calculations is among the key directions of improving the performance for the exascale systems and computational codes [2]. The publication [2] reviews the popular linear algebra algorithms and libraries of numerical methods capable of using mixed precision calculations and highlights the current achievements and future research trends.

The iterative methods for solving large sparse SLAEs, which this paper focuses on, belong to the memory-bound applications with very low compute intensity of the order $O(10^{-1})$ [7, 18]. Thus, the use of mixed precision can be an attractive way to increase the compute intensity

[1] Lomonosov Moscow State University, Moscow, Russian Federation

and significantly improve the performance of the calculations. Reducing the precision of the floating-point numbers for the whole linear solver, however, may lead to a significant iterative process convergence rate degradation or even to divergence. The potential of using the single (or even lower) precision calculations depends on the properties of the specific SLAE, but for many practical applications, like structural mechanics problems or incompressible turbulent flow calculations, the use of single-precision for the whole solver is unacceptable.

The compromise and widely accepted variant is the use of reduced precision floating-point calculations when performing the preconditioning while preserving operations with the SLAE matrix and solution vector in the basic precision. This approach does not affect the resulting solution tolerance and is more robust compared to performing the whole solver with reduced precision. However, a significant portion of calculations is still performed with the basic precision, which reduces significantly the potential calculations speedup.

An alternative way of utilizing the mixed precision calculations is the use of the iterative refinement procedure [17]. Initially developed in the 1940s, this method receives great attention in the light of recent activity with introducing the mixed precision calculations in iterative methods for solving SLAEs [3, 5]. These publications show promising results with attractive speedup numbers. The conjugate gradient (CG) method with adaptive geometric multigrid preconditioner was considered by the developers of the QUDA library, and the combination of the top-level 64-bit solver with inner 32-bit CG solver and 16-bit preconditioner was analyzed. The proposed methods combination allowed to speed up the corresponding SLAE solver for NVIDIA Volta GPUs compared to the basic solver performed in double precision by a factor of 4–6. In [3] the authors investigated the combination of CG solver with a lightweight plain Jacobi (diagonal) preconditioner and the top-level iterative refinement procedure. The outer solver calculations were performed in double precision, and the preconditioned CG solver was performed in the single precision. The solver was evaluated with 123 SLAEs, and in about 60 % of the cases, the total energy consumption improvement by a factor of 1.1–1.8 was demonstrated compared to the preconditioned CG solver performed in double precision.

While the results mentioned above show good potential in using mixed precision calculations, the numerical methods and test matrices considered have their specific, which hardens the conclusion on the applicability of the methods proposed to the wider range of applications. Additionally, the energy consumption results presented in [3] can only give a rough estimate to the real calculation time improvement by using the mixed precision iterative refinement procedure. The current paper focuses on the applicability of mixed precision calculations to solve the elliptic equations, and, specifically, the pressure Poisson equation occurring in incompressible flow simulations. For example, high-fidelity turbulent flow simulations require multiple solutions of SLAEs, and the corresponding time to solve these SLAEs typically prevails in the overall calculation time. This aspect motivates the further tuning of numerical methods applicable for solving the corresponding SLAEs, and the use of iterative refinement with reduced precision solvers can be an attractive option.

The rest of the paper is organized as follows. The first section outlines the potential speedup due to the use of reduced precision calculations. The second section presents the iterative refinement procedure and highlights the main algorithm pitfalls. Various residual replacement strategies used to resolve some of these problems are discussed. The description of the XAMG library used in the numerical experiments to demonstrate the efficiency of the iterative refinement procedure is presented in the third section. The fourth section discusses the results of the numerical experiments and the evaluation of the mixed precision iterative refinement procedure for modeling incompressible turbulent flows. The conclusion section summarizes the paper.

1. Potential of Mixed Precision Calculations

Before discussing the aspects of the iterative refinement procedure, it is important to estimate the potential performance gain due to reducing the precision of floating-point calculations. The object of interest is the algorithm combining the Krylov subspace method (specifically, BiCGStab [15]) with the classical algebraic multigrid preconditioner and Gauss-Seidel smoother. This combination of the methods represents a robust and scalable solver, which can be effectively used to solve various SLAEs, including the ones derived from the elliptic partial differential equations.

The methods mentioned above consist of the combination of linear operations with vectors and matrix-vector multiplications. Accounting that both of these operations are memory-bound, the expected calculation time reduction would be in proportion with the corresponding reduction in the amount of memory accesses. The overall execution time, T_{exec}, is a sum of vector operations time, T_{vec}, and matrix-vector multiplications time, T_{mat}:

$$T_{exec} = T_{vec} + T_{mat} = \frac{\Sigma_{vec} + \Sigma_{mat}}{B}, \tag{1}$$

where Σ is the amount of memory traffic when performing the corresponding operations, and B is the memory bandwidth of the compute system.

Reduction of the floating-point data size, e.g., from 8 to 4 bytes, leads to a twofold decrease of the amount of memory traffic for the vector operations and the corresponding calculation time. The matrix-vector operations also demonstrate significant calculation speedup, but it is typically less than by a factor of 2 due to the use of specialized sparse matrix storage formats and the need to store additional integer indices. For example, for the CSR data storage format [12], the overall amount of memory traffic when performing matrix-vector multiplication can be represented as:

$$\Sigma_{mat} = n\,I(C+1) + n\,F(2C+1). \tag{2}$$

Here n is the matrix size, I and F are the sizes of the integer and floating-point data types respectively, and C is the average number of nonzero elements per each matrix row. The ratio of the amount of data for 8 and 4 byte floating-point numbers shows the potential of the reduced precision matrix-vector operations speedup (expecting here that I is equal to 4 bytes):

$$P = \frac{n\,4(C+1) + n\,8(2C+1)}{n\,4(C+1) + n\,4(2C+1)} = \frac{5C+3}{3C+2}. \tag{3}$$

In the typical case $C \gg 1$ the expression above reduces to $P \approx 1.67$.

Using the expression (1), one can expect the overall speedup due to using the reduced precision floating-point calculations. Generally, the speedup lies in the range of 1.7–1.9 in case the overall convergence rate is not affected by the round-off errors. The use of mixed precision calculations for the preconditioner only further decreases the potential speedup, as only a portion of vector and matrix-vector operations in (1) is performed with reduced precision. In practice, this speedup depends on the size of the multigrid matrix hierarchy and can be roughly estimated in the range of 5–30 %.

Algorithm 1 Mixed precision iterative refinement algorithm.

```
1: $x_0$ – initial guess        (basic precision)
2: $r_0 = b - Ax_0$             (basic precision)
3: $k = 0$
4: while not converged do
5:     $r_k \to \hat{r}_k$                  (convert to reduced precision)
6:     solve $\hat{A}\hat{y}_k = \hat{r}_k$      (reduced precision)
7:     $\hat{y}_k \to y_k$                  (convert to basic precision)
8:     $x_{k+1} = x_k + y_k$          (basic precision)
9:     $r_{k+1} = b - Ax_{k+1}$       (basic precision)
10:    $k = k + 1$
11: end while
```

The iterative refinement procedure, discussed in detail in the sections below, requires some additional calculations compared to the pure 32-bit solver. Even in case the SLAE solver is not suffering the convergence rate degradation due to lower precision calculations, the expected speedup will be lower than for the 32-bit solver. In the negative scenario, the overall calculation time can be even higher than that for the basic precision solver.

Summarizing the discussion above, the following conclusions can be stated:

- performance of iterative refinement procedure combining 64-bit and 32-bit floating-point calculations is expected to be lower than for the pure 32-bit solver and to not exceed the range of 1.7–1.9;
- mixed precision iterative refinement procedure can be found useful in case the obtained calculations speedup will exceed the one for the method configuration with the preconditioner only performed with reduced precision.

2. Iterative Refinement Procedure

The main idea of the mixed precision iterative refinement procedure is to combine two iterative algorithms, the inner iterative method operating with reduced precision and the outer method operating with the basic precision. The mixed precision iterative refinement algorithm is presented in Algorithm 1. The calculations start with computing the initial residual vector r in basic precision (line 2). The residual is converted to lower precision (line 5) and the reduced precision solver is used to get the solution update $\hat{y}$ (line 6). The obtained vector is converted back to basic precision (line 7) and accumulated with solution vector x (line 8). Finally, the residual vector is recalculated in basic precision (line 9) and is used to control the overall convergence.

This rather simple algorithm, however, has several pitfalls when used in practice:

1. The outer loop of the iterative refinement procedure performs some additional calculations, like the calculation of residual vector, requiring matrix-vector multiplication, vector updates, and floating-point data conversion. These operations provide some overhead compared to the original solver performed in basic precision, and the number of outer iterations should be minimized to minimize the corresponding overhead.
2. To compete with the reduced precision preconditioner calculations, the number of inner solver iterations should not exceed 20–30 %. For the iterative methods considered in this paper the typical number of iterations of the basic solver is in the range 10–20. This means,

an increase in the cumulative number of inner solver iterations in the range 3–5 is permissible, otherwise the use of iterative refinement procedure will lead to the calculations slowdown.

3. The Krylov subspace iterative methods (e.g., CG, BiCGStab) use recurrent expressions to calculate the residual vector at the next iteration. The long recurrent chains are sensitive to the accumulation of the round-off errors, and as a result, the deviation of this residual from the true one, $r = b - Ax$, may be observed. This issue can lead to the situation when the convergence of the recurrent residuals will be accompanied by the true residuals stagnation. The use of reduced precision calculations for the inner solver can even complicate the situation, as the calculations with reduced precision can accelerate the accumulation of the round-off errors.

The problem of residuals deviation for the Krylov subspace iterative methods has been analyzed in the literature, e.g., in [3, 13, 16]. The observed convergence issue in some cases can be resolved by the correction of the recurrent residuals, which can be done in several ways. In [13, 16] the authors propose the reliable updates to correct the recurrent residual with the true one and several criteria when these updates must be performed. Alternatively, in [3] the inner solver restarts are considered for the mixed precision iterative refinement algorithm. Following [3], the proper choice of the conditions to perform the restarts allows this approach to outperform the reliable updates strategy.

It should be noted that the publications mentioned above use for the numerical evaluation the lightweight iterative methods with the simple preconditioner or even without it. The typical number of iterations till convergence in the presented results is counted by at least several hundred, and several additional residual checks or extra iterations do not play a significant role. The current paper focuses on the algorithms with robust preconditioners performing typically 10–20 iterations. In that case, the penalty for the several extra iterations can be critical, and the applicability of the corresponding residual replacement algorithms must be analyzed in detail. To the best of our knowledge, there are no publications discussing the applicability of the mixed precision iterative refinement algorithms with robust and fast converging iterative methods.

3. XAMG Library

The performance evaluation results presented in this paper are performed with the XAMG library [9, 10]. The XAMG library is a novel open-source project which implements sparse linear algebra subroutines and solvers in modern C++. The library provides the implementation of a set of widely used numerical methods for solving large sparse SLAEs, including the Krylov subspace iterative methods, classical algebraic multigrid method, and others. The XAMG library focuses on the optimized implementation for the solve part of the methods and reuses the *hypre* library [1] for the multigrid hierarchy matrices construction.

The library design feature is a template-based generalization of basic objects and subroutines. This feature makes it possible to implement a few important design principles. First, the specialization of the number of right-hand side vectors as a template parameter allows for a generalization of the subroutines for an arbitrary number of right-hand sides in the solution. The specialization of the generalized subroutines takes place automatically in compile time, and the automated vectorization of loops is being done without additional effort. Second, all the data types for base library objects are also specialized with template parameters, so the features like the advanced sparse matrix formats and reduced precision storage options can be implemented with ease.

The library consists of four major groups of elements: (i) matrices and vectors data structures, representing basic data objects; (ii) basic sparse linear algebra subroutines gathered in “blas” and “blas2” groups; (iii) solver classes inherited from an abstract interface, which exposes the main library function “solve()”; (iv) solver parameter classes. The solver classes are arranged in a structure allowing easy addition of the new numerical methods. The solvers can be combined with each other as well. The established solver parameters subsystem supports the solver code extension and code combinations.

An important design feature of the XAMG library is the hierarchical hybrid parallel programming model. This model, called MPI+ShM, is integrated into basic data objects and basic subroutines of the library. The MPI+ShM model implies that MPI ranks, which appear to be placed on the same compute node, allocate and use the common POSIX shared memory regions to store the vector data structures. The communication and data access within a single compute node is performed using these shared memory regions. This hybrid approach to parallel programming improves the productivity and scalability of basic library subroutines compared to a pure MPI-only parallel programming model. It makes it possible to get leading scalability figures on modern multicore and manycore HPC systems.

To summarize, the XAMG library is a flexible tool to implement new iterative methods and their combinations and do experiments with their traits and parameters. On the other hand, the MPI+ShM hybrid model ensures the good productivity and scalability of the resulting code. It makes the XAMG library an attractive platform for the research made in this work.

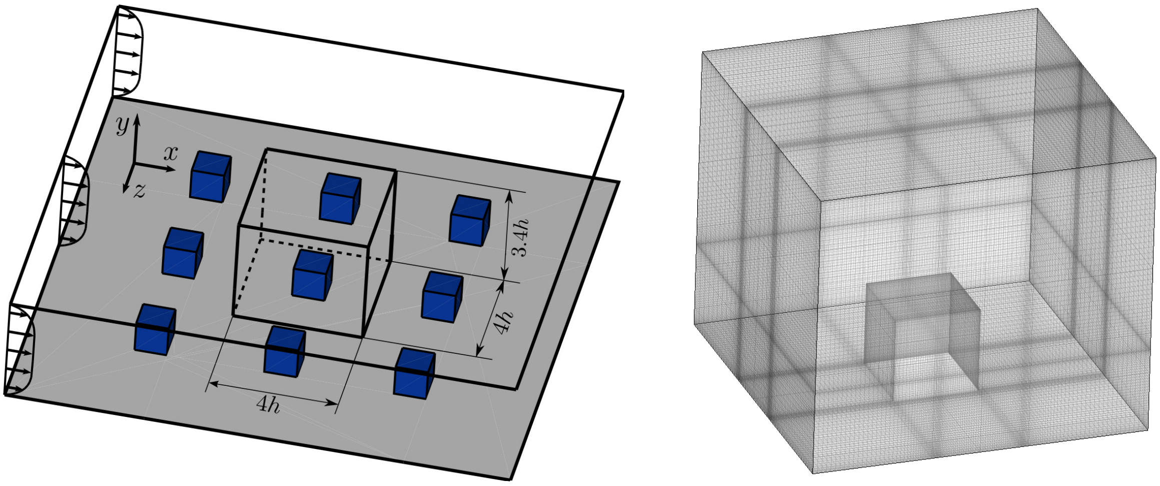
4. Numerical Experiments

The current section presents results of the numerical experiments investigating the applicability of the mixed precision iterative refinement procedure to accelerate the solution of the SLAEs, and, specifically, the ones obtained as a result of the spatial discretization of the pressure Poisson equation when modeling incompressible turbulent flows. The investigation is performed for the SLAE solver comprising the BiCGStab method with algebraic multigrid preconditioner and symmetric Gauss-Seidel smoother. The four solver configurations with the same numerical method configurations are considered. The *basic* solver configuration performs all the calculations with double precision. The *mixed* solver configuration assumes the use of reduced precision for the preconditioning when operating with the constructed multigrid matrices hierarchy. The *IR* solver utilizes the mixed precision iterative refinement procedure with the inner solver, corresponding to the *basic* solver, but performed in the single precision. Finally, the *reduced* solver configuration performs all the calculations with single precision. All the numerical experiments are performed on the Lomonosov-2 supercomputer (compute nodes with single 14-core Intel E5-2697v3 processor); the calculation runs utilize all available 14 physical CPU cores per node.

4.1. Testing Methodology

The evaluation of different approaches of introducing the reduced precision calculations in the iterative SLAE solvers is performed in several steps. The first test series confirms the potential speedup limits by performing the same calculations in double precision (*basic*), in single precision (*reduced*), and for the double-precision solver with reduced precision preconditioning calculations (*mixed*). The various inner solver stopping criteria, affecting the frequency of residual replacement in the mixed precision iterative refinement algorithm (*IR*) and the corresponding solver convergence rate, are investigated in the second test series. The third group of tests compares the convergence rates and calculation times for *basic*, *mixed*, and *IR* solver configurations. Finally, the turbulent flow simulations are performed with various solver configurations to demonstrate the potential decrease in the solution time in the real calculations.

The two groups of test SLAEs are used in the tests. The first one corresponds to the discretization of the 3D Poisson equation in the cubic domain ("cube") with the uniform grid and constant right-hand side vector. The size of the grid varies in the range of 100^3–500^3. The second group includes the two test systems corresponding to the pressure Poisson equation in the computational domain used for the simulation of the turbulent flow in a channel with a matrix of wall-mounted cubes [6] ("channel with cube"; the geometry of the computational domain and the corresponding computational grid are presented in Fig. 1); the size of the grids is equal to 2.3 and 9.7 million unknowns. The right-hand side vectors are chosen corresponding to the physical problem statement, and they are defined as the divergence of the random velocity fields. Both groups of the test matrices are available with the internal data generator of the integration test application provided with the XAMG library [10].



(a) Computational domain

(b) Computational grid

Figure 1. Sketch of the computational domain and the computational grid

4.2. Performance Tests

4.2.1. Performance limitations

The performance evaluation starts with the numerical experiments, demonstrating the real calculation time reduction with mixed precision calculations and the calculations with the solver operating in single precision. This test series performs the constant number of iterations and focuses on the calculation time reduction as an upper bound estimate for the calculation results that can be observed with the iterative refinement procedure (in case the convergence rate is not affected by the reduced precision calculations).

The calculations presented in this section are performed with the single node of the Lomonosov-2 supercomputer for the "cube" test system with 150^3 unknowns and the "channel with cube" system with 2.3 million unknowns, and with 4 compute nodes for the "cube" system with 500^3 unknowns. The obtained calculation results are summarized in Tab. 1. These include the single iteration times and the corresponding speedup compared with the *basic* solver configuration performed in double precision. The speedup by a factor of about 1.1 can be expected for the *mixed* solver configuration and by a factor of 1.65 – for the *reduced* solver configuration operating with 32-bit floating-point numbers. The results presented are similar for all test matrices considered and are in agreement with the proposed theoretical estimates (see, section 1).

The obtained calculation times are also compared with the ones for the well-known open-source *hypre* library, which allows using the same numerical method configurations as in XAMG. The *hypre* library does not provide the functionality to utilize the mixed precision calculations, thus only double precision calculations are performed, which correspond to the *basic* configuration of the XAMG library. The results presented in Tab. 1 demonstrate an advantage of the XAMG library compared to *hypre*: the decrease in the execution time is about 10–15 %. Some more details on comparing the performance of the XAMG and *hypre* libraries can be found in [9].

Table 1. The single iteration calculation times for the XAMG and *hypre* libraries and the corresponding decrease in the solution time compared to the double precision solver (speedup) for different solver configurations and test matrices

Test case	*basic*	*mixed*		*reduced*		*hypre*
	time, s	time, s	speedup	time, s	speedup	time, s
Cube, 150^3	0.151	0.143	1.06	0.091	1.66	0.170
Cube*, 500^3	1.65	1.51	1.09	1.01	1.64	1.91
Channel with cube, 2M	0.123	0.109	1.12	0.076	1.62	0.139

*The corresponding calculations are performed with 4 compute nodes

4.2.2. Residual replacement

The choice of the inner solver stopping criteria is among the key practical aspects affecting the usability of the mixed precision iterative refinement procedure. The current paragraph analyzes and compares some of them, including the periodic restarts and the explicit residual replacement [3]. The corresponding experiments are performed for two test matrices, the "cube" case with 150^3 unknowns and the "channel with cube" case with 2.3 million unknowns and investigate 8 different combinations of the parameters responsible for stopping the inner solver and residual replacement. The frequency of the periodic restarts is chosen according to the typical number of iterations till convergence, which varies in the range 10–20: the experiments with restarts every 1, 2, 3, and 5 iterations are performed. The explicit residual deviation checks once in $t = 100$ iterations with the allowed deviation ratio $\gamma = 10$ were used in [3]. In our case, these values are inapplicable because the periodicity of the residual checks outnumbers significantly the expected number of iterations till convergence. Alternatively, the residual deviation checks every $t = 2$ and 3 iterations with the allowed deviation ratio $\gamma = 1.01$ and 2 are considered. The decrease of the initial residual norm by a factor of $\varepsilon = 10^{-8}$ is used as the convergence criterion.

The obtained calculation results are presented in Tab. 2. The restarts for the inner solver performed every iteration lead to the convergence degradation due to the loss of the information about the already constructed Krylov subspace basis. An increase in the number of inner solver iterations leads to the overall solver convergence acceleration. However, reduction of the frequency of restarts strengthens the residual deviation effect, and for the case with restarts once in 5 iterations, the overall number of iterations starts to increase. Summarizing the results presented in the table, the frequency of restarts every 2–3 iterations can be outlined as the one providing the best convergence rate for the SLAEs and numerical method configurations considered.

The explicit residual deviation check assumes explicit control of the deviation of the true and recurrent residuals every t iterations and the inner solver restart only in case the residual deviation exceeds the limiting value γ. The presented calculation results show that the use of a high deviation ratio is inefficient: the higher value γ only tightens the inner solver restart, while performing the inner solver with even slightly deviated residuals reduces the solver efficiency in terms of the true residual decay. The decrease of the deviation value γ allows reducing in some cases the overall number of iterations, and, consequently, the calculation time. However, the obtained calculation times with explicit residual deviation checks are systematically higher than the ones for the periodic restarts approach, which provides a simple and efficient way to avoid the residual deviation issue and convergence rate degradation.

Table 2. Comparison of various inner solver stopping criteria

Stopping criteria		Cube, 150^3		Channel with cube, 2M	
Method	Parameters	Time, s	Iter.	Time, s	Iter.
Periodic restart	$t = 1$	1.54	15 (15)	1.01	12 (12)
	$t = 2$	1.14	6 (12)	0.79	5 (10)
	$t = 3$	1.13	4 (12)	0.79	4 (10)
	$t = 5$	1.12	3 (12)	0.86	3 (11)
Explicit residuals check	$t = 2, \gamma = 2$	1.32	2 (14)	1.04	3 (13)
	$t = 3, \gamma = 2$	1.22	2 (13)	1.09	3 (14)
	$t = 2, \gamma = 1.01$	1.24	3 (13)	0.88	3 (11)
	$t = 3, \gamma = 1.01$	1.22	2 (13)	0.87	3 (11)

4.2.3. Single SLAE tests

The next step of performance evaluation focuses on the investigation of various solver configurations with a set of test matrices. The three solver configurations (*basic*, *mixed*, and *IR* solvers) with four “cube” SLAEs and two “channel with cube” SLAEs are analyzed. The *IR* solver is performed with the inner solver periodic restarts every 3 iterations.

The overall solution times and the number of iterations (for the *IR* solver – the number of outer iterations and the cumulative number of inner iterations) are collected in Tab. 3. For all the test cases considered the observed calculation results, in general, demonstrate similar behavior. The *basic* and *mixed* solver configurations converge in the same number of iterations: the use of reduced precision for the multigrid hierarchy preconditioning calculations does not affect the overall convergence rate. The speedup for the *mixed* solver varies in the range 1.06–1.13, and these values correspond to the basic theoretical estimates. The *IR* solver demonstrates even faster convergence than the *basic* one, decreasing the number of iterations by 1–3. The reduction in the number of iterations for the *IR* solver on par with the lower calculation time for the inner solver performed in reduced precision allows obtaining the significant speedup for all the test systems considered, which varies in the range 1.67–1.88.

Table 3. The calculation times, the number of iterations, and the achieved relative speedup for the several test matrices and solver configurations

Test case	*basic*		*mixed*			*IR*		
	time, s	iter.	time, s	iter.	speedup	time, s	iter.	speedup
Cube, 100^3	0.53	13	0.50	13	1.06	0.30	4 (11)	1.74
Cube, 150^3	2.11	14	1.99	14	1.06	1.12	4 (12)	1.88
Cube, 200^3	5.69	15	5.34	15	1.07	3.10	5 (13)	1.83
Cube*, 500^3	39.5	24	36.3	24	1.09	22.0	7 (21)	1.80
Channel with cube, 2M	1.35	11	1.20	11	1.13	0.79	4 (10)	1.71
Channel with cube*, 9M	2.12	14	1.88	14	1.13	1.27	5 (13)	1.67

*The corresponding calculations are performed with 4 compute nodes

The calculation times presented in Tab. 3 show a stable tendency in decreasing the number of iterations for the *IR* solver configuration. Despite the use of lower precision for the inner solver calculations, the corresponding number of iterations till convergence is decreased in all the cases considered by about 10–15 %. To investigate the reasons for this effect, the same calculations were repeated for the double precision solver with residual replacement. This test series reproduced the same effect of reducing the number of iterations for the basic solver with residual replacement, observed for the *IR* solver configuration (Tab. 4). Despite an additional amount of calculations to perform the residual replacement and solver restarts, the use of this technique can be advantageous even for double precision calculations.

Table 4. The calculation times and the number of iterations for the *IR* and double precision and solver configurations with residual replacement

Test case	*basic + RR*		*IR*	
	time, s	iterations	time, s	iterations
Cube, 100^3	0.46	4 (11)	0.30	4 (11)
Cube, 150^3	1.87	4 (12)	1.12	4 (12)
Cube, 200^3	5.11	5 (13)	3.10	5 (13)
Cube*, 500^3	35.5	7 (21)	22.0	7 (21)
Channel with cube, 2M	1.26	4 (10)	0.79	4 (10)
Channel with cube*, 9M	1.85	4 (12)	1.27	5 (13)

*The corresponding calculations are performed with 4 compute nodes

4.2.4. Turbulent flow calculations

The results presented in the previous paragraph demonstrate attractive speedup for the mixed precision iterative refinement procedure which can be easily incorporated into practical applications. The final stage of the performance evaluation considers the test runs for the direct numerical simulation of incompressible turbulent flow with three solver configurations and shows the practical calculations speedup.

The turbulent flow simulations are performed with in-house computational code, based on the second order in space and third order in time computational algorithm, operating with curvilinear orthogonal coordinates [11]. This algorithm requires three solutions of pressure Poisson equation per each time step; the SLAE matrix remains unchanged during the calculations allowing to perform the construction of the multigrid matrix hierarchy only once at the initialization stage. The calculations are performed for the test case of modeling the turbulent flow in a channel with a matrix of wall-mounted cubes, described in subsection 4.1, on the grid with 2.3 million unknowns. The test runs are performed with 8 compute nodes and model short time interval of $T = 3$ time units, which corresponds to about 1100 time steps.

The results of the three test runs are outlined in Tab. 5. These include the linear solver calculation times and the overall simulation time. The simulation with the *basic* solver configuration takes 486 seconds, where 455 seconds are spent with the SLAE solver. The cumulative number of iterations performed by the linear solver is about 30000. The run with the *mixed* precision solver takes 452 seconds and shows the speedup by a factor of 1.08. The convergence rate of the linear solver remains unaffected by the reduction of the preconditioner calculations precision – the number of iterations to solve the corresponding SLAEs is about the same as for the *basic* solver configuration. The lowest calculation times are observed with the *IR* solver configuration. The overall simulation time takes 323 seconds, which is 1.5 times faster than the one with the *basic* solver configuration. The linear solver fraction of the calculation time is reduced by a factor of 1.56 and takes 292 seconds. The *IR* solver requires about 9800 iterations for the outer solver and about 26000 iterations for the inner solver. The overall number of iterations for the *IR* solver is about 10 % less than for the *basic* solver, which indicates that the regular residual replacement allows to reduce the round-off errors and increase the convergence rate.

Table 5. The calculation times, the total number of iterations, and the achieved speedup for the direct numerical simulation of turbulent flow performed with several linear solver configurations

Stage	*basic*		*mixed*			*IR*		
	time, s	iter.	time, s	iter.	speedup	time, s	iter.	speedup
SLAE solver	455	29978	421	29982	1.08	292	9816 (25687)	1.56
Total	486		452		1.08	323		1.50

Conclusions

The paper investigates the efficiency of the several approaches of introducing the reduced precision calculations for solving large sparse systems of linear algebraic equations, and, specifically, the ones occurring when solving the elliptic differential equations. These include the use of reduced precision calculations for the preconditioning and the mixed precision iterative refinement procedure, combining the outer solver performed with basic precision and the inner solver operating with reduced precision. The paper reviews the examples of using the mixed precision iterative refinement procedure known in the literature and significantly extends them with the obtained calculation results for the numerical methods with robust preconditioners. The various residual replacement strategies are discussed and the optimal one for the test cases considered is proposed.

The results of the numerical experiments performed for several test matrices are presented. These results show that the use of reduced precision calculations for the preconditioner allows decreasing the solution time by only about 10 %, while the use of mixed precision iterative refinement procedure with the proper choice of the residual replacement strategy can provide the SLAE solver speedup by a factor of 1.8. The proposed residual replacement algorithm is examined by performing the calculations of incompressible turbulent flow. The several runs performed for the direct numerical simulation of the turbulent flow in a channel with a matrix of wall-mounted cubes demonstrate about 1.5 times decrease in the solution time due to acceleration of the pressure Poisson equation SLAE solver when using the mixed precision iterative refinement procedure.

Acknowledgements

The presented work is supported by the RSF grant No. 18-71-10075. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. HYPRE: High performance preconditioners (2020). `http://www.llnl.gov/CASC/hypre/`, accessed: 2020-12-27

2. Abdelfattah, A., Anzt, H., Boman, E.G., et al.: A survey of numerical linear algebra methods utilizing mixed-precision arithmetic. The International Journal of High Performance Computing Applications 35(4), 344–369 (2021). `https://doi.org/10.1177/10943420211003313`

3. Anzt, H., Flegar, G., Novaković, V., et al.: Residual replacement in mixed-precision iterative refinement for sparse linear systems. In: Yokota, R., Weiland, M., Shalf, J., Alam, S. (eds.) High Performance Computing. pp. 554–561. Springer, Cham (2018). `https://doi.org/10.1007/978-3-030-02465-9_39`

4. Baboulin, M., Buttari, A., Dongarra, J., et al.: Accelerating scientific computations with mixed precision algorithms. Computer Physics Communications 180(12), 2526–2533 (2009). `https://doi.org/10.1016/j.cpc.2008.11.005`

5. Clark, M., Babich, R., Barros, K., et al.: Solving lattice QCD systems of equations using mixed precision solvers on GPUs. Computer Physics Communications 181(9), 1517–1528 (2010). `https://doi.org/10.1016/j.cpc.2010.05.002`

6. Krasnopolsky, B.: An approach for accelerating incompressible turbulent flow simulations based on simultaneous modelling of multiple ensembles. Computer Physics Communications 229, 8–19 (2018). `https://doi.org/10.1016/j.cpc.2018.03.023`

7. Krasnopolsky, B.: Revisiting performance of BiCGStab methods for solving systems with multiple right-hand sides. Computers & Mathematics with Applications 79(9), 2574–2597 (2020). https://doi.org/10.1016/j.camwa.2019.11.025

8. Krasnopolsky, B., Medvedev, A.: Acceleration of large scale OpenFOAM simulations on distributed systems with multicore CPUs and GPUs. In: Parallel Computing: On the Road to Exascale. Advances in Parallel Computing, vol. 27, pp. 93–102 (2016). https://doi.org/10.3233/978-1-61499-621-7-93

9. Krasnopolsky, B., Medvedev, A.: XAMG: A library for solving linear systems with multiple right-hand side vectors. SoftwareX 14 (2021). https://doi.org/10.1016/j.softx.2021.100695

10. Krasnopolsky, B., Medvedev, A.: XAMG: Source code repository (2021). https://gitlab.com/xamg/xamg, accessed: 2021-03-31

11. Nikitin, N.: Finite-difference method for incompressible Navier-Stokes equations in arbitrary orthogonal curvilinear coordinates. Journal of Computational Physics 217, 759–781 (2006). https://doi.org/10.1016/j.jcp.2006.01.036

12. Saad, Y.: Iterative methods for sparse linear systems, 2nd edition. SIAM, Philadelpha, PA (2003)

13. Sleijpen, G.L.G., van der Vorst, H.A.: Reliable updated residuals in hybrid Bi-CG methods. Computing 56, 141–163 (1996). https://doi.org/10.1007/BF02309342

14. Sumiyoshi, Y., Fujii, A., Nukada, A., Tanaka, T.: Mixed-precision AMG method for many core accelerators. In: 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14. pp. 127–132. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2642769.2642794

15. van der Vorst, H.A.: BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 13(2), 631–644 (1992). https://doi.org/10.1137/0913035

16. van der Vorst, H.A., Ye, Q.: Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals. SIAM Journal on Scientific Computing 22(3), 835–852 (2000). https://doi.org/10.1137/S1064827599353865

17. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Englewood Cliffs, Prentice-Hall, New Jersey (1963)

18. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Communications of the ACM 52(4), 65–76 (2009). https://doi.org/10.1145/1498765.1498785

19. Zarechnev, S., Krasnopolsky, B.: Improving performance of linear solvers by using indices compression for storing sparse matrices. In: Voevodin, V. (ed.) Russian Supercomputing Days: Proceedings of the International Conference. pp. 119–120. MAKS Press, Moscow (2020). https://doi.org/10.29003/m1406.RussianSCDays-2020

DOI: 10.14529/jsfi210302

Fog Computing State of the Art: Concept and Classification of Platforms to Support Distributed Computing Systems

Alexandra A. Kirsanova[1], *Gleb I. Radchenko*[1], *Andrei N. Tchernykh*[1,2,3]

As the Internet of Things (IoT) becomes a part of our daily life, there is a rapid growth in the connected devices. A well-established approach based on cloud computing technologies cannot provide the necessary quality of service in such an environment, particularly in terms of reducing data latency. Today, fog computing technology is seen as a novel approach for processing large amounts of critical and time-sensitive data. This article reviews cloud computing technology and analyzes the prerequisites for the evolution of this approach and the emergence of the concept of fog computing. As part of an overview of the critical features of fog computing, we analyze the frequent confusion of the concepts of fog and edge computing. We provide an overview of fog computing technologies: virtualization, containerization, orchestration, scalability, parallel computing environments, as well as systematic analysis of the most popular platforms that support fog computing. As a result of the analysis, we offer two approaches to classification of the fog computing platforms: by the principle of openness/closure of components and by the three-level classification based on the provided platform functionality (Deploy-, Platform- and Ecosystem as a Service).

Keywords: big data processing, fog computing, scheduling, cloud computing, edge computing, Internet of Things.

Introduction

Data is a major commodity today. Having more data and the ability to intelligently analyze it effectively creates significant value for data-managed enterprises [40]. According to the International Data Corporation (IDC), the amount of digital data generated in 202 exceeded 59 zettabytes (ZB) of data [5]. Cisco estimated that there will be about 50 billion connected devices in 2020 [27]. These connected devices form the Internet of Things (IoT) and generate a vast amount of data in real-time. Modern mobile networks are already being designed considering the loads that arise in the transmission and processing of such astronomical volumes of data.

Within the cloud computing concept, most of the data that requires storage, analysis, and decision making is sent to data centers in the cloud [74]. As the data volume increases, moving information between an IoT device and the cloud may be inefficient or even impossible in some cases due to bandwidth limitations or latency requirements. As time-sensitive applications (such as patient monitoring, autopilot vehicles, etc.) become more common, the remote cloud will not be able to meet the need for ultra-reliable communications with minimal delay [99]. Moreover, some applications may not be able to send data to the cloud because of privacy issues.

To solve the challenges of applications that require high network bandwidth, access to geographically distributed data sources, ultra-low latency, and localized data processing, there is a specific need for a computing paradigm that provides a one-size-fits-all approach to the organization of computing, both in the cloud and in computing nodes closer to connected devices. The concept of Fog computing has been proposed by industry and academia to bridge the gap

[1] South Ural State University, Chelyabinsk, Russia

[2] CICESE Research Center Ensenada, Mexico

[3] Ivannikov Institute for System Programming of the RAS, Russia

between the cloud and IoT devices by providing computing capabilities, storage, networking, and data management at network nodes closely located to IoT devices [15, 68]. The research community has proposed several computing paradigms to address these problems, such as edge computing, fog computing, and dew computing. A common feature of these concepts is the use of distributed heterogeneous systems that provide highly scalable clusters of computing nodes located closely (either networked or geographically) to data sources. In this review, we provide an analysis of the most popular platforms that support fog computing solutions. Based on this analysis, we propose two approaches to classify fog computing platforms: by the principle of openness/closure of components and by the three-tier classification based on the provided platform functionality (Deploy-, Platform- and Ecosystem as a Service).

The article is organized as follows. Section 1 discusses cloud computing as the basis for new computing concepts, prerequisites for the emergence, and key characteristics of cloud computing. Section 2 is devoted to fog and edge computing, their origins, definition, and critical characteristics. Section 3 discusses technologies that support fog computing, including virtualization, orchestration, security, and computation scalability issues. Section 4 provides an overview of fog computing platforms: private, public, open-source, and proposes a classification of fog platforms. In section 5 we focus on the current challenges faced by the fog computing researchers. In conclusion, we summarize the results obtained in the context of this study and indicate directions for further research.

1. Cloud Computing as a Basis for New Computational Concepts

1.1. The Prerequisites for Cloud Computing

The utility computing concept, originating in the 1960s, is considered to be the earliest ancestor of cloud technologies [31, 93]. This concept was not generally adopted until the 90s due to the technical constraints of the deployment and use of this architecture [13, 17, 41, 58, 60, 93]. Improvements in network technology and data transfer rates in the mid-'90s led to a new round of research in utility computing in the framework of the grid computing concept [33, 41, 58]. These shortcomings have led to further evolutionary development and the emergence of cloud computing, which often uses the grid computing model to expand computing resources [55].

1.2. Key Features of Cloud Computing

Today, 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 multiclouds [73]. 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.

The National Institute of Standards and Technology (NIST) published a definition of cloud computing, its main characteristics, and its deployment and maintenance models in 2011. Cloud computing has been defined as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

The NIST model comprises five essential characteristics, three service models, and four deployment models for clouds [63]. The following key cloud deployment models can be identified as: private cloud, public cloud, and hybrid cloud [26, 94].

A private cloud is deployed within a single organization, is available only to internal users, and does not share its resources outside the organization. The public cloud is developed by third parties and provides the resources to external users under the terms of the contract on the right of use. A hybrid cloud combines two types of deployment described above, which allows to build a balance between private and public computing [26].

Private clouds are commonly deployed as close to the end-user of the cloud as possible. That reduces the response time of the computing platform and increases the speed of data transfer between the nodes of the system. However, a private cloud is tightly interconnected with the computing needs of its owner. Not every organization has enough resources to maintain its private cloud, which must meet the requirements for availability, reliability, and the law's requirements in the country where the cloud is located [44, 78].

On the other hand, public cloud users often lack direct control over the underlying computing infrastructure. This can lead to several problems, including uncontrolled access by third parties to the private data hosted in a public cloud; blocking user servers that can be deployed on the same subnet with hosts banned in a particular country; the uncertainty of the quality of cloud resources as they are deployed on servers shared with third parties [44]. It is also challenging to ensure a change of cloud provider, as it is necessary to solve the problem of migration and conversion of data and computing services.

These features of each type of deployment are the reason why cloud providers that provide clouds to private organizations often support the ability to create hybrid clouds [84], which can be configured to a particular mode of operation, depending on the customer's requirements. This approach addresses data latency, security, and migration issues while maintaining the flexibility to customize computing resources for each task.

1.3. Preconditions for New Computing Concepts

Despite all the significant advantages guaranteed by public cloud platforms, problems that such approaches cannot effectively solve have emerged over the last five years. Thus, a large number of users of "smart" systems such as "smart home", "smart enterprise", "smart city" and other IoT solutions cannot always be satisfied with the quality of services provided by cloud solutions, in particular, due to the increase in the amount of data sent between the user/device and the cloud [46].

The emergence of the smart systems approach, populated with a variety of Internet-connected sensors and actuators, led to a revision of the architectural concept of data collection and analysis systems. The Internet of Things concept requires new approaches to storage solutions, fast data processing, and the ability to respond quickly to changes in the state of end devices [69, 70, 98]. Also, the spread of mobile devices as the main platforms for client applications makes it difficult to transfer and process large amounts of data without causing problems with response delays due to the constant movement of mobile devices.

As the amount of data sent between IoT devices, clients, and the cloud increases, problems associated with increased response time due to physical bandwidth limitations appear [60]. On the other hand, there are response time-sensitive applications and devices such as life support systems, autopilots, drones and others. Under these conditions, a remote centralized cloud has become unable to meet the ultra-low latency requirements [98]. Also, data transmission through multiple gateways and subnets raises the issue of sensitive data transmission [51].

In response to these problems, private enterprises and the academic community have raised the need to develop a computing paradigm that meets new concepts such as IoT [15, 61, 70]. This paradigm had to fill the gap between the cloud and the end devices, providing computing, storage, and data transfer in intermediate network nodes closest to the end devices. Several paradigms have been developed and applied to solve this problem, including fog and edge computing [23]. Each of these paradigms has its specific features, but all of them derive from a common principle – reducing time delays in data processing and transmission by moving computing tasks closer to the final device.

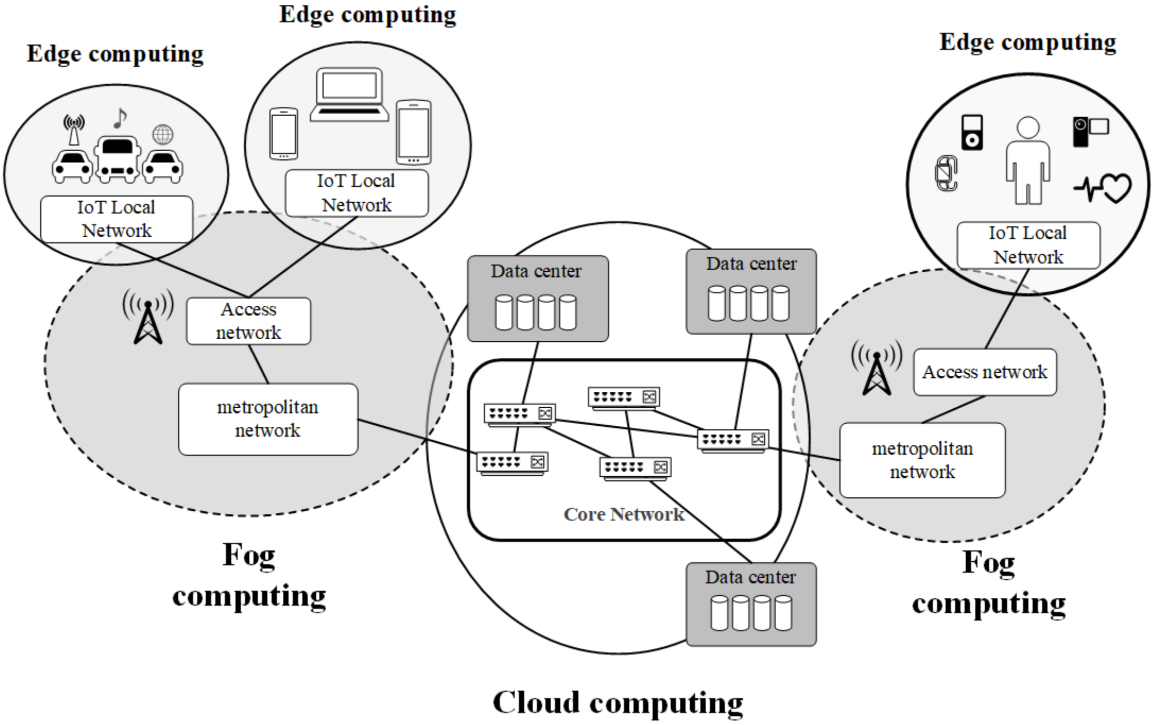


Figure 1. Comparison of the infrastructure of fog computing and its related computing paradigms from the networking perspective [93]

Figure 1 shows a diagram of the relative distribution of computational resources defined by edge, fog and cloud computing concepts. Cloud computing is a separate data center (DC) or a network of data centers located far from the user but providing high computing capabilities. On the other hand, edge computing is located right at the edge of the computing system and provides small computing capabilities, but near the consumer of those resources. Fog computing is located between the edge of the network and the cloud data center, providing significant computing resources close to the end-user, which, on the other hand, is not comparable to the total amount of cloud computing resources but can be customized and scale depending on the objectives of the end-user. This article considers Fog computing as a more general concept that includes the edge computing paradigm [45].

2. Fog and Edge Computing

2.1. History and Definition

In 1961 (see Tab. 1), John McCarthy spoke at the MIT Centennial: “If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.” [77] His concept was the basis for the idea of Douglas Parkhill [41, 43, 77, 89] to create a grid computing paradigm that was described later in 1966 and was a set of computers connected over a grid that take the computing decisions collectively.

The fog computing approach was one of the first technologies to solve the latency issues of cloud computing. The “Fog Computing” term was first proposed by CISCO in 2012 [42] and had been described as “a highly virtualized platform that provides compute storage, and networking services between end devices and traditional Cloud Computing Data Centers, typically, but not exclusively located at the edge of the network” [15]. The OpenFog group was established in 2015 to develop standards in the field of fog computing. It included companies and academic organizations such as Cisco, Dell, Intel, Microsoft Corp, and Princeton University. On December 18, 2018, the OpenFog consortium became part of The Industrial Internet Consortium [50].

Table 1. Fog computing timeline

1961	1990’s	2012	2015	2018
John McCarthy. Utility computing Definition [76]	**Ian Foster et. al**. Definition of the grid computing [33]	**Flavio Bonomi et. al**. CISCO proposed the definition of the cloud computing [15] **Mell Peter**. The NIST published the definition of the cloud computing [63]	**The OpenFog group** was established [68]	**Machaela Iorga et. al**. The NIST published the definition of the fog computing [51] **Mahmoudi Charid**. The formal definition of the edge computing was published [62]

In 2018, the National Institute of Standards and Technology of the United States had formulated an official definition of the fog computing term: “Fog computing is a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services, and consists of fog nodes (physical or virtual), residing between smart end-devices and centralized (cloud) services. The fog nodes are context-aware and support a common data management and communication system. They can be organized in clusters – either vertically (to support isolation), horizontally (to support federation), or relative to fog nodes latency-distance to the smart end-devices. Fog computing minimizes the request-response time from/to supported applications, and provides, for the end-devices, local computing resources and, when needed, network connectivity to centralized services” [51].

Bridging the gap between the cloud and end devices through computing, storage, and data management not only in the cloud but also in intermediate nodes [59] has expanded the scope of fog computing, which allowed its application in new tasks such as IoT, smart vehicles [47], smart cities [22], health care [37], smart delivery (including the use of drones) [92], video surveillance, etc. [100]. These systems benefit significantly from Big Data processing [76], allowing them to extract new knowledge and decision-making information from the data streams generated by clusters of IoT devices. Fog computing supports this challenge by enabling distributed computing resources for lightweight data processing tasks, including filtering and preprocessing data before sending it to the cloud. But the geographical distribution, heterogeneity of computing nodes, and high instability of network communications at the edge level lead to the need to solve complex problems associated with monitoring, scheduling, and ensuring the necessary quality of service of such services.

2.2. Key Characteristics of Fog Computing

Due to the late separation of the fog and edge computing concepts, many companies introduced their characteristics [9] and definitions for fog and edge computing, often combining them into one [59]. Table 2 presents the key characteristics that different authors distinguished for fog and edge computing.

In 2017, the OpenFog Consortium released a reference architecture for fog computing, which is based on eight basic principles: security, scalability, openness, autonomy, RAS (reliability, availability, and serviceability), agility, hierarchy, and programmability [68].

Table 2. Characteristics of Fog Computing [65]

Method	Properties									
	Highly virtualized	**Generally used for IoT**	**Extends the cloud**	**Not exclusively located at the edge**	**Resides at network ends**	**Fog device consists of processing, storage, and network connectivity**	**Run in a sandboxed environment**	**Leasing part of users devices and provide an incentive**	**Fog and Edge computing as similar**	**Can be deployed anywhere**
Bonomi et al. [15]	+	+		+						
Cisco Systems [21]		+	+			+				+
Vaquero and Rodero-Merino [36]						+	+	+		
IBM [49]			+	+	+				+	
Synthesis [65]	+		+	+	+					

In [45] and [11], the following key characteristics of fog computing are highlighted.

- *Contextual location awareness and low latency.* Fog computing offers the lowest-possible latency due to the fog nodes awareness of their logical location in the context of the entire system and of the latency costs for communicating with other nodes.
- *Geographical distribution.* In sharp contrast to the more centralized cloud, the services and applications targeted by fog computing demand widely but geographically identifiable, distributed deployments.

- *Heterogeneity.* Fog computing supports the collection and processing of data of different form factors acquired through multiple types of network communication capabilities
- *Interoperability and federation.* Seamless support of certain services (real-time streaming services is a good example) requires the cooperation of different providers. Hence, fog computing components must be able to interoperate, and services must be federated across domains.
- *Real-time interactions.* Fog computing applications involve real-time interactions rather than batch processing.
- *Scalability and agility of federated, fog-node clusters.* Fog computing is adaptive, at cluster or cluster-of-clusters level, supporting elastic compute, resource pooling, data-load changes, and network condition variations, to list a few of the supported adaptive functions.
- *Cognition.* Cognition is responsiveness to client-centric objectives. Fog-based data access and analytics give a better alert about customer requirements, best position handling for transmitting, storing, and controlling functions throughout the cloud to the IoT continuum. Applications, due to proximity, at end devices provide a better conscious and responsive reproduced customer requirement relation [97].
- *Support for Mobility.* Mobility support is a vital fog computational advantage that can enable direct communication between mobile devices using SDN protocols (i.e., CISCO Locator/ID Separation Protocol) that decouples host identity from location identity with a dispersed indexing system [102].
- *Large Scale Sensor Network.* The fog has a feature applicable when an environment monitoring system, in near smart grid applications, inherently extends its monitoring systems caused by hierarchical computing and storage resource requirements.
- *Widespread Wireless Access.* In this scenario, wireless access protocols (WAP) and cellular mobile gateways can act as typical examples of fog node proximity to the end-users.
- *Interoperable Technology.* Fog components must work in an interoperating environment to guarantee support for a wide range of services like data streaming and real-time processing for best data analyses and predictive decisions.

2.3. Fog and Edge Computing Concepts Definitions

Some sources refer to fog computing as edge computing, relying on the critical technology feature that data collection and analysis is not organized in a centralized cloud, but as close to the end device as possible, “at the edge of the network” [15, 34, 46, 51].

However, [98] indicates that although fog and edge computing move computation and data storage closer to the network edge, these paradigms are not identical. Within the Fog Computing paradigm, fog nodes are located at the edge of the local network, often they are deployed based on routers and wireless access points (if these devices support the required technologies for deployment of the fog node) [92]. In contrast to fog computing, edge computing is deployed even “closer” to the end devices, already inside the local network itself on the intermediate access points. Sometimes the end devices themselves can act as edge computing nodes. Smartphones, tablets, and other computing devices with sufficient computing capabilities and support for the deployment of computing nodes can handle edge computing tasks [88]. However, this also limits their computational power, and therefore there are some limitations in their application scope.

So, edge computing is used to solve such tasks as video surveillance, video caching, and traffic control [98].

The OpenFog Consortium claims that edge computing is often erroneously referred to as fog computing and determine that the main difference is that fog computing is the overall architecture of distributing resources across the network, whereas edge computing is specifically focused on executing compute processes close to end-users outside the core of the network [62]. In [20], the authors note on fog and edge computing that "fog is inclusive of cloud, core, metro, edge, clients, and things" and "the fog seeks to realize a seamless continuum of computing services from the cloud to the things rather than treating the network edges as isolated computing platforms".

Thus, the term "edge computing" is mainly used in the telecommunications industry and usually refers to 4G/5G, RAN (Radio Access Network), and ISP (Internet Service Provider) base stations [20, 56]. However, this term has recently been used in the subject area of IoT [35, 56, 75] concerning the local network where sensors and IoT devices are located. In other words, "edge computing" is located within the first of the IoT device of the transit section of the network, for example, at WiFi access points or gateways.

2.4. Classification of Fog Computing Applications

Fog computing enables new applications, especially those with strict latency constraints and those involving mobility. These new applications have heterogeneous QoS requirements and demand Fog management mechanisms to cope efficiently with that heterogeneity. Thus, resource management in Fog computing is quite challenging, calling for integrated mechanisms capable of dynamically adapting the allocation of resources. The very first step in resource management is to separate the incoming flow of requests into Classes of Service (CoS) according to their QoS requirements. The mapping of applications into a set of classes of service is the first step in creating a resource management system capable of coping with the heterogeneity of Fog applications. The authors of [38] proposed the following critical classes of fog computing applications:

- *Mission-critical.* Applications in which a component failure would cause a significant increase in the safety risk for people and the environment. Those are healthcare systems, criminal justice, drone operations, industrial control, financial transactions, military, and emergency operations. Those applications should implement distribution features to ensure duplication of functionality.
- *Real-time.* The speed of response in these applications is critical since data are processed at the same time they are generated but can tolerate a certain amount of data loss (online gaming, virtual and augmented reality applications).
- *Interactive.* Responsiveness is critical; the time between when the user requests and actions is less than a few seconds. Those are interactive television, web browsing, database retrieval, server access applications.
- *Conversational.* Characterized by being delay-sensitive but loss-tolerant with slight delays (about 100–200 ms). E.g., video and Voice-over-IP (VoIP) applications where losses cause occasional glitches in audio or video playback.
- *Streaming* class applications are accessed by users on-demand and must guarantee interactivity and continuous playout. The network must provide each stream with an average throughput that is larger than the content consumption rate. In such a case, data should

be located as close to the end-user as possible, and new nodes should easily be created and removed from the environment.

- *CPU-bound.* Involves complex processing models, such as those in decision making, which may demand hours, days, or even months of processing. Face recognition, animation rendering, speech processing, and distributed camera networks are examples of this applications class.
- *Best-effort.* For these applications, long delays are annoying but not particularly harmful; however, the completeness and integrity of the transferred data are of paramount importance. Some examples of the Best-Effort class are e-mail downloads, chats, SMS delivery, FTP, P2P file sharing.

3. Technologies that Support Fog and Edge Computing

3.1. Virtualization

The key technology that supports cloud and fog computing is virtualization [87], which allows to use the resources of one physical machine by several logical virtual machines (VMs) at the level of Hardware Abstraction Layer (HAL). Virtualization technology uses a hypervisor – a software layer that provides the operation of virtual machines based on hardware resources. A machine with a hypervisor is called a host machine. A virtual machine running on the host machine is called a guest machine, on which in turn the guest operating systems (OS) can be installed. This type of virtualization is called hypervisor-based virtualization.

There is also container-based virtualization [25], representing a packaged, standalone, deployable set of application components that can also include middleware and business logic in binary files and libraries to run applications.

Authors of [73] present a comparative analysis of both types of virtualization, based on which we can highlight some of the advantages of container-based virtualization.

- *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.
- *Scalability.* A single container engine can efficiently manage large numbers of containers, enabling additional containers to be created as needed.
- *Spatial isolation.* Containers support lightweight spatial isolation by providing each container with its resources (e.g., core processing unit, memory, and network access) and container-specific namespaces.
- *Storage.* Compared with virtual machines, containers are lightweight concerning storage size. The applications within containers share both binaries and libraries.
- *Real-time applications.* Containers provide more consistent timing than virtual machines, although this advantage is lost when using hybrid virtualization which uses both types of virtualization (hypervisor and container-based).
- *Portability.* Containers support portability from development to production environments, especially for cloud-based applications.

Thus, two main virtualization technologies are currently used to support fog computing [53]: hypervisor-based and container-based. Cloud computing mainly uses hypervisor-based virtualization to share limited hardware resources among several virtual machines. Fog computing that commonly hosted on low-performance hardware prefers container-based virtualization to create node instances on new hardware devices. That is why container-based virtualization is becoming more and more widespread in fog computing. Due to lower hardware performance requirements to ensure the deployment of computing nodes, intermediate devices may not have high computing power. This is especially relevant for edge computing nodes because they are not even run on the IoT devices themselves [71] but on intermediate access points closest to the IoT devices.

3.2. Fog Computing Orchestration

With containerization evolving as one of the technologies to support fog computing, the challenge arose to manage the computational load to ensure efficient use of geographically dispersed resources [52]. Fog computing implementation requires a different level of computing resource management compared to the cloud, for example [91].

The first complex task that arises when working with fog computing, as opposed to cloud computing, is managing the distribution of computational load (orchestration) between nodes of the fog [56, 58] by placing the fog services on them, as well as orchestration of these services, i.e. ensuring efficient collaboration of computational services for solving tasks assigned to the fog environment. Authors of [95] formulate that orchestration provides the centralized arrangement of the resource pool, mapping applications with specific requests and providing an automated workflow to physical resources (deployment and scheduling); workload execution management with runtime QoS control; and time-efficient directive generation to manipulate specific objects.

Let us consider the key tasks to be solved by the Fog Orchestrator [24, 91].

- *Scheduling.* It is necessary to consider how to exploit the collaboration between nodes to offload applications (which were not used for a long time and should be deleted to save resources) efficiently in Fog environments. In general, the processing nodes should be managed by a resource broker in the Orchestrator to perform smart scheduling of the resource, considering the applications workflows.
- *Path computation's* main objectives are: maintaining end-to-end connectivity, adapting to dynamic topologies, maximizing network and application traffic performance and providing network resilience.
- *Discovery and allocation* of the physical and virtual devices in the Fog, as well as the resources associated with them.
- *Interoperability* is the ability that distributed system elements are able to interact with each other. Several factors influence the interoperability of a system, such as the heterogeneity of its elements.
- *Latency.* One of the characteristics of Fog environments is that they provide low levels of latency. This allows the deployment of a different kind of services with real-time and low latency restrictions that are not necessarily fit for the cloud; but also requires a new set of mechanisms that guarantee that these low latency levels are met.
- *Resilience.* To guarantee a smooth work of the complex and diverse environment where the IoT acts from the resilience perspective, an Orchestrator should be in charge of intelligent migration and instantiation of resources and services providing a global view on the status of the IoT.
- *Prediction and optimization.* Proper management of resources and services in an IoT environment, where these are geographically distributed, generating multi-dimensional data in enormous quantities, is only possible if the orchestration process takes into consideration

prediction and optimization mechanisms of all overlapping and interconnected layers in the IoT.

- *Security and privacy.* From the privacy perspective, the main challenge lies in preserving the end-user privacy since the Fog nodes are deployed near them, collecting sensitive data concerning identity and usage patterns. Regarding security, a significant challenge is how to deal with the massively distributed approach of the Fog to guarantee the proper authentication mechanisms and avoid massive distributed attacks.
- *Authentication, access, and account.* To perform activities related to application life cycle management (i.e. deployment, migration, application of policies), the Orchestrator interacts with the fog nodes in the environment.

Optimization of various metrics (latency, bandwidth, energy consumption etc.) plays a vital role in fog computing orchestration. The following key tasks related to the distribution of tasks and data by the level of fog computing are currently being identified [14]:

- Offloading computing tasks from end devices to fog nodes and cloud.
- Scheduling of tasks within a fog node.
- Clustering of fog nodes: how to determine the size of a cluster of fog nodes to handle the relevant requests.
- Migration of data/applications between fog nodes.
- Geographical distribution of physical resources (before operation).
- Distributing applications/data among fog nodes and cloud.

3.3. Fog Computing and Security Issues

Due to the significant degree of decentralization of the computing process, security in fog computing differs in some critical aspects from mechanisms used, for example, in cloud computing. The design of a secure fog architecture must take into account the security features of each layer of the computing architecture, including the features of lightweight wireless data transfer at the sensing/edge layer; data transfer over middleware mesh networks; preprocessing of data using clusters of fog nodes on the application level; possible data transfer over the WAN for processing in the public cloud [10, 72].

Each of these layers has its security issues and vulnerabilities. The sensing layer is vulnerable to sensors and devices which are targets of outcoming threats, including device tampering, spoofing attacks, signal attacks, malicious data, etc. At the middleware level, the secure transmission of sensed data and its storage are the primary concerns. This layer deals with confidentiality, integrity, and availability issues. The security requirements at the application layer are determined directly by the application being executed. Figure 2 presents a classification of possible security issues and their solutions for each of the fog architecture layers listed above [10].

The authors of [96] state that the most promising research directions for security solutions in fog computing are cryptographic techniques and machine-learning for intrusion detection. Cryptographic processing includes encryption, decryption, key and hash generation, and verification of hashes used to guarantee data privacy.

As an example of this technique, the Configurable Reliable Distributed Data Storage System [19] was designed to secure data flows in whole fog. Such a system uses the AR-RRNS (Approximation of the Rank – Redundant Residue Number System) method to encrypt and decrypt data using error correction codes and secret sharing schemes. Machine-learning techniques are proposed to analyze data flow and node states to detect outside intrusion. To implement such a traffic analysis, the fog orchestrator can act as a tool to detect an intrusion or data corruption [29].

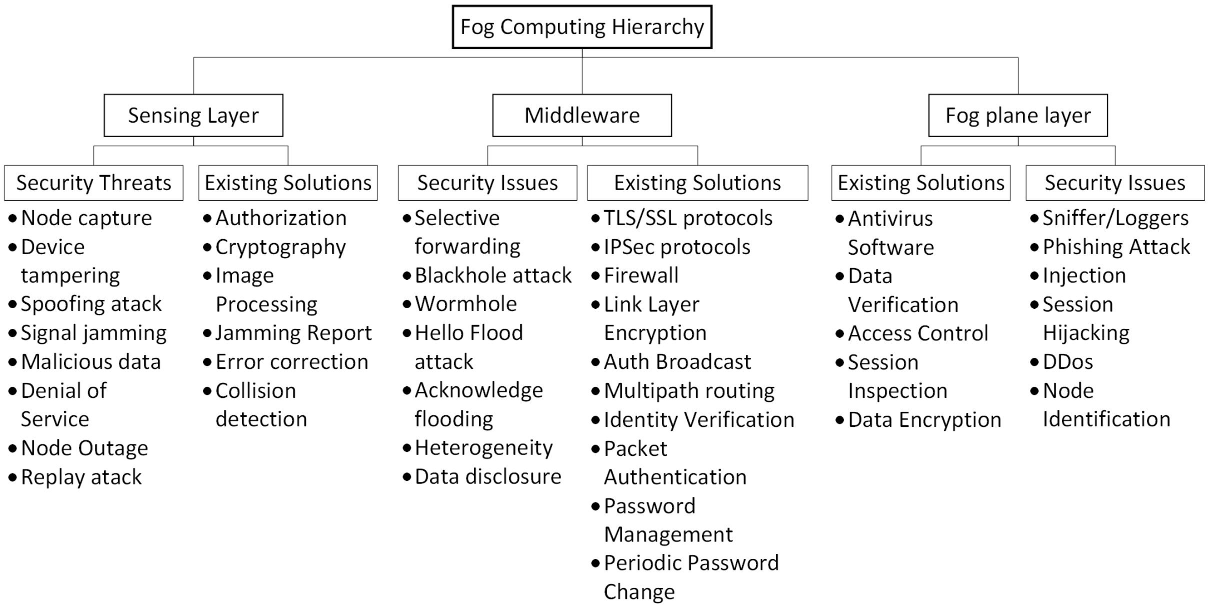


Figure 2. The security threats and solutions classifications in fog computing. DDoS: distributed DoS; TLS: transport layer security; SSL: secure sockets layer; IPsec: Internet Protocol security [10]

3.4. Fog Computing and Scalability

Scalability is another essential feature for fog computing systems to adapt workload, system cost, performance, and business needs. Based on fog computing hierarchical properties, we can highlight the following key elements of fog architecture that can be scaled [85]:

- *Virtual Nodes:* through software reconfiguration, specifying if several virtual nodes can be placed on one physical device;
- *Physical Nodes:* vertical scalability trough hardware upgrade;
- *Networks:* horizontal scaling of fog nodes and adapting to environmental changes and dynamic workloads.

Adding new fog nodes to the fog network affects all three main aspects of scalability discussed above (the question concerning fog storage resources won't be reviewed in this paper). However, this task commonly requires manual workload from network administrators, while it is hard to effectively identify the location or cluster of the new fog node. In [85] the fog model that helps to overcome this difficulty was proposed. It automates the introduction of new fog nodes into an existing network based on the current network conditions and services required by customers. Concretely, the newly added fog node can detect its geographical location (e.g., using its network scan capability or via its GPS module) and identify the most suitable cluster to connect with it.

Kubernetes platform is now the de-facto standard for service management in centralized distributed systems such as clouds. In this regard, its application to the management of fog infrastructures is of definite scientific interest. Kubernetes has a native mechanism for autoscaling that considers only CPU usage. Users can specify the maximal number of application instances, but the actual number of application instances activated is under the control of

Kubernetes. Authors of [101] developed a modification of the scheduling algorithm based on the Kubernetes platform to manage resource autoscaling tasks in fog computing systems. A fog computing platform has been designed as a collection of physically distributed containers that are orchestrated by Kubernetes and AS Broker – a service running in a Pod on Master. It communicates through APIServer with the Controller and Scheduler of Kubernetes to obtain a list of nodes where application instances are currently running. It then collects node information from all nodes. If the number of application instances should be adjusted, it sends a request through APIServer for Pod number adjustment.

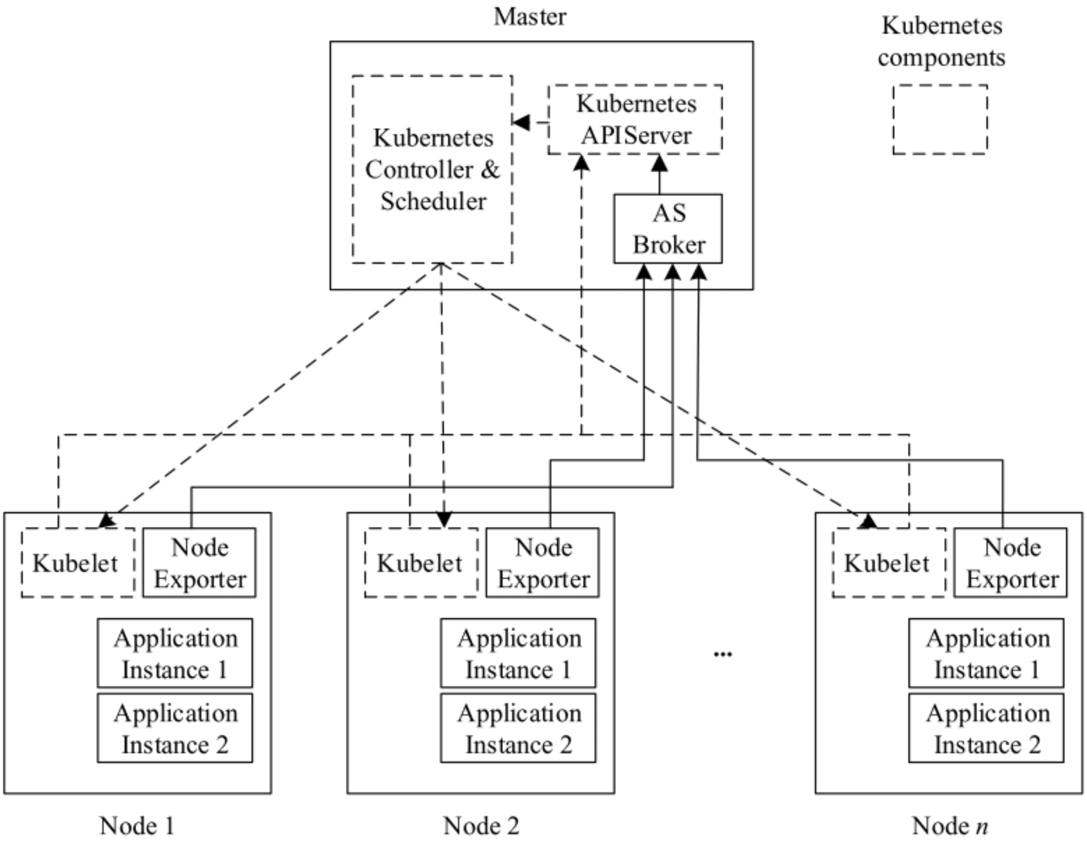


Figure 3. Proposed architecture of fog network based on Kubernetes [101]

The scalability experiment in [101] included four independent fog nodes. A stress program was used to generate CPU and memory load to emulate the processing of requests. Every request took a 15-second execution time and a 50 MB memory amount. Figure 4 shows tested application response time with and without AS Broker. Though response time dynamically changes, the result with AS Broker is better than that without almost at every time point. This result demonstrates the effectiveness of the proposed scheme.

The authors of [28] also investigate the possibilities of automatic scaling of computing resources of fog platforms developing their own Voilà platform. The objective of their work was to dynamically scale and place an applications replicas in a cluster of geo-distributed fog nodes to predominantly minimize the number of slow requests while maintaining efficient resource utilization. As a critical parameter determining the quality of service, the authors use the average response time parameter whether the latency and the processing capacity requirements are still

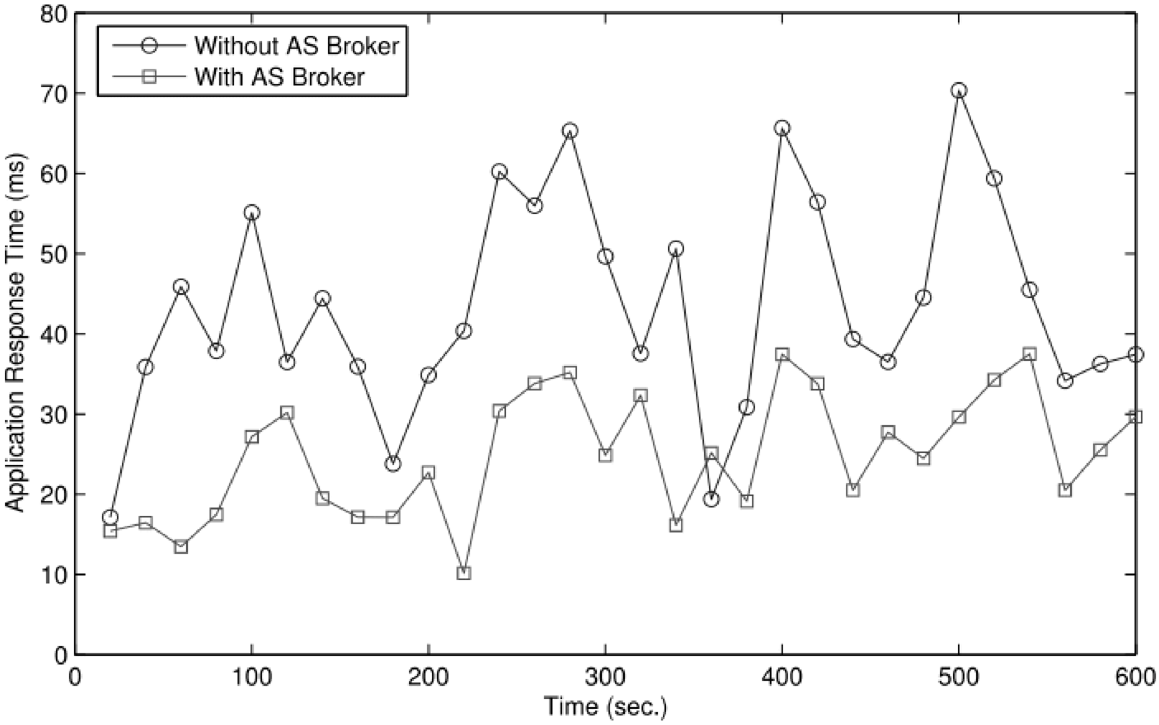


Figure 4. Application response time with and without AS Broker [101]

met. The following methods have been used to improve the quality of service: transferring service from one node to another, load redirection to the nearest lightly loaded servers, or scaling, by creating new replicas of computing services. The experimental setup consisted of 22 Raspberry Pi (RPi) model 3B+ single-board computers acting as fog computing servers. The RPis were organized with one master node and 21 worker nodes capable of hosting replicas. Node deployment was controlled with Kubernetes including node status, average number of placement,etc. as part of its normal operations. Replica placement quality evaluation was presented in [28] as main part of their experiment evaluations. Figure 5 shows the average number of placements that could be studied per second for various system sizes with the average number of placements that had to be evaluated to repair a latency or capacity violation. When the cluster size increased, the time needed to study any single placement also increased. However, even for a large system with 500 nodes, author's system evaluated approximately 100 placements per second.

The idea of using fog computing as a computing swarm and architecture of organizing fog nodes and application was described in [16]. In the proposed architecture, the fog consists of fog nodes (FNs), fog node controllers (FNCs), applications (Apps) and application controllers (ACs). FNCs control when FNs can be attached and detached. With the help of ACs FNCs can scale up and down the set of resources assigned to an App (e.g. by decreasing/increasing the cores, CPU time, and bandwidth assigned to such App) by simply changing the resources assigned to the corresponding Docker container. If the computational resources available in a FN are no more capable of satisfying the requirements of all Apps running on it, the FNC of the overloaded FN will interact with the other FNCs in the system to decide which Apps can be migrated and on which FNs.

Attempts to use fog computing with low-power devices to solve resource-intensive computational problems have been made since the beginning of the concept of fog computing. Authors

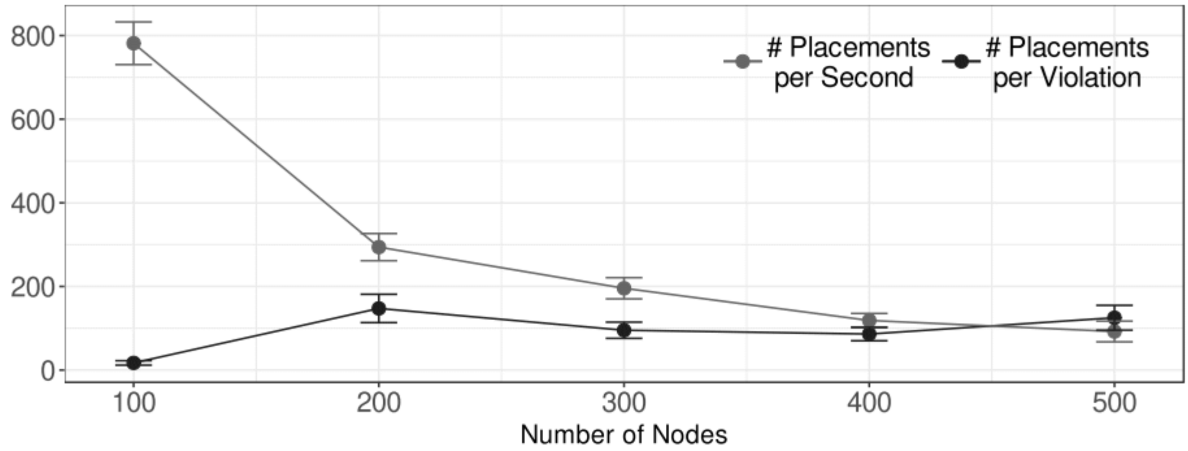


Figure 5. Evalutation of average number of placement that can be studied per second [28]

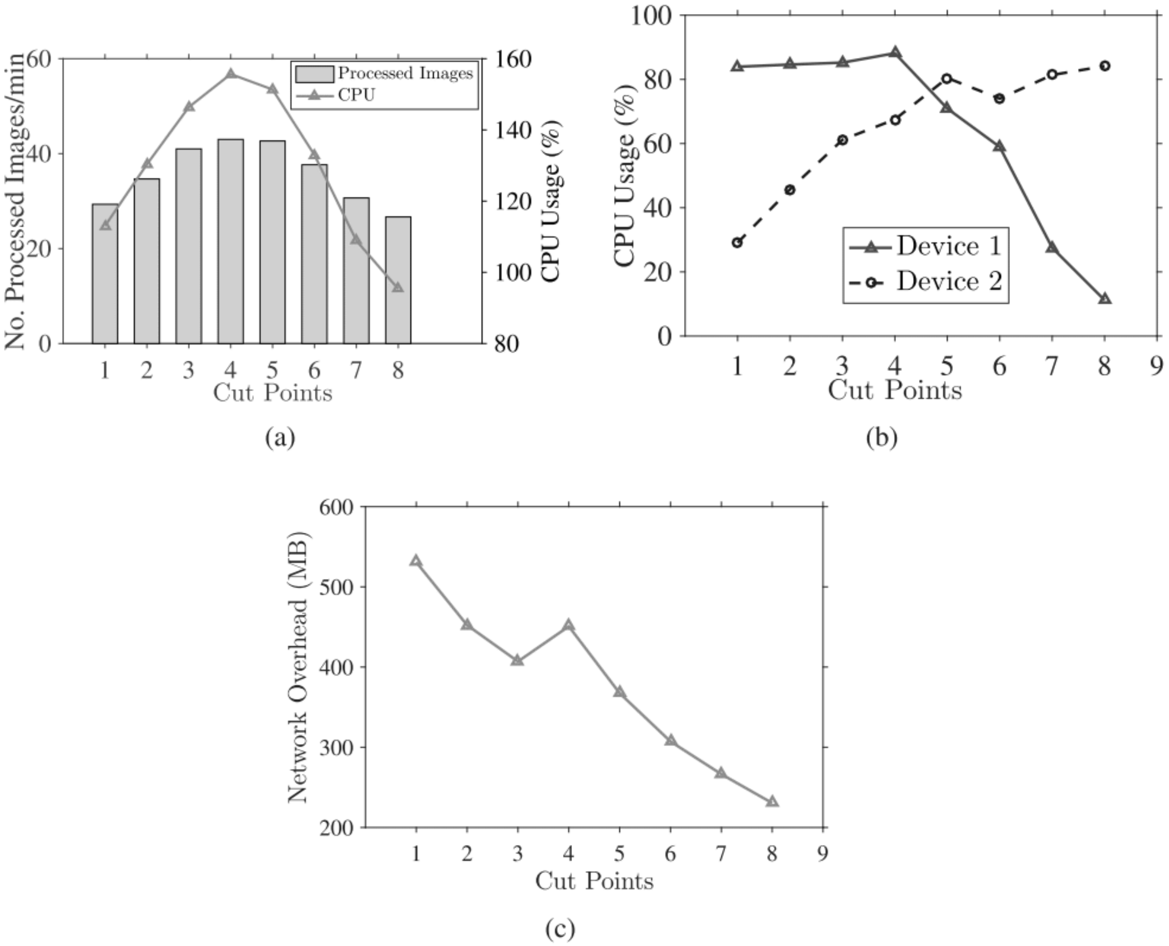


Figure 6. Different service quality caused by different cutting points: (a) the number of processed images, (b) the CPU and RAM usages, and (c) the network overhead [86]

of [86] use the resources of low-power Raspberry Pi-based nodes for Machine Learning Data Analytics using Deep Learning. They took the SCALE (Safe Community and Alerting Network) TensorFlow Application that uses various sensors (movement detection, gas, temperature, heartbeat, etc.) to build a security system as an example. The authors enhanced this project to support two crucial sensors: camera and microphone. They collected sensor data along the path where the person with the camera passed. Figure 6a shows the number of processed images per minute when running enhanced application on two devices with eight different cutting points. The application implemented a 9-layers network, so cuts are made between layers. When the cutting point went from 1 to 8, more complicated operators were put on the first device. The first device processed images before the second device. As shown in Fig. 6a, cutting points 4 and 5 resulted in the best performance. It is explained by Fig. 6b, which shows that cutting an application into smaller operators with similar complexity results in the best performance. Moreover, Fig. 6c reports the network overhead caused by distributed analytics. It shows that if more loads were put on the first device, it resulted in lower network overhead. Hence, when network resources were the bottleneck, equally-loaded splitting decisions were not preferred.

4. Overview of Fog Computing Platforms

While reviewing the existing fog computing deployment platforms, we would consider the commercial ones as well as open-source platforms. The complexity of the analysis of commercial platforms is the lack of information about their architecture and the technical solutions used, which constitute a trade secret. However, the analysis of commercial solutions has shown that among commercial fog platforms, there are platforms with the full support of fog computing (computing, analytics, and organization of the transport layer of the fog network) and platforms that provide only the transport layer of the fog network and do not provide management of computing nodes and fog computing itself. Platforms that provide only the transport layer of fog computing will not be considered in this paper.

The following key characteristics of private and public commercial fog platforms can be highlighted (see Tab. 3 and Tab. 4).

- *Supported hardware platforms* – the platform can work with any device that supports virtualization or containerization, or only with a limited list of devices – through drivers or branded devices. Smartly Fog, ThingWorx, and Cisco IOx only work with their proprietary hardware.
- *Basic development technology* – which executable environment is used to create, deploy and run fog applications.
- *Open communication protocols and SDK* – is there any restriction on the applications that can be used in the fog: whether it is necessary to port applications, or in principle can be executed only applications written using special supplied SDK, as in the case of ThingWorx, whose fog applications should be written using a proprietary SDK to run in the fog.
- *Deployment technology* – which of the technologies is used to deploy fog nodes, if known.
- Integration options – is it possible to integrate with other platforms, such as enterprise solutions or public clouds?
- *Connecting of external data sources* – the platform's ability to connect to third-party databases and data warehouses physically located outside the central cloud for data storage and processing.
- *Availability of additional services (Machine Learning, Analytics, etc.)* – the ability to connect and use additional services, which provide additional functionality for analysis and work with data in the fog.

- *Edge support* – the ability to connect and use edge devices and edge computing, and further collect and process information from them.

4.1. Private Fog Platforms

Private fog platforms provide private fog solutions based on computing infrastructure deployed directly on the customer's resources.

Table 3. Overview of private fog platforms

Feature	**ClearBlade**	**Smartiply Fog**	**LoopEdge**	**ThingWorx**	**Nebbiolo Technologies**	**Cisco IOx**
Supported hardware platforms	Universal	Own equipment	Universal	Own equipment	Universal	Own equipment
Basic development technology	JavaScript	No data	Universal (Docker)	Java VM	Universal (Docker)	Docker, Linux, IOx
Open communication protocols and SDK	+	+	+		+	+
Deployment technology	Linux KVM	No data	Docker	No data	Docker	Linux KVM
Integration opportunities	Oracle, SAP, Microsoft, Salesforce			Microsoft, Azure IoT, Hub		Microsoft, Azure IoT, Hub
Connecting external data sources	+		+	+	+	+
Availability of additional services	No data.	+		+	+	+
Edge Support	+	+	+	+	+	+

The Cisco IOx platform was presented by Cisco in 2014 [12] as a network infrastructure development due to the expected growth of IoT. The platform's focus is to reduce the labor costs of porting applications to the fog nodes, achieved through containerization technologies and based on its operating system based on the Linux OS.

The Cisco IOx is an application environment that combines Cisco IOS (a mini operating system of Cisco hardware) and Linux. Open-source Linux utilities are used to develop applications. It uses a single protocol for the interaction of fog applications throughout the network, organized using Cisco IoT technologies. Both Cisco and its partners supply IOx infrastructure fog applications. A variety of general-purpose programming languages is supported to develop Cisco IOx applications.

The Docker is used for deploying applications. Various types of applications are supported, including Docker containers and virtual machines (if network equipment has such a capability). It is also possible to use your IOx executable environment to write applications in high-level programming languages (such as Python).

The Nebbiolo Technologies platform is aimed at the corporate industrial market, supporting the Industry 4.0 concept [67]. Nebbiolo Technologies closely cooperates with Toshiba Digital Solutions [66] in supplying complete computing solutions for the industrial and IoT sectors.

The platform consists of fogNode hardware, fogOS software stack, and fogSM system administrator, deployed in the cloud or locally [45]. Fog System Manager (fogSM) provides a cloud-based, centralized management platform that allows you to deploy and configure devices on the periphery.

The platform's key feature is fogOS [45] – a software stack that provides communication, data management and application deployment at the fog level. Based on a hypervisor, fogOS provides a set of functions in a virtualized form. It supports a wide range of device connectivity standards and allows applications to be hosted and managed in real-time.

The ClearBlade Platform is a technology stack that provides fast development and deployment of enterprise IoT solutions, from edge devices to cloud services. It includes software components installed on the entire IoT device stack and the ability to connect third-party systems through the provided API for integration with devices, internal business applications, and cloud services. The ClearBlade Platform provides a centralized console for managing IoT applications, with the ability to deploy locally or in the cloud. Platform management functions are delegated to the edge nodes (or on the end devices themselves or their gateways) using ClearBlade Edge fog and edge computing [48].

The platform supports a serverless computing approach to the development of services based on the JavaScript language, which can be configured to implement machine learning and data analysis methods. The platform provides mechanisms for exporting data and analytics collected by the system to widely used business systems, applications, and databases through integration with corporate platform solutions from Oracle, SAP, Microsoft, and Salesforce. ClearBlade also provides in-house dashboards, business applications, and database management systems for integrated monitoring and management of the IoT ecosystem.

ClearBlade uses the OAuth model for access control, where each user and device receives a token that must be authorized to gain access to the system or its node. The data is encrypted on the devices themselves as well as on network transmissions. Transmitted data is encrypted using OpenSSL libraries with TLS encryption.

The Smartiply Fog platform is a cloud computing platform that focuses on optimizing resources and keeping your devices running even without connecting to the cloud. The platform provides greater reliability for online environments by optimizing resources and computing based on proprietary hardware [82]. The platform enables point-to-point interaction between devices. In this way, the node system can continue to operate autonomously to receive, analyze and store data, up to restoring communication with the external network [83].

The LoopEdge platform from Litmus Automation [4, 6] allows to connect different devices in a single system, collect and analyze data from them. Litmus Automation also provides a Loop platform allowing to manage the life cycle of any IoT device and export real-time data to internal analytical and business applications. This platform is widely distributed among well-known engineering concerns: Nissan, Renault, Mitsubishi Corporation Techno.

The platform developers emphasize that it can work with virtually any device, industrial and domestic consumers. For example, the platform supports the connection of devices based on Arduino and Raspberry Pi. Even if some device is not supported, connecting it to the platform is relatively easy due to the executable packages installed on the device itself, which can be expanded and created from scratch for a particular device.

The PTC ThingWorx platform [7] is an IoT platform that offers the connection possibility to more than 150 types of devices. However, since devices are connected through drivers that require installation, this platform is not universal and has limitations on the devices used.

Applications for the platform should be written using the supplied SDKs. Further data analysis and business process management also go through the tools provided by the platform itself. The platform has an extensive developer section with instructions, tutorials, and assistance from specialists from the company itself to install, configure, and expand the platform. Also "out of the box" is the possibility of connecting to Microsoft Azure IoT Hub.

4.2. Public Fog Platforms

Table 4. Overview of public fog platforms

Feature	**AWS Greengrass**	**Azure IoT**	**Google**	**Yandex**	**Mail.ru**
Supported hardware platforms	Universal	Universal	Universal	Universal	Universal
Basic development technology	Universal (Docker)	Universal (Docker)	Universal	Universal	Universal
Open communication protocols and SDK	+	+	+	+	+
Deployment technology	Docker	Docker	Docker	Docker	Docker
Integration capability	Amazon Elastic Compute 2	Azure, via an API	Services of Google and partners, through API	Universally via API	Universally via API
Connecting external data sources			+	+	
Availability of additional services (Machine Learning, Analytics, etc.).	+	+	+	+	+
Support Edge	+	+	+	+	+

Today, public fog platforms are the solutions of major players in the fog computing market, focused on solving data processing tasks from IoT systems linked to the capabilities of the corresponding cloud platform. The key characteristics of the considered public fog platform are given in Tab. 4.

The Azure IoT platform provides a platform for fog and edge computing based on Microsoft's technology stack. It consists of several extensive subsystems such as IoT Central and IoT Edge, which base their work on Microsoft Azure cloud technology. Connection of devices from Microsoft partners is possible without using drivers or software code due to IoT Plug and Play technology. This approach is possible for devices running any OS, including Linux, Android, Azure Sphere OS, Windows IoT, RTOS, and others.

Creation, installation, and management of fog applications are performed through the Azure IoT Hub portal. The IoT Hub is a cloud-based managed service that acts as a central message processor for bidirectional communication between an IoT application and the devices it manages. IoT Hub supports both device-to-cloud and cloud-to-device transfers. The IoT Hub supports multiple messaging templates, such as telemetry between devices and the cloud, downloading files from devices, and query-answer technology for managing devices from the cloud.

To deploy computing closer to the devices themselves or on the devices themselves, Azure IoT Edge system is used, allowing to deploy applications with their business logic, or already available in the directory ready-made applications on end devices using containerization technology.

The Amazon AWS IoT Greengrass platform allows to extend the capabilities of AWS (Amazon Web Services) to one's peripherals, enabling them to work locally with one's data while using the cloud to manage, analyze and securely store one's data. AWS IoT Greengrass allows connected devices to perform AWS Lambda functions, run Docker containers, generate forecasts based on machine learning models, synchronize these devices and interact securely with other devices even without an Internet connection.

AWS IoT Greengrass allows to create IoT solutions that connect different types of devices to the cloud and each other. AWS IoT Greengrass Core can be used on Linux devices (including Ubuntu and Raspbian distributions) that support Arm or x86 architectures. The AWS IoT Greengrass Core service provides local AWS Lambda code execution, messaging, data management, and security. Devices with AWS IoT Greengrass Core serve as portals of the service and interact with other devices that run FreeRTOS (Real-time operating system for microcontrollers) or installed SDK package AWS IoT for devices. The size of such devices can be very different: from small devices based on microcontrollers to large household appliances. When a device with AWS IoT Greengrass Core loses contact with the cloud, devices in the AWS IoT Greengrass group can continue to communicate with each other over a local network.

Google, Yandex, and Mail.ru platforms provide their cloud and fog solutions for data collection, storage, processing, analysis, and visualization. Collected data from devices is integrated into the public cloud system for deeper processing and analysis (including machine learning and artificial intelligence) due to the high computing power of the cloud. These platforms support multiple protocols for connectivity and communication through the provided API. There are many ready-to-use services available for installation in the platform directory itself, which can be connected to your cloud solution by combining them.

4.3. Open Source Fog Platforms

During the analysis of existing solutions, we reviewed existing open-source fog platforms. In contrast to commercial solutions, for open-source platforms, there are complete descriptions of architectures, requirements to computing resources, as well as technologies used, both on hardware and software levels (see Tab. 5). Open Source approach often implies that technologies used to develop, maintain and deploy systems are free and available to contributors.

The FogFrame2.0 is an open-source fog platform [3] aimed at deployment on single-board computers (Raspberry Pi). Authors designed architecture and implemented a representative framework to resolve the following challenges [79]:

- enable the coordinated cooperation among computational, storage, and networking resources in the fog [80, 81];
- implement heuristic algorithms for service placement in the fog, namely a first-fit algorithm and a genetic algorithm;
- introduce mechanisms for adapting to dynamic changes in the fog landscape and for recovering from overloads and failures.

To evaluate the behavior of FogFrame, authors apply different arrival patterns of application requests, i.e., constant, pyramid, and random walk, and observe service placement. The platform dynamically reacts to events at runtime, i.e. when new devices appear or are disabled when devices experience failures or overloads, necessary node redeployments are performed.

The FogFlow platform is an open-source fog platform [2]. The developers' main task was to provide a flexible and straightforward way of development, deployment, and orchestration of fog services [15]. The uniqueness of their approach is as follows:

- standard-based programming model for fog computing with declarative hints;

- scalable context management: to overcome the limitations of centralized context management, FogFlow introduces a distributed context management approach.

The data structure of all data flows is described based on the same standardized data model called the NGSI. Therefore, FogFlow can learn which type of data is created at which edge node. It then triggers and launches dynamic data processing flows for each edge node based on the availability of registered context metadata, which gives service developers two advantages:

- fast and easy development of fog computing applications, because the proposed hints hide configuration and deployment complexity tasks from service developers;
- good openness and interoperability for information sharing and data source integration with the use of the NGSI – a standardized open data model and API and it has been widely adopted by more than 30 cities worldwide.

FogFlow is one of the components of FIWARE open infrastructure [32], which provides the development and implementation of various smart solutions [18, 27, 30]. This infrastructure is one of the modern cloud frameworks along with Amazon Web Services [39]. A wide library of ready-made solutions from the developer community and detailed implementation instructions are available for implementation and use of FogFlow [1].

The FogBus platform (supported by Melbourn Clouds Lab) integrates various hardware tools through software components that provide structured interaction and platform-independent application execution [88]. FogBus uses blockchain to ensure data integrity when transmitting sensitive data. The platform-independent architecture of application execution and interaction between nodes allows to overcome heterogeneity in the integrated environment.

FogBus supports implementing various resource management and scheduling policies to run IoT applications compiled using parallel programming models such as SPMD (single program, multiple data).

To evaluate the performance of the FogBus platform, a prototype application system is used to analyze the Sleep Apnea data. This example illustrates how an application (in the healthcare sector) built using the SPMD model can be implemented using different FogBus settings to process IoT data in an integrated computing environment.

This framework makes it easy to deploy IoT applications, monitor and manage resources. FogBus system services are developed in cross-platform programming languages (PHP and Java). Thay are used with the Extensible Application Layer Protocol (HTTP), which helps FogBus overcome heterogeneity in the communication level of the OS and P2P of different nodes of fog. Besides, the FogBus platform functions as a “Platform as a Service” (PaaS) model for the Fog Cloud integrated environment, which not only helps application developers create different types of IoT applications but also supports users to configure services, and service providers to manage resources according to system conditions without maintaining the infrastructure.

Table 5. Overview of Open Source Fog Platforms

	Goal	Deployment
FogFrame2.0	Check the conceptual model	
FogFlow	Simpler and more flexible orchestration of services	+
FogBus	Overcome heterogeneity at the communication level between OS and P2P of different nodes of the fog	

4.4. Classification Methods for Fog Platforms

To form a unified approach to the classification of fog platforms, we have considered the key fog platforms and their key characteristics. For example, AWS Greengrass can work without access to the public cloud[4], but it is possible only to store local data in this mode of operation. Central device management, as well as centralized data collection and processing, becomes impossible. The Entire platform operation requires access to AWS IoT Core, which acts as a central service for the management and organization of fog and a public cloud.

Azure IoT can also operate on private networks[5], but only if there is a gateway within the private network that must connect to the central management and data collection node, and that node is also a public cloud. What distinguishes IoT from a public cloud is that it has a single point of access to the external network, rather than many different gateways that communicate with the public cloud.

Other public fog platforms have the same limitations as private and open-source fog platforms, the central control node of which can be deployed on any server in the local network or not at all (in this case, management and orchestration tasks are separated between intermediate fog nodes, as is done, for example, in FogFlow2.0).

Therefore, all fog platforms can be classified according to the openness or closedness of the deployment of the hub, a service that is responsible for connecting, monitoring and managing devices connected in the fog. In one form or another, almost all commercial fog platforms has the hub: LoopEdge and Azure IoT call this service – the Hub. ClearBlade and FogHorn platforms have a service with the same functionality, but it is called Device Manager. At AWS Greengrass, this service is called AWS IoT Core.

Another criterion for classification may be the requirements for the underlying hardware on which fog platform services can be deployed. Some of these platforms are tightly bound to a limited list of supported hardware devices. On the other hand, other platforms allow their services to be deployed on any end-user hardware as long as it meets the necessary minimum requirements for platform deployment. We define this characteristic as an indicator of the classification of fog platforms according to the openness of the hardware infrastructure.

The same principle is observed when comparing platforms based on openness or closed software infrastructure: the platform can support open protocols of data exchange between nodes of fog or fog programs are supplied exclusively by the developers of the platform itself and licensed partners.

Thus, any fog platform can be classified according to the principle of openness or closeness of its components (see Fig. 2). It should also be noted that platforms with a public hub are more likely to be open to their hardware and software infrastructures.

In addition to the openness or closeness of their components, some platforms have focused on the availability of the various features or services provided by the platform. The Azure IoT Hub, which is an integral part of the Azure IoT platform, explicitly calls itself **PaaS (Platform as a Service)**, providing ready-made solutions for the user's required tasks. It should be noted that none of the public fog platforms positions their platforms as pure fog. They provide fog computing as a certain basic functionality, which is the basis for other provided platform functions and services.

[4] `https://aws.amazon.com/ru/greengrass/faqs/Local_Resource_Access`

[5] `https://azure.microsoft.com/en-us/blog/introducing-iot-hub-device-streams-in-public-preview/`

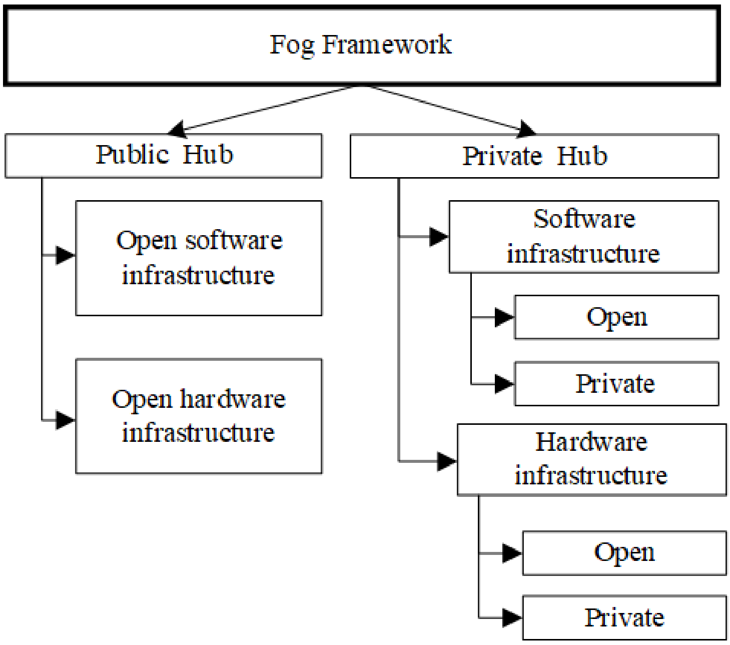


Figure 7. Classification of fog platforms according to the principle of openness or closeness of its components

Thus, the platforms themselves position some functionality as basic, which should be in any fog platform, and the user is interested not only in simple deployment and basic management of fog nodes but also in solving their specific tasks: Industry 4.0, Medicine, Smart City, etc. Platforms should provide ready-made solutions for each of the user's tasks as much as possible.

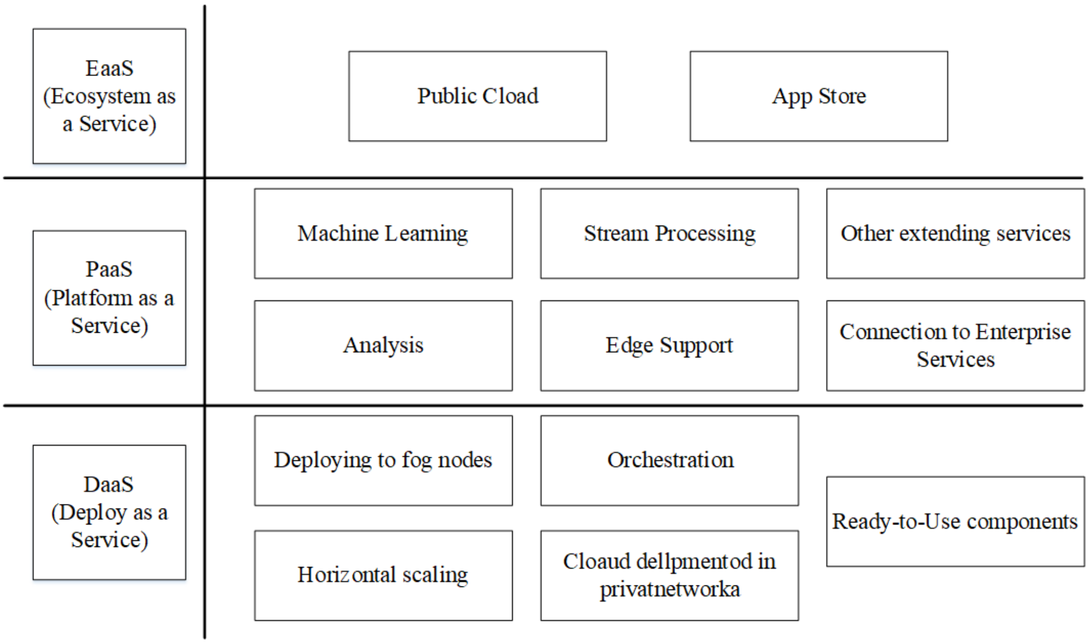


Figure 8. Classification of fog platforms based on provided platform functionality

Among other things, some platforms have allowed users to share their ready-made solutions created within the platform with the help of "stores" – resources where the user can publish his readymade fog application. This has led to the emergence of entire fog ecosystems – **EaaS**

Table 6. Reviewed platform classification

	Classification "as a service"	**Hub**	**Software Infrastructure**	**Hardware Infrastructure**
FogFrame2.0	DaaS	No Hub	Public	Public
FogFlow	PaaS	Private	Public	Public
FogBus	PaaS	Private	Public	Public
AWS Greengrass	EaaS	Public	Public	Public
Azure IoT	EaaS	Public	Public	Public
Google	EaaS	Public	Public	Public
Yandex	EaaS	Public	Public	Public
Mail.ru	EaaS	Public	Public	Public
ClearBlade	PaaS	Private	Private	Public
Smartiply Fog	PaaS	Private	Private	Private
LoopEdge	PaaS	Private	Public	Public
ThingWorx	PaaS	Private	Private	Private
Nebbiolo	PaaS	Private	Public	Public
Cisco IOx	PaaS	Private	Public	Private

(Ecosystem as a Service), which allow users to create their fog solution from ready-made components available on the platform.

This description also includes Open Source solutions that provide only a basic level of functionality – **DaaS (Deploy as a Service)**: deployment of fog nodes on existing devices, orchestration, etc. On the other hand, FogFlow has wider functionality and even its ecosystem, which includes ready-to-install components from both platform developers and the community.

Classification "as a service" can be used as a classification method based on the provided platform functionality (see Fig. 8).

This classification then can be applied to reviewed in this paper platforms (Tab. 6). Assuming that term "public" to hub means if the hub can be publicly accessed or "private" if there's only an option of deploying your own hub without ability to share your solution with others. "Private" software means that platform uses only limited list of applications and "public" if any software can be installed without publishing your application to some centralized hub. "Private" hardware if fog nodes can be deployed only on specific hardware and "public" hardware infrastructure can use a wide range of devices that meet requirements.

5. Fog Computing Challenges

In this section, we discuss several future research directions that are considered to be most promising for future research in other works and in this research likewise.

Artificial Intelligence Application Management is currently receiving considerable attention because of its ability to solve complex problems. The data needed to build an AI system is quickly accumulated in a fog [57]. Artificial Intelligence application management can help predict future resource requirements, context variations, and node failures more accurately and manage applications accordingly.

Fog nodes are limited in resources. Adding more fog nodes to the fog may reduce this limitation. However, it increases the cost of deployment, complexness of node communication, and power consumption at the network edge [8]. In this case, it may be helpful to **dynamically consolidate and scale the fog nodes** according to computational and storage needs. Fog computing is developed to execute various complex IoT applications from different domains, including smart healthcare, city, agriculture, and industry [64]. These IoT applications have specific requirements and the need for specialized support. **Application-specific management strategies** can help deal with them in Fog.

Task and data processing is decentralized in the Fog. The task may begin on one node and going through several others end on the last one. When an emergency happens in the fog, the developer and fog designer need access to log information to locate the problem in minimal time. Thus total logging helps with this task, but then the problem appears to maintain a data lake supporting storage and analysis of such data. Thats the question of **logging and monitoring of highly-distributed fog applications**.

On the other hand, there is also the issue of **task sharing and re-usability**. Applications can share a particular task to optimize the computational load on fog nodes [90]. Besides, the task executables of recently terminated applications can also be reused for other applications. To perform such operations, shared caching techniques and policies are required to be developed in the context of fog computing.

The above opens another question. If the fog node faces a software or hardware problem and shuts down other nodes wont have any information on the checkpoint the node was in. But most applications are state-dependent and stateful. So there is a challenge to organize **state management and sharing** between nodes to support the continuous and flowless work of the fog.

Most Fog applications do not consider security as part of a system but rather focus on functionality, which results in many fog platforms being vulnerable [54]. That leads to sensitive data leakage, user loss of privacy, and other **security issues** that are very significant in most IoT domains. Future work could lead towards the development of knowledge-based supplementary references, which can provide decision support for developers in designing a secure and performance efficient fog infrastructure. Such decision support would require a large systematic knowledge acquisition of best practices, known security threats and their solutions, which can be formalized as either a statistical-based system or rules, policies and facts.

Conclusion

The increase of transferred data volumes and the increased load on the cloud for client services became a prerequisite for the concept of fog computing. In this paper, the concept of fog computing, its definition and key characteristics were considered. Also, there were considered, classified and generalized some fog platforms, which are subjects of research or already used by business and private clients. In the end, the general architectural characteristics inherent in all the platforms reviewed were described.

Fog computing is a more flexible and efficient type of computation compared to cloud computing due to the solution of tasks requiring high bandwidth of the computing network, the ability to work with geographically dispersed data sources, ultra-low latency and providing local data processing. In this review paper, we not only gave an extended point of view over the fog computing paradigm but also analyzed the growing diverse number of open source and enterprise solutions for deploying fog platforms. On the basis of this review, we proposed a classification of fog solutions by their cloud layer, hardware and software publicity level and by a provided service and functionality they grant.

Acknowledgements

The research was carried out with the financial support of the RFBR and the Chelyabinsk Region, project number 20-47-740005 (chapters 1–3) and the Ministry of Science and Higher Education of the Russian Federation (state assignment FENU-2020-0022) (chapters 4, 5).

References

1. FogFlow - FogFlow v2.0 documentation. `https://fogflow.readthedocs.io/en/latest/`, accessed: 2020-02-27

2. Smartfog/fogflow: FogFlow is a standard-based IoT fog computing framework that supports serverless computing and edge computing with advanced programming models. `https://github.com/smartfog/fogflow`, accessed: 2020-02-27

3. Softls/fogframe-2.0: Fogframe framework (with extensions). `https://github.com/softls/FogFrame-2.0`, accessed: 2020-02-27

4. Litmus Automation Releases Next Generation of LoopEdge Industrial IoT Gateway Software. `https://litmus.io/loopedge-update-release/` (2017), accessed: 2020-02-27

5. IDC's Global DataSphere forecast shows continued steady growth in the creation and consumption of data. `https://www.idc.com/getdoc.jsp?containerId=prUS46286020` (2020), accessed: 2020-06-20

6. Litmus Automation - Platform Features. `https://litmus.io/litmus-edge/features/` (2020), accessed: 2020-02-27

7. PTC Thingworx. `https://www.ptc.com/ru/products/thingworx` (2020), accessed: 2020-02-27

8. Afrin, M., Jin, J., Rahman, A., et al.: Multi-objective resource allocation for Edge Cloud based robotic workflow in smart factory. Future Generation Computer Systems 97, 119–130 (2019). `https://doi.org/10.1016/j.future.2019.02.062`

9. Al-Doghman, F., Chaczko, Z., Ajayan, A.R., Klempous, R.: A review on Fog Computing technology. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings. pp. 1525–1530. IEEE (2017). `https://doi.org/10.1109/SMC.2016.7844455`

10. Aljumah, A., Ahanger, T.A.: Fog computing and security issues: A review. In: 2018 7th International Conference on Computers Communications and Control, ICCCC. pp. 237–239. IEEE (2018). `https://doi.org/10.1109/ICCCC.2018.8390464`

11. Anawar, M.R., Wang, S., Azam Zia, M., et al.: Fog computing: An overview of big IoT data analytics. Wireless Communications and Mobile Computing 2018 (2018). https://doi.org/10.1155/2018/7157192

12. Antonio, S.: Cisco delivers vision of fog computing to accelerate value from billions of connected devices pp. 1–4 (2014)

13. Armbrust, M., Fox, A., Griffith, R., et al.: A view of cloud computing. Communications of the ACM 53(4), 50–58 (2010). https://doi.org/10.1145/1721654.1721672

14. Bellendorf, J., Mann, Z.Á.: Classification of optimization problems in fog computing. Future Generation Computer Systems 107, 158–176 (2020). https://doi.org/10.1016/j.future.2020.01.036

15. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: MCC'12 - Proceedings of the 1st ACM Mobile Cloud Computing Workshop. pp. 13–15. ACM Press, Helsinki (2012). https://doi.org/10.1145/2342509.2342513

16. Brogi, A., Mencagli, G., Neri, D., et al.: Container-based support for autonomic data stream processing through the fog. In: Euro-Par 2017: Parallel Processing Workshops. Euro-Par 2017. Lecture Notes in Computer Science, vol. 10659, pp. 17–28. Springer Verlag (2018). https://doi.org/10.1007/978-3-319-75178-8_2

17. Brynjolfsson, E., Hofmann, P., Jordan, J.: Cloud computing and electricity: Beyond the utility model. Communications of the ACM 53(5), 32–34 (2010). https://doi.org/10.1145/1735223.1735234

18. Celesti, A., Fazio, M., Márquez, F.G., et al.: How to develop IoT cloud e-health systems based on FIWARE: A lesson learnt. Journal of Sensor and Actuator Networks 8(1) (2019). https://doi.org/10.3390/jsan8010007

19. Chervyakov, N., Babenko, M., Tchernykh, A., et al.: AR-RRNS: Configurable reliable distributed data storage systems for Internet of Things to ensure security. Future Generation Computer Systems 92, 1080–1092 (2019). https://doi.org/10.1016/j.future.2017.09.061

20. Chiang, M., Ha, S., Risso, F., et al.: Clarifying fog computing and networking: 10 questions and answers. IEEE Commun. Mag. 55(4), 18–20 (2017). https://doi.org/10.1109/MCOM.2017.7901470

21. Cisco Systems: Fog computing and the Internet of Things: Extend the cloud to where the things are. http://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computingoverview.pdf (2016), accessed: 2020-02-27

22. Dantas, L., Cavalcante, E., Batista, T.: A Development Environment for FIWARE-based Internet of Things Applications. In: M4IoT 2019 - Proceedings of the 2019 Workshop on Middleware and Applications for the Internet of Things, Part of Middleware 2019 Conference. pp. 21–26. ACM, Davis CA (2019). https://doi.org/10.1145/3366610.3368100

23. Dar, B.K., Shah, M.A., Islam, S.U., et al.: Delay-aware accident detection and response system using fog computing. IEEE Access 7, 70975–70985 (2019). https://doi.org/10.1109/ACCESS.2019.2910862

24. De Brito, M.S., Hoque, S., Magedanz, T., et al.: A service orchestration architecture for Fog-enabled infrastructures. In: 2017 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017. pp. 127–132. IEEE, Valencia (2017). https://doi.org/10.1109/FMEC.2017.7946419

25. De Donno, M., Tange, K., Dragoni, N.: Foundations and Evolution of Modern Computing Paradigms: Cloud, IoT, Edge, and Fog. IEEE Access 7, 150936–150948 (2019). https://doi.org/10.1109/ACCESS.2019.2947652

26. Eugene, G.: Cloud computing models. Tech. Rep. January (2013). https://doi.org/10.1201/b11149

27. Evans, D.: The Internet of Things: How the next evolution of the internet is changing everything. CISCO white paper 1, 1–11 (2011)

28. Fahs, A.J., Pierre, G., Elmroth, E.: Voilà: Tail-latency-aware fog application replicas autoscaler. In: Proceedings - IEEE Computer Society's Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, MASCOTS. pp. 1–8. IEEE Computer Society (2020). https://doi.org/10.1109/MASCOTS50786.2020.9285953

29. Fakude, N.C., Tarwireyi, P., Adigun, M.O., Abu-Mahfouz, A.M.: Fog orchestrator as an enabler for security in fog computing: A review. In: Proceedings - 2019 International Multidisciplinary Information Technology and Engineering Conference, IMITEC 2019. pp. 1–6. IEEE (2019). https://doi.org/10.1109/IMITEC45504.2019.9015896

30. Fazio, M., Celesti, A., Marquez, F.G., et al.: Exploiting the FIWARE cloud platform to develop a remote patient monitoring system. In: Proceedings - IEEE Symposium on Computers and Communications. pp. 264–270. IEEE (2016). https://doi.org/10.1109/ISCC.2015.7405526

31. Feeney, G.J.: Utility computinga superior alternative? In: AFIPS Conference Proceedings - 1974 National Computer Conference, AFIPS 1974. pp. 1003–1004. ACM Press, Chicago (1974). https://doi.org/10.1145/1500175.1500370

32. FIWARE: What is FIWARE? (2015), https://www.fiware.org/about-us/

33. Foster, I.T., Kesselman, C.: The history of the grid. In: High Performance Computing: From Grids and Clouds to Exascale - Selected Papers from the High Performance Computing Workshop. Advances in Parallel Computing, vol. 20, pp. 3–30. IOS Press (2010). https://doi.org/10.3233/978-1-60750-803-8-3

34. Garcia, J., Simo, E., Masip-Bruin, X., et al.: Do we really need cloud? estimating the fog computing capacities in the city of Barcelona. In: Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018. pp. 290–295. IEEE (2019). https://doi.org/10.1109/UCC-Companion.2018.00070

35. GE Digital: What is edge computing? (2018), https://www.ge.com/digital/blog/what-edge-computing

36. González, L.M.V., Rodero-Merino, L.: Finding your way in the fog: Towards a comprehensive definition of fog computing. Comput. Commun. Rev. 44(5), 27–32 (2014). https://doi.org/10.1145/2677046.2677052

37. Gu, L., Zeng, D., Guo, S., Barnawi, A., Xiang, Y.: Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Transactions on Emerging Topics in Computing 5(1), 108–119 (2017). https://doi.org/10.1109/TETC.2015.2508382

38. Guevara, J.C., Torres, R.d.S., da Fonseca, N.L.: On the classification of fog computing applications: A machine learning perspective. Journal of Network and Computer Applications 159 (2020). https://doi.org/10.1016/j.jnca.2020.102596

39. Guth, J., Breitenbucher, U., Falkenthal, M., et al.: Comparison of IoT platform architectures: A field study based on a reference architecture. In: 2016 Cloudification of the Internet of Things, CIoT 2016. pp. 1–6. IEEE (2017). https://doi.org/10.1109/CIOT.2016.7872918

40. Hagiu, A., Wright, J.: When data creates competitive advantage ... ... and when it doesn't. Harvard Business Review 98(1), 94–101 (2020)

41. Hannabuss, S.: The Big Switch: Rewiring the World, from Edison to Google. Library Review 58(2), 136–137 (2009). https://doi.org/10.1108/00242530910936989

42. Haouari, F., Faraj, R., Alja'Am, J.M.: Fog computing potentials, applications, and challenges. In: 2018 International Conference on Computer and Applications, ICCA 2018. pp. 399–406. IEEE, Beirut (2018). https://doi.org/10.1109/COMAPP.2018.8460182

43. Hashemi, S.M., Bardsiri, A.K.: Cloud computing vs. grid computing. ARPN Journal of Systems and Software 2(5), 188–194 (2012)

44. Hofmann, P., Woods, D.: Cloud computing: The limits of public clouds for business applications. IEEE Internet Computing 14(6), 90–93 (2010). https://doi.org/10.1109/MIC.2010.136

45. Hong, C.H., Varghese, B.: Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. ACM Computing Surveys 52(5), 1–37 (2019). https://doi.org/10.1145/3326066

46. Hong, H.J.: From cloud computing to fog computing: Unleash the power of edge and end devices. In: Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom. pp. 331–334. IEEE Computer Society, Hong Kong (2017). https://doi.org/10.1109/CloudCom.2017.53

47. Huang, C., Lu, R., Choo, K.K.R.: Vehicular fog computing: Architecture, use case, and security and forensic challenges. IEEE Communications Magazine 55(11), 105–111 (2017). https://doi.org/10.1109/MCOM.2017.1700322

48. Hughes, I., Immerman, D., Daly, P.: ClearBlade demonstrates scalability and edge analytics with IoT platform. Tech. rep. (2017)

49. IBM: What is fog computing? https://www.ibm.com/blogs/cloud-computing/2014/08/fog-computing/ (2016), accessed: 2020-02-27

50. Industrial Internet Consortium: The Industrial Internet Consortium and Openfog Consortium Join Forces. https://www.iiconsortium.org/press-room/01-31-19.htm (2019), accessed: 2020-02-27

51. Iorga, M., Feldman, L., Barton, R., et al.: Fog computing conceptual model. Tech. rep., Gaithersburg (2018). https://doi.org/10.6028/NIST.SP.500-325

52. Jiang, Y., Huang, Z., Tsang, D.H.K.: Challenges and solutions in fog computing orchestration. IEEE Network 32(3), 122–129 (2018). https://doi.org/10.1109/MNET.2017.1700271

53. Kakakhel, S.R.U., Mukkala, L., Westerlund, T., et al.: Virtualization at the network edge: A technology perspective. In: 2018 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018. pp. 87–92. IEEE, Barcelona (2018). https://doi.org/10.1109/FMEC.2018.8364049

54. Khan, S., Parkinson, S., Qin, Y.: Fog computing security: a review of current applications and security solutions. Journal of Cloud Computing 6(1) (2017). https://doi.org/10.1186/s13677-017-0090-3

55. Kumar, R., Charu, S.: An importance of using virtualization technology in cloud computing. Global Journal of Computers & Technology 1(2), 56–60 (2015)

56. Li, C., Xue, Y., Wang, J., et al.: Edge-oriented computing paradigms: A survey on architecture design and system management. ACM Computing Surveys 51(2), 39:1–39:34 (2018). https://doi.org/10.1145/3154815

57. Li, H., Ota, K., Dong, M.: Deep reinforcement scheduling for mobile crowdsensing in fog computing. ACM Transactions on Internet Technology 19(2), 1–18 (2019). https://doi.org/10.1145/3234463

58. Liu, L., Wang, Y., Yang, Y., Tian, Z.: Utility-based computing model for grid. In: 2005 International Conference on Semantics, Knowledge and Grid, SKG 2005. p. 109. IEEE Computer Society (2005). https://doi.org/10.1109/SKG.2005.140

59. Liu, Y., Fieldsend, J.E., Min, G.: A framework of fog computing: Architecture, challenges, and optimization. IEEE Access 5, 25445–25454 (2017). https://doi.org/10.1109/ACCESS.2017.2766923

60. Madsen, H., Albeanu, G., Burtschy, B., Popentiu-Vladicescu, F.: Reliability in the utility computing era: Towards reliable fog computing. In: International Conference on Systems, Signals, and Image Processing. pp. 43–46. IEEE Computer Society, Rio de Janeiro (2013). https://doi.org/10.1109/IWSSIP.2013.6623445

61. Mahmood, Z., Ramachandran, M.: Fog computing: Concepts, principles and related paradigms. In: Fog Computing: Concepts, Frameworks and Technologies, pp. 3–21. Springer (2018). https://doi.org/10.1007/978-3-319-94890-4_1

62. Mahmoudi, C., Mourlin, F., Battou, A.: Formal definition of edge computing: An emphasis on mobile cloud and IoT composition. In: 2018 3rd International Conference on Fog and Mobile Edge Computing, FMEC 2018. pp. 34–42. IEEE, Barcelona (2018). https://doi.org/10.1109/FMEC.2018.8364042

63. Mell, P., Grance, T.: The NIST definition of cloud computing: Recommendations of the National Institute of Standards and Technology. In: Public Cloud Computing: Security and Privacy Guidelines, pp. 97–101 (2012)

64. Mohamed, N., Al-Jaroodi, J., Jawhar, I.: Towards fault tolerant fog computing for IoT-based smart city applications. In: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference, CCWC 2019. pp. 752–757. IEEE (2019). https://doi.org/10.1109/CCWC.2019.8666447

65. Naha, R.K., Garg, S., Georgakopoulos, D., et al.: Fog computing: Survey of trends, architectures, requirements, and research directions. IEEE Access 6(c), 47980–48009 (2018). https://doi.org/10.1109/ACCESS.2018.2866491

66. Nebbiolo Technologies: Toshiba Digital Solutions Corporation and Nebbiolo Technologies Inc. Sign an Industrial IoT Strategic Partnership Agreement. https://www.ibm.com/blogs/internet-of-things/edge-iot-analytics/ (2018), accessed: 2020-02-27

67. Nebbiolo Technologies Inc.: Fog vs edge computing p. 8 (2016)

68. OpenFog Consortium Architecture Working Group: OpenFog reference architecture for fog computing. Tech. Rep. February (2017). https://doi.org/OPFRA001.020817

69. Pinchuk, A., Sokolov, N., Freinkman, V.: General principles of foggy computing. LastMile (3), 38–45 (2018). https://doi.org/10.22184/2070-8963.2018.72.3.38.45

70. Proferansov, D.Y., Safonova, I.E.: To the question of fog computing and the Internet of Things. Educational Resources and Technology 4(21), 30–39 (2017). https://doi.org/10.21777/2500-2112-2017-4-30-39

71. Puliafito, C., Mingozzi, E., Vallati, C., et al.: Virtualization and migration at the network edge: An overview. In: Proceedings - 2018 IEEE International Conference on Smart Computing, SMARTCOMP 2018. pp. 368–374. IEEE, Taormina, Sicily (2018). https://doi.org/10.1109/SMARTCOMP.2018.00031

72. Puthal, D., Mohanty, S.P., Bhavake, S.A., et al.: Fog computing security challenges and future directions [energy and security]. IEEE Consumer Electronics Magazine 8(3), 92–96 (2019). https://doi.org/10.1109/MCE.2019.2893674

73. Radchenko, G.I., Alaasam, A.B., Tchernykh, A.N.: Comparative analysis of virtualization methods in big data processing. Supercomputing Frontiers and Innovations 6(1), 48–79 (2019). https://doi.org/10.14529/jsfi190107

74. Ravandi, B., Papapanagiotou, I.: A self-learning scheduling in cloud software defined block storage. In: 2017 IEEE 10th International Conference on Cloud Computing, CLOUD. pp. 415–422. IEEE, Honolulu, Hawaii (2017). https://doi.org/10.1109/CLOUD.2017.60

75. Reale, A.: A guide to Edge IoT analytics: Internet of Things blog. https://www.ibm.com/blogs/internet-of-things/edge-iot-analytics/ (2017), accessed: 2020-02-27

76. Russo, G.R.: Model-based auto-scaling of distributed data stream processing applications. In: Middleware 2020 Doctoral Symposium - Proceedings of the 2020 21st International Middleware Conference Doctoral Symposium, Part of Middleware 2020. pp. 5–8. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3429351.3431741

77. Sadashiv, N., Kumar, S.M.: Cluster, grid and cloud computing: A detailed comparison. In: ICCSE 2011 - 6th International Conference on Computer Science and Education, Final Program and Proceedings. pp. 477–482. IEEE, Chennai (2011). https://doi.org/10.1109/ICCSE.2011.6028683

78. Sehgal, N.K., Bhatt, P.C.P., Sehgal, N.K., Bhatt, P.C.P.: Features of private and public clouds. In: Cloud Computing, pp. 51–60. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77839-6_4

79. Skarlat, O., Karagiannis, V., Rausch, T., et al.: A framework for optimization, service placement, and runtime operation in the fog. In: Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2018. pp. 164–173. IEEE, Zurich (2019). https://doi.org/10.1109/UCC.2018.00025

80. Skarlat, O., Nardelli, M., Schulte, S., et al.: Optimized IoT service placement in the fog. Service Oriented Computing and Applications 11(4), 427–443 (2017). https://doi.org/10.1007/s11761-017-0219-8

81. Skarlat, O., Schulte, S., Borkowski, M., et al.: Resource provisioning for IoT services in the fog. In: Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016. pp. 32–39. IEEE, Macau (2016). https://doi.org/10.1109/SOCA.2016.10

82. Smartiply: Edge gateway. https://www.smartiply.com/gateway, accessed: 2020-02-27

83. Smartiply: Mobile platform. https://www.smartiply.com/mobile, accessed: 2020-02-27

84. Sotomayor, B., Montero, R.S., Llorente, I.M., Foster, I.: Virtual infrastructure management in private and hybrid clouds. IEEE Internet Computing 13(5), 14–22 (2009). https://doi.org/10.1109/MIC.2009.119

85. Tran, Q.M., Nguyen, P.H., Tsuchiya, T., Toulouse, M.: Designed features for improving openness, scalability and programmability in the fog computing-based IoT systems. SN Computer Science 1(4), 194 (2020). https://doi.org/10.1007/s42979-020-00197-w

86. Tsai, P.H., Hong, H.J., Cheng, A.C., et al.: Distributed analytics in fog computing platforms using tensorflow and kubernetes. In: 19th Asia-Pacific Network Operations and Management Symposium: Managing a World of Things, APNOMS 2017. pp. 145–150. IEEE (2017). https://doi.org/10.1109/APNOMS.2017.8094194

87. Tseng, F.H., Tsai, M.S., Tseng, C.W., et al.: A lightweight autoscaling mechanism for fog computing in industrial applications. IEEE Transactions on Industrial Informatics 14(10), 4529–4537 (2018). https://doi.org/10.1109/TII.2018.2799230

88. Tuli, S., Basumatary, N., Buyya, R.: EdgeLens: Deep learning based object detection in integrated IoT, fog and cloud computing environments. CoRR abs/1906.11056 (2019), http://arxiv.org/abs/1906.11056

89. Vandenberg, A.: Grid computing for all. In: Guimarães, M. (ed.) Proceedings of the 43nd Annual Southeast Regional Conference, 2005, Kennesaw, Georgia, USA, March 18-20, 2005, Volume 1. p. 3. ACM (2005). https://doi.org/10.1145/1167350.1167353

90. Varshney, P., Simmhan, Y.: Demystifying fog computing: Characterizing architectures, applications and abstractions. In: Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017. pp. 115–124. IEEE (2017). https://doi.org/10.1109/ICFEC.2017.20

91. Velasquez, K., Abreu, D.P., Assis, M.R., et al.: Fog orchestration for the Internet of Everything: state-of-the-art and research challenges. Journal of Internet Services and Applications 9(1), 14:1–14:23 (2018). https://doi.org/10.1186/s13174-018-0086-3

92. Wadhwa, H., Aron, R.: Fog computing with the integration of Internet of Things: Architecture, applications and future directions. In: Proceedings - 16th IEEE International Symposium on Parallel and Distributed Processing with Applications, 17th IEEE International Conference on Ubiquitous Computing and Communications, 8th IEEE International Conference on Big Data and Cloud Computing, 11t. pp. 987–994. IEEE, Melbourne (2019). https://doi.org/10.1109/BDCloud.2018.00144

93. Webb, K.: Reviews. Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT. Internet Research 10(1), 169–174 (2000). https://doi.org/10.1108/intr.2000.17210aaf.006

94. Weinhardt, C., Anandasivam, A., Blau, B., et al.: Cloud Computing - A Classification, Business Models, and Research Directions. Business & Information Systems Engineering 1(5), 391–399 (2009). https://doi.org/10.1007/s12599-009-0071-2

95. Wen, Z., Yang, R., Garraghan, P., et al.: Fog orchestration for Internet of Things services. IEEE Internet Computing 21(2), 16–24 (2017). https://doi.org/10.1109/MIC.2017.36

96. Yakubu, J., Abdulhamid, S.M., Christopher, H.A., et al.: Security challenges in fog-computing environment: a systematic appraisal of current developments. Journal of Reliable Intelligent Environments 5(4), 209–233 (2019). https://doi.org/10.1007/s40860-019-00081-2

97. Yin, S., Kaynak, O.: Big data for modern industry: Challenges and trends [point of view]. Proc. IEEE 103(2), 143–146 (2015). https://doi.org/10.1109/JPROC.2015.2388958

98. Yousefpour, A., Fung, C., Nguyen, T., et al.: All one needs to know about fog computing and related edge computing paradigms: A complete survey. Journal of Systems Architecture 98, 289–330 (2019). https://doi.org/10.1016/j.sysarc.2019.02.009

99. Zhang, B., Mor, N., Kolb, J., et al.: The cloud is not enough: Saving IoT from the cloud. In: 7th USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 2015 (2020)

100. Zhang, P., Liu, J.K., Richard Yu, F., et al.: A survey on access control in fog computing. IEEE Communications Magazine 56(2), 144–149 (2018). https://doi.org/10.1109/MCOM.2018.1700333

101. Zheng, W.S., Yen, L.H.: Auto-scaling in Kubernetes-based fog computing platform. In: New Trends in Computer Technologies and Applications, ICS 2018. Communications in Computer and Information Science, vol. 1013, pp. 338–345. Springer (2019). https://doi.org/10.1007/978-981-13-9190-3_35

102. Zhu, J., Chan, D.S., Prabhu, M.S., et al.: Improving web sites performance using edge servers in fog computing architecture. In: Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013. pp. 320–323. IEEE (2013). https://doi.org/10.1109/SOSE.2013.73

DOI: 10.14529/jsfi210303

VaLiPro: Linear Programming Validator for Cluster Computing Systems

Leonid B. Sokolinsky[1], *Irina M. Sokolinskaya*[1]

The article presents and evaluates a scalable algorithm for validating solutions to linear programming problems on cluster computing systems. The main idea of the method is to generate a regular set of points (validation set) on a small-radius hypersphere centered at the solution point submitted to validation. The objective function is computed at each point of the validation that belongs to the feasible region. If all the values are less than or equal to the value of the objective function at the point that is to be validated, then this point is the correct solution. The parallel implementation of the VaLiPro algorithm is written in C++ through the parallel BSF-skeleton, which encapsulates all aspects related to the MPI-based parallelization of the program. We provide the results of large-scale computational experiments on a cluster computing system to study the scalability of the VaLiPro algorithm.

Keywords: linear programming, solution validator, VaLiPro, parallel algorithm, cluster computing system, BSF-skeleton.

Introduction

The era of big data [1, 2] has generated large-scale linear programming (LP) problems [3]. Such problems arise in economics, industry, logistics, statistics, quantum physics, and other fields. To solve them, high-performance computing systems and parallel algorithms are required. Thus, the development of new parallel algorithms for solving LP problems and the revision of current algorithms have become imperative. As examples, we can cite the works [4–9]. The development of new parallel algorithms for solving large-scale linear programming problems involves testing them on various benchmarks. One of the most well-known benchmark repositories of linear programming problems is the Netlib-Lp benchmark suite [10]. The solutions to all the problems from this repository are known. At the same time, in practice, it is often necessary to test a new algorithm on certain problems with unknown solutions. When testing an LP solver on such classes of problems, there is a need for validation (certification) and refinement of the obtained solution.

Several works have been devoted to the problem of certification and refinement of LP solutions. The paper [11] presents the LPlex system, which verifies and repairs a given solution to an LP problem for feasibility and optimality using exact arithmetic to guarantee the correctness of the results. The LPlex system can solve medium to large LP problems to optimality. Based on exact arithmetic (integer, rational, or modular), LPlex implements a module to detect block structures in matrices [12] and supports LU-factorizations of sparse matrices, the Bareiss method [13, 14], and the Wiedemann method [15]. The main drawback of the approach is that LPlex fails if the certified solution is not close enough to the optimal one. Koch [16] modified this approach to computing optimal solutions for the full set of Netlib-Lp instances. Rather than attempting to repair a nonoptimal basis with rational pivots, Koch recomputes a floating-point solution using greater precision in the floating-point representations. He employed the long double type that specifies 128-bit values. In [17], Applegate and co-authors extend Koch's methodology with an implementation that dynamically increases the precision of floating-point

[1] South Ural State University (National Research University), Chelyabinsk, Russian Federation

computations until a rational solution satisfying the optimality condition is obtained. They modify the conventional simplex algorithm by changing every floating-point type into the rational type provided by the GNU multiple precision arithmetic library (GMP) [18] and replacing every arithmetic operation in the original code with the corresponding GMP operations. The program starts with the best native floating-point precision and then increases it by about 50 % at each iteration (keeping the precision value a multiple of 32 bits to align with the typical word size). The main drawback of this approach is that the use of the multiple-precision arithmetic in the case of large and complex LP problems has high overheads. In [19], Panyukov and Gorbik try to overcome this disadvantage by using parallel computing on distributed memory. In this paper, they utilize rational arithmetic and propose two approaches for parallelizing the simplex method. The first method is based on the decomposition of the simplex tableau by columns. The second method is based on the modified simplex method using the inverse matrix and exploits the decomposition of the original matrix by columns and that of the inverse matrix by rows. However, the results of computational experiments are not sufficiently convincing for the following reasons: there is no comparison with the best sequential solutions; the computations were performed using only three sparse LP problems (the number of nonzero elements did not exceed 5 %); and, in addition, the scalability bound of the proposed parallel algorithm was only 16 processors. Another original approach is suggested by Gleixner and co-authors in [20]. This paper describes an iterative refinement procedure for computing extended-precision or exact solutions to LP problems. Arbitrarily precise solutions can be computed by solving a sequence of closely related LPs with limited-precision arithmetic. These LPs share the same constraint matrix as the original problem instance and are transformed only by modification of the objective function, right-hand sides, and variable bounds. This implementation is publicly available as an extension of the academic LP solver SoPlex.

All the methods discussed above concentrate on refining the approximate solution that has already been found. If the found solution is too far from the correct one, which means that there is an error in the algorithm, then the use of these methods becomes impractical. In addition, all of these algorithms have high computational complexity and do not allow efficient parallelization on large cluster computing systems. The method proposed in this article focuses on debugging and validating new LP algorithms on cluster computing systems. It is implemented as a parallel program, VaLiPro (Validator of Linear Program), which shows good scalability on multiprocessor computing systems with distributed memory. The rest of the paper is organized as follows. Section 1 provides a formal description of the proposed method for validating solutions to LP problems and presents a sequential version of the VaLiPro algorithm. The parallel version of the VaLiPro algorithm is discussed in section 2. Section 3 describes the implementation of the VaLiPro parallel algorithm in C++ using the BSF-skeleton. Here, we present the results of computational experiments on a cluster computing system, which confirm the efficiency of the proposed approach. In conclusion, we summarize the obtained results and expose plans for using the VaLiPro validator in the development of an artificial neural network capable of solving large-scale LP problems.

1. Method for Validating Solutions to LP Problems

Let the following linear programming problem be given in the Euclidean space $\mathbb{R}^n$:

$$\bar{x} = \arg\max \left\{ \langle c, x \rangle \mid Ax \leqslant b, x \in \mathbb{R}^n \right\}, \tag{1}$$

where c is the vector of the objective function coefficients. Here and below, $\langle \cdot, \cdot \rangle$ stands for the dot product of vectors. Let us define $M = \{x \in \mathbb{R}^n \mid Ax \leqslant b\}$ as the feasible region of problem (1). By definition, the set M is convex and closed. From now on, we assume that M is a nonempty bounded set, i.e., problem (1) has at least one solution. Let $\tilde{x} \in \mathbb{R}^n$ be an approximate solution of problem (1) obtained using some LP solver that must be certified.

The main idea of the VaLiPro validation method is to construct a finite set of points V covering a hypersphere S of small (compared to the size of the polytope M) radius ρ centered at the certified solution point $\tilde{x}$:

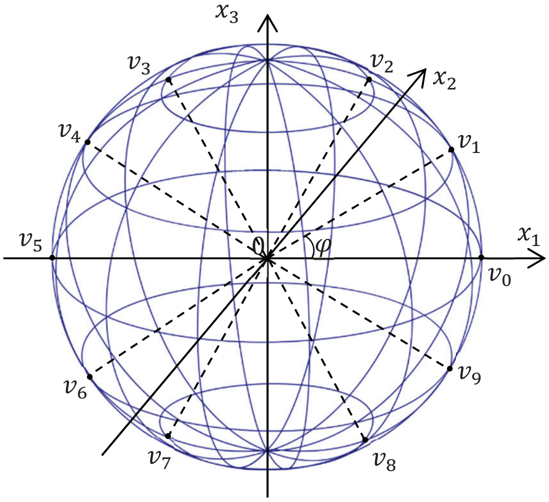


Figure 1. Plotting the points of the validation set V on a three-dimensional sphere with $d = 5$

$$V \subset S = \left\{ x \in \mathbb{R}^n \;\middle|\; \|x - \tilde{x}\|^2 = \rho^2 \right\}.$$

Here and below, $\|\cdot\|$ denotes the Euclidean norm. Let us compute the maximum of the objective function on the set $V \cap M$:

$$\bar{v} = \arg\max \left\{ \langle c, v \rangle \;\middle|\; v \in V \cap M \right\}.$$

If $\left| \langle c, \bar{v} \rangle - \langle c, \tilde{x} \rangle \right| < \varepsilon$, then the approximation $\tilde{x}$ is considered correct. Otherwise, $\tilde{x}$ is considered an incorrect solution. Here, $\varepsilon \in \mathbb{R}_{>0}$ is a small positive constant that is a parameter of the validation algorithm.

Let us describe the method for constructing the validation set V. It is known [21] that the coordinates of any point $v = (v_1, \dots, v_n)$ lying on the surface of the hypersphere S defined by the equation

$$\|x - \tilde{x}\|^2 = \rho^2$$

can be represented as follows:

$$\begin{aligned}
v_1 &= \rho \cos(\phi_1); \\
v_j &= \rho \cos(\phi_j) \prod_{i=1}^{j-1} \sin(\phi_i) \ (j = 2, \dots, n-2); \\
v_{n-1} &= \rho \sin(\theta) \prod_{i=1}^{n-2} \sin(\phi_i); \\
v_n &= \rho \cos(\theta) \prod_{i=1}^{n-2} \sin(\phi_i),
\end{aligned} \tag{2}$$

where $0 \leqslant \phi_j \leqslant \pi$ $(j = 1, \dots, n-2)$ and $0 \leqslant \theta < 2\pi$. Let us explain the method for generating the validation set V using a three-dimensional sphere (see Fig. 1). Fix an odd number of *parallels* $d \geqslant 3$ (poles are excluded). Set

$$\varphi = \pi/d. \tag{3}$$

Algorithm 1 Generating points of the validation set

Parameters: d, ρ

a) With duplicates

```
 1: $\varphi := \pi/d$
 2: for $j_{n-1} = 0 \ldots (2d-1)$ do
 3:     $\theta := j_{n-1}\varphi$
 4:     for $j_{n-2} = 0 \ldots d$ do
 5:         $\phi_{n-2} := j_{(n-2)}\varphi$
 6:         $\ldots$
 7:         for $j_2 = 0 \ldots d$ do
 8:             $\phi_2 := j_2\varphi$
 9:             for $j_1 = 0 \ldots d$ do
10:                 $\phi_1 := j_1\varphi$
11:                 $\varpi := 1$
12:                 $v_1 := \rho\cos(\phi_1)$
13:                 for $l = 2 \ldots n-2$ do
14:                     $\varpi := \sin(\phi_{l-1})\varpi$
15:                     $v_l := \rho\cos(\phi_l)\varpi$
16:                 end for
17:                 $v_{n-1} := \rho\sin(\theta)\varpi$
18:                 $v_n := \rho\cos(\theta)\varpi$
19:                 output $v$
20:             end for
21:         end for
22:         $\ldots$
23:     end for
24: end for
25: stop
```

b) Without duplicates

```
 1: $\varphi := \pi/d$
 2: for $j_{n-1} = 0 \ldots (2d-1)$ do
 3:     $\theta := j_{n-1}\varphi$
 4:     for $j_{n-2} = 1 \ldots d-1$ do
 5:         $\phi_{n-2} := j_{(n-2)}\varphi$
 6:         $\ldots$
 7:         for $j_2 = 1 \ldots d-1$ do
 8:             $\phi_2 := j_2\varphi$
 9:             for $j_1 = 1 \ldots d-1$ do
10:                 $\phi_1 := j_1\varphi$
11:                 $\varpi := 1$
12:                 $v_1 := \rho\cos(\phi_1)$
13:                 for $l = 2 \ldots n-2$ do
14:                     $\varpi := \sin(\phi_{l-1})\varpi$
15:                     $v_l := \rho\cos(\phi_l)\varpi$
16:                 end for
17:                 $v_{n-1} := \rho\sin(\theta)\varpi$
18:                 $v_n := \rho\cos(\theta)\varpi$
19:                 output $v$
20:             end for
21:         end for
22:         $\ldots$
23:     end for
24: end for
25: stop
```

In the plane $(x_1, 0, x_3)$, we set the angles $0, \varphi, \ldots, (2d-1)\varphi$ starting from the axis $(0, x)$. At the intersection with the sphere, the resulting rays give the set of points $\{v_0, \ldots, v_{2d-1}\}$, which uniquely define d parallels. Then, on the plane $(x_1, 0, x_2)$, set the angles $0, \varphi, \ldots, (2d-1)\varphi$ from the axis $(0, x_1)$ and define d *meridians* in the same way. The intersections of parallels and meridians, excluding the poles, give the points that form a validation set on the three-dimensional sphere.

The described method for generating points of the validation set for $n \geqslant 3$ in the general case is given in Algorithm 1a. The nested loops with the headers in steps 2, 4, ..., 7, and 9 generate the following spherical coordinates of a validation point:

$$(\rho, \phi_1, \phi_2, \ldots, \phi_{n-2}, \theta)\,. \tag{4}$$

Algorithm 2 Function g (calculating the point v by its number k)

```
 1: function $g(k, d, \rho)$
 2:     $u_{n-1} := \left\lfloor k/\left(d-1\right)^{n-2} \right\rfloor$
 3:     $u_n := u_{n-1}$
 4:     $k := k \mod (d-1)^{n-2}$
 5:     for $j = (n-3)\ldots 0$ do
 6:         $u_j := \lfloor k/(d-1)^j \rfloor + 1$
 7:         $k := k \mod (d-1)^j$
 8:     end for
 9:     $\varpi := 1$
10:     $\varphi := \pi/d$
11:     $v_1 := \rho\cos(u_1\varphi)$
12:     for $j = 2\ldots(n-2)$ do
13:         $\varpi := \varpi\sin(u_{j-1}\varphi)$
14:         $v_j := \rho\cos(u_j\varphi)\varpi$
15:     end for
16:     $\varpi := \varpi\sin(u_{n-2}\varphi)$
17:     $v_{n-1} := \rho\sin(u_{n-1}\varphi)\varpi$
18:     $v_n := \rho\ \cos(u_n\varphi)\varpi$
19:     return $(v_1, \ldots, v_n)$
20: end function
```

In steps 11–18, the spherical coordinates are converted to Cartesian coordinates by Equations (2). Multiplying the quantities of iterations of for-loops with headers 2, $4, \ldots,$ 7, and 9, we conclude that Algorithm 1a outputs $2d(d+1)^{n-2}$ validation points. However, there will be duplicates among the output points. The computational experiments showed that if one sets the dimension $n = 4$ and the number of parallels $d = 5$, then Algorithm 1a generates 189 duplicates with a total number of points equal to 360, which is more than 50 %. The duplicates are generated at iterations in which $\phi_i = 0$ or $\phi_i = \pi$, which corresponds to $j_i = 0$ and $j_i = d$ $(i = 1, \ldots, n-2)$. The reason is that one of the factors $\sin(\phi_i)$ in (2) is equal to zero in this case, and therefore the variations of other factors cannot change the value of the corresponding coordinate. This issue can be solved without a major revision of Algorithm 1a, by changing the start values and end values of the control variables in loop headers $4, \ldots,$ 7, and 9, as is done in Algorithm 1b. This algorithm generates a validation set without duplicates but, at the same time, it loses a certain number of unique points. With $n = 4$ and $d = 5$, this quantity is 11, which is less than 7 % of the whole set after removing duplicates. Experiments have shown that such a loss does not significantly affect the accuracy of the validation algorithm. The number of points of the validation set V generated by Algorithm 1b is determined by the following equation:

$$|V| = 2d(d-1)^{n-2}. \tag{5}$$

The main drawback of Algorithm 1b is that the number of nested loops depends on the problem dimension, which does not allow using the dimension as a program parameter. To overcome this drawback, we use a vector-valued function that calculates the coordinates of a point by its sequential number k in the point sequence generated by Algorithm 1b (counting starts from zero). The definition of this function is given in Algorithm 2.

The final implementation of the VaLiPro method, using the vector-valued function g, is given in Algorithm 3. An additional parameter of this algorithm is the small positive constant ε (by default, $\varepsilon = 10^{-6}$), which compensates for possible numerical errors when comparing the values of the objective function in Step 5. Let us make several brief comments on the steps of Algorithm 3. Step 1 reads the source data of LP problem (1), the algorithm parameters, and the solution $\tilde{x}$ that is to be certified. Step 2 calculates the angle φ according to Equation (3). Step 3 begins the loop that varies the point number k from 0 to $2d(d-1)^{n-2} - 1$ as per Equation (5). Using the vector-valued function g (see Algorithm 2), Step 4 computes the next validation point v. Step 5 checks whether v belongs to the feasible region of problem (1) and compares the objective-function values at the points v and $\tilde{x}$. If the objective function takes a larger value at the point v, and this point is feasible, then the control is passed to Step 9, which prints a message stating that the certified solution is not correct. Otherwise, the next iteration of the loop

Algorithm 3 Validation of the LP solution $\widetilde{x}$

```
 1: input $n, A, b, c, d, \rho, \varepsilon, \widetilde{x}$
 2: $\varphi := \pi/d$
 3: for $k = 0 \dots 2d(d-1)^{n-2} - 1$ do
 4:     $v := g(k, d, \rho)$
 5:     if $Av \leqslant b$ & $\langle c, v \rangle > \langle c, \widetilde{x} \rangle + \varepsilon$ goto 9
 6: end for
 7: output "Solution is correct"
 8: goto 10
 9: output "Solution is incorrect"
10: stop
```

proceeds. If the loop ends naturally, the control is passed to Step 7, which outputs a message saying that the solution is correct. After that, the control is passed to Step 10, which completes the execution of the algorithm.

2. Parallel Algorithm for Validating LP Solutions

According to Equation (5), the cardinality of the validation set generated by Algorithm 3 depends exponentially on the space dimension. Therefore, Algorithm 3 has high computational complexity for large dimensions. To reduce computational overheads, we developed a parallel version of Algorithm 3, given as Algorithm 4 below. The parallel algorithm is based on the BSF parallel computation model [22, 23], which exploits the master–slave paradigm [24]. According to the BSF model, the master node serves as a control and communication center. All slave nodes execute the same code but on different data. The BSF model assumes the algorithm representation in the form of operations on lists using the higher-order functions *Map* and *Reduce* defined by the Bird–Meertens formalism [25]. The higher-order function *Map* transforms the original list $W = [w_0, \dots, w_{K-1}]$ into the list $Z = [z_0, \dots, z_{K-1}]$ by applying the function $f_{\tilde{x}}$ to each element:

$$Z = Map(f_{\tilde{x}}, W) = [f_{\tilde{x}}(w_0), \dots, f_{\tilde{x}}(w_{K-1})] .$$

In the case considered here, the elements of the list W are the sequential numbers of the validation set points, that is,

$$W = [0, \dots, K-1] ,$$

where $K = 2d(d-1)^{n-2}$. The Boolean function $f_{\tilde{x}} : \{0, \dots, K-1\} \to \{true, false\}$ is defined as follows:

$$f_{\tilde{x}}(w) = \begin{cases} true \mid A \cdot g(w) \leqslant b \wedge \langle c, g(w) \rangle \leqslant \langle c, \tilde{x} \rangle ; \\ false \mid A \cdot g(w) > b \vee \langle c, g(w) \rangle > \langle c, \tilde{x} \rangle , \end{cases}$$

where the vector-valued function g computes the coordinates of the validation point by its number w. The function $f_{\tilde{x}}$ returns *true* if the point $g(w)$ belongs to the feasible region and if the value of the objective function at this point is less than or equal to the value of the objective function at the point $\tilde{x}$. Otherwise, the function $f_{\tilde{x}}$ returns *false*. Thus, the list $Z = [z_0, \dots, z_{K-1}]$ contains Boolean indicators for all points of the validation set. If at least one element in this list has the value *false*, then the point $\tilde{x}$ is an incorrect solution of problem (1).

Algorithm 4 Parallel algorithm for validating an LP solution

```
                 Master                                    Slave (l=0,...,L-1)

 1:                                            1: input $n, A, b, c, d, \rho, \varepsilon, \widetilde{x}$
 2:                                            2: $L :=$ NumberOfSlaves
 3:                                            3: $K := 2d(d-1)^{n-2}$
 4:                                            4: $W_l := [lK/L, \ldots, (l+1)K/L - 1]$
 5:                                            5: $Z_l := Map\,(f_{\widetilde{x}}, W_l)$
 6:                                            6: $s_l := Reduce(\wedge, Z_l)$
 7: RecvFromSlaves $[s_0, \ldots, s_{L-1}]$    7: SendToMaster $s_l$
 8: $s := Reduce(\wedge, [s_0, \ldots, s_{L-1}])$  8:
 9: if $s = true$ then                         9:
10:     output "Solution is correct"          10:
11: else                                      11:
12:     output "Solution is incorrect"        12:
13: end if                                    13:
14: stop                                      14: stop
```

The higher-order function *Reduce* transforms the list $Z = [z_0, \ldots, z_{K-1}]$ into a single Boolean value s by iteratively applying the conjunction operation to all the elements of the list Z:

$$s = Reduce(\wedge, Z) = z_0 \wedge \ldots \wedge z_{K-1}.$$

In Step 4 of Algorithm 4, the l-th slave sets its own part W_l of the list W:

$$W_l = [lK/L, \ldots, (l+1)K/L - 1]\,.$$

Here, L denotes the number of slaves. For simplicity, we assume that K is a multiple of L. In Step 5, the slave applies the *Map* function to its sublist W_l. In Step 6, the resulting sublist of Boolean values is folded into a single Boolean value s_l by applying the *Reduce* function, taking the conjunction operation as the first parameter. In Step 7, the l-th slave sends the value s_l to the master. In the same Step 7, the master receives all the calculated values from the slaves. In Step 8, the master folds the list of received values into a single Boolean value s using the *Reduce* function. In steps 9–12, the master examines the calculated Boolean value s and outputs the corresponding conclusion.

3. Software Implementation and Computational Experiments

We implemented the parallel Algorithm 4 in C++ through the parallel BSF-skeleton [26], which is based on the BSF parallel computation model [22, 23] and encapsulates all aspects related to the parallelization of the program using the MPI library [27]. The source code of the VaLiPro parallel program is freely available at `https://github.com/leonid-sokolinsky/BSF-LPP-Validator`. Using this program, we conducted large-scale computational experiments on the cluster computing system "Tornado SUSU" [28]. The specifications of this system are given in Tab. 1.

Table 1. Specifications of the "Tornado SUSU" computing cluster

Parameter	Value
Number of processor nodes	480
Processor	Intel Xeon X5680 (6 cores, 3.33 GHz)
Processors per node	2
Memory per node	24 GB DDR3
Interconnect	InfiniBand QDR (40 Gbit/s)
Operating system	Linux CentOS

For experiments, we used random LP problems generated by the program FRaGenLP [29] with the following parameters: $\alpha = 200$ (the length of the bounding-hypercube edge), $\theta = 100$ (the radius of the large hypersphere), $\rho = 50$ (the radius of the small hypersphere), $L_{\max} = 0.35$ (the upper bound of *near parallelism* for hyperplanes), $S_{\min} = 100$ (the minimum acceptable *closeness* for hyperplanes), $a_{\max} = 1000$ (the upper absolute bound for the coefficients), and $b_{\max} = 10\,000$ (the upper absolute bound for the constant terms). The experiments were conducted for the following dimensions: $n = 15$, $n = 17$, and $n = 19$. The numbers of inequalities were 46, 52, and 58, respectively. The solutions to the LP problems were obtained using the apex method [8]. Throughout the experiments, we used the following VaLiPro parameters: $d = 5$, $\rho = 1$, and $\varepsilon = 10^{-6}$. The results of the experiments are shown in Fig. 2. The verification of a solution for a problem of dimension $n = 19$ with a configuration consisting of the master node and one slave node took 17 minutes. The verification of a solution for the same problem with a configuration consisting of the master node and 310 slave nodes took 4 seconds. The analysis of the results showed that the scalability bound (the maximum of the speedup curve) of the algorithm significantly depends on the dimension of the problem. For $n = 19$, the parallel version of the VaLiPro algorithm demonstrated near-linear scalability up to 310 processor nodes. For $n = 17$, the scalability bound was approximately 260 nodes, and for $n = 15$, this bound decreased to 60 processor nodes. This is because a problem of such a small dimension is not able to load such a large number of processor nodes: the time spent on data transfer over the network begins to dominate over the time spent on calculations, and the processors begin to stand idle.

Conclusions

The article presents the parallel algorithm VaLiPro for validating linear programming solutions on cluster computing systems. The main idea of the validation algorithm is to generate a regular set of points on a small-radius hypersphere centered at the solution point that is to be certified. The solution is considered correct if all points of the validation set belonging to the feasible region have lower values of the objective function than does the solution point being certified. The implementation of the parallel algorithm VaLiPro was performed in C++ using the parallel BSF-skeleton, which encapsulates in the problem-independent part of its code all aspects related to the parallelization of a program using the MPI library. The source code of the developed parallel program is freely available at `https://github.com/leonid-sokolinsky/BSF-LPP-Validator`. The proposed validation method is generic and suitable for linear programming problems of any kind. The advantage of the parallel VaLiPro algorithm is the near-linear speedup starting with

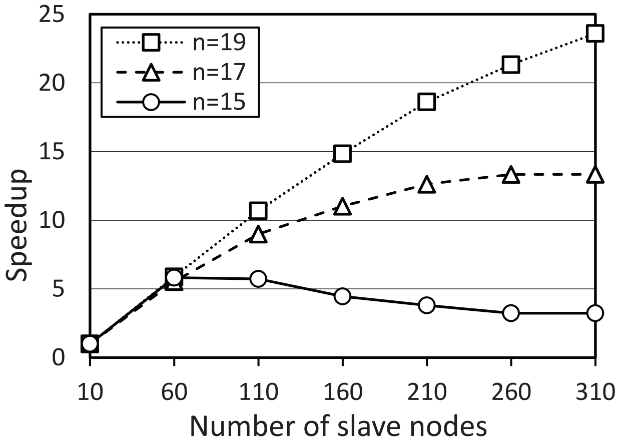


Figure 2. Speedup curves of the VaLiPro parallel algorithm for various dimensions

a problem dimension of 19. The main drawback that limits the practical use of the suggested method is the exponential growth of the number of points in the validation set as the dimension of the space increases; this results in the exponential growth of the computational complexity. In practice, the proposed algorithm can be effectively used for LP problems with a space dimension of no more than 20. The described algorithm was used together with the FRaGenLP generator and the apex method to prepare a training dataset of 70 000 examples, which will be used to develop an artificial neural network capable of solving multidimensional linear programming problems.

Acknowledgements

The reported study was partially funded by the Russian Foundation for Basic Research (project No. 20-07-00092-a) and the Ministry of Science and Higher Education of the Russian Federation (government order FENU-2020-0022).

References

1. Jagadish, H.V., Gehrke, J., Labrinidis, A., et al.: Big data and its technical challenges. Communications of the ACM 57(7), 86–94 (2014). https://doi.org/10.1145/2611567

2. Hartung, T.: Making Big Sense grom Big Data. Frontiers in Big Data 1, 5 (2018). https://doi.org/10.3389/fdata.2018.00005

3. Sokolinskaya, I., Sokolinsky, L.B.: On the Solution of Linear Programming Problems in the Age of Big Data. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies, PCT 2017. Communications in Computer and Information Science, vol. 753. pp. 86–100. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67035-5_7

4. Mamalis, B., Pantziou, G.: Advances in the Parallelization of the Simplex Method. In: Zaroliagis, C., Pantziou, G., Kontogiannis, S. (eds.) Algorithms, Probability, Networks, and

Games. Lecture Notes in Computer Science, vol. 9295. pp. 281–307. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24024-4_17

5. Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. Mathematical Programming Computation 10(1), 119–142 (2018). https://doi.org/10.1007/s12532-017-0130-5

6. Tar, P., Stagel, B., Maros, I.: Parallel search paths for the simplex algorithm. Central European Journal of Operations Research 25(4), 967–984 (2017). https://doi.org/10.1007/s10100-016-0452-9

7. Yang, L., Li, T., Li, J.: Parallel predictor-corrector interior-point algorithm of structured optimization problems. In: 3rd International Conference on Genetic and Evolutionary Computing, WGEC 2009. pp. 256–259 (2009). https://doi.org/10.1109/WGEC.2009.68

8. Sokolinsky, L.B., Sokolinskaya, I.M.: Scalable Method for Linear Optimization of Industrial Processes. In: Proceedings - 2020 Global Smart Industry Conference, GloSIC 2020. pp. 20–26. Article number 9267854. IEEE (2020). https://doi.org/10.1109/GloSIC50886.2020.9267854

9. Sokolinskaya, I., Sokolinsky, L.B.: Scalability Evaluation of NSLP Algorithm for Solving Non-Stationary Linear Programming Problems on Cluster Computing Systems. In: Voevodin, V., Sobolev, S. (eds.) Supercomputing, RuSCDays 2017. Communications in Computer and Information Science, vol. 793. pp. 40–53. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71255-0_4

10. Gay, D.M.: Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Bulletin (13), 10–12 (1985)

11. Dhiflaoui, M., Funke, S., Kwappik, C., et al.: Certifying and Repairing Solutions to Large LPs How Good are LP-solvers? In: SODA'03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 255–256. Society for Industrial and Applied Mathematics, USA (2003)

12. Rose, D.J., Tarjan, R.E.: Algorithmic Aspects of Vertex Elimination on Directed Graphs. SIAM Journal on Applied Mathematics 34(1), 176–197 (1978). https://doi.org/10.1137/0134014

13. Bareiss, E.H.: Sylvester's Identity and Multistep Integer-Preserving Gaussian Elimination. Mathematics of Computation 22(103), 565 (1968). https://doi.org/10.2307/2004533

14. Middeke, J., Jeffrey, D.J., Koutschan, C.: Common Factors in Fraction-Free Matrix Decompositions. Mathematics in Computer Science 1–20 (2020). https://doi.org/10.1007/s11786-020-00495-9

15. Wiedemann, D.H.: Solving Sparse Linear Equations over Finite Fields. IEEE Transactions on Information Theory 32(1), 54–62 (1986). https://doi.org/10.1109/TIT.1986.1057137

16. Koch, T.: The final NETLIB-LP results. Operations Research Letters 32(2), 138–142 (2004). https://doi.org/10.1016/S0167-6377(03)00094-4

17. Applegate, D.L., Cook, W., Dash, S., Espinoza, D.G.: Exact solutions to linear programming problems. Operations Research Letters 35(6), 693–699 (2007). https://doi.org/10.1016/j.orl.2006.12.010

18. The GNU Multiple Precision Arithmetic Library. https://gmplib.org

19. Panyukov, A.V., Gorbik, V.V.: Using massively parallel computations for absolutely precise solution of the linear programming problems. Automation and Remote Control 73(2), 276–290 (2012). https://doi.org/10.1134/S0005117912020063

20. Gleixner, A.M., Steffy, D.E., Wolter, K.: Iterative refinement for linear programming. INFORMS Journal on Computing 28(3), 449–464 (2016). https://doi.org/10.1287/ijoc.2016.0692

21. Blumenson, L.E.: A Derivation of n-Dimensional Spherical Coordinates. The American Mathematical Monthly 67(1), 63–66 (1960). https://doi.org/10.2307/2308932

22. Sokolinsky, L.B.: BSF: A parallel computation model for scalability estimation of iterative numerical algorithms on cluster computing systems. Journal of Parallel and Distributed Computing 149, 193–206 (2021). https://doi.org/10.1016/j.jpdc.2020.12.009

23. Sokolinsky, L.B.: Analytical Estimation of the Scalability of Iterative Numerical Algorithms on Distributed Memory Multiprocessors. Lobachevskii Journal of Mathematics 39(4), 571–575 (2018). https://doi.org/10.1134/S1995080218040121

24. Sahni, S., Vairaktarakis, G.: The master-slave paradigm in parallel computer and industrial settings. Journal of Global Optimization 9(3-4), 357–377 (1996). https://doi.org/10.1007/BF00121679

25. Bird, R.S.: Lectures on Constructive Functional Programming. In: Broy, M. (ed.) Constructive Methods in Computing Science. NATO ASI Series F: Computer and Systems Sciences, vol. 55. pp. 151–216. Springer, Berlin, Heidelberg (1988)

26. Sokolinsky, L.B.: BSF-skeleton: A Template for Parallelization of Iterative Numerical Algorithms on Cluster Computing Systems. MethodsX 8. Art. no. 101437 (2021). https://doi.org/10.1016/j.mex.2021.101437

27. Gropp, W.: MPI 3 and Beyond: Why MPI is Successful and What Challenges It Faces. In: Traff, J.L., Benkner, S., Dongarra, J.J. (eds.) Recent Advances in the Message Passing Interface, EuroMPI 2012. Lecture Notes in Computer Science, vol. 7490. pp. 1–9. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33518-1_1

28. Kostenetskiy, P., Semenikhina, P.: SUSU Supercomputer Resources for Industry and Fundamental Science. In: Proceedings - 2018 Global Smart Industry Conference, GloSIC 2018, art. no. 8570068. p. 7. IEEE (2018). https://doi.org/10.1109/GloSIC.2018.8570068

29. Sokolinsky, L.B., Sokolinskaya, I.M.: FRaGenLP: A Generator of Random Linear Programming Problems for Cluster Computing Systems. In: Sokolinsky L., Zymbler M. (eds) Parallel Computational Technologies, PCT 2021. Communications in Computer and Information Science, vol 1437. pp. 164–177. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81691-9_12

DOI: 10.14529/jsfi210304

A Review of Supercomputer Performance Monitoring Systems

Konstantin S. Stefanov[1], *Sucheta Pawar*[2], *Ashish Ranjan*[2], *Sanjay Wandhekar*[2], *Vladimir V. Voevodin*[1]

High Performance Computing is now one of the emerging fields in computer science and its applications. Top HPC facilities, supercomputers, offer great opportunities in modeling diverse processes thus allowing to create more and greater products without full-scale experiments. Current supercomputers and applications for them are very complex and thus are hard to use efficiently. Performance monitoring systems are the tools that help to understand the efficiency of supercomputing applications and overall supercomputer functioning. These systems collect data on what happens on a supercomputer (performance data, performance metrics) and present them in a way allowing to make conclusions about performance issues in programs running on the supercomputer. In this paper we give an overview of existing performance monitoring systems designed for or used on supercomputers. We give a comparison of performance monitoring systems found in literature, describe problems emerging in monitoring large scale HPC systems, and outline our vision on future direction of HPC monitoring systems development.

Keywords: monitoring, supercomputers, performance monitoring, review.

Introduction

High Performance Computing is now one of the emerging fields in computer science and its applications. Top HPC facilities, supercomputers, offer great opportunities in modeling diverse processes thus allowing to create more and greater products without full-scale experiments. Current supercomputers are very complex, and their efficient usage is very complicated. One of the tools helping to understand the efficiency of supercomputing applications is the performance monitoring systems. They collect data on what happens on a supercomputer (performance data, performance metrics) and present them in a way allowing to make conclusions about performance issues in programs running on the supercomputer.

In this paper, a performance monitoring system is a software package that continuously gathers performance metrics for at least compute nodes of an HPC compute system. Data from other sources may be collected as well. Those data are then used to provide an insight into what is happening in a supercomputer from a performance viewpoint, for individual nodes, parts or the whole supercomputer or specific jobs. In this paper we do not consider as a performance monitoring system a system which is aimed at monitoring the health of the supercomputer, i.e. that the components of a supercomputer are in a state suitable for running jobs. Nagios [13] is an example of such a system. We also do not consider systems aimed at profiling and tuning the specific jobs and which collect performance metrics only for the job in consideration. PTP [53], Tau [46], and HPCTOOLKIT [23] are the examples.

This paper is organized as follows. In section 1 we give a brief overview of historic and current performance monitoring systems. In section 2 we compare the features of the systems described in section 1. In section 3 we try to outline the performance impact of large scale monitoring systems. In section 4 we focus on design directions of large scale monitoring systems. And then we give a conclusion.

[1] Lomonosov Moscow State University, Moscow, Russian Federation

[2] HPC-Tech Group, Centre for Development of Advanced Computing, Pune, India

1. An Overview of Performance Monitoring Systems

Let us introduce some terms to describe different monitoring systems in a consistent way.

A *node agent* is a part of a monitoring system that runs on compute nodes of an HPC cluster and gets data related to that node. If the data related to other parts of the supercomputer (like network equipment, storage systems, central servers, etc.) are needed, they may be obtained by some node agents or by some dedicated agents running not on a compute node. Sometimes a monitoring system is claimed to be *agentless*. Generally, it means that instead of a custom agent some standard part of an OS or a hardware being monitored is used, e.g. SNMP agent.

Data from node agents and agents collecting other data flow to a *central*, or *server part* of a performance monitoring system. In the server part, the data are processed and presented to a user via some interface (not necessarily a GUI).

Another important part of a performance monitoring system is data storage. It is used to store the performance data and to process them in a case when some past data are needed.

Performance metrics are data about performance. The most frequent type of data is numeric data. A *primary metric* is a metric obtained directly from a data source: Operating System, hardware, etc. CPU loads, free RAM are the examples. A *secondary* or *derived*, metric is a metric which values are calculated based on values of primary and/or other secondary metrics. For example, a maximum value of free RAM over some period of time is an example of a derived metric.

Performance data from agents can be transferred to the server part of performance monitoring systems in *push* or *pull* modes. A *push* mode is a mode when agents send data to the server part without a command (by their own schedule). A *pull* mode is when agents are queried for data by the server part. There may be mixed cases: for example, node agents work in push mode while data from network equipment are obtained via SNMP in pull mode. A more elaborate example of mixing the modes is given in [22].

Concrete performance monitoring systems will be considered further. The performance monitoring systems we selected for this section were chosen with the following guideline in mind. We selected a specific tool if there is a scientific paper describing it or it is referenced in a paper describing other tool and has an active website. We do not consider stale systems like CLUMON although it is referenced in other papers as we did not manage to find a paper describing it and its website is only available as a copy in the Internet Archive [1].

1.1. PARMON

PARMON [28] was introduced in 1998. It is a tool for monitoring a cluster of UNIX workstations. It was used in Center for Development of Advanced Computing (C-DAC) in India for PARAM 10000, a HPC cluster consisting of nodes with UltraSPARC CPUs working with Solaris OS.

PARMON includes two main components.

Parmon-server runs on every node (a node agent) and provides information about node performance metrics on request in a client-server fashion (pull mode).

Parmon-client is a custom Java-based GUI client which connects to *parmon-servers* running on nodes, retrieves information and presents it to a user. It can collect and present information on one specific node or on a set of nodes.

All performance data processing is done by the *parmon-client* online while running a GUI session for the user. There is no database for storing data for subsequent analysis.

1.2. SuperMon

SuperMon [41, 50], introduced in 2001, is described as 'a flexible set of tools for high speed, scalable cluster monitoring'. It can be considered as one of the first monitoring systems designed for HPC clusters, or at least the first which gained wide visibility. It was created in the Advanced Computing Laboratory of Los Alamos National Laboratory and tested on 128 nodes Alpha Linux cluster. It is not a full performance monitoring system as it provides only the way to obtain and aggregate performance data; no processing of those data is described.

SuperMon includes a Linux kernel module that produces performance monitoring data, *mon* – node level data server and *Supermon* – several node data concentrators. These components form a tree-like hierarchical structure in which *mon* retrieves data from the kernel module on the same node, *Supermon* retrieves data from several *mons* or *Supermons*.

SuperMon has some interesting features.

All its components (kernel module, *mon* and *Supermon*) speak the same text-based protocol. It uses s-expressions introduced in LISP programming language.

Its components form a hierarchy that can be used for multi-level aggregation of monitoring data thus making SuperMon scalable.

SuperMon was tested on a heterogeneous cluster. The cluster was composed of nodes with 1, 2 and 4 Alpha CPUs [50]. But this point is not discussed in SuperMon paper. One of the possible reasons is that the paper does not describe the part which processes monitoring data, only the means to get the data are described. And the specifics of monitoring a heterogeneous cluster should be in a data processing part.

SuperMon was tested for extremely high (even for today) data sampling rates up to 3500 Hz. Somewhat counterintuitive, SuperMon authors claim that the higher the sampling rate the less the perturbation it causes to user jobs. Unfortunately only the theoretical support for this claim is given and no experiments have been done.

Despite all those features SuperMon is just a tool for retrieving and aggregating data, no database that may store the data for future analysis is not described.

1.3. Ganglia

Ganglia is one of the oldest but still widely used cluster performance monitoring systems. Its description [37] was published in 2004 when Ganglia was quite mature. The oldest release announcement (`http://ganglia.info/?m=200202`) found on Ganglia website is the announcement of version 2.0.3 in February 2002. The website [4] shows an impressive list of organizations using Ganglia.

Ganglia can be used for clusters and federation of clusters. Its components are *gmond*, agent running on nodes, and *gmetad*, which aggregates data from *gmonds* or other *gmetads*.

Communication between *gmetad* and *gmond* is done via multicast UDP. This allows several data receivers to operate on data coming from a single data producer or a single set of producers from several nodes. *Gmetad* may poll for data and *gmonds* respond, or *gmonds* send data by themselves. Multiple *gmetads* from a tree hierarchy and communicate via TCP. All data are transmitted in XDR encoding. A special crafted client library communicates to *gmetad*. *gmetad* store the collected data using RRDtool [20], a tool for storing and visualizing time series data.

Ganglia also has a web front-end that can show collected data for all or part of the nodes.

Ganglia authors provide experimental data for the overhead (they call it local overhead) incurred by *gmond* running on compute nodes. They measure the overhead as a percentage of CPU load consumed by *gmond* and its size in physical and virtual memory. The reported numbers are up to 0.4 % of CPU load, 16 MB of physical RAM and 16.7 MB of virtual memory (maximum values for different clusters).

1.4. NWPerf

NWPerf [42] was introduced in 2004. At that time it was used on MPP2 [10], 977-node Linux 11.8 TFLOPS cluster located at Pacific Northwest National Laboratory (PNNL).

Generally, NWPerf data collecting part looks very similar to that of Ganglia. NWPerf uses agents running on compute nodes that send XDR-encoded data via multicast UDP to data receivers. Data are stored in PostgreSQL [18] relational database thus allowing using SQL for data processing.

NWPerf adds data about a job being run on a cluster to that database. By combining performance data and data about jobs it becomes possible to analyze the specific jobs performance and calculate performance metrics of jobs.

One of NWPerf design goals was to lessen the overhead (perturbation in time of jobs) caused by the monitoring system. NWPerf authors did quite an extensive experimental evaluation of such overhead. They used MPI collective operations slowdown as a measure of the influence of the monitoring system on user jobs. To reduce the overhead they synchronized the moment when all node agents wake up to collect and transmit performance data. As a result they claim that they can collect performance data once per minute causing not more than a 1 % slowdown of user jobs on a 1000+ node cluster.

1.5. Ovis-2

Ovis-2 [26], introduced in 2008, is a redesign of Ovis [16]. Ovis is a monitoring system with sophisticated analytic tools allowing setting complex conditions on collected metrics as a trigger for notification and reaction. Ovis-2 is a framework for performance cluster monitoring and analysis of performance data. It addresses scalability and fault-tolerance.

Ovis-2 contains *sheep* processes (node agents) that run on components and collect data. *Shepherd* is an aggregator which gets data from *sheep* and stores them in a database.

Sheep, when run, search for a *shepherd* in mDNS (local multicast name-resolving system) and register themselves with a random *shepherd*, which, in turn, may redirect to another *shepherd*. After registering *sheep* begin to push performance data. Each *sheep* has all possible data source library compiled-in and on startup tries to instantiate as many data sources as possible. *Shepherds*, in turn, aggregate data and store them into a replicated MySQL [12] database.

Scalability is achieved by distributing the load from many *sheep* between *shepherds*. Fault-tolerance is achieved by many *shepherds* which may take over other failed *shepherd*, and by replicated database storage.

GUI and analytical tools for collected data analysis are mentioned in the paper. They are oriented on analysis of the whole cluster health. The job-centric view is mentioned in the Future Work section of [26].

Ovis and Ovis-2 do not seem to be developed further, but one of their component, Lightweight Distributed Metrics Service (LDMS) became an independent tool (section 1.8).

1.6. TACC Stats and SUPReMM

TACC Stats [31, 32], introduced in 2011 by Texas Advanced Computing Center, is a package that collects performance-related data from compute nodes of a cluster and presents them in a job-centric view. TACC stats consists of four components: *monitor*, *pickler*, *analysis*, and *site*.

Monitor is a small modular executable aimed to collect performance data. It is run in the job prolog, every ten minutes, and the job epilog. *Monitor* stores the collected data locally on every node.

Pickler runs every 24 hours to collect the data saved by *monitors* on compute nodes and stores them in central storage in per-job files. The data are stored in Python pickle format.

Analysis is a set of tests and plotting routines that can be run on a set of jobs to show possible performance issues in tested jobs.

Site module is a web interface to the data provided by TACC stats.

As a part of SUPReMM project [27], TACC stats was integrated with Open XDMoD [15, 43], a tool aimed at providing per-job data from cluster scheduler. With such integration, Open XDMoD is able to enrich its data with the performance data from TACC stats.

To map jobs to software packages and other properties of the executable files used for the job, a XALT [25] tool is used. XALT can collect the data like the libraries used for build the binary used for the job, the compiler used to compile it, tries to determine the exact version of the software package used, etc.

1.7. Dataheap

Dataheap, presented in 2012 paper [33], is focused on combining performance monitoring data from compute nodes and other, mostly I/O-related, sources like RAID controllers, storage area networks (SAN) switches, parallel file systems, etc.

It has compute node agents which send performance data from nodes and agents which send performance data from other sources. All those data are processed to calculate secondary data and then stored in a database. MySQL and SQLite are mentioned as possible choices. Additional tools exist for access to stored data like standalone GUI application, PHP-based web interface, and a command-line tool.

Dataheap authors pay special attention to the calculating of secondary values (or derived metrics). These are values that are not received directly from some sources like OS and hardware (primary values) but are calculated based on them. Such secondary metrics can be aggregated like average over a time interval or updated when a single new value of primary metric arrives. The latter case is specifically considered by Dataheap authors. Dataheap can calculate such secondary metrics on-the-fly. A scheme is proposed for updating secondary values based on interpolation of primary values.

1.8. Lightweight Distributed Metrics Service (LDMS)

Lightweight Distributed Metrics Service (LDMS) [24] is the data collection, transport, and storage component of Ovis (section 1.5). LDMS, as a separate component, was introduced in 2014. At the time of publication it was used on The National Center for Supercomputing Applications (NCSA) Cray XE6/XK7 Blue Waters and Sandia National Laboratories Linux cluster Chama. Blue Waters consists of 24 648 nodes and Chama consists of 1296 nodes.

LDMS is comprised of *ldmsd* daemons which can be configured to run in either *sampler* or *aggregator* modes.

A *sampler* includes sampling plugins each combing a specific set of metrics. *Samplers* can run several sampling plugins. The sampling frequency is used-defined and can be reconfigured on-the-fly.

Aggregators collect data from *samplers* and/or other *aggregators* in pull mode. They can use TCP, InfiniBand RDMA or Crays Gemini RDMA transports. Aggregation frequency is set on startup and cannot be changed without a restart. *Aggregators* support failover connection to samplers to take over data pulling if another *aggregator* is down.

The collected data are stored by *aggregators* with storage plugins. Possible storage formats are MySQL, Comma Separated Value (CSV) files and Scalable Object Store (SOS) [5].

1.9. LIKWID Monitoring Stack

Originally, LIKWID [52] was a set of tools for getting hardware performance counters for a specific job running on an x86-based compute node. After some development by 2017, it was transformed to a LIKWID monitoring stack [44] (LMS) which collects performance data from nodes and presents them in a per-job view.

LMS contains a host agent which uses a component from the original LIKWID to obtain performance data. The data from nodes or other sources like servers are sent to a central router. The router receives a signal about job start and finish from the cluster resource management system. The router is responsible for tagging the performance data with job id tags and storing them in a database. Another function of the router is to pull performance data from the sources which do not support push mode of operations like Ganglia *gmond* and mix that data with the data received from sources using push mode.

All data are saved in InfluxDB [7] time-series database. The protocol used in communication between LMS components is the InfluxDB line protocol. The protocol is based on HTTP and is widely used. Hence adding a filter into the data processing chain should be quite easy.

The performance data stored in InfluxDB are tagged with hostnames and job ids. These data are used to represent the metrics on all or part of the cluster or specific jobs. Grafana [6] toolkit is used to create charts and dashboards and show them to a user.

1.10. Performance Co-Pilot

Performance Co-Pilot (PCP) [17, 40] is a performance monitoring toolkit tracing its history since 1999 and is still being actively developed. While not targeted specifically for HPC clusters and supercomputers, it is often compared to while describing HPC performance monitoring systems – that is the reason for including it in this review.

PCP includes a Performance Metrics Collector Daemon (*pmcd*), an agent running on hosts to be monitored. *Pmcd* includes several Performance Metrics Domain Agents (*PMDA*), which are dynamically loaded libraries with a specified API. Each PMDA is responsible for collecting metrics from a performance metric domain like a kernel, a database server, etc. PCP uses pull mode: a client application connects to *pmcd* and requests some metrics. *Pmcd* routes the request to the appropriate *PMDA* and returns the response.

One of the client applications included in PCP is *pmlogger*, which can create performance metrics archive. One interesting feature of PCP is the ability to replay the archive created by *pmlogger*, thus enabling to reproduce the events from the past.

Other modules of PCP include *pmie* which can make notification on rules for metric values; *pmie* to generate periodic reports and a PCP GUI package.

1.11. Examon

Examon [3, 29], which stands for Exascale Monitoring, is a framework for the performance monitoring of HPC systems. Its distinctive feature is that it uses MQTT [11] protocol as a transport for performance data.

Its main executable *pmu_pub* collects performance data and publishes it to the MQTT broker. Then the data are exported to KairosDB [8] (a NoSQL time-series database built on top of Apache Cassandra [35]) and can be retrieved or analyzed by other tools which are not part of the project.

As MQTT uses publish/subscribe model, other clients can use the data in parallel to storing them in a database to make other services like alerting.

1.12. DiMMon and TASC

Distributed Modular Monitoring (DiMMon) [51], introduced in 2015, is a framework aimed at creating performance monitoring configurations. It is designed to be modular in all aspects. It has several types of modules: sensors that collect performance data, processing modules that make some calculations with the data, communication modules that send or receive data to components of the system working on other nodes. But this distinction is purely logical: from the frameworks point of view all modules are the same and developed using the same API. This modular design allows to create different paths for different data or having several databases for data aggregated on different periods of time.

DiMMon is used as a data source in TASC [47, 48]. TASC (Tuning Application for Supercomputers) is a system for visualizing performance monitoring data and produce advice to users if there are some performance issues in their jobs. DiMMon pushes data to TASC, which uses PostgreSQL and MongoDB [21] to store performance data. Redash [19] is used for data visualization.

1.13. C-CHAKSHU

C-CHAKSHU, a multi-cluster monitoring platform, was introduced as a part of the software stack for the National Supercomputing Mission (NSM) [14], which is implemented by C-DAC and supported by the Ministry of Electronics and Information Technology (MeitY) and Department of Science and Technology (DST), Government of India, in the year 2019. It monitors multiple HPC/Supercomputing systems, which are geographically separated, from a single dashboard. Currently, this tool is deployed on HPC systems installed at different scientific institutions and research organizations across India under the NSM project. C-CHAKSHU designed intelligently to address and cater to the needs of different users ranging from system administrators, application developers, domain scientists, and system architects. It has scalability for the pre-exascale/ Petascale system.

This tool collects real-time system-wide performance metrics, compute nodes health assessment-related metrics with action, verification of essential service/daemons. In addition, it also collects job/application execution snapshots and related performance counters, information from resource managers, and presents them in a Dashboard.

In C-CHAKSHU, two major components are used, one is on the server node to pull data whenever needed. The second component runs on all compute nodes all the time which collects different system-wide metrics and other subsystems over a network. Furthermore, Compute Nodes process data on their own and insert it into a NoSQL database whereas the aggregation of data for system-wide monitoring is carried out by the single server. C-CHAKSHU only causes a negligible network overhead on the cluster management node, incurs no overhead on the computing nodes, and reveals insightful knowledge of how HPC components interact with each other.

C-CHAKSHU has a loosely coupled architecture that supports the integration of any third-party tool and different back-ends easily. This tool has more capability of analyzing system performance and identification of bottlenecks.

2. Comparison of Performance Monitoring Systems

In this section we will compare the performance monitoring systems from several points of view.

First, we will outline data sets available in performance monitoring systems. Second, we will outline similarities and differences in data path inside monitoring systems and in the modes used to obtain the data. Next, we will compare the methods used to measure an overhead incurred by performance monitoring systems. And finally, we will give a table summarizing the comparison.

2.1. Metrics Used for HPC Performance Monitoring

The full list of metrics collected by a monitoring system is generally not easy to obtain: it is not usually included in a paper. Frequently, the list can be mined from the documentation if it is available. And still the exact metrics often depend on OS, on the hardware, etc. So we will not try to provide the full exact lists of metrics. Instead, we will outline the general groups of metrics used for performance monitoring.

The first group is what NWPerf authors call 'vmstat like information': 'percent of time spent on kernel processes, percent of time spent on user space processes, swap utilization and swap blocks in and out, and block device KB in and out' [42]. In modern Linux-based OS and some UNIX variants these metrics are available via `/proc` interface. Generally these metrics include CPU usage parameters (up to 10 metrics); network interface counters (bytes in/out, packet in/out, errors), load average for 1, 5, and 15 minutes; various types of memory usage (free RAM, RAM active, RAM inactive, swap usage, etc.), block device statistics. This group is basic for any performance analysis, and all performance monitoring systems mentioned in section 1 provide such data.

The second group is metrics provided by Performance Monitoring Units (PMU), Performance Monitoring Counters (PMC), Hardware Monitoring Counters (HMC), or simply hardware counters. All these are the names for basically the same thing: a set of hardware resources in modern CPUs that can count hardware events which occur while the program is being executed. Examples of these metrics are: a number of CPU cycles, a number of executed operations, a number of floating-point operations, a number of last level cache misses, etc. While a CPU can support a large number of events (several hundreds) that can be counted in general, a number of events that can be counted simultaneously is small. For example, CPUs based on Intel Haswell microarchitecture have three fixed-purpose counters (an event that is tied to that counters cannot be changed) per thread and four general purpose counters (an event to count on them can be programmed) per thread. So a maximum of seven events per thread can be counted simultaneously or 11 if HyperThreading is switched off. This limitation can be relaxed with so-called multiplexing [39]: measuring metrics in a round-robin way and extrapolating the results. This leads to a loss of accuracy, which is usually tolerable. There are techniques aimed at improving the accuracy, e.g. [38].

The first mention of metrics from this group in context of performance monitoring we found is for NWPerf [42]. The mentioned metrics are ‘CPU Performance Counters including: percent of peak flops (for CPU 0 and 1), memory bytes per cycle (for CPU 0 and 1)’. Then we can mention TACC stats [32] which collects ’core and socket perf. counters’. Since that we may assume that every performance monitoring system can collect such data. With regard to PMCs we must pay special attention to LIKWID. Originally, LIKWID [52] was developed as a set of tools to deal with hardware performance counters on x86_64 CPUs. Later it was transformed to a full monitoring system [44].

Taking into account a relatively small number of performance metrics that can be gathered simultaneously even with multiplexing, it is important not to waste valuable resources and wisely choose the appropriate metrics to measure. Unfortunately, there is no common view of what metrics to collect. Hence, a decision should be made for every setup. For example, some ideas may be borrowed from top-down analysis methodology [54].

And the last group of performance metrics is data from other sources. It may be data from compute nodes gathered via IPMI. We place IPMI data in this group due to two reasons. First, it is usually gathered by out-of-band means, not via compute node agent. Second, these data are more relevant to health monitoring, not the performance monitoring per se. Another source for data in this group may be sources common for several components of a supercomputer system: network equipment (usually gathered via SNMP or InfiniBand-specific tools), shared storage, etc. As these metrics are influenced by several (or all) jobs simultaneously, a separate problem of correlating these data to jobs behavior arises, see, for example [34].

2.2. Data Paths in Performance Monitoring Systems

In this section we will outline data paths used in performance monitoring systems. A data path is a way that the performance data traverse in a performance monitoring system. This includes data being communicated from node agent to a server part, aggregating data, storing data for subsequent use and presenting the results to a user.

The first step in a data path in a performance monitoring system is passing data from compute nodes for subsequent processing. As we have said earlier, basically there are two modes for doing this: push, when the data are sent by compute node agents on its own, without a request, and pull, when the data are sent as a reply to a request for the data. Generally, push mode is more efficient and leads to more scalable solutions, as making a request incurs additional overhead compared to just sending the data. Due to this reason, most of the performance systems considered in section 1 use push mode. In fact, of the modern systems, only LDMS uses pull mode. LDMS pulls the data from compute nodes by means of RDMA, which allows to pull the data with very low overhead and without request-reply type interaction with the agent on compute nodes. This is achieved by the means of special interconnect hardware supporting such features like InfiniBand or Cray Gemini. LDMS can also be used in mixed modes [22]. Ganglia supports both modes. Its main mode is push, the data is sent by gmond working on compute nodes. Gmetad (server part of Ganglia) can request the data by sending a request. It is done to get recent data held in memory by gmonds and missed for some reason. When gmetad form a tree for serving large-scale clusters, the data between gmetad is transferred with pull mode: the upper-level gmetad requests the data from lower-level gmetad.

What happens after the data are collected from their sources (compute nodes and others) differs. PARMON was designed to make online analysis only. Its GUI can show only online data, no means for data storage and historical data analysis was provided. SuperMon is a set of tools for retrieving and aggregating data, no data storage is provided, and no tools for presenting collected data to users are mentioned in its description. All newer performance monitoring systems or analysis tools built on top of them provide data storage option. Gagnlia storage is based on RRDtool, which is a library for working with time-series data but cant be called a full database. Some monitoring systems use relational database management systems (RDBMS) like MySQL, PostgreSQL or SQLite. While RDMS are not very performant when dealing with time series data, which are the most part of monitoring data, use of RDBMS is justified because of their widespread and their familiarity to many developers. Other databases are used as well: TASC and C-CHAKSHU use MongoDB, LIKWID monitoring stack uses InfluxDB, Examon uses KairosDB. InfluxDB and KairosDB are specialized databases for time-series data.

Another step in processing monitoring data is calculating derived metrics. The most common type of derived metrics is one metric aggregated data. The data can be aggregated over a period of time for reducing data storage volume. Or the data from several nodes can be aggregated to get an overview of some part of the compute system. This may include a partition with a specific hardware, e.g. accelerators, or a part in a specific room, rack row, etc. An aggregation may be done over all nodes occupied by one compute job. In this case the goal of aggregation is to get the data related to one job or a set of jobs. Most common type of aggregation is the calculation of average, minimum, and maximum. Another type of metric that is derived from a single other metric is transforming counters to its derivative with time. Some metrics are counters, i.e. they only increase when some event happens. An example of such a metric is a counter of bytes received or sent via a network interface. Absolute values of such metrics are generally not interesting. What is useful for performance analysis is a time derivative of such a counter. In case of network interface bytes counter such a derivative will produce an amount of traffic passing via an interface in a time period (bytes/s).

Other derived metrics are calculated from several primary metrics. An example of such a metric is an Instructions Per Cycle (IPC) metric. It represents an average number of instructions a CPU executes in one cycle. It is quite a complex metric. To calculate it, one needs a time derivative of executed instructions counter, a time derivative of CPU cycles counters and to divide the former by the latter. In modern CPUs cycle frequency is not constant due to power saving settings, waiting for RAM access, etc., so the time derivative of CPU cycles counters is not constant. If both counters are retrieved at the same moment, calculating derivatives is not needed, one can simply divide differences of counter values. Other examples of metrics derived from several other metrics are an average network packet size, and FLOPS per Watt ratio.

Another difference between performance monitoring systems is where on the data path the derived metrics are calculated. We see three distinct places where it can be done. We also should note that not every derived metric can be calculated at any of these places.

The first possible solution is to calculate derived metrics directly at node agents immediately after the metrics have been obtained. This is feasible for derived metrics which can be calculated from primary metrics available at a single node. Such derived metrics include an average over a specific period of time or an average over all similar metrics from a node (e.g. percentage of time a core spent in user mode averaged over all cores). This approach is criticized for producing greater overhead and thus affecting compute jobs. Still this is used at least while calculating time derivatives from counters. DiMMon has modules that can calculate derivatives and averages on node agents.

The second possibility is to calculate derived metrics in a server part of a monitoring system after receiving data from node agents. This is done by the majority of monitoring systems. Note that for a big supercomputer this produces a serious requirement in compute power of the server part. As monitoring systems usually run on dedicated servers, this does not incur overhead visible to a main, computing part of a supercomputer, this requirement can be fulfilled with a required number of separate servers and generally is tolerable from a hardware requirements perspective. As we have already mentioned, sometimes sophisticated techniques are used to account for different data obtained at different timestamps [33, section 4]. Ganglia, which is built on RRDtool as data storage, uses RRDtool's built-in features to calculate derived metrics while storing data.

The last possibility is to calculate the derived metrics after the data are stored in a permanent storage. In this scenario, the data are retrieved from the database, the derived metrics are calculated, and the results are saved back to the database (or to another database). This approach is commonly used when calculating metrics related to compute jobs or long periods of time. C-CHAKSHU uses this approach for derived metrics and uses MongoDB non-relational database for scaling. From the systems described in section 1, the first paper that proposed such an approach is the description of NWPerf [42]. After that, all performance systems that produce per job data use this approach. This solution has a drawback that it incurs much load on a data storage thus requiring quite a capable storage system to be used for performance monitoring data.

MRNet [45], a tool for creating overlay networks operating in multicast/reduction manner and aimed at performing scalable calculations, is often mentioned as a possible solution to calculating derived metrics in performance monitoring systems. Still we are not aware of any real monitoring system that uses MRNet.

2.3. An Overhead Caused by Performance Monitoring Systems

In this subsection we will not directly compare the overhead of different performance monitoring systems. It is a very difficult task as no methodology exists to make such a comparison. We will rather try to describe methods which performance monitoring system authors use to measure overhead of their systems.

First, we have to admit that most of the monitoring system authors do not give any measurements regarding an overhead of a system. More common is to give general description like 'insignificant' or 'negligible' or not to say anything at all. Exceptions are the object of this description.

SuperMon authors make a theoretical study of what they call 'perturbation' from a monitoring system [50, section 4.1]. They make a conclusion that a node agent with a high peak sampling rate is preferable as it causes less perturbation when used with sampling rates much less than the peak rate. Hence peak sampling rate may be viewed as an indirect metric of monitoring system overhead. Unfortunately, SuperMon authors do not provide experimental data to support their claim.

Ganglia authors measure several value as overhead metrics [37, section 5.2]. They measure the percentage of CPU time consumed by Ganglia processes and their memory footprint.

They also provide data on network bandwidth consumed by communications between Ganglia components. They present data for different clusters and confederations of clusters.

NWPerf authors try to measure the overhead (‘perturbation effects’) directly [42, section 4]. They run a test which executes a cycle of MPI collective operations (All-to-All and AllReduce were used) and measure its execution time without NWPerf agent or with it running with different sampling rates. The conclusion made from these results is that in its normal configuration (data are obtained at 1-minute intervals) NWPerfs effects are ‘immeasurably small’.

The authors of LDMS provide extensive data on influence of LDMS on various benchmarks [24, section V]. They run more than 10 different benchmarks (artificial tests as well as real applications) and measure time changes with different LDMS modes. Overall conclusion is that LDMS obtaining data once per second does not really affect running jobs.

2.4. Comparison Summary

In Tab. 1 we provide a short comparison of different features of performance monitoring systems. The columns denote the following properties of the performance monitoring system:

- Year – when the system was introduced or when it was mentioned for the first time.
- Cluster size, nodes – a maximum number of cluster the system was tested on (per available information).
- Heterogeneous – if a system was tested on a heterogeneous cluster.
- PMC – if the system can collect Performance Monitoring Counters data.
- Data Storage – if the system has storage for performance metrics and what type of storage is used (if the information is available).
- GUI – if the system offers a Graphical User Interface.
- Per-job – can the system correlate the performance data to the jobs executed on the cluster.
- Push/pull – which mode for transmitting data does the system use.
- Reconfig on-the-fly – can the system be reconfigured without restart.
- Overhead measured – if the overhead produced by the system is measured and the results are available.

An empty cell means that no information was found. Plus and minus signs mean ‘Yes’ and ‘No’, respectively.

3. Problems with Monitoring Large Scale HPC Systems

This section describes the challenges with the current monitoring software.

3.1. Requirement of Multiple Tools

Currently, we are working with many monitoring tools like Ganglia, Nagios, XDMoD, C-CHAKSHU along with custom scripts in a single HPC facility.

Existing monitoring software can be categorized broadly into the following types.

- Tools that provide job level details through resource manager like XDMoD [15, 43], TACC-stats [31, 32], VisQueue [49].
- Tools that provide only node level stats that use third-party API/tools to get the metrics like Ganglia [37], PARMON [28], SuperMON [41, 50]. It may include proprietary tools like NVIDIA DCGM [2] for monitoring sub-system components like graphics card/storage/ High-performance networks like Infiniband.

Table 1. A comparison of performance monitoring features

System	Year	Cluster size, nodes	Heterogeneous	PMC	Data storage	GUI	Per-job	Push/pull	Reconfig on-the-fly	Overhead measured
PARMON	1998	48+	-	-	-	+	-	Pull	-	-
SuperMon	2001	128	+	-	-	-	-	Pull	-	±[1]
Ganglia	2002[2]		+		RRDTool	+	-	Mixed	-	+
NWPerf	2004	1000+	-	+	PostgreSQL	-	+	Push	-	+
Ovis-2	2008	920			MySQL	+	-	Push		-
TACC Stats & SUPReMM	2011	6400	+	+	Pickle	+	+	Pull	-	-
Dataheap	2012			+	MySQL SQLite	+	+	Push	-	-
LDMS	2014	24648	+	+	MySQL CSV SOS			Pull[3]	+	+
LIKWID Monitoring Stack	2017			+	InfluxDB	+	+	Push		
PCP	1999			+	Custom	+	+	Pull		
Examon		1022	+	+	KairosDB	+	+	Push		
DiMMon & TASC	2015	6300	+	+	PostgreSQL MongoDB	+	+	Push	+	
C-CHAKSHU	2019		+	+	MongoDB	+	-	Pull	-	-

[1] Only theoretical study given

[2] The first found release announcement date

[3] Generally, LDMS uses pull mode, but in some cases the mixed mode can also be used

- Tools like Nagios [13] that provide node component level health metrics and also provide alerting services and event-based recovery.

As per the study mentioned in section 1 and the broad category mentioned above, there is no single tool to achieve all aspects of monitoring, analysis, and alerting. Currently, we require multiple tools along with custom scripts to get comprehensive monitoring of the HPC systems.

3.2. Performance

The monitoring software should not introduce any substantial performance penalty and unpredictable noises to the HPC system. In-band communication increases overhead on compute nodes.

- A higher sampling rate for collecting metrics has a performance penalty on the overall monitoring system in addition to an effect on application execution and also poses a challenge on scalability. Many times out-of-band measurements do not enable high-frequency sampling [30] and we cannot increase the frequency of the measurement.
- Reading metrics from OS and temp files should not introduce any lock to user applications.

3.3. Large Data Storage

Storing the generated log/metrics requires a separate infrastructure. An enormous amount of data from a few hundreds to thousands nodes generating logs and various performance metrics.

If we want to capture comprehensive metrics and logs data can be of size to the tune of TBs per day. If we want to save it for off-line analysis and correlations for a few months, data storage requirements will be huge. Managing multiple databases/structures/formats as every monitoring tool requires different types. It is challenging to handle multiple databases in a single monitoring environment.

3.4. Non-dedicated Monitoring Infrastructure

Typically monitoring software deployment is an afterthought. We provision monitoring software on master or management servers already having dedicated workloads. As we evolve in our monitoring requirements, existing infrastructure becomes insufficient for dedicated monitoring systems. Traditional HPC facilities do not have dedicated hardware clusters with big data analytics capability along with deep learning for processing real-time events for monitoring infrastructure.

3.5. Data Duplication and Redundancy

Multiple monitoring software in the current HPC environment collect similar kinds of data and store them in different data formats/databases. Similarly, we collect the same kind of logs from multiple nodes/sources without any incremental information. This redundant data makes our monitoring infrastructure slow for querying, analyzing and processing. It also consumes unnecessary space in the storage.

3.6. Data Correlation

Monitoring systems retrieve data from all hardware subsystems and system software. As observed in the study presented in section 1, there is no correlation provided among different metrics or logs to capture the HPC system behavior in totality. There are also requirements for data analysis to be done to find out trends in system behavior over time.

3.7. Limited Metrics Set

Due to multiple limitations like out of band management interface [36], processor [38], or network, we are monitoring limited metrics. There is no universal set of metrics or standardization to collect specific metrics set from different subsystems of the HPC. Every HPC facility has its own custom set of metrics as per its need. A large-scale monitoring system requires new capabilities in metric-gathering.

4. Future Directions of Monitoring Systems Development

Going forward beyond the petascale systems, we will have thousands of compute nodes and many more components and subcomponents in a single HPC system. To monitor overall efficiency and get deep insight into the complete system will be a must.

4.1. Comprehensive Monitoring

Large supercomputing sites require a monitoring framework to get all kinds of monitoring information through simple API-based queries. The visualization of monitored data over a dashboard is more of a drag and drop type along with the integration of alerting mechanisms. It should be able to perform proactive actions based on monitoring events to improve system efficiency and failure. Currently, we are using multiple tools like Nagios, Ganglia, resource manager stats, and many custom scripts to monitor the HPC systems. Typically the same or similar kind of data across multiple monitoring tools are kept in different data formats/databases. The framework must remove the need for maintaining multiple copies of the same metric data and should be available to all the consumers of data through a single provider with redundancy. This has been further discussed in the data management section below.

We are monitoring many metrics in the isolation. We will see the deployment of methods for correlating different metrics in real-time to understand the holistic behavior of the system. It will also help us classify the HPC workload based on similar correlated behaviors.

4.2. Long Term Trend Analysis

As HPC systems are operational for multiple years, trend analysis over a longer period is necessary to understand behavioral changes in the overall system. They still do not give long-term statistics for drawing perceptions for policy decisions. Further, almost all of them lack the decision-making capability based on dynamic system events.

4.3. Scalable and Modular Framework

Creation of a scalable and modular framework for accommodating a large number of nodes and metrics. We keep on increasing parameters to be monitored as more sensors are available in the next-gen server and sub-systems.

- Hardware Infrastructure: dedicated cluster for monitoring infrastructure for real and offline monitoring.
- High Memory servers for In-memory real-time data analysis.
- Broker services: Message broker services for allowing multiple consumers to access the same data and avoiding data duplications.
- Ability to over-provision and failover capability.
- Lightweight collector daemons at compute side with a minimal performance penalty and use maximum out-of-band capabilities.
- Support for ingestion of data at variable time-interval with techniques to enable only meaningful and incremental information flow.
- We need to provision monitoring infrastructure before the establishment of the HPC system.

4.4. Data Management

Furthermore, data mining and reduction techniques will become a necessity in exascale to perform the on-the-fly information reduction that will be a requirement to deliver scalable, automated online performance analysis. As the system grows, it generates a large volume of data from various sub-systems. Identification of data storage and defining optimal data path in addition to data processing before storing.

4.5. Optimization

As the HPC system becomes bigger and the Monitoring system also becomes complex with multiple stages of collection, pre-processing, data segregation, aggregation, and multiple consumers of the data. We must perform an iterative way of optimization [36] of all the stages to remove any performance bottleneck in the overall monitoring systems. Understanding the overhead of each stage of the monitoring system and reduction of overhead in each stage will require optimization at each stage in an iterative fashion. Daemons or collectors installed at compute nodes should not introduce any noise or jitter for the application performance. Large data collection or spurt due to some event in the HPC system should not introduce any network overhead.

4.6. Resiliency Mitigation

Exascale systems are likely to experience even higher fault rates due to increased component count and density. Which component will fail and how it will impact the system is not known ahead of time which is important nowadays. Triggering resilience-mitigating techniques remains a challenge due to the absence of well-defined failure indicators. Examination of event logs can be part of a monitoring feature that can be used with a data mining framework for failure detection. Desh [30] framework for efficient failure prediction and enablement of proactive recovery mechanisms to increase reliability.

4.7. Visualization

Representation of meaningful data is also an important aspect to understand that gives performance statistics of HPC systems. Nowadays easily pluggable and drag and drop visualization technologies are adopted. Grafana [6] and Kibana [9] are tools that will be essential for the creation of a customized dashboard. Radar [36] kind of view will be useful to understand the complete picture of applications behavior during the execution.

Conclusion

Unlike other components of the HPC software stack monitoring is an afterthought in HPC cluster. Each HPC facility has its own list of components to be measured. Every supercomputing facility has expertise in different monitoring tools, and they use a lot of custom methods to measure cluster efficiency. There is a need for standardization and collaborative effort for monitoring tools to measure various hardware and different architectures in a similar technique. Data collection and store format should be compatible across HPC sites and future monitoring tools should be configurable on the HPC system without much effort.

Acknowledgements

The reported study was funded by the Russian Foundation for Basic Research and Department of Science and Technology, India, according to the research project No. 19-57-45020 (RFBR project number).

References

1. CLUMON. https://web.archive.org/web/20090517125016/http://clumon.ncsa.uiuc.edu/, accessed: 2021-06-16

2. Data Center GPU Manager Documentation. https://docs.nvidia.com/datacenter/dcgm/latest/index.html, accessed: 2021-07-19

3. ExaMon | Exascale Monitoring Framework for HPC. http://projects.eees.dei.unibo.it/monitoring/wordpress/, accessed: 2021-06-08

4. Ganglia Monitoring System. http://ganglia.info/, accessed: 2021-06-09

5. GitHub - ovis-hpc/sos: sos pre-release stable. https://github.com/ovis-hpc/sos, accessed: 2021-06-11

6. Grafana - The open platform for analytics and monitoring. https://grafana.com/, accessed: 2021-06-11

7. InfluxData (InfluxDB) | Time Series Database Monitoring & Analytics. https://www.influxdata.com/, accessed: 2021-03-17

8. KairosDB. https://kairosdb.github.io/, accessed: 2021-08-24

9. Kibana: Explore, Visualize, Discover Data | Elastic. https://www.elastic.co/kibana/, accessed: 2021-07-19

10. Mpp2 - Cluster Platform 6000 rx2600 Itanium2 1.5 GHz, Quadrics | TOP500. https://www.top500.org/system/173082/, accessed: 2021-05-28

11. MQTT - The Standard for IoT Messaging. https://mqtt.org/, accessed: 2021-06-17

12. MySQL. https://www.mysql.com/, accessed: 2021-06-11

13. Nagios - The Industry Standard in IT Infrastructure Monitoring. http://www.nagios.org/, accessed: 2021-06-18

14. National Supercomputing Mission. https://nsmindia.in/, accessed: 2021-07-19

15. Open XDMoD. https://open.xdmod.org/9.5/index.html, accessed: 2021-06-11

16. OVISWiki. https://ovis.ca.sandia.gov/index.php/Main_Page, accessed: 2021-06-11

17. Performance Co-Pilot. http://pcp.io/, accessed: 2021-06-11

18. PostgreSQL: The world's most advanced open source database. https://www.postgresql.org/, accessed: 2021-06-23

19. Redash helps you make sense of your data. https://redash.io/, accessed: 2021-06-17

20. RRDtool - About RRDtool. http://oss.oetiker.ch/rrdtool/, accessed: 2021-06-11

21. The most popular database for modern apps | MongoDB. https://www.mongodb.com/, accessed: 2021-06-17

22. Aaziz, O., Cook, J., Sharifi, H.: Push Me Pull You: Integrating Opposing Data Transport Modes for Efficient HPC Application Monitoring. In: 2015 IEEE International Conference on Cluster Computing. pp. 674–681. IEEE (2015). https://doi.org/10.1109/CLUSTER.2015.118

23. Adhianto, L., Banerjee, S., Fagan, M., et al.: HPCTOOLKIT: tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience 22(6), 685–701 (2010). https://doi.org/10.1002/cpe.1553

24. Agelastos, A., Allan, B., Brandt, J., et al.: The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014. pp. 154–165. IEEE (2014). https://doi.org/10.1109/SC.2014.18

25. Agrawal, K., Fahey, M.R., McLay, R., James, D.: User Environment Tracking and Problem Detection with XALT. In: 2014 First International Workshop on HPC User Support Tools. pp. 32–40. IEEE (2014). https://doi.org/10.1109/HUST.2014.6

26. Brandt, J.M., Debusschere, B.J., Gentile, A.C., et al.: Ovis-2: A robust distributed architecture for scalable RAS. In: 2008 IEEE International Symposium on Parallel and Distributed Processing. pp. 1–8. IEEE (2008). https://doi.org/10.1109/IPDPS.2008.4536549

27. Browne, J.C., DeLeon, R.L., Lu, C.D., et al.: Enabling comprehensive data-driven system management for large computational facilities. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. pp. 1–11. ACM, New York, NY, USA (2013). https://doi.org/10.1145/2503210.2503230

28. Buyya, R.: PARMON: a portable and scalable monitoring system for clusters. Software: Practice and Experience 30(7), 723–739 (2000). https://doi.org/10.1002/(SICI)1097-024X(200006)30:7<723::AID-SPE314>3.0.CO;2-5

29. Byford, N., Popov, S., Paterson, A.: Anomaly Detection in High Performance Computing Systems. In: Kos, L. (ed.) Summer of HPC 2020, pp. 12–14 (2020). https://summerofhpc.prace-ri.eu/wp-content/uploads/2020/12/SoHPC2020-reports.pdf

30. Das, A., Mueller, F., Siegel, C., Vishnu, A.: Desh: deep learning for system health prediction of lead times to failure in HPC. In: HPDC'18: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing. pp. 40–51. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3208040.3208051

31. Evans, T., Barth, W., Browne, J., et al.: Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In: Proceedings of the First International Workshop on HPC User Support Tools, HUST '14, New Orleans, Louisiana, USA, November 16-21, 2014. pp. 13–21. IEEE (2014). https://doi.org/10.1109/HUST.2014.7

32. Hammond, J.: Tacc stats: I/O performance monitoring for the instransigent. In: Invited Keynote for the 3rd IASDS Workshop. pp. 1–29. Austin, TX (2011)

33. Kluge, M., Hackenberg, D., Nagel, W.E.: Collecting Distributed Performance Data with Dataheap: Generating and Exploiting a Holistic System View. Procedia Computer Science 9, 1969–1978 (2012). https://doi.org/10.1016/j.procs.2012.04.215

34. Kluge, M., Hartung, M.: Mapping of RAID Controller Performance Data to the Job History on Large Computing Systems. In: 2014 International Workshop on Data Intensive Scalable Computing Systems. pp. 73–80. New Orleans, Louisiana, USA (2014). http://conferences.computer.org/discs/2014/papers/7038a073.pdf

35. Lakshman, A., Malik, P.: Cassandra. ACM SIGOPS Operating Systems Review 44(2), 35–40 (2010). https://doi.org/10.1145/1773912.1773922

36. Li, J., Ali, G., Nguyen, N., et al.: MonSTer: An Out-of-the-Box Monitoring Tool for High Performance Computing Systems. In: IEEE International Conference on Cluster Computing, CLUSTER 2020. pp. 119–129. IEEE (2020). https://doi.org/10.1109/CLUSTER49012.2020.00022

37. 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). https://doi.org/10.1016/j.parco.2004.04.001

38. Mathur, W., Cook, J.: Improved Estimation for Software Multiplexing of Performance Counters. In: 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. vol. 2005, pp. 23–34. IEEE (2005). https://doi.org/10.1109/MASCOTS.2005.34

39. May, J.: MPX: Software for multiplexing hardware performance counters in multithreaded programs. In: Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001. p. 8. IEEE (2001). https://doi.org/10.1109/IPDPS.2001.924955

40. McDonnel, K.: System Level Performance Management (1999). http://mirror.linux.org.au/pub/linux.conf.au/1999/

41. Minnich, R.G.: Supermon: High-Performance Monitoring for Linux Clusters. In: 5th Annual Linux Showcase & Conference 2001, Oakland, California, USA, November 5-10, 2001. USENIX Association, USA (2001)

42. Mooney, R., Schmidt, K., Studham, R.: NWPerf: a system wide performance monitoring tool for large Linux clusters. In: 2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935). pp. 379–389. IEEE (2004). https://doi.org/10.1109/CLUSTR.2004.1392637

43. Palmer, J.T., Gallo, S.M., Furlani, T.R., et al.: Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. Computing in Science & Engineering 17(4), 52–62 (2015). https://doi.org/10.1109/MCSE.2015.68

44. Rohl, T., Eitzinger, J., Hager, G., Wellein, G.: LIKWID Monitoring Stack: A Flexible Framework Enabling Job Specific Performance monitoring for the masses. In: 2017 IEEE International Conference on Cluster Computing, CLUSTER 2017. pp. 781–784. IEEE (2017). https://doi.org/10.1109/CLUSTER.2017.115

45. Roth, P., Arnold, D., Miller, B.: MRNet: A Software-Based Multicast/Reduction Network for Scalable Tools. In: Proceedings of the ACM/IEEE SC2003 Conference on High Performance Networking and Computing. p. 21. IEEE (2003). https://doi.org/10.1109/SC.2003.10039

46. Shende, S.S., Malony, A.D.: The Tau Parallel Performance System. International Journal of High Performance Computing Applications 20(2), 287–311 (2006). https://doi.org/10.1177/1094342006064482

47. Shvets, P., Voevodin, V., Zhumatiy, S.: Primary Automatic Analysis of the Entire Flow of Supercomputer Applications. In: Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists. pp. 20–32. CEUR Workshop Proceedings, Yekaterinburg (2018), http://ceur-ws.org/Vol-2281/

48. Shvets, P., Voevodin, V., Zhumatiy, S.: HPC Software for Massive Analysis of the Parallel Efficiency of Applications. In: Parallel Computational Technologies. PCT 2019. Communications in Computer and Information Science, vol. 1063, pp. 3–18. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-28163-2_1

49. Solis, A.J., Foss, G., Jansen, C., Stelmaszek, M.: VisQueue. In: Practice and Experience in Advanced Research Computing. pp. 293–298. ACM, New York, NY, USA (2020). https://doi.org/10.1145/3311790.3396618

50. Sottile, M., Minnich, R.: Supermon: a high-speed cluster monitoring system. In: 2002 IEEE International Conference on Cluster Computing, CLUSTER 2002. pp. 39–46. IEEE (2002). https://doi.org/10.1109/CLUSTR.2002.1137727

51. Stefanov, K., Voevodin, V., Zhumatiy, S., Voevodin, V.: Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). In: Sloot, P., Boukhanovsky, A., Athanassoulis, G., Klimentov, A. (eds.) 4th International Young Scientist Conference on Computational Science. Procedia Computer Science, vol. 66, pp. 625–634. Elsevier B.V. (2015). https://doi.org/10.1016/j.procs.2015.11.071

52. Treibig, J., Hager, G., Wellein, G.: LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In: 2010 39th International Conference on Parallel Processing Workshops. pp. 207–216. IEEE (2010). https://doi.org/10.1109/ICPPW.2010.38

53. Watson, G.R., Frings, W., Knobloch, C., et al.: Scalable Control and Monitoring of Supercomputer Applications Using an Integrated Tool Framework. In: 2011 40th International Conference on Parallel Processing Workshops. pp. 457–466. IEEE (2011). https://doi.org/10.1109/ICPPW.2011.53

54. Yasin, A.: A Top-Down method for performance analysis and counters architecture. In: 2014 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2014. pp. 35–44. IEEE (2014). https://doi.org/10.1109/ISPASS.2014.6844459

DOI: 10.14529/jsfi210305

Administration, Monitoring and Analysis of Supercomputers in Russia: a Survey of 10 HPC Centers

Vadim V. Voevodin[1], *Roman A. Chulkevich*[2], *Pavel S. Kostenetskiy*[2], *Vyacheslav I. Kozyrev*[2], *Anton K. Maliutin*[3], *Dmitry A. Nikitenko*[1], *Sergey G. Rykovanov*[3], *Artemiy B. Shamsutdinov*[2], *Yurii N. Shkandybin*[3], *Sergey A. Zhumatiy*[1]

Supercomputer technologies are in demand for solving many important and computationally-intensive tasks in various fields of science and technology. Therefore, it is not surprising that there are several dozen supercomputer centers only in Russia. However, the goals of creating such centers, as well as the range of tasks solved in them, can vary greatly, therefore the structure of supercomputers and the policies for their usage can significantly differ. This leads to the fact that many supercomputer centers live an isolated life – the administrators of such centers tend to solve administration-related tasks on their own, despite the fact that solutions for many similar tasks have already been developed and applied in other centers. This can happen due to different reasons, but in any case, this situation could and should be improved. To do this, it is worth establishing a closer connection between supercomputer centers, which will allow more actively exchanging experience or jointly developing desired system software. In order to understand the current situation in this area, a survey was conducted of representatives among 10 large supercomputer centers in Russia, and its results are presented in this paper. Two relevant topics about using monitoring data in practice and real-life examples of supercomputer functioning improvement are also discussed here in more detail. Their vision on these topics is provided by the system administrators of HSE University, Skoltech and Moscow State University.

Keywords: supercomputer, high-performance computing, administration, survey, monitoring, performance.

Introduction

The high-performance computing (HPC) area is in great demand in the modern world [19]. There are various important problems in all major subject areas that cannot be solved without supercomputer technologies, and the number of such problems is constantly growing. For this reason, the HPC area itself also grows [11]. Each year supercomputers are becoming more powerful and more complex, and there are more and more of them, which contributes to the faster development of science and technology.

At the same time, the architecture of many supercomputers is noticeably different – in some cases, it is more important to use accelerators, but for someone a fast memory or a powerful interconnect is of main interest; in some organizations, a universal system is needed that allows effectively solving a variety of problems from different scientific areas, in others – a specialized system designed for extremely efficient solving of one specific class of problems [9]. Therefore, approaches to dealing with the issues of maintaining, monitoring and ensuring the efficient functioning of supercomputers also often differ.

All this leads to the fact that many supercomputer centers (SC) live a rather isolated life – if it is necessary to solve a certain administration-related task (e.g., implement new user access policies or quotas, configure file system or resource manager, monitor different aspects of compute

[1] Lomonosov Moscow State University, Moscow, Russian Federation
[2] HSE University, Moscow, Russian Federation
[3] Skolkovo Institute of Science and Technology (Skoltech), Moscow, Russian Federation

nodes utilization, etc.), new solutions are often developed and implemented, despite the fact that similar solutions have already been proposed in another center. This often happens because either a previously developed solution was not designed to be portable, or that solution is not directly suitable and needs to be adapted. In such cases, it is usually easier for administrators to develop their own solution than to try to adapt an existing one. Also, very often administrators were simply not aware that such a solution had already been found and implemented by someone. Moreover, administrators often have to solely struggle with the issues of deciding which system software is most suitable or how to optimally configure it, although exchange of experience with other colleagues could significantly simplify this task.

In our opinion, this situation can be improved. It can be achieved by exchanging experience in organizing the efficient functioning of SCs, as well as using the best practices from various SCs when developing new software tools and methods. For these purposes, at the end of year 2020, a working group on the analysis and quality assurance of supercomputer center functioning was created [3]. This group brings together system administrators and analysts from different Russian supercomputing centers. The circle of major interests of this group can be outlined as follows:

- efficiency of using supercomputer resources in general and executing HPC applications in particular;
- technologies for holistic monitoring of a supercomputer functioning as well as its individual components (both performance and operability issues are of interest);
- methods and system software for a comprehensive performance analysis of supercomputer applications;
- effective organization of the supercomputer work (project management, access rights, quotas, policies, etc.).

In order to study the current situation with the administration of supercomputers in practice, this working group has conducted a survey of 10 different Russian supercomputer centers on the above issues. The results, as well as a more detailed discussion of several questions raised in the survey, are given in this paper.

At the moment, only one similar work has been discovered [14], which has focused on studying operational data measurement, collection and analysis in 9 different large centers in the USA, Germany, Italy and Japan.

The main contribution of this paper is the collection and detailed analysis of various information about the current situation with the administration of modern Russian HPC centers. This information can be primarily useful in practice for system administrators who want to learn from the experience of others. And it can as well be useful for HPC center management and common users in order to understand the general picture of what is happening in this area and the relevance of the issues solved there.

The rest of the paper is organized as follows. Section 1 briefly describes how a survey has been conducted. In section 2, the most interesting results from the conducted survey are presented and analyzed. Section 3 discusses two actual topics about using monitoring data in practice and real-life examples of supercomputer functioning improvement in more detail. In this section, the vision on these topics is provided by the system administrators of HSE University, Skoltech (Skolkovo Institute of Science and Technology) and Lomonosov Moscow State University. The conclusions are made in the last section.

1. Conducting a Survey

The survey was completed by system administrators of 10 different supercomputer centers. Brief description of these centers (showing affiliation and the most powerful system in each case) is presented in Tab. 1. "# in Top50" column refers to the position of the corresponding system in the Top50 supercomputing rating [6].

Table 1. List of supercomputing centers that participated in the survey and corresponding most powerful systems with their parameters

#	Top HPC sites at each center	Performance, TFlop/s	Nodes	# in Top50
1	Lomonosov-2, Supercomputing center of M.V. Lomonosov Moscow State University, Moscow	max: 6669 peak: 8789.76	1696	2
2	Polytechnic RSC Tornado, Peter the Great St. Petersburg Polytechnic University, St. Petersburg	max: 910.31 peak: 1309	784	4
3	cHARISMa, HSE University, Moscow	max: 653.7 peak: 1003.2	48	6
4	Zhores, Skoltech, Moscow	max: 495.9 peak: 1011.6	82	8
5	Govorun, SKYLAKE component Joint Institute for Nuclear Research, Dubna	max: 312.62 peak: 463.26	104	12
6	Lobachevsky, Lobachevsky Nizhni Novgorod State University, Nizhny Novgorod	max: 289.5 peak: 573	180	13
7	Tornado SUSU, South Ural State University, Chelyabinsk	max: 288.2 peak: 473.64	384	14
8	Uran, Krasovskii Institute of Mathematics and Mechanics, UB RAS, Ekaterinburg	max: 194.77 peak: 326.85	76	18
9	NKS-1P, SSCC, SB RAS, Novosibirsk	max: 85.45 peak: 136.94	48	37
10	Polus, M.V. Lomonosov Moscow State University, Faculty of Computational Mathematics and Cybernetics, Moscow	max: 40.39 peak: 55.84	5	N/A

The survey was conducted using Google Forms and included 25 questions concerning the following topics:

- monitoring data collection;
- usage of different system software;
- automation of administrative routines;
- understanding the general behavior of supercomputer systems;
- resource management;
- supercomputer support.

Most of the questions had multiple predefined options to choose from, but almost always there was an opportunity to give your own answer. There were cases when several administrators from one center completed the survey; their answers were combined and presented as one.

2. Analyzing Survey Results

This section discusses the most interesting topics covered in the survey. A presentation with all questions and answers (in Russian) can be found on the web-site of the working group [5].

The first two questions of the survey were related to the data collection, on the compute nodes (Fig. 1) and the engineering infrastructure (Fig. 2) correspondingly.

Figure 1 shows that most of the supercomputer centers collect information about CPU load and file systems usage – 7 out of 10 administrators have chosen these options. A little less (6 out of 10) centers collect more information about memory usage (DRAM, I/O), network load, power consumption and temperature. In our opinion, these characteristics are quite expectedly the most popular ones, since they reflect in general the usage of main supercomputer resources as well as basic operability characteristics. It is interesting to notice that 6 out of 10 centers can analyze the performance of user applications since they bind the collected monitoring data to the tasks launched on compute nodes.

It is also worth noting that not so many supercomputer centers are interested in getting more detailed information about node and/or task behavior using processor counters. Example of what data could be of interest and how such data can be used is provided in section 3.1.3.

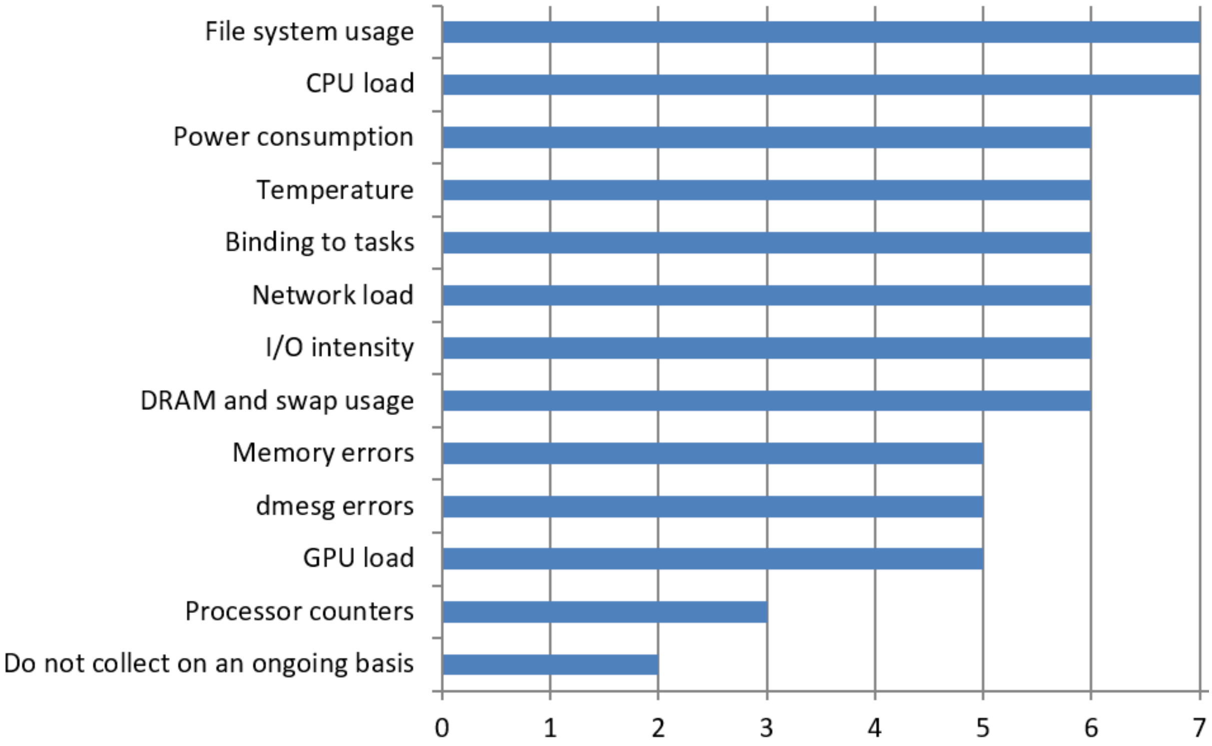


Figure 1. Answers to the question “What data do you collect on the compute nodes?”

The statistics shown in Fig. 2 are quite expected as well. The most crucial question for administrators is the condition and operability of the supercomputer, so almost every center collects data on the temperature (8 out of 10), estimated battery life (7 out of 10) and humidity (7 out of 10). The security question is less crucial but still important, that is why different centers collect indoor video (4 out of 10) as well as information about who entered the room (3 out of 10) and from motion sensors (1 out of 10).

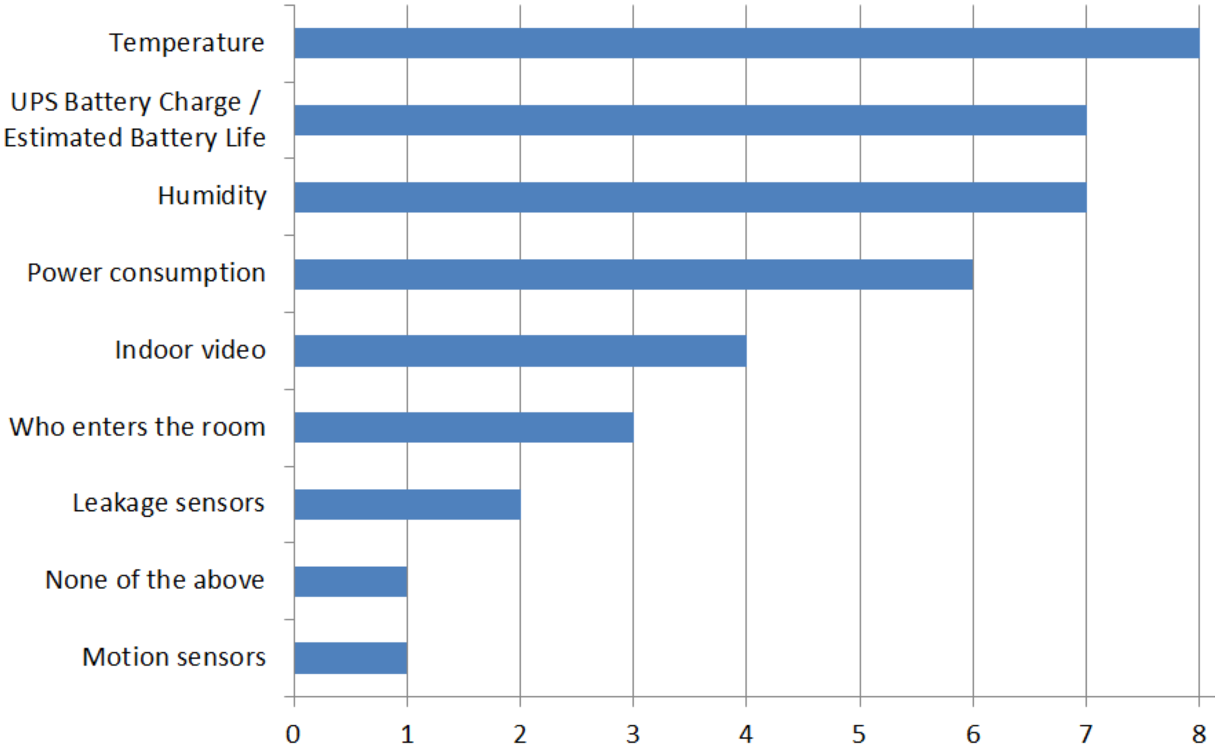


Figure 2. Answers to the question “What data do you collect about the engineering infrastructure?”

One of the most interesting topics to investigate was to find out about the usage of different system software that can help to obtain more insights about the state, behavior and usage of a supercomputer. For example, it was interesting to study whether there are generally accepted ready-to-use solutions that are widely used, and what proprietary solutions are used. The survey included questions about the use of the following software:

- monitoring systems;
- database management systems (DBMS);
- data visualization;
- data stream processing;
- data analysis.

Every question had a predefined list of the most accepted software, with the ability to specify other solutions as well. For each option, it was necessary to indicate how it is used: 1) as a main solution; 2) as a supplementary tool; 3) planned to be used in future.

First, let us take a closer look at monitoring systems. Figure 3 provides the distribution of answers to the following question: “What monitoring systems do you use to collect data on the work of your supercomputer center?”.

These results were quite surprising for us. As seen, the most popular option is the “proprietary solutions”, which means that administrators use their own developed software more often than any of existing ready-to-use monitoring systems. In our opinion, this is a rather alarming situation, since the development of your own monitoring system is most often cumbersome. The question arises why administrators are not satisfied with ready-made solutions and whether

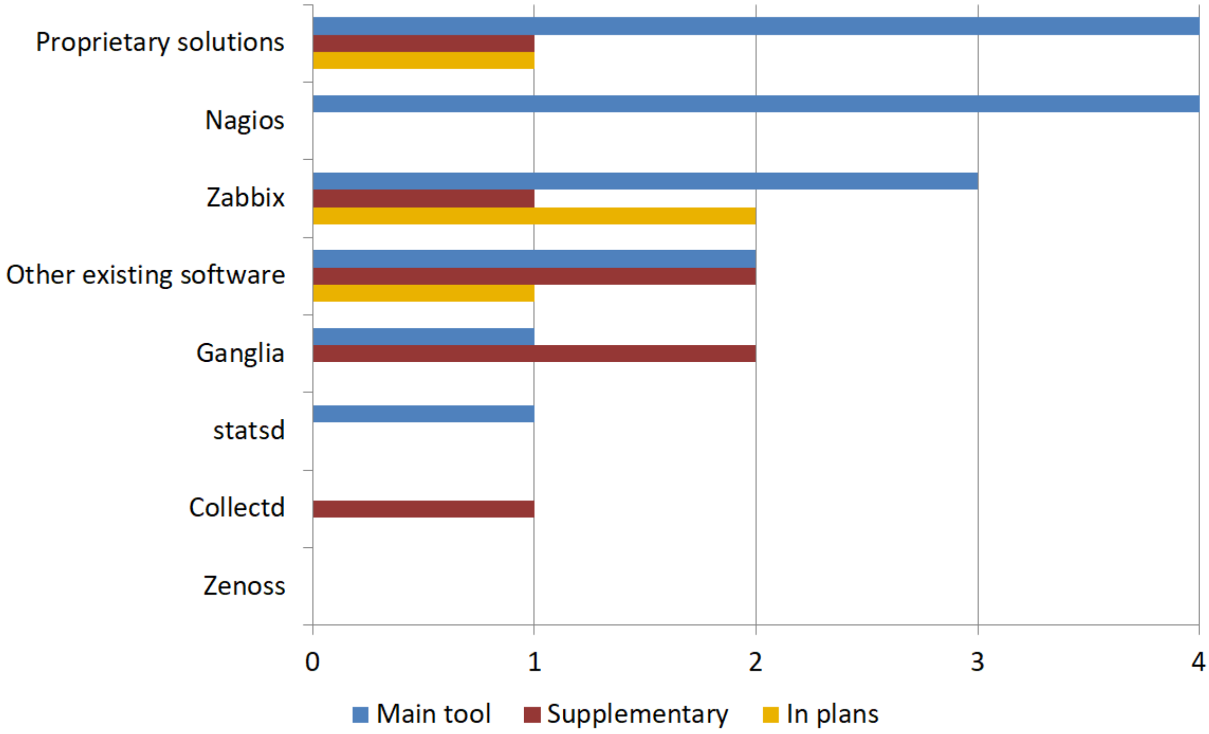


Figure 3. Information about monitoring systems used in different supercomputer centers

it is possible to simplify the solution of this problem by exchanging experience or sharing the developed ideas. This issue is further discussed in the section 3.1.

Other results in Fig. 3 show that Nagios and Zabbix are by far the most popular existing solutions, both mentioned 4 times as being used (2 centers also plan to use Zabbix in future). It should be mentioned that Telegraf and Icinga were mentioned twice each in the "other existing software" category, what makes them more popular than statsd and Collectd.

The situation with DBMS used to store data on SC work is shown in Fig. 4. SQL-related databases are traditionally very popular, with MySQL/MariaDB on the first place (used by 4 out of 10 centers) and PostgreSQL on the second place (3 out of 10). Time-series databases are used quite rarely – only 3 centers in total claim to use them, with InfluxDB being the most popular. This was quite surprising: they were expected to be used more often since most of the collected data is presented in the form of time series. Also, MongoDB is used by 2 centers, but only as the supplementary one. The ElasticSearch solution was also expected to be used more often, since the ELK stack (ElasticSearch, Logstash, Kibana) seemed to be quite famous for solving similar tasks, but it turned out not to be the case.

The next question was about visualization systems that help to present the information about different aspects of supercomputer functioning. The situation here is quite expected: Graphana is by far the most commonly used solution (5 out of 10 centers), followed by Kibana (2 centers). All other mentioned software – Redash, Jupyter Notebook, and two commercial solutions (IBM Blue Gene Navigator and HPE CMU) – is used only in one center each. Only one center uses its own developed solution in this case.

The situation with data stream processing tools is even more expected – most of the centers do not use such tools. This can be explained by the fact that the collected data streams are currently not so large and complicated that they can be processed without such tools. Only two ready-to-use solutions were mentioned – kapacitor and RabbitMQ, used by one center each. Also,

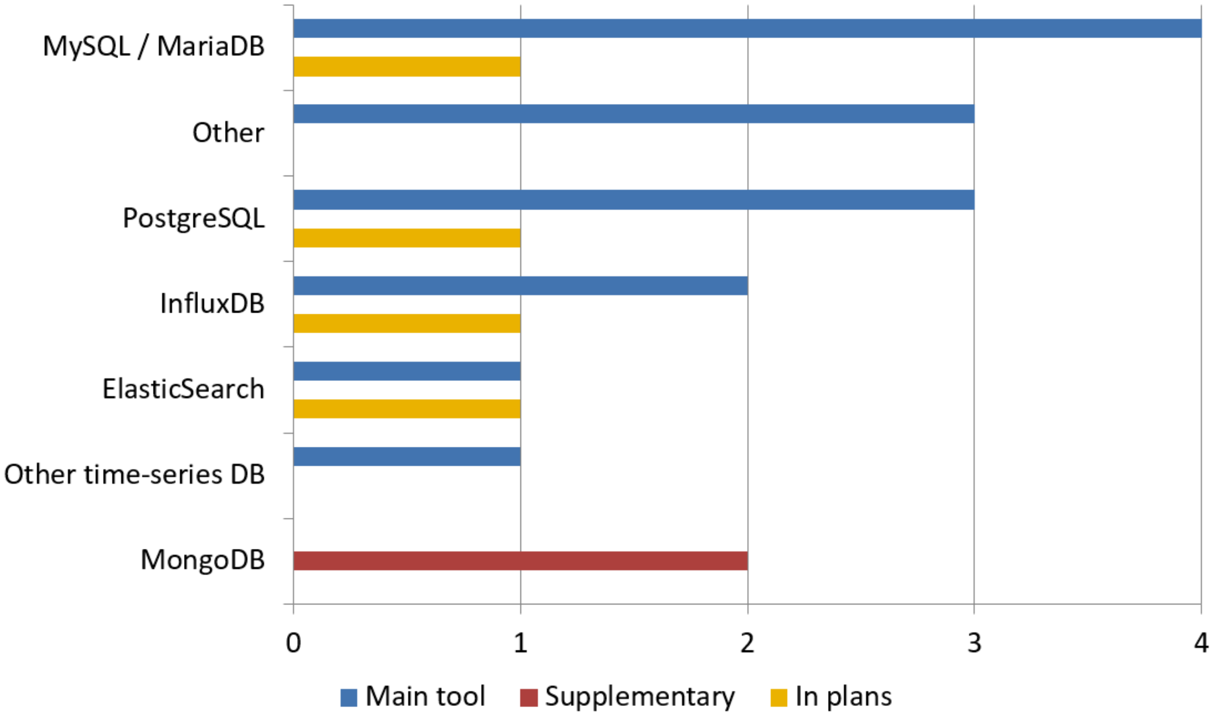


Figure 4. Information about DBMS systems used in different supercomputer centers

two centers claim to apply their own developed solutions, and in this case it would be interesting to drill into further details to find out why existing software does not fit their needs.

Interesting answers were received to the question "What software do you use to analyze data on supercomputer work?", shown in Fig. 5. By far the most common answer in this case is "proprietary solutions", with 4 centers using them as main or supplementary solutions. This is caused by different reasons, and, in our opinion, one of the main reasons is that there is no existing solution that can fully address this issue. All existing software mentioned in Fig. 5 help to process and analyze data, but they require additional work (sometimes a lot) to integrate them into the whole flow of working with supercomputer data. This shows that this area is currently the least developed one among those considered, and it is worth paying special attention to the joint solution of this issue.

The last question we want to discuss in this section is a general one – "What do you lack for a more complete understanding of the state of your supercomputer center?". The distribution of answers is shown in Fig. 6. The most important conclusion from the data presented is that not a single administrator responded that they had everything they needed to fully understand the behavior of their supercomputers. This means that the potential for development in this area is great, and it is worthwhile to join efforts to improve this situation. Otherwise, the distribution of answers in general is quite natural: many centers are already collecting data on the correctness of the supercomputer's operation, so this option received the least number of votes. At the same time, the main problem that prevents administrators from understanding in more detail the state of supercomputers is the lack of time and people to help conduct such an analysis. We would like to point out that many centers also want more "intellectual" data analysis software, which once again emphasizes the lack of such ready-to-use solutions and their relevance in practice.

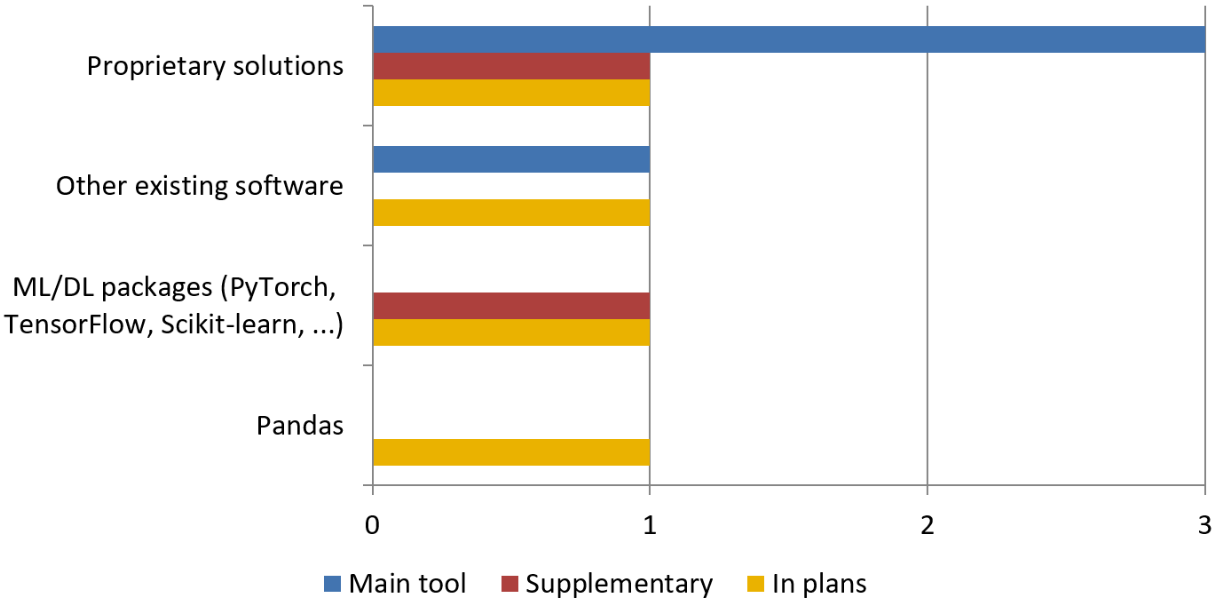


Figure 5. Information about data analysis systems used in different supercomputer centers

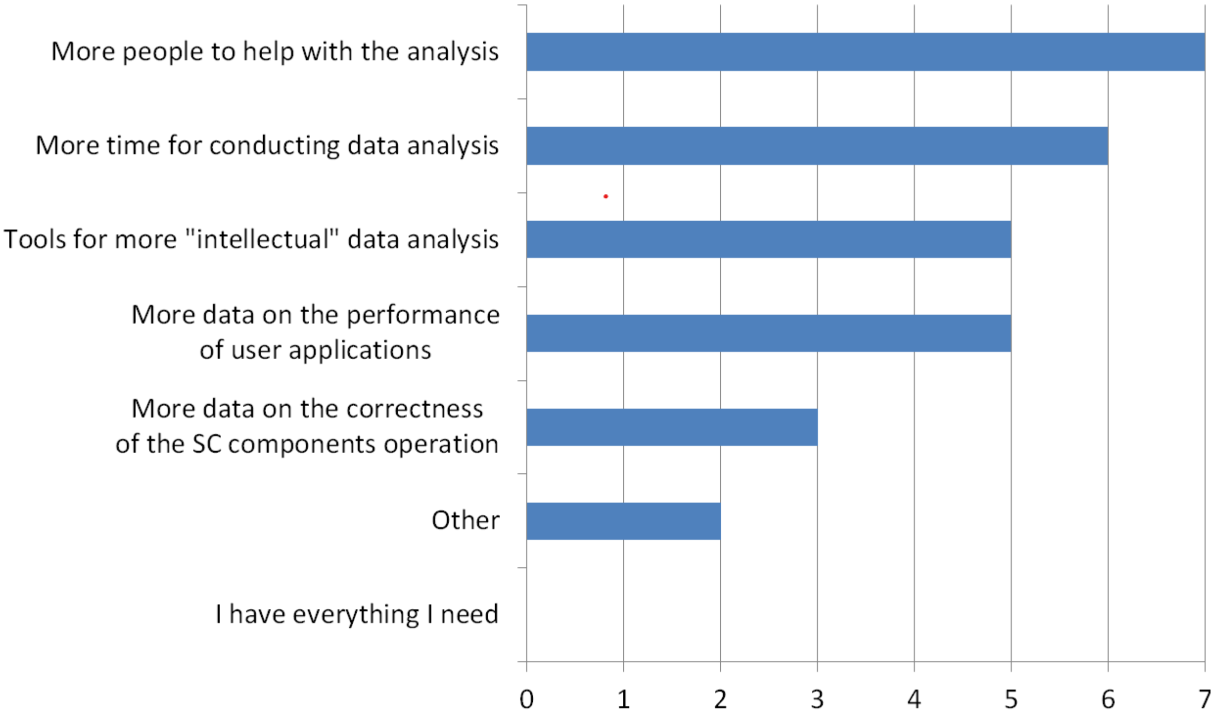


Figure 6. Answers to the question “What do you lack for a more complete understanding of the state of your supercomputer center?”

3. Diving into Details of Supercomputer Administration in Practice

In this section, the following questions of interest raised in the survey are considered in more detail:

1. Using monitoring data in practice.
2. Real-life examples of supercomputer functioning improvement.

In order to get a more complete picture of each of them, their vision on these issues is provided by the system administrators of HSE Univesity, Skoltech and Moscow State University.

3.1. Using Monitoring Data in Practice

One of the main questions that arises when administering supercomputers is what data to collect about its health and performance, how to collect this data and what insights can be obtained from it. We will talk about this in this section.

3.1.1. HSE University

HSE University has developed its own HPC cluster monitoring system. The key feature of the system is the interactive dashboard, which displays the state of the entire supercomputer on one screen (see Fig. 7). The system collects data from various sources, processes and logically combines it. This allows simultaneously visualizing the real load, the distribution of resources between users, the status of the task queue, and the health of computing nodes. The system's dashboard has functionality for managing the task queue. For example, the head of an HPC department can directly raise an urgent user task in the queue through the web interface, and the system will automatically recalculate the priorities of tasks and pass them to the task scheduler. Also, the system allows centrally changing the resource limits for different types of users (students, employees), which is convenient during holidays, when the cluster load decreases, and the restrictions on the number of computing resources used by users can be loosened.

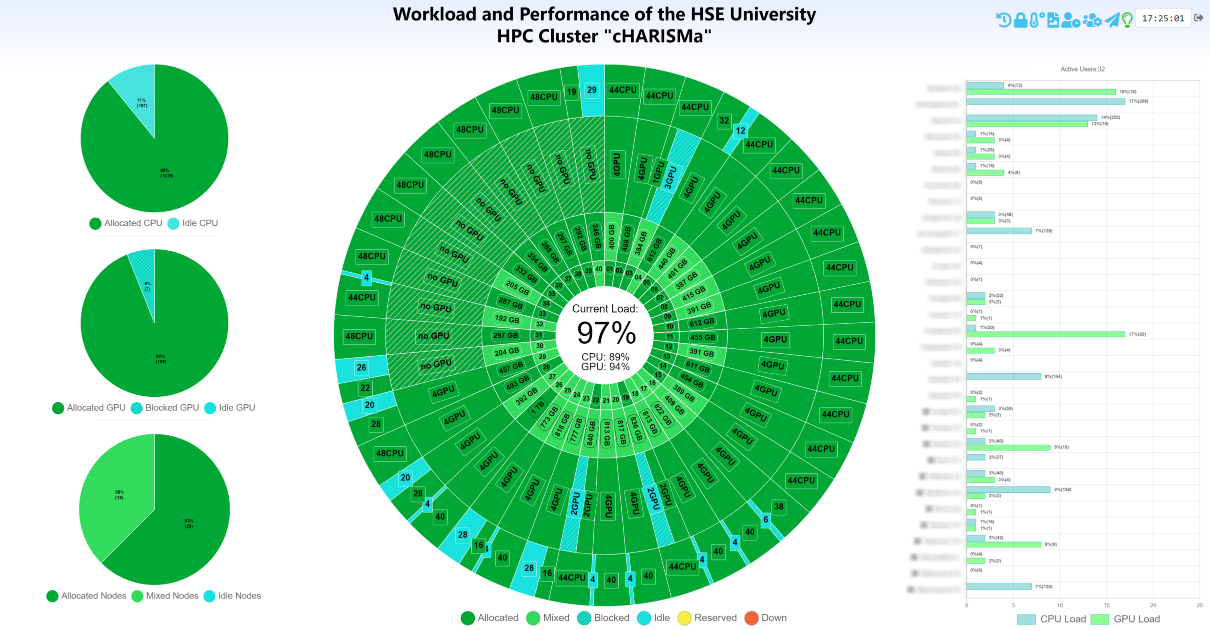


Figure 7. Monitoring system of the HSE university supercomputer facilities

Access to the monitoring system of the HSE University supercomputer center is available not only to the system administrators of the HPC center but also to the top management of the university. Different users of the monitoring system receive various functionality. The system allows managers to create summary graphs and reports on the use of the supercomputer.

With the help of the developed monitoring system, it was possible to detect and eliminate many non-optimal settings in the configuration of the supercomputer. As a result, many cluster optimizations were performed, some of which are listed below.

Running multiple tasks on the same node. The compute nodes in the HSE HPC cluster have a configuration that differs from many other high-performance installations. Instead of a large number of nodes with average performance and small memory, *cHARISMa* has powerful nodes with a large amount of RAM (up to 1.5 TB). Therefore, the minimum entity available to the user is not the entire computing node, but 1 processor core, or 1 GPU. As a result, up to 48 tasks of different users can run simultaneously on one node. Thus, it became possible to significantly increase the number of simultaneously solved tasks, which greatly increased the efficiency of the whole computing cluster.

Optimization of the queue system. The fact that the task manager in the HSE HPC cluster does not allocate the entire node, but specific cores or GPUs, can lead to the high fragmentation of the computational field. By default, SLURM tries to place new tasks on the least loaded nodes. As a result, the cluster can be loaded by 30–40 %, but a large number of tasks will wait in the queue, which requires many processor cores on one node. To reduce resource downtime, an improved task grouping (sched/backfill + pack-serial-at-end) was configured. Tasks are now placed in such a way as to maximize a load of already partially occupied dedicated compute nodes.

Protection of resources from accidental use. Another feature of the system is the protection from the possible accidental usage of other user's resources. By default, the control over which resources on the node are available to the user is carried out using the operating system environment variables. At the same time, the user can accidentally or intentionally change these variables and get access to resources that are not allocated to him. To solve this problem, isolation of tasks from each other on a specific compute node (group, etc.) was implemented. As a result, the executed task process sees only his computing resources and does not imply the existence of any others.

Fair distribution of computing resources. Monitoring the statistics in the monitoring system allowed us to discover that some users sometimes abuse the available computing resources, for example, when one user launches several hundred tasks of the same type, which occupy the cluster for a long time. As a solution, a limit on the number of resources used simultaneously by one user was implemented. For the convenience of administrators and managers, the ability to manage this restriction in the web interface of the monitoring system was added.

Protection against GPU blocking. Another point that the system monitors is the blocking of graphics accelerators. To perform a task on a graphics accelerator, the user needs at least one processor core. However, if the tasks of another user have occupied all the available processor cores on this node, then the graphics accelerators will not be available for allocation. To solve this problem, an additional task queue plugin was implemented. When a task appears that can block the graphics accelerator on the node, the plugin issues a warning and slightly changes the number of resources to avoid blocking.

Script automation. The administration of a cluster complex is usually a time-consuming process. The complexity is a consequence of the fact that a cluster consists of many compute nodes and network equipment. Automation of routine processes using scripts allows simplifying the administration of cluster. An example of script automation is the simple Supercomputer

Emergency Shutdown System (ESS), developed at HSE University for the *cHARISMa* HPC cluster. The system is essential because there are events that require immediate response:

- failure of the cooling system of the compute node;
- failure of the indoor air conditioning system;
- switching the UPS to battery mode;
- low level of battery power of the UPS.

The emergency shutdown system makes asynchronous requests and checks:

- The temperature at the input of the compute node (RedFish REST API). A high temperature indicates a general malfunction of the indoor air conditioning system.
- The temperature at the output of the compute node (RedFish REST API). A high temperature indicates a malfunction of the cooling system of a particular node.
- UPS input status and remaining battery life (SNMP). The lack of input voltage and low battery charge indicates a possible accident in the power supply system.

The developed ESS has successfully supplemented the local overheating prevention system built into the firmware of compute nodes. It automated and streamlined the process of shutting down the supercomputer in the event of an accident, preventing equipment breakdowns. The system has already been triggered during a power outage and has shown its effectiveness and usefulness.

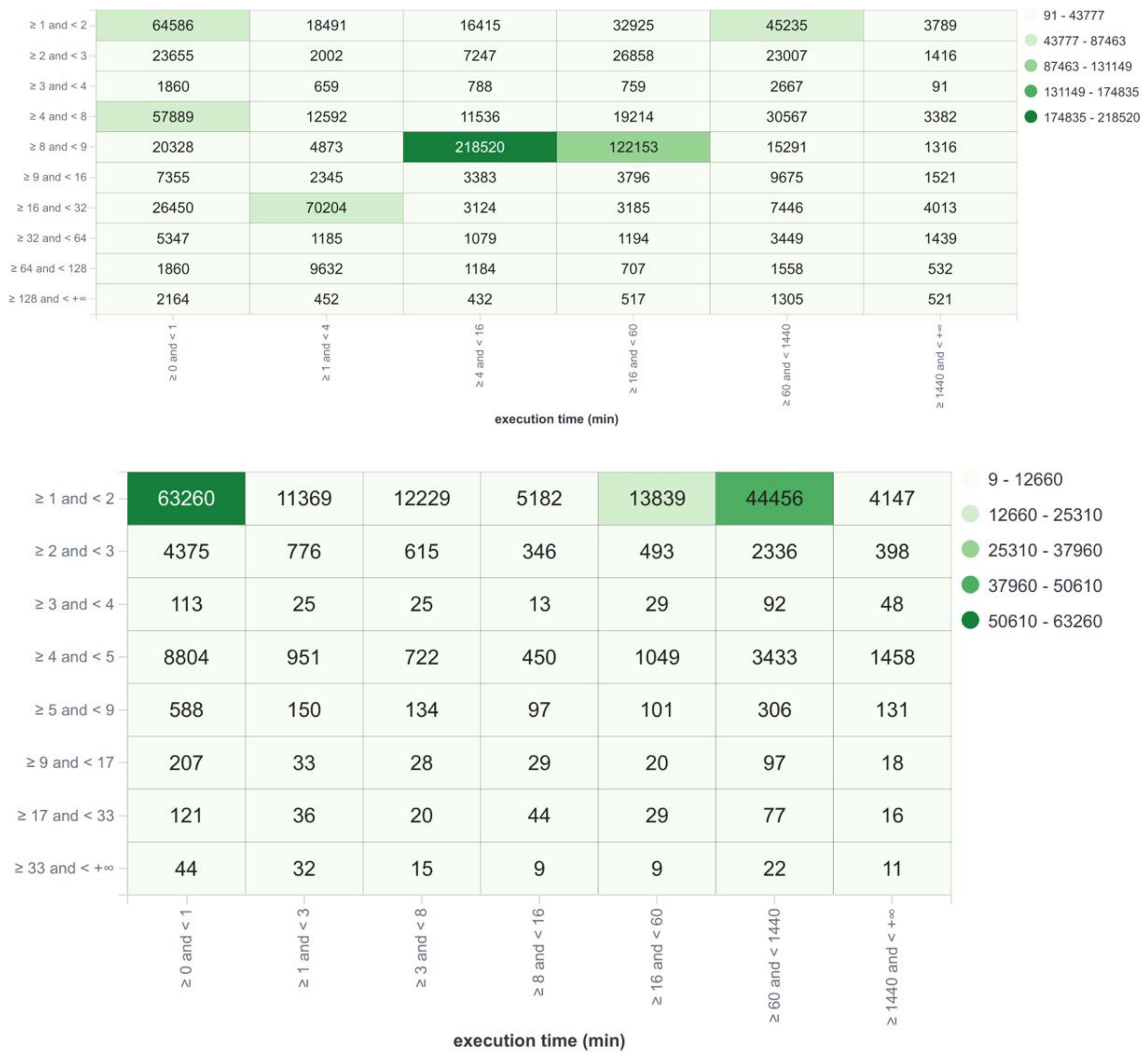
3.1.2. Skoltech

The range of tasks that a modern supercomputer solves goes beyond the scope of classical HPC tasks. In Skoltech researchers and engineers working on fundamental and applied science are faced with different types of problems. These might be high throughput computing (HTC) tasks that require a large number of single CPU cores or GPUs running independently, HPC tasks that require a large fraction of all resources to work on a single problem, or data-intensive tasks with ML/DL approach applied to solving them. The difficulty is that all types of tasks have to be solved effectively within a single supercomputer both from the user and from the economical/environmental perspectives (the latter can be described as the amount of time the system is running idle and amount of time the system is performing productive calculations). In Skoltech the task of increasing the "Zhores" supercomputer load efficiency has been gradually performed since the start of the supercomputing facility late in 2018 [23].

First, the inventory of tools that can obtain data on computing efficiency was performed giving the profile of typical tasks performed on a supercomputer and an estimate of their average efficiency in terms of resource utilization. The tools that were used for gathering analyzing data are the following:

- Slurm workload manager [22];
- Elasticsearch [10] for job properties aggregation;
- Kibana for data visualization;
- Zabbix for infrastructure monitoring;
- a home brewed tool for a more detailed "Zhores" cluster; monitoring [24];
- users surveys and individual interaction.

An example of the heatmap of number of launched jobs as a function of running time and requested resources for the "Zhores" cluster is presented in Fig. 8. This allowed us to create a "profile" of an average user, find out the needs of the users and tune the Slurm queues correspondingly (see sec. 3.2.2 for further details). After an initial assessment of the collected data, it



	≥ 0 and < 1	≥ 1 and < 4	≥ 4 and < 16	≥ 16 and < 60	≥ 60 and < 1440	≥ 1440 and < +∞
≥ 1 and < 2	64586	18491	16415	32925	45235	3789
≥ 2 and < 3	23655	2002	7247	26858	23007	1416
≥ 3 and < 4	1860	659	788	759	2667	91
≥ 4 and < 8	57889	12592	11536	19214	30567	3382
≥ 8 and < 9	20328	4873	218520	122153	15291	1316
≥ 9 and < 16	7355	2345	3383	3796	9675	1521
≥ 16 and < 32	26450	70204	3124	3185	7446	4013
≥ 32 and < 64	5347	1185	1079	1194	3449	1439
≥ 64 and < 128	1860	9632	1184	707	1558	532
≥ 128 and < +∞	2164	452	432	517	1305	521

	≥ 0 and < 1	≥ 1 and < 3	≥ 3 and < 8	≥ 8 and < 16	≥ 16 and < 60	≥ 60 and < 1440	≥ 1440 and < +∞
≥ 1 and < 2	63260	11369	12229	5182	13839	44456	4147
≥ 2 and < 3	4375	776	615	346	493	2336	398
≥ 3 and < 4	113	25	25	13	29	92	48
≥ 4 and < 5	8804	951	722	450	1049	3433	1458
≥ 5 and < 9	588	150	134	97	101	306	131
≥ 9 and < 17	207	33	28	29	20	97	18
≥ 17 and < 33	121	36	20	44	29	77	16
≥ 33 and < +∞	44	32	15	9	9	22	11

Figure 8. Heatmaps of the number of jobs launched on the "Zhores" cluster as function of time spent on the running time (longitudinal axis) and requested resources (number of CPU cores for upper plot and number of GPUs for the lower plot, vertical axis). Each cell shows the number of launched jobs

was found that the performance of tasks using GPU accelerators is significantly lower compared to tasks using exclusively CPU computing resources. For CPU tasks it was 55–60 %, for GPU tasks – less than 25 %. Three main problems that get in the way of efficient cluster usage were identified:

1. Tasks interfere with each other, large tasks block the launch of small ones, and vice versa. The queuing time increases, and large windows of equipment downtime appear.
2. Resources are allocated by the user, but tasks are launched with a long delay or poorly load the computational resources, which is especially true for GPU tasks for deep learning often run using Jupyter notebooks [8].
3. Under certain conditions some tasks work inefficiently. For example, they might lack a needed amount of RAM or are poorly optimized for parallel performance. As a result, the resources utilization is low while the allocation is high.

Possible solutions to these three identified problems are given below in sec. 3.2.2.

3.1.3. Moscow State University

Collecting data. Data collection in the Lomonosov-2 supercomputer [21] installed in MSU has been organized since the beginning and is constantly being improved. At the current moment, the performance monitoring of compute nodes is conducted using our own DiMMon monitoring system [18]. We were not satisfied with existing common solutions like Zabbix, Nagios, etc, because we wanted to have a more flexible and lightweight solution with richer functionality, such as:

- ability to change data collection interval on-the-fly for certain nodes, e.g., nodes running a specific job;
- ability to turn off data collection for nodes not running any jobs;
- automatic aggregation of data from nodes running one job;
- ability to turn off data collecting for nodes running a specific job (e.g., to collect data via trace collector or similar tools).

On each node, we collect the following types of data:

- different types of CPU load (user, iowait, idle, system) and load average;
- GPU usage (GPU load and memory utilization);
- Infiniband usage intensity (amount of bytes/packets sent/received per second);
- Lustre file system usage (number of opened and closed files, amount of read/written bytes plus service data like page faults);
- performance hardware counters (number of retired instructions and unhalted cycles per second, number of cache misses to L1 and LLC per second);
- other memory characteristics (ECC errors and free memory).

This data is collected once every second, being aggregated once every minute and stored in PostgreSQL database. It should be noted that this performance data from compute nodes could be efficiently stored in time-series DB as well (since it is a set of time series), but PostgreSQL, initially selected for different reasons, copes without problems with this amount of data (several GBs per day).

This node performance data is also then binded to the user jobs running on these nodes, and integral information about each job is stored separately in the MongoDB database, together with more data describing different features of these jobs: Slurm data with launch parameters, information about the project and organization for which the user works, data on software packages and libraries being used on jobs.

Besides monitoring of compute nodes, service servers are also monitored, in order to control their load and correctness of their work. We check ping as well as load average and disk space on each server, using Telegraph monitoring, with data being stored in VictoriaMetrics [4] database. On Lustre servers disks activity, Infiniband stats, CPU system time, Lustre read/writes, Lustre cache access rate and memory statistics also are collected via Telegraph service and stored in the VictoriaMetrics database.

How we use the data we collect. There are several different ways all this collected data is used in practice in order to help us to control and improve the performance of the Lomonosov-2 supercomputer.

One of the obvious and quite efficient ways to analyze job performance based on the collected monitoring data is to investigate various graphs showing timelines with job behavior according to a specific characteristic. This idea was implemented in the JobDigest [13] – a software for generating web reports presenting different information on the performance of the chosen job. JobDigest automatically creates such reports for all jobs running on the Lomonosov-2, which are available both for users (job owners only) and system administrators.

One example of how this data can be used is shown in Fig. 9. It is a heatmap showing the change of a number of L1 cache misses per second, with distribution among nodes used in the job. It should be noted that this data is collected using processor counters via PAPI [20]. As seen, most nodes show very low number of cache misses, while one node constantly shows over 280 million of misses per second, which is a very high value. In this case, we can assume that a program uses memory both very intensively and inefficiently on this node. Judging from this graph alone we cannot be sure that this leads to a decrease in the performance of the program, but this is definitely a potential bottleneck which should be primarily investigated during performance analysis.

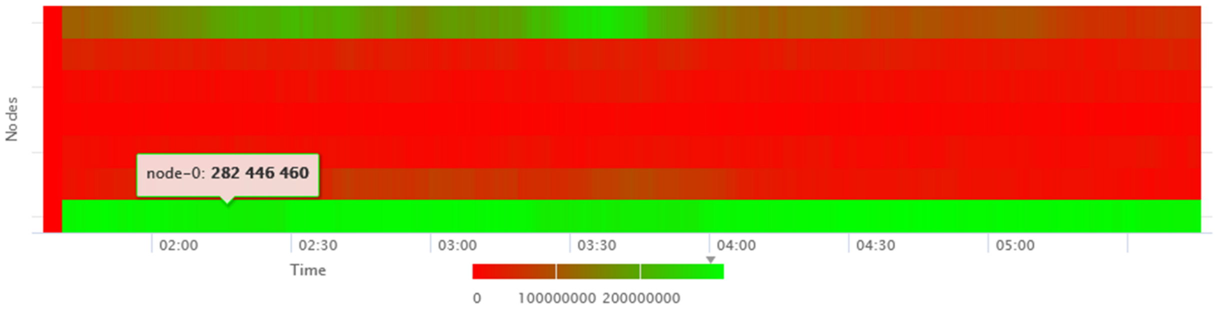
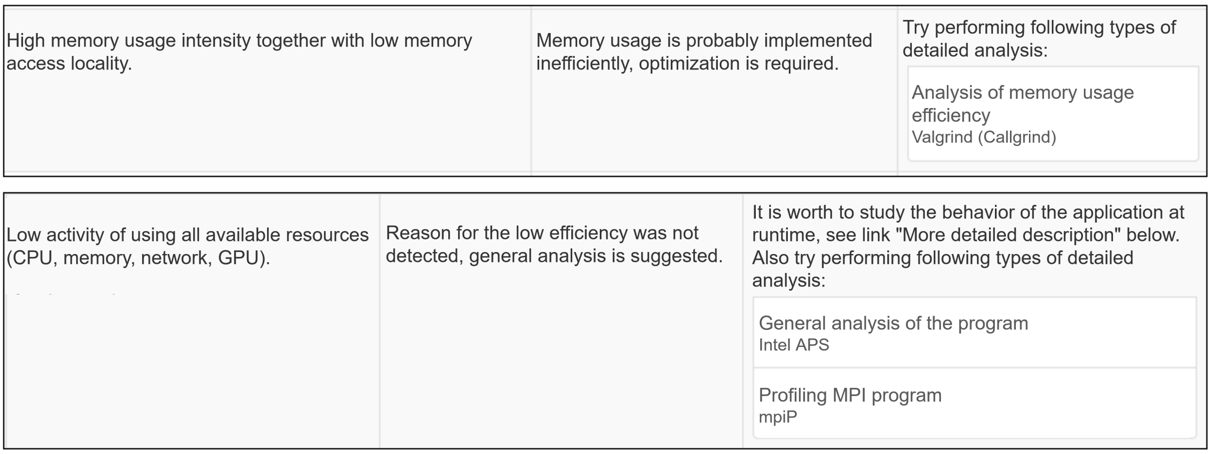


Figure 9. Distribution of the maximum number of L1 cache misses per second between nodes during job execution

JobDigest reports can be useful, but it is usually hard for users to properly analyze such low-level information and get insights from it. To help with this, a software package called TASC (Tuning Applications for SuperComputers) [17] for more “intellectual” data analysis was developed at the Moscow State University. Its main purpose is to help administrators and users of a supercomputer in detecting and eliminating a variety of issues with the performance of a supercomputer in general and individual applications in particular. For this purpose, it provides users with a detailed information about different performance characteristics of their applications as well as automatically detects potential performance issues (concerning CPU or memory utilization, efficiency of network or I/O usage, optimality and correctness of job launches, etc.) and notifies users about them.

An example of performance issues automatically detected in a job is shown in Fig. 10. Such information is available (in Russian) within the Octoshell system for all users of Lomonosov-2 supercomputer about their jobs. For each detected issue, there is a description of what seems to be the problem, what potentially could cause it and recommendations on its further analysis. Note that when clicking on the desired type of further analysis, a detailed manual on how to conduct it and what to look at is provided to a user.

The aforementioned examples show how the collected data can be useful for the supercomputer users. But also TASC, by integrating and analyzing together a huge amount of data on each main aspect of supercomputer behavior, allows system administrators to quickly study any aspect of interest with the desired level of details, as well as automatically notifies them about different critical issues automatically detected on the level of the whole supercomputer. Among such issues that were detected in practice are the inefficient usage of software packages by sev-



High memory usage intensity together with low memory access locality.	Memory usage is probably implemented inefficiently, optimization is required.	Try performing following types of detailed analysis: Analysis of memory usage efficiency Valgrind (Callgrind)

Low activity of using all available resources (CPU, memory, network, GPU).	Reason for the low efficiency was not detected, general analysis is suggested.	It is worth to study the behavior of the application at runtime, see link "More detailed description" below. Also try performing following types of detailed analysis: General analysis of the program Intel APS Profiling MPI program mpiP

Figure 10. Description of performance issues detected in a user job with recommendations on its further study (originally available in Russian)

eral users, failures in file system functioning, as well as inefficient job launches with excessively high load of compute nodes. Another direction of analyzing the collected monitoring data being of interest for the administrators, which is also studied in MSU, is using data mining methods for detecting similar jobs in order to get more insights on the structure and peculiarities of supercomputer job flow [16].

Data on service servers is visualized using Grafana package [2]. Most critical monitored metrics are controlled using Grafana alerts – in any suspicious or dangerous situation an alert message is being sent via email and Telegram to the administrators and engineers. We are planning to change alerting procedure to another service, because of low flexibility of Grafana alerting abilities – now we are testing the open-source software package “balerter” [1].

Grafana is also used to visualize current and historical supercomputer usage – number of used, free and disabled nodes, number of running and waiting jobs. Special dashboard shows current statuses of all nodes and list of reasons for node disabling. Another important dashboard shows incoming and outgoing water temperatures in the air conditioners and current energy consumption. Data for these dashboards are collected via SNMP protocol from air conditioners and power distribution units and stored into VictoriaMetrics database. All these dashboards are used on a daily basis by our system administrators to monitor and analyze the behavior of the Lomonosov-2 supercomputer.

3.2. Real-life Examples of Supercomputer Functioning Improvement

This topic is based on the survey question stated as “Please describe examples of real-life situations when the analysis of data on the supercomputer functioning helped you to distinctly improve the quality of its work.”. Such examples appeared in three stated supercomputer centers are provided below.

3.2.1. HSE University

HSE University has a complex task efficiency monitoring system of its own development on the *cHARISMa* HPC cluster. It is called *HPC TaskMaster*. The system allows monitoring tasks of *cHARISMa* users, creating reports with task information and dynamic graphs, and automatically detects inefficient tasks by finding problems related to resource utilization and creating conclusions about the work of the task. The system is designed to help cluster users run their tasks more efficiently, therefore saving expensive supercomputer machine time.

Finding inefficient tasks is an important problem that all large clusters face. To classify a task as inefficient, it is necessary to rely on its various indicators. Such indicators can be low or extremely high rates of components utilization and too short or too long task duration. For many tasks, it is essential to determine their type before analyzing its effectiveness: for example, Jupyter Notebook and Gromacs will have completely different behavior and their indicators must be analyzed in different ways.

The principle of operation of *HPC TaskMaster* is briefly described in Fig. 11. From each compute node of the *cHARISMa* cluster, the *HPC TaskMaster* system collects time series of such characteristics as usage of CPU, GPU, file system, RAM, InfiniBand network. After that, the system aggregates the time series, obtaining the average, minimum and maximum values for the characteristics. The resulting aggregated metrics are stored as a tuple of values, which is further processed by the indicator definition functions. The resulting tuple contains indicator levels for each type of processed value from 0 to 1 according to the level of manifestation.

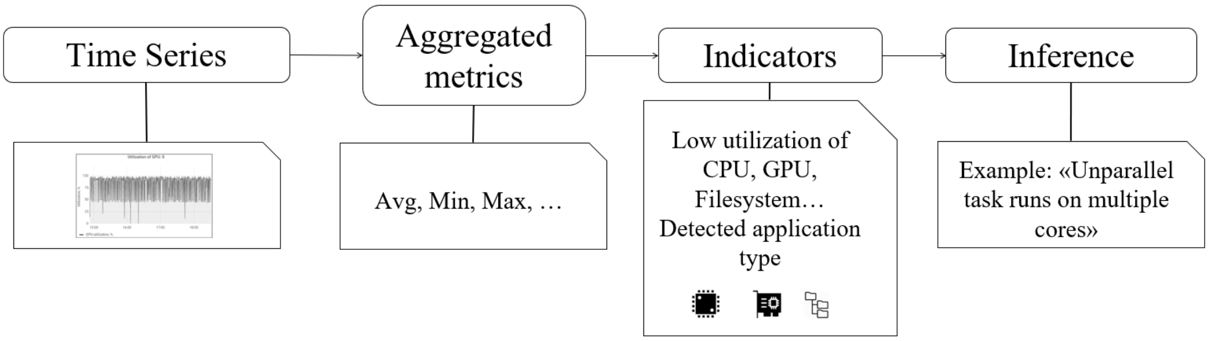

Time Series
Aggregated metrics
Indicators
Inference
Avg, Min, Max, …
Low utilization of CPU, GPU, Filesystem… Detected application type
Example: «Unparallel task runs on multiple cores»

Figure 11. Stages of data processing

After the tuple of indicators is created, it is compared with a list of all possible inferences specified in the system. When all the indicator values fall within the specified limits, the corresponding inference is assigned to the task.

There are some examples of inferences that can be generated by the system:

- *Incorrect launch of jupyter-notebook application.* The running task type is detected as a jupyter-notebook application, and there are indicators of low utilization of CPU or GPU.
- *Allocating resources without running computations.* The user has allocated resources for the task, but there are indicators of low resource utilization.
- *Running unparallel tasks on multiple nodes.* Several nodes are allocated for the task, but there are indicators of low InfiniBand network usage and low resource utilization.

If the indicator values fit several inferences, the one with the highest priority is selected. This system allows the administrator to conduct a flexible configuration of inferences for various types of tasks, without changing the source code of the system and quickly adding new inferences when new types of tasks appear on the cluster.

The start-up of this system allowed increasing the effective load of the HSE supercomputer by 16 %. This was made possible by detecting computing tasks with incorrect startup parameters and notifying users about them. In the near future, when the system is trained to find problems that are more complicated and issue expanded inferences, it will be possible to achieve even greater savings in computing resources.

3.2.2. Skoltech

The collected data and arising problems for the Skoltech supercomputer "Zhores" were described above in sec. 3.1.2. First, optimization of resource usage was performed by allowing out-of-order job launching in Slurm using the backfill method (see Fig. 12) as well as changing the properties of the queues. Backfill method drastically reduced queuing times and increased computation density.

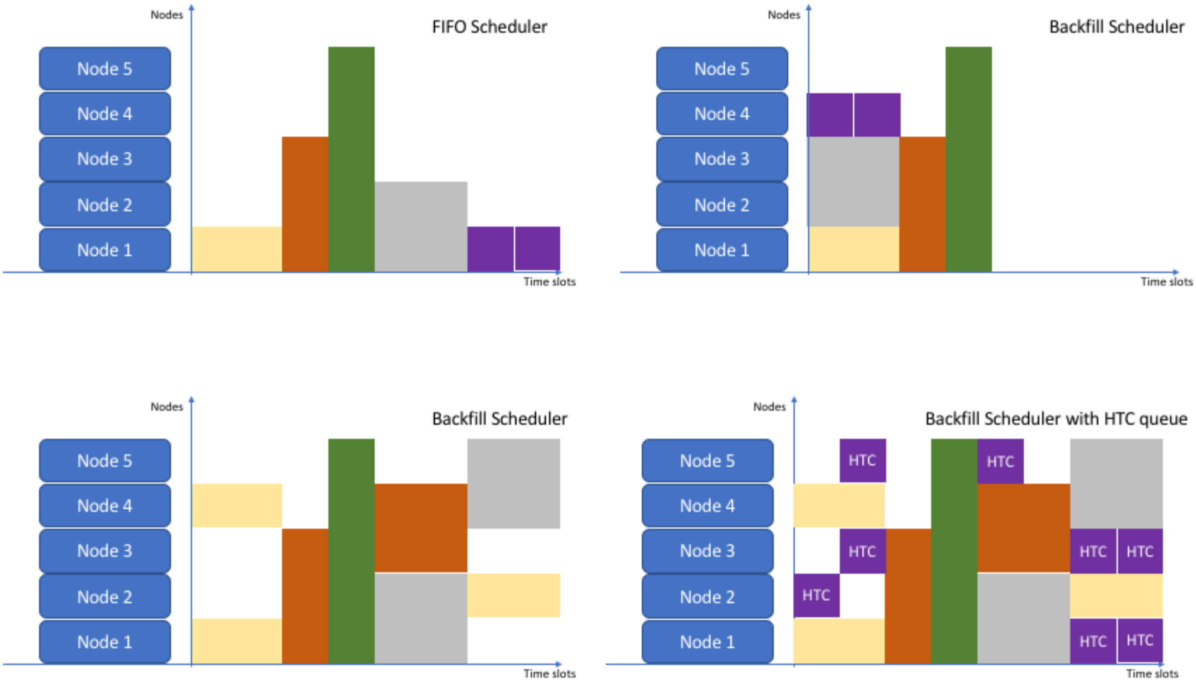


Figure 12. Backfill vs FIFO scheduler schematics. Backfill allows us to significantly densify the task launching (top left row for FIFO scheduler vs top right row for backfill scheduler). Allowing HTC tasks to be launched in free spots further leads to a more effective supercomputer usage (bottom left row for backfill scheduler without HTC vs bottom right row for backfill scheduler with HTC)

"Zhores" cluster has a hybrid architecture and consists of CPU-only nodes as well as nodes with modern GPUs. That is why the entire flow of computational tasks was divided into classes according to the required architecture (CPU, GPU and later big memory nodes) as well as required runtime. Before the majority of users started to use "Zhores" supercomputer the queues presented in Tab. 2 (only columns with the asterisk) were envisioned.

Later on the queuing scheme evolved into the one presented in Tab. 2 (only columns without the asterisk) and the following restrictions:

- one can not occupy more than half of all the resources of the queue;
- queues with lower time limit have higher priority;
- there is a minimum amount of RAM per allocated core.

Additionally, HTC tasks can be launched in all queues. HTC tasks are typically short and not resource-intensive but there is a huge amount of them. By filling all the free slots with HTC tasks, the utilization of CPU resources was increased by 15 %, and the time spent in the queue for HTC tasks was reduced by 6 times. Altogether, backfilling and restructuring the queues

Table 2. Queues distribution in “Zhores” cluster

*Queue name	*Time limit	Name	Queues description	DefaultTime	MaxTime	MaxNodesPerJob
cpu_debug cpu_small cpu_big	30 min. 24 hours 6 days	cpu	CPU nodes only	24h	6d	22
gpu_debug gpu_small gpu_big	30 min 24 hours 6 days	gpu	GPU nodes only	24h	6d	6
mem_debug mem_small mem_big	30 min 24 hours 6 days	mem	Big mem nodes	24h	6d	4
		htc	HTC jobs, 1 node per task all nodes above, less priority		24h	1

helped to reduce queuing times and increase CPU utilization by approximately 20–25 % and GPU utilization by approximately 5 %.

The next step was to increase the utilization of the most expensive resource – the GPUs. In the process of supercomputer performance monitoring it was found that users (mostly those working on ML/DL tasks and using Jupyter notebook environment) use nodes with powerful NVidia Tesla V100 GPUs for code prototyping and development. This leads to GPU resource allocation with almost zero utilization during the code development stage that can sometimes last longer than the actual running time. This is the usual difference between the traditional HPC workflow and “AI” workflow. To solve this problem, several dedicated servers each containing from 8 to 10 less powerful and lower cost NVidia GPUs (typically NVidia GTX 1080 or 2080 Ti) with support of the same software libraries that are used for the V100 (CUDA, Torch, Tensorflow, etc.) were purchased. Thus code prototyping and development zone was established on those servers with dynamic deployment of the Jupyter Hub environment and a set of all the necessary tools for using the GPU. By doing this it was achieved that in the vast majority only well debugged production codes were launched on the Tesla V100 GPUs and their utilization rate raised by approximately 30 %.

Third of the problems identified in sec. 3.1.2 was tackled by purchasing high-density servers with four processors and up to 3 terabytes RAM per each node. Such nodes could effectively solve problems where it was needed to keep a large amount of data in RAM, as well as problems that were poorly optimized for parallelization to several nodes. This differentiation of tools helped to solve user problems faster. With an identical load on computing resources, the average CPU task execution speed increased by 15 %.

Even if there is limited budget for hardware purchase, one can improve the efficiency of the supercomputer by increasing user awareness. They should not treat the cluster as a black box. Knowledge of its architecture, strengths and weaknesses enables the developer to create more efficient code. To increase HPC users knowledge and awareness, regular HPC seminars are held in Skoltech, an online system of interaction with users has been established, an HPC Wiki has been developed with examples of best practice. A Telegram channel has been created with news about the cluster and an overview of new approaches to solving computational problems. To better prepare students, lectures and laboratory works on basic architecture of a supercomputer were introduced into the HPC track of the Advanced Computational Science MSc educational program, within which students receive not only theoretical knowledge, but also build and learn to administer their own (small scale) supercomputer.

All the aforementioned steps have lead to the reduction of the difference between resources allocation percentage and utilization percentage. Average CPU utilization is approximately 90 % and average GPU utilization is more than 65 %.

3.2.3. Moscow State University

An aforementioned TASC software has been used on the Lomonosov-2 supercomputer for a couple years, and during this time it has helped to find and eliminate different performance issues. Let us discuss three of them.

TASC automatically detects different performance issues in all user applications running on the supercomputer. One of such issues, which is considered as critical, detects suspiciously low utilization of both CPU and GPU. TASC also provides web reports that allow analyzing the appearance of such issues among users, and such statistics for the first half of May 2020 has shown that 95 % of all "suspicious" node-hours belonged to one user. Further analysis of this user has revealed that he has made only 5 job launches during this time period, but they occupied 12000 node-hours (which is quite a lot) and all of them were "suspicious". This means that too many resources are idle, so this user was contacted. It turned out that these jobs were launched on 50 nodes each, but due to an error in a program only one node was actually involved. The user was unaware of it; after we contacted him this issue was eliminated.

Another example happened in July 2020. Among other things, TASC-based reports provide general statistics on overall user activity on Lomonosov-2. Figure 13 shows top 10 users based on the number of job launches during selected time period. It can be seen that the most active user has made over 1500 launches (leftmost column), which is almost 6 times more than the second most active user. At the same time, these jobs occupied only a small amount of node-hours (black line). Such situation is quite unusual, so it was decided to investigate it further. It appeared that almost all of the jobs were using Gromacs package [7] and lasted less than 5 minutes, which is even more unusual, especially for Gromacs users. We contacted this user, and it turned out that he was using a script that was automatically launching jobs which almost immediately fell with an error, forcing the script to start new jobs. After our request, the user fixed his script.

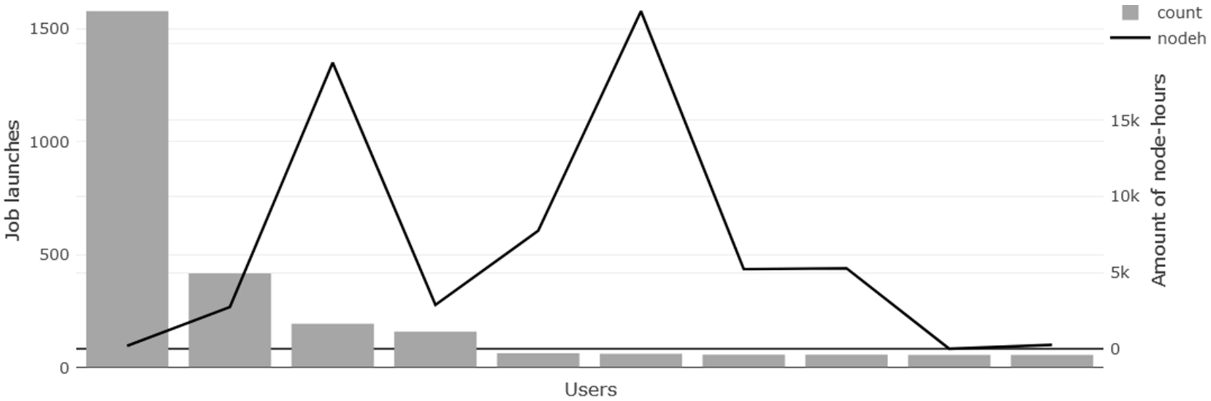


Figure 13. Top ten users of Lomonosov-2 supercomputer based on the number of job launches during beginning of July 2020

Another interesting example concerns the NAMD package [15] for solving molecular dynamics tasks. The Lomonosov-2 supercomputer has a special partition for GPU-intensive jobs. While studying the GPU utilization of this partition within year 2020, it was discovered that jobs of one active user was showing only 7 % of GPU load. Further analysis using TASC-based reports allowed us to see that almost all jobs were using NAMD package, while showing less than 10 % of GPU load each. It was the only user of NAMD package in this partition, but many other packages launched in this partition were showing much higher GPU load (Gromacs – 57 %, LAMMPS – 34 %). Next, we discovered that the GPU load in this user jobs when launching NAMD in another (main) partition was significantly higher – 57 %. This suggested that the problem was with NAMD build for the GPU-based partition, and it turned out to be so. After tuning this package and therefore fixing this issue, the GPU load of this package returned to normal.

Conclusions

This paper shows the results of a survey of system administrators from 10 large supercomputer centers in Russia regarding the issues of maintenance, monitoring and analysis of supercomputer behavior. This review allows, as a first approximation, to capture the general picture of the current state in this area. The information collected made it possible to find out what monitoring data is most interesting in practice, and what information is practically not collected; what existing systems for monitoring, storing, visualizing and analyzing data are most often used; etc. Of particular interest are the areas in which you most often have to develop your own solutions (i.e., data monitoring and analysis) – these are the areas in which there are no ready-made suitable solutions, and further development of these areas is worthwhile to be carried out collectively. At the same time, the results of the survey show that in all centers without exception there is a need for a more complete understanding of the state of supercomputers, which suggests that this area is important and needs to be further developed.

Two important topics are considered separately – using monitoring data in practice and real-life examples of supercomputer functioning improvement. For each topic, the administrators of three large centers (HSE University, Skoltech and Moscow State University) describe how they approach these issues in practice. Such a description helps to better understand the complexities and challenges in this area, as well as possible approaches to their solution.

Acknowledgements

This research was supported in part through computational resources of HPC facilities at HSE University [12]. The research is partially carried out using the equipment of shared research facilities of HPC computing resources at Lomonosov Moscow State University. The MSU part of the survey and the reported study was funded by the Russian Foundation for Basic Research, project no. 20-07-00864.

References

1. Balerter homepage. `https://balerter.com/0.8.1/getting_started/about.html`, accessed: 2021-08-26

2. Grafana: The open observability platform. `https://grafana.com/`, accessed: 2021-08-26

3. The working group on the analysis and quality assurance of supercomputer center functioning. `https://scc-efficiency.parallel.ru/`, accessed: 2021-08-26

4. VictoriaMetrics documentation. `https://docs.victoriametrics.com/`, accessed: 2021-08-26

5. Presentation with final survey results (in Russian). Tech. rep. (2021), `https://scc-efficiency.parallel.ru/assets/final_scc_survey.pdf`

6. Top 50 supercomputers list. `http://top50.supercomputers.ru/list` (2021), accessed: 2021-08-26

7. Abraham, M.J., Murtola, T., Schulz, R., et al.: GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1-2, 19–25 (2015). `https://doi.org/10.1016/j.softx.2015.06.001`

8. Community, E.B.: Jupyter Book (2020). `https://doi.org/10.5281/zenodo.4539666`

9. Deneroff, M.M., Shaw, D.E., Dror, R.O., et al.: Anton: A specialized ASIC for molecular dynamics. In: 2008 IEEE Hot Chips 20 Symposium (HCS). pp. 1–34 (2008). `https://doi.org/10.1109/HOTCHIPS.2008.7476542`

10. Gormley, C., Tong, Z.: Elasticsearch: The Definitive Guide. O'Reilly Media, Inc., 1st edn. (2015)

11. Joseph, E., Conway, S.: Major Trends in the Worldwide HPC Market. Tech. rep. (2017), `https://hpcuserforum.com/presentations/stuttgart2017/IDC-update-HLRS.pdf`

12. Kostenetskiy, P.S., Chulkevich, R.A., Kozyrev, V.I.: HPC Resources of the Higher School of Economics. Journal of Physics: Conference Series 1740, 012050 (2021). `https://doi.org/10.1088/1742-6596/1740/1/012050`

13. Nikitenko, D., Antonov, A., Shvets, P., et al.: JobDigest – Detailed System Monitoring-Based Supercomputer Application Behavior Analysis. In: Supercomputing. Third Russian Supercomputing Days, RuSCDays 2017, Moscow, Russia, September 25-26, 2017, Revised Selected Papers. pp. 516–529. Springer, Cham (2017). `https://doi.org/10.1007/978-3-319-71255-0_42`

14. Ott, M., Shin, W., Bourassa, N., et al.: Global Experiences with HPC Operational Data Measurement, Collection and Analysis. In: IEEE International Conference on Cluster Computing, CLUSTER 2020. pp. 499–508. IEEE (2020). `https://doi.org/10.1109/CLUSTER49012.2020.00071`

15. Phillips, J.C., Braun, R., Wang, W., et al.: Scalable molecular dynamics with NAMD. Journal of Computational Chemistry 26(16), 1781–1802 (2005). `https://doi.org/10.1002/jcc.20289`

16. Shaikhislamov, D., Voevodin, V.: Solving the problem of detecting similar supercomputer applications using machine learning methods. In: Parallel Computational Technologies, PCT 2020. Communications in Computer and Information Science, vol. 1263, pp. 46–57. Springer, Cham (2020). `https://doi.org/10.1007/978-3-030-55326-5_4`

17. Shvets, P., Voevodin, V., Nikitenko, D.: Approach to Workload Analysis of Large HPC Centers. In: Parallel Computational Technologies, PCT 2020. Communications in Computer and Information Science, vol. 1263, pp. 16–30. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55326-5_2

18. Stefanov, K., Voevodin, V., Zhumatiy, S., et al.: Dynamically Reconfigurable Distributed Modular Monitoring System for Supercomputers (DiMMon). Procedia Computer Science 66, 625–634 (2015). https://doi.org/10.1016/j.procs.2015.11.071

19. Sterling, T., Anderson, M., Brodowicz, M.: High Performance Computing: Modern Systems and Practices. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2017). https://doi.org/10.1016/C2013-0-09704-6

20. Terpstra, D., Jagode, H., You, H., et al.: Collecting performance data with PAPI-C. In: Müller, M.S., Resch, M.M., Schulz, A., Nagel, W.E. (eds.) Tools for High Performance Computing 2009. pp. 157–173. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11261-4_11

21. Voevodin, V.V., Antonov, A.S., Nikitenko, D.A., et al.: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomputing Frontiers and Innovations 6(2), 4–11 (2019). https://doi.org/10.14529/jsfi190201

22. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer, Berlin, Heidelberg (2003). https://doi.org/10.1007/10968987_3

23. Zacharov, I., Arslanov, R., Gunin, M., et al.: "Zhores" – Petaflops supercomputer for data-driven modeling, machine learning and artificial intelligence installed in Skolkovo Institute of Science and Technology. Open Engineering 9(1), 512–520 (2019). https://doi.org/10.1515/eng-2019-0059

24. Zacharov, I., Panarin, O., Rykovanov, S., et al.: Monitoring applications on the ZHORES cluster at Skoltech. Program Systems: Theory and Applications 12(2), 73–103 (2021). https://doi.org/10.25209/2079-3316-2021-12-2-73-103

DOI: 10.14529/jsfi210306

Efficient Implementation of Liquid Crystal Simulation Software on Modern HPC Platforms

Ilya V. Afanasyev[1,2], *Dmitry I. Lichmanov*[1,2], *Vladimir Yu. Rudyak*[3], *Vadim V. Voevodin*[1,2]

In this paper we demonstrate the process of efficient porting a software package for Markov chain Monte Carlo (MCMC) simulations on a finite cubic lattice on multiple modern architectures: Pascal, Volta and Turing NVIDIA GPUs, NEC SX-Aurora TSUBASA vector engines and Intel Xeon Gold processors. In the studied software, MCMC methodology is used for simulations of liquid crystal structures, but it can be as well employed in a wide range of problems of mathematical physics and numerical methods. The main goals of this work are to determine the best software optimization strategy for this class of algorithms and to examine the speed and the efficiency of such simulations on modern HPC platforms. We evaluate the effects of various optimizations, such as using more suitable memory access patterns, multitasking for efficient utilization of massive parallelism on the target architectures, improved cache hit-rates, parallel workload balancing, etc. We perform a detailed performance analysis for each target platform using software tools such as nvprof, Ftrace and VTune. On this basis, we evaluate and compare the efficiency of the developed computational kernels on different platforms and subsequently rank these platforms by their performance. The results show that NVIDIA GPU and NEC SX-Aurora TSUBASA platforms, although at first glance seem very different, require similar optimization approaches in many cases due to similarities in data processing principles.

Keywords: NVIDIA GPU, NEC SX-Aurora TSUBASA, liquid crystals, HPC, co-design, performance optimization, Monte Carlo, cubic lattice.

Introduction

The program under study is a software package developed specifically for meso- and macroscopic computer simulations of structure and properties of nematic and cholesteric liquid crystal droplets. Although this software is aimed to solve the specific physical problem, the computational task beyond it belongs to the one of the most important classes of computational problems. Optimization of a functional defined on a finite space cubic lattice relates to a wide range of problems from mathematics and physics to economics and operational research, which require high efficiency implementations. For example, in mathematics and natural sciences, Markov chain Monte Carlo (MCMC) simulations on cubic lattice are used in methodological studies of novel Monte Carlo techniques [13, 14, 37], spin models [9, 17, 44], quantum Monte Carlo [7, 26, 29], material design [47], bio-chemistry [31], etc. Keeping that in mind, we will focus on both the current implementation and general approach to the optimization of this type of algorithms. We believe that the optimization techniques demonstrated below are applicable to a wide range of computational problems dealing with Markov chain Monte Carlo simulations and stochastic optimization on finite space cubic lattice.

Liquid crystals (LCs) are the perfect example of soft matter materials that combine the typical properties of a crystalline solid and a viscous liquid [12]. This gives LCs unique physical properties and allows many applications of LC materials. LCs are mostly known for their use in liquid crystal displays, embedded today in almost every device. At the same time, there is a broad variety of more sophisticated applications nowadays: chemical sensors [38, 40], tunable optical

[1] Moscow Center of Fundamental and Applied Mathematics, Moscow, Russia
[2] Research Computing Center, Lomonosov Moscow State University, Moscow, Russia
[3] Faculty of Physics, Lomonosov Moscow State University, Moscow, Russia

devices [11, 22], biomedicine applications [18], and many others [21]. For the most complex applications, it is crucial to precisely understand the fundamental behaviour of LCs in various conditions, their orientational structure and properties. While there are theoretical descriptions for such problems, the analytical solutions are typically unavailable due to the complexity of the problems. At this point, computer simulation techniques are typically used to solve such problems.

The program under study implements the extended Frank elastic continuum approach to describe the energy of the system [34]. Markov chain Monte Carlo simulated annealing is used to find energy-optimal structures of droplets of liquid crystal (LC droplets) droplet structures. It previously showed good results for nematic and cholesteric LC [20, 35, 39]. The use of MCMC stochastic optimization by simulated annealing [10, 30] allows us to almost completely ignore the problem of the trapping into local minima, in contrast to the Newtonian-like iterative methods. Moreover, checkerboard decomposition algorithms [45] result in great computational efficiency and parallelizability, which is critically important for GPU implementation [5, 33]. The original software package was developed back in 2012 [34] for NVIDIA Fermi GPU architecture. Since that time, both GPU hardware and software has changed significantly, which makes it reasonable to update the computational core of the software to fit the abilities of Pascal, Volta and Turing GPUs. Moreover, it seems promising to try to port this software to vector processor architectures (VCPUs). VCPUs show significant progress in solving vectorizable problems like Markov chain Monte Carlo simulations [2, 27, 32, 41]. In particular, massively parallel NEC SX-Aurora TSUBASA vector processors equipped with high-bandwidth memory (HBM) as well as the newest Intel Xeon processors with AVX-512 vector instructions may provide significant acceleration for cubic lattice Monte Carlo problems [16].

In this paper, we use supercomputing co-design approach (i.e., porting the evaluated software to multiple target platforms with the subsequent selection of the platform which demonstrates the highest performance on the particular problem) in order to develop an efficient and high-performance implementation of the software package for simulation of LC droplets. For this purpose, three modern target platforms are investigated in this paper: NVIDIA GPUs of Volta, Pascal and Turing architectures, NEC SX-Aurora TSUBASA vector processors, and Intel Xeon Skylake multicore CPUs. These platforms belong to a significant and representative subclass of modern supercomputing architectures, which provide high-performance computational units and high-bandwidth memory. For each target architecture we describe implementation and optimization approaches, typically required to achieve high performance for the studied class of problems: selecting efficient memory access patterns, using multitasking to efficiently utilize massive parallelism, improving cache hit-rates, parallel workload balancing, and several others. Finally, we present a comparative analysis of these implementations for different target platforms and discuss the efficiency of these platforms for this class of computational problems.

The rest of the paper is organized as follows. Section 1 describes hardware features and properties of target architectures used in this work. Section 2 explains in detail what kind of calculations forms the main computationally-intensive part of the program being analyzed and why the question of its optimization is not trivial but actual. In section 3, we briefly describe a physical task solved within the entire program package as well as its implementation features. Section 4 covers a detailed description of a typical computational kernel and its contribution to a whole program package, while section 5 describes the pipeline of optimizations applied to this typical kernel on all target architectures. Conclusion summarizes the study and points out the main results of this work.

1. Target Architectures

1.1. NVIDIA GPU

Modern NVIDIA GPU consists of a set of identical Streaming Multiprocessors (SM), each of which has multiple CUDA cores, capable of performing various types of operations. CUDA (Compute Unified Device Architecture) programming model allows users to define kernels – special functions, which are executed on GPUs, and grids – configurations of these kernels, which define the number of threads used by each kernel. NVIDIA GPUs employ Single Instruction Multiple Thread (SIMT) computational model, in which threads are organized in groups of size 32 (called warps), and all threads of one warp execute the same instruction at any given moment of time. Typically each thread performs almost identical computational workflow over its own data – so called data-driven parallelism.

At the time of this research, the most recent NVIDIA GPU microarchitectures included Pascal, Volta, Turing and Ampere. For the performance evaluations in this paper, machines equipped with Pascal (P100), Volta GPUs (V100) and desktop Turing (GeForce RTX 2080 Ti) are used. Unfortunately, Ampere GPU servers were unavailable to us at the moment of this writing. An execution model is similar for all recent generations of GPUs, however, they have different hardware characteristics, the most important for our paper being theoretical peak performances and the structure of memory hierarchy. Specifications of these GPUs are shown in Tab. 1.

Table 1. Specifications of the GPU architectures used in this work

Architecture	**Pascal**	**Volta**	**Turing**
Release year	2016	2017	2018
Model	P100	V100	RTX 2080 Ti
Memory type	HBM2	HBM2	GDDR6
Memory bus, bit	4096	4096	352
Memory size, GB	16	32	11
Theoretical peak memory bandwidth, GB/s	720	900	616
L1 cache per SM, KB	64	128	64
L2 cache, KB	4096	6144	5632
Peak performance (float), TFlop/s	9.5	14.3	11.8

1.2. NEC SX-Aurora TSUBASA

The NEC SX-Aurora TSUBASA architecture with dedicated vector processors [19, 46] inherits the design concepts of a vector supercomputer and enhances its advantages to achieve higher sustained performance and higher usability. NEC SX-Aurora TSUBASA mainly consists of vector engines (VEs), equipped with a vector processor and a vector host (VH) of an x86 node. VE is used as a primary processor for executing applications, while the VH is used as a secondary processor for executing basic operating system functions that are offloaded from the

VE. VE has eight powerful vector cores with the total peak performance of 4.91 TFlop/s on single precision and 2.45 TFlop/s on double precision.

Each SX-Aurora vector core consists of three components: a scalar processing unit (SPU), a vector processing unit (VPU), and a memory subsystem. The majority of computations are performed by VPUs, while SPUs provide the functionality of a typical CPU. Since SX-Aurora is not just a typical accelerator but rather a self-sufficient processor, SPUs are designed to provide relatively high performance on scalar computations. In order to store the results of intermediate calculations, each vector core is equipped with 64 vector registers with a total register capacity equal to 128 KB. Each register is designed to store a vector of 256 double precision elements. On the memory subsystem side, six HBM modules in the vector processor can deliver up to 1.22 TB/s theoretical memory bandwidth [8] with up to 48 GB total capacity. Parallel programs for the NEC SX-Aurora TSUBASA architecture are implemented via OpenMP programming model, while vectorization is performed by NEC compiler (the user inserts specific directives, which help the compiler to perform automatic vectorization).

1.3. Intel Xeon

Intel Xeon Gold 6126 processors of Skylake microarchitecture have been used in order to evaluate the performance of modern CPUs in this paper. These 12-core processors achieve theoretical peak performance of almost 2 TFlop/s on double precision and 4 TFlop/s on single precision. Each core has a private L1 data cache of 32 KB size, a private L2 cache of 1 MB size, while all cores share a 19.25 MB non-inclusive L3 cache. Intel Xeon Gold 6126 processors support Advanced Vector Extensions 512 (AVX-512), capable of processing 16 single precision values on each cycle. Theoretical peak DRAM memory bandwidth of these processors is equal to 125 GB/s in the configuration available to us.

2. Formulation of the Problem

Since the beginning of the era of general-purpose computing on graphics accelerators, various algorithms of computational physics have been ported to NVIDIA GPUs and thoroughly optimized for this architecture [15, 28]. Mesh methods and particularly cubic lattice methods allow researchers to utilize the massive parallelizm of GPUs for the two following reasons [6, 23, 42]. First, a parallelization method is natively embedded into the lattice decomposition. Second, these methods often do not require storing and exchanging any additional data between threads (in contrast to the molecular dynamics, for example). The most common approaches used for the physical problems formulated on cubic lattice are solutions of transport equations (for example, lattice Boltzmann methods used for computational fluid dynamics, CFD [43]), Newtonian-like function optimization [10], and Monte Carlo methods (used for wide range of statistical physics problems in soft matter, bio-chemistry and quantum physics [24, 26, 31, 36, 48], and also for function optimization [30]).

In the case of the software studied in this paper, cubic lattice Monte Carlo methods are used. At the first glance, it seems that, in terms of optimization approaches, these methods should be very similar to the stencil calculations typically present in transport equations. For example, in the problem of liquid crystal structure optimizations, the free energy of the system (the value to be minimized) is a function of 3D distribution of a director vector (so called “structure”, which is varied to deliver the minimum to the free energy). The free energy in each sub-volume of the lattice is calculated on the basis of the nearest values of director field, effectively forming $3 \times 3 \times 3$ size stencil for the calculations. And the optimization of stencil calculations for GPUs is thoroughly studied [25], consisting primarily of the usage of shared memory, utilizing registers to increase volume of cached data, block tiling, depending on the order of a numerical scheme (effectively, the size of a stencil) and the form of the used equations.

However, there are critical differences between traditional stencil calculations and Monte Carlo approach, which also makes the optimization strategies for these methods very different. Let us take a closer look at the problem to understand these differences.

First, approaches to choosing the optimal stencil size are nearly opposite. In stencil calculations, the use of numerical scheme of higher order grants higher stability of the numerical scheme and larger stencil size. In turn, using a larger stencil size increases data reuse without changing the available degree of parallelism, leading to a more efficient implementation (see Fig. 1, where stencils are shown as square brackets).

Monte Carlo calculations, for the most of physical problems, require small stencil size. Each MC step consists of three intrinsic parts: (1) trial change of the state, (2) the update of the energy function (or other function), and (3) the decision step by probabilistically accepting some trial changes (mostly favorable) and declining others (mostly unfavorable). MC steps at lattice points located at a stencil size and beyond can be performed independently in parallel. In this case, using smaller stencil size increases the available degree of parallelism without changing the data reuse (which is almost absent in this type of MC calculations), see Fig. 1. This type of space decomposition-based parallelizm is called checkerboard algorithm [45]. Practically important stochastic optimization problems often require as many MC steps as possible, which makes the checkerboard parallelization strategy one of the most useful for this type of problems [3, 5, 33]. Thus, for stencil calculations larger stencil is optimal as it allows better utilizing the limited data transfer rate, but for Monte Carlo the smaller stencil size is optimal as it allows using higher number of parallel threads.

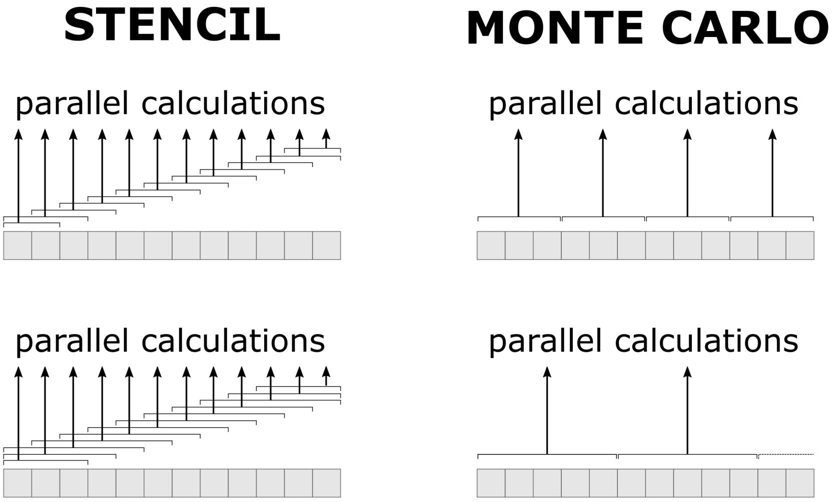


Figure 1. Simplified parallelization schemes for stencil-like calculations (left column) and Monte Carlo with checkerboard decomposition (right column) for 1D problem. Two rows demonstrate changes in parallelizm with increasing stencil size

Second difference concerns the operations with the data. In traditional stencil calculations, the data operations are pretty straightforward: the changes in the state turns to the changes in the function, which turns in the changes in the state on the next iteration. The memory operations are similar: read the state, then write the output values after calculations. In Monte Carlo, each change will be accepted or declined with some probability. It raises the need for more memory operations: (1) each step starts with copying the current state, its energy and auxiliary parameters; (2) each step may end with restoring the state if the step is declined, which effectively is copying back all the data. It results into massive memory operations in the beginning and the end of each MC step, which is unavoidable due to the nature of Monte Carlo method.

Third, traditional stencil calculations require relatively low amount of memory per calculation and produce relatively low number of operations per step. In this case, the processing power is often limited by the memory bandwidth of the device, and the use of shared memory improves the situation significantly. In contrast, in our simulations the mathematical equations lying behind the calculations are rather complex. Calculations in each point require $3 \times 3 \times 3$ values of two 3D vector fields, large number of temporary data for the minimization of recurring calculations, and around 10 auxiliary output values to make the result not just valid but also meaningful for the end user. This forms a large amount of memory used per lattice point per step, which is far beyond the capabilities of shared memory of modern GPU devices. For example, NVIDIA V100 and RTX 2080 Ti contain only 64 KB of L1 cache per SM, while the suggested numerical scheme requires at least 100 float values per thread.

It can be seen that each iteration of MC calculations in our case requires a lot of computations as well as intensive memory usage. It should be noted, that the high memory usage for calculations is not a sequence of non-optimality of the mathematical approach or the technical implementation. For many real world problems, it turns out the same way due to the continuously increasing complexity of physical phenomena taken into account, even when the basic equations seem very simple. Clearly, it is applicable for all lattice Monte Carlo problems, not just for the topic of liquid crystals.

To sum it up, nowadays the arsenal of lattice algorithms is optimized for NVIDIA GPUs for the two major cases: stencil-type and Monte Carlo calculations with relatively low memory usage and relatively low computational intensity per step. However, many physical problems require Monte Carlo calculations with both intense memory usage and computational operations. There are serious optimization issues with applying traditional optimizations to this class of problems. Our aim is to study its optimization possibilities. So, we need to seek optimal memory access patterns inside computational kernels and efficient parallelization scheme.

3. Brief Description of the Evaluated Program Package

The evaluated program package utilizes the cubic lattice approach, in which the whole 3D volume of an LC droplet is divided into cells by a cubic lattice of a predefined size. A special first-order scheme is used, thus the calculations utilize the data on the director vector value and its first spatial derivative. This scheme was optimized for unit length quasi-vector fields [34]. It helps to efficiently implement highly parallel Monte Carlo simulations with the degree of parallelism of $N_xN_yN_z/8$, where N_x, N_y and N_z are the dimensions of the lattice. The program package has been implemented in CUDA C, with CPU host used only for an initialization and input/output functions. The original version of the package from 2012 required double precision calculations for accurate results and has been briefly optimized in terms of mathematical algorithms complexity and memory consumption.

When we use the evaluated package to solve real-life physical problems, we typically produce preliminary, main and precise calculations. Preliminary calculations consist of tens to hundreds of tasks on small and medium lattices (from 16^3 to 32^3). Typically, these calculations are used to check the physical problem, task parameters and also to find basic physical regimes of the system. Main calculations consist of hundreds to thousands of tasks on medium and large lattices (32^3 to 64^3). These calculations are used to scan the physical system over parameters under study (for example, application of electric field) and find stable and metastable states of the system. Precise calculations consist of tens of tasks on large to extra large lattices (from 64^3 to 256^3). These calculations are used to re-evaluate the energy of the system with higher precision in a certain state.

It should be noted that each optimization task consists of many similar independent runs by the nature of stochastic optimization. Usually, each task requires from 4 to 10 runs; however, in some situations up to 100 runs may be required (for example, when the frustration of the system between ground and metastable states is studied).

4. Typical Computational Kernel of the Program Package

At the first stage of this research, we selected a primary computational kernel (device function) of the evaluated program, which has two important characteristics. First, this kernel performs the largest part of computations among other GPU activities in the evaluated program. Second, this kernel utilizes memory access patterns and computational workflow similar to other important kernels of the program package. Thus, it is reasonable to apply optimizations and evaluate their effects on this specific kernel, and later to generalize and apply them for the remaining kernels. This kernel will be further referred to as “typical computational kernel” (TCK).

TCK produces a trial change in the director field $\bar{n}(\bar{r})$ (2D simplification is shown as color bars in Fig. 2a), then calculates the difference in free energy of interaction between LC and external electric field ($F_{ext} = const \times \int_V \left[(\bar{n}(\bar{r}) \cdot \bar{E}(\bar{r}))\right] d\bar{r}$) between the new state (trial) and the previous one, and finally accepts or declines each of these changes (by Metropolis algorithm). Here V is the droplet volume and $E(\bar{r})$ is external electric field distribution. Green circles in Fig. 2a denote the points in which the director field is changed by the trial (so-called pivot points). This procedure seeks for the optimal distribution of director field $\bar{n}(\bar{r})$ which delivers a minimum to F_{ext}, when Monte Carlo is coupled with simulated annealing. According to the formulation of free energy, we use sparse 3D placement of pivot points to implement checkerboard decomposition technique. It allows processing these points in parallel, thus producing multiple independent local MCMC trials at one step.

Figure 2b illustrates the placement of the stored data in 2D simplification. Solid green squares represent pivot points processed simultaneously. During the next step, the set of pivot points will be shifted. Eight steps (also called ticks) cover the full 3D lattice, shown in Fig. 2c. On each step, the tick vector shows the current shift of the first pivot point from the bottom-most point of the lattice. The tick vector varies from $(0, 0, 0)$ to $(1, 1, 1)$ and cyclically increments from step to step. Thus, the most of device kernels are launched on a cubic CUDA grid of dimensions twice smaller than corresponding (physical) lattice size. A technical layer (blue squares in Fig. 2b) was introduced to make calculations on the border equivalent to the one in the inner part of the lattice. The zero contribution of technical layer to the total energy is granted by zero volumetric coefficients V_i in corresponding cells.

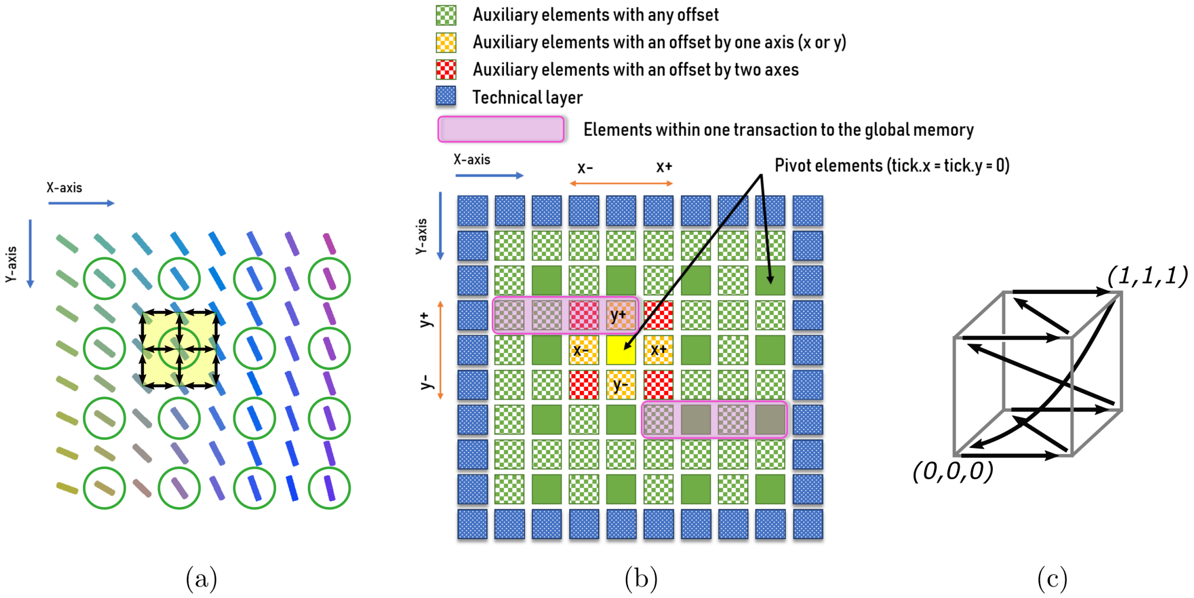

Auxiliary elements with any offset
Auxiliary elements with an offset by one axis (x or y)
Auxiliary elements with an offset by two axes
Technical layer
Elements within one transaction to the global memory
X-axis
Y-axis
x-
x+
y+
y-
Pivot elements (tick.x = tick.y = 0)
(1,1,1)
(0,0,0)
(a)
(b)
(c)

Figure 2. (a) 2D simplification of director field on a cubic lattice. Green circles show pivot points, to which trial changes of director vectors are applied in parallel. (b) Lattice configuration of the TCK (2D simplification). Pivot elements (tick.x = tick.y = 0) are solid squares on the picture, auxiliary elements (tick.x = 1 or tick.y = 1) are marked as checkered squares. Transactions to global memory from a warp of adjacent threads are shown as magenta rectangles. (c) Sequence of shifts of pivot points in 3D

In the implemented numerical scheme, the integral over droplet volume V is replaced with the sum over lattice cells (shown as yellow boxes in Fig. 2a): $F_{ext} = const \times \sum_i \left[\left\langle(\bar{n}(\bar{r}) \cdot \bar{E}(\bar{r}))\right\rangle_i \times V_i\right]$, where V_i is the volume of i-th cell, and $\langle ... \rangle_i$ is averaging over i-th cell. For better scheme convergence, the value $\left\langle(\bar{n}(\bar{r}) \cdot \bar{E}(\bar{r}))\right\rangle_i$ is averaged by 26 points per cell: cell corners (i.e., lattice points), middles of cell edges (so-called secondary lattice points), and middles of cell facets (so-called tertiary lattice points). Thus, the calculations of the energy change require not only the pivot point, but also neighbor points (illustrated by black arrows in Fig. 2a). It makes the TCK computations heavily memory-intensive.

The software implementation of the TCK consists of three major parts. In the first part of the typical kernel, each CUDA thread calculates indexes, used to access pivots and adjacent lattice elements during the following calculation. Since this kernel utilizes the lattice approach, each CUDA thread operates with a specific pivot element of the three-dimensional lattice and its neighborhood. Thus, 3^3 indexes are calculated, including the pivot (central) one. These indexes are stored on registers of streaming multiprocessors. Figure 2 shows a simplified mapping scheme of the relation between CUDA threads and data arrays processed by a kernel.

In the second part of the TCK, the data describing the state before the change in the director field is copied into separate arrays. This operation is necessary since the trial change can be declined by Metropolis algorithm, and then the previous state should be restored.

In the third part, the kernel calculates the values of F_{ext} for the trial director configuration. To finalize MCMC step, the Metropolis algorithm should be applied, and trial change of the director in each pivot point should be accepted or declined probabilistically.

Contribution of the TCK into the whole program package runtime can be evaluated using nvprof profiling tool. This contribution slightly changes when processing different physical problems, for example different lattice sizes or the requirement to process the surface of the droplet.

Our measurments demonstrated that the runtime contribution of the TCK varies between 12 % and 20 % for most of the tests.

5. Co-design of the Typical Kernel of the Evaluated Software Package

5.1. Detecting Primary Optimization Techniques

At first it is necessary to highlight primary optimization techniques, which can be implemented to increase the performance of the TCK. As mentioned in the previous section, the TCK shares various computational properties and features with a vast majority of other kernels of the evaluated program package, and thus the proposed optimizations can be later easily applied to the whole program package.

First of all we optimized the TCK for the NVIDIA GPU architecture, since it provides an extremely convenient profiling toolkit, which allows easily determining performance issues and bottlenecks of CUDA programs. In this paper we used nvprof and nvvp profiling tools in order to collect various dynamic characteristics of the program, which are necessary in the process of the optimization.

The dynamic characteristics and therefore performance and efficiency of the investigated kernel significantly depend on the grid size, as shown in Tab. 2.

Table 2. Utilization of P100 GPU resources of the TCK for various problem sizes

Grid parameters	Compute utilization	Memory utilization
128^3	10 %	55 %
64^3	10 %	35 %
16^3	15 %	25 %

Table 2 shows that a significant part of kernel runtime is spent on loading information from various levels of GPU memory (device and caches), and thus memory subsystem bandwidth is the primary performance bottleneck. Kernels with memory utilization ratio higher than 50 % are usually called *memory-bound*. Since the TCK launched on the large grids (128^3) is memory bound, its interaction with memory subsystem needs to be carefully investigated and optimized.

Memory subsystem usage of any kernel highly depends on memory access patterns, which can be characterized via efficiency of GPU-transactions. GPU-transaction is a process of loading continuous chunk of data (usually 128 bytes) from memory subsystem. The efficiency of the transaction can be estimated as the amount of useful data loaded from memory divided by the transaction size. When the kernel uses a sequential memory access pattern, transactions efficiency of such kernel is equal to 100 % in the case when the initial transaction addresses are 128-byte aligned. In the worst case (random memory access pattern), each GPU warp needs to generate the amount of transactions equal to the size of warp, and the transaction efficiency for such program is roughly equal to 3 %. Figure 2b highlights two transactions for the TCK: the first one (1) loads elements within vertical offset (by Y-axis), while the second one (2) loads pivot elements and simultaneously prefetches elements within horizontal offset (by X-axis) into L1 cache.

To assist developers in calculating the transaction efficiency of the specific kernel, special gld_efficiency and gst_efficiency hardware metrics of nvprof tool can be used. Elements of the physical lattice are sparsely located in the data arrays, for example pivot elements alternate with horizontal offset elements, as shown in Fig. 2. Hence, if the warp loads pivot elements from the global memory, the transactions will contain both pivots and elements within X-axis offsets, and the latter being redundant to load, resulting in the efficiency of such transactions being twice lower compared to the case when all required elements are stored densely.

Transaction efficiency also depends on the type of the requested elements. In the original program, the director field direction in each lattice point is stored in three double precision variables. Thus unrolling arrays of structures (AoS) into structures of arrays (SoA) allows us to further increase transaction efficiency and significantly accelerate the investigated kernel.

Despite the fact that transactions to global memory in the TCK have low efficiency, memory access pattern of the kernel has an important advantage. Elements, which neighbor pivots and fall into a requested transaction when pivots are loaded (shaded purple-colored elements in Fig. 2) are prefetched into L1 GPU cache, resulting into further transactions to these elements being processed significantly faster, since they load the required data from caches instead of global memory. Due to the fact that a significant amount of elements neighboring pivots reside in L1 cache (L1 hit rate of the TCK is 72 %), only a small number of transactions will be directed to L2 cache or device memory.

Another important feature of every CUDA kernel is occupancy. Occupancy is a ratio of active warps resident on a single streaming multiprocessor to a maximum theoretical value of active warps supported by a single SM. Occupancy of the TCK launched on the smallest computational grid of size 16 is very low – only 64 blocks of size 512 are distributed over 56 SMs, while each SM allows processing up to 4 blocks of similar size simultaneously. Such small number of blocks can not fully utilize GPU resources, for example global memory bandwidth. On the other hand, when the kernel is launched on the largest grid (128^3), it fully occupies each SM with 512 blocks launched in total. Occupancy for the TCK launched on small grids can be increased using multitasking, when two or more independent CUDA-kernels are concurrently launched on a single GPU.

Occupancy of the investigated kernel is also limited by the number of registers available on a single SM. Registers are heavily used in the TCK to store both indexes and intermediate elements of physical lattice, which are accessed multiple times in different parts of the kernel. Thus, using compiler directives to limit a number of registers used by each thread is another important direction of further optimization.

5.2. Optimizations for NVIDIA GPU Architecture

Implementing a coalesced memory access pattern to global memory. Figure 2 and subsection 5.1 explain why the existing memory access pattern used in the initial version of the TCK has low efficiency of load transactions. To avoid this problem, the lattice storage configuration has been changed in the following way: the lattice was split into 8 independent parts, each corresponding to a particular shift in 3D-version of checkerboard algorithm (value of the *tick* variable), as shown in Fig. 2c. Figure 3 demonstrates a simplified 2D version of storage splitting scheme, where checkerboard algorithm supposes the use of four ticks. Here, on each tick, all pivot points are stored in the same part of lattice. This way load transactions to elements in different parts of lattice do not contain gaps filled with elements with another offset. As a result, memory accesses to elements within different offsets will be coalesced; such optimization approximately doubles the efficiency of load and store transactions for the investigated kernel, according to the results from the Tab. 3. Since the described optimization changes the structure of the kernel significantly, all further optimizations in this paper will be applied to this kernel with this new memory access pattern.

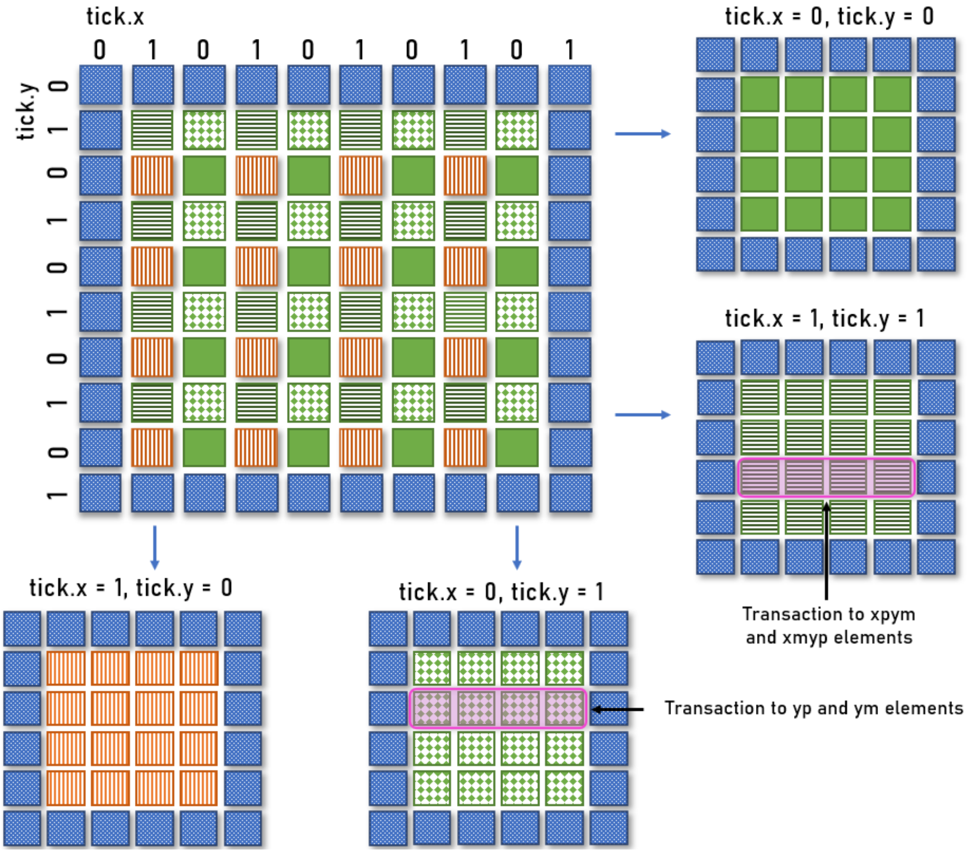


Figure 3. Lattice configuration in the investigated kernel (2D simplification). Pivot elements (tick.x=tick.y=0) are solid squares on the picture, auxiliary elements (tick.x $\neq$ 0 or tick.y $\neq$ 0) are marked as checkered squares. Transactions to global memory from a warp of adjacent threads are shown as rectangles

Table 3. The comparison of transaction efficiency for different memory access patterns

	Initial memory access pattern	**Improved memory access pattern**
Store transactions efficiency	34 %	80 %
Load transactions efficiency	43 %	70 %

Using shared memory. Shared memory of NVIDIA GPUs is a hand-driven L1 cache. Stencil kernels can benefit a lot from using this memory, since inner neighbor threads in a warp generate a transaction with highly reused elements, and with the use of shared memory we can avoid sending more than 1 transaction to global memory and do not care if required elements reside in non-programmable L1 cache. Typically each thread copies its central element from global to shared memory, allowing other threads to access this element with lower latency and higher bandwidth.

However, kernels of optimized package (and thus TCK) are not pure stencils and perform other operations, such as copying energies from previous step and making a Monte-Carlo decision. These parts can not benefit from using shared memory, since they are based on typical sequential memory access patterns and are memory-bound. This is the first problem of using shared-memory based optimizations in the kernels of our package: due to Amdal's law, the obtained acceleration will be much lower compared to the cases of usual stencil codes [25].

When applying shared memory optimizations to the stencil part of TCK, we face another problem: it uses a huge amount of input arrays (around 20 float arrays for TCK), required for stencil computations. Certainly, all these arrays can not be stored in shared memory due to its limited size of 64KB or 96KB per one SM. A possible solution can be the usage of registers [25], but according to the nvprof the majority of them is already used for storing other intermediate data.

A possible solution involves storing arrays in shared memory step-by-step, implementing a sort of pipeline. Unfortunately, computational-intensive functions operate only with 2 elements of stencil radius at once to calculate parts of each pivot element, which results in low data reuse. In addition, the implementation of such pipeline requires frequent thread synchronization. For these reasons shared memory optimizations can not be applied to TCK and other kernels of the package, which is confirmed by our implementation experiments.

Changing precision from *double* to *single*. Since the initial kernel launched on a large computational grid is memory-bound, it is reasonable to try to change the type of lattice elements from double to float in order to significantly reduce the amount of loaded data. We determined the critical parts of the code, where changing the precision from double to float affected the results. It corresponds to the parts where exponential functions are taken, and where reduction over large number of values is produced. It required to remain about 1 % of the variables in double precision. The other variables were switched to float. After this optimization the amount of load transactions is 1.3 times lower for single precision compared to double precision (30.7M transactions for double and 23.2M transactions for float), which is proportional to the acceleration obtained by this optimization. It is important to notice that physical results of calculations are correct when obtained via both single and double precision.

Unrolling array of structures into structure of arrays. Unrolling array of structures (AoS) into structure of arrays (SoA) allows us to further reduce the amount of memory transactions. With this optimization applied, the ratio of the requested data to the required data loaded from the global memory has doubled – from 24 % to 46 %. However, this optimization did not lead to the proportional acceleration, since this optimization also decreases the efficiency of using L1 cache: the accesses to fields of structures occur in the TCK close to each other, and thus the remaining fields are typically prefetched into L1 cache in the case of using AoS.

Decreasing the transaction size. NVIDIA GPUs allow turning off L1 cache in order to decrease the size of memory transaction and thus improving the effective bandwidth for non-coalesed memory access pattern. When L1 cache is disabled, the transaction has 4 times larger size (128 against 32 bytes). In this paper we evaluated kernel performance with L1 cache both turned on and turned off, and the experiments demonstrated a 15 % speedup with L1 cache turned on.

Increasing the occupancy by limiting the register usage. One of the reasons why occupancy of the investigated kernel is low is that each CUDA thread uses many GPU registers – approximately 70 of them, and consequently a block of 512 threads requires 35840 registers. Each streaming multiprocessor of P100 GPU has only 32768 registers available, which does not allow running two or more blocks concurrently on the same SM. The *launch_bounds (maxThreadsPerBlock, minBlocksPerMultiprocessor)* directive can be used to limit the amount of registers utilized by a single CUDA-block, which allows running up to 4 blocks of size 512 on a single P100 SM concurrently (since there is also a hardware thread limit of 2048 threads per SM). However, applying this directive with a parameter *minBlocksPerMultiprocessor=4* turns the kernel into being more memory-bound, since the ratio of memory operations among the whole kernel increases from 35 % to 55 % and does not lead to any significant acceleration.

Applying multitasking for small grids. Occupancy values are significantly higher for large computational grids, since the amount of CUDA threads launched for such grids is also high. Insufficient number of threads launched on small grids decreases the efficiency of using available throughput of GPU memory. The achieved global memory throughput can be calculated as a sum of two nvprof metrics – gld_throughput and gst_throughput. On computational grids of size 64^3 this sum is approximately equal to 420 GB/s and remains the same for all larger grids, thus this value can be viewed as an achievable limit for the investigated kernel on a P100 GPU. This value is also relatively close to the achieved throughput of 540 GB/s on Stream benchmark [4], which also has a coalesced access pattern, but transactions of which are aligned (unlike the TCK).

For smaller grids the investigated kernel achieves throughput equal to 295 GB/s, which is 1.4 times lower compared to throughput on large grids. This gap can be eliminated by using multitasking: multiple independent instances of kernel can be launched in parallel on a single GPU, thus utilizing its hardware resources more efficiently. This way GPU occupancy is increased, which in turn allows us to hide latency. We implemented multitasking using OpenMP directives and CUDA Streams. Table 4 shows that an expected 1.4 times acceleration has been achieved.

Table 4. Theoretical acceleration which can be achieved by launching independent kernels (tasks) in parallel

	Single task	Multiple tasks	Theoretical speedup
gst+gld throughput on 64^3 lattice	975.1 GB/s	1375.82 GB/s	1.41
gst+gld throughput on 128^3 lattice	321.414 GB/s	474.97 GB/s	1.47

The overall comparison of effects from applying different types of optimizations to the TCK (launched on NVIDIA P100 GPU) is shown in Fig. 5.

5.3. Porting the Typical Kernel to NEC SX-Aurora TSUBASA

Since vector architectures and GPUs have a significant number of similar architectural and computing features [1], the TCK can be ported to the NEC SX-Aurora TSUBASA architecture

Table 5. The comparison of bandwidth (BW) and execution time values for different versions of the TCK launched on NVIDIA P100 GPU

lattice size	**initial version**	**new pattern**	**AoS to SoA**	**register usage**	**changed precision**	**multi-tasking**
64^3(BW)	319.87 GB/s	551.78 GB/s	569.05 GB/s	576.60 GB/s	409.76 GB/s	594.84 GB/s
64^3(time)	319.22 s	172.17 s	179.21 s	171.52 s	126.75 s	87.35 s
128^3(BW)	479.18 GB/s	710.67 GB/s	716.16 GB/s	558.38 GB/s	460.24 GB/s	754.21 GB/s
128^3(time)	678.1 s	457.20 s	453.7 s	581.9 s	352.98 s	339.43 s

in a relatively straightforward way. The investigated CUDA-kernel is transformed into a 3-dimensional nested loop; at each iteration of the innermost loop the same program code is executed by each of CUDA threads. Since the initial version of the kernel does not use any architecture-dependent features of the GPUs (such as shared memory, special instructions, etc.), the program code can be used on NEC SX-Aurora TSUBASA architecture without any changes. The number of iterations inside each nested loop is equal to the size of the CUDA grid in one dimension (x, y, z), and the innermost loop corresponds to X dimension of CUDA grid. This means that the innermost loop is used to process X-axis elements of lattice shown in Fig. 2. Since all iterations inside each of the nested loop are independent, computations inside the loops can be parallelized and vectorized in several different ways. NEC SX-Aurora TSUBASA architecture does not allow efficient loading information from arrays of structures. Thus, the initial version of the program was implemented with coordinate data structures unrolled into three separate arrays.

The following subsections describe the most important optimizations required to obtain a high-performance version of the TCK for NEC SX-Aurora TSUBASA vector engines. Since the investigated computational kernel is memory-bound (as demonstrated in the previous sections for GPUs), the sustained memory bandwidth values will be used during its performance evaluation. The speedup achieved by implementing each of the further discussed optimizations is listed in Tab. 7 in the end of this section.

Parallelization and vectorization of the program. When porting algorithms consisting of multiple nested loops to vector architectures, the innermost loop is usually a subject to the vectorization. According to the previously discussed kernel transformation, the innermost loop processes adjacent lattice elements located in adjacent cells, which allows vector instructions to follow a relatively efficient memory access pattern. The outer loop is parallelized using OpenMP *#pragma omp parallel for* clause and *schedule (static)* mode for distributing iterations.

Improving memory access pattern. Since the investigated kernel is memory-bound, vector instructions should have a specific vector-friendly memory access pattern. For the NEC SX-Aurora TSUBASA architecture, the linear dependence between the loop iteration indexes and the indexes of the accessed arrays is the necessary condition: all arrays indexes should be represented as $i+offset$, where i is the index of the innermost (vectorized) loop. If this condition is satisfied, then vector LOAD and STORE instructions are used, otherwise – GATHER and SCATTER instructions, which are significantly less efficient. When vectorizing the initial version of the kernel, memory access indexes can be represented as $i*2+tick+1$, as illustrated in Fig. 2. This pattern leads to the usage of GATHER and SCATTER instructions which significantly reduces the performance of the kernel. Thus, proposed in section 5.2 data-layout transformation is required for the NEC SX-Aurora TSUBASA architecture.

Collapsing nested loops. Vectorizing only the innermost loop has a significant downside: in the case of small lattices, vector instructions have relatively low length (16–32 compared to the desirable value of 256, equal to the maximum vector length of NEC SX-Aurora TSUBASA architecture), since the innermost loop does not have enough iterations to be vectorized with instructions of maximum length. However, all three-dimensional loops can be collapsed (merged) into a single linear loop, which typically has enough iterations for simultaneous vectorization and parallelization. This optimization also allowed us to achieve a significant speedup as shown in Tab. 7.

For discussed optimizations, Tab. 6 demonstrates the comparison of three important metrics: average vector length, vector operation ratio and and last level cache hit rate. These values are received with special tool for NEC Vector architectures called Ftrace.

Table 6. Performance metrics for different versions of the TCK, implemented for the NEC SX-Aurora TSUBASA architecture

metric	**initial version**	**vectorisation and parallelisation**	**improved memory access pattern**	**loops collapsed**
average vector length	1	33.9	249.6	249.6
vector operation ratio	0 %	99 %	99 %	99 %
LLC error rate	N/A	86 %	77 %	58 %

Using multitasking for small lattices. Despite applying loop collapse optimization, the performance of the developed kernel on small lattices is still limited by the insufficient amount of computational work. Thus, in such cases multitasking optimization can be implemented, similar as for NVIDIA GPUs: different kernel runs are executed on independent vector cores of SX-Aurora, while each kernel is vectorized, but not parallelized. However, the acceleration obtained by this optimization is significantly lower compared to GPUs. The number of runs in parallel affects neither the average length of a vector operation, nor the ratio of vector operations in a kernel and LLC hit rate, so obtained values by Ftrace in Tab.6 are sufficient.

Converting precision from *double* to *single*. Same as for NVIDIA GPUs, changing the kernel to operate with single precision instead of double allows halving the amount of data loaded from memory during kernel execution. However, in case of NEC SX-Aurora TSUBASA, LOAD and STORE vector instructions for double precision are capable of loading twice the amount of data compared to single precision, in approximately the same time. This causes a relatively low speedup when applying this optimization. As the number of elements and strategy of loading elements from memory remains the same as for previous optimizations, we do not need to update Tab.6 for such optimization.

The overall comparison of effects from applying different types of optimizations to the TCK (launched on NEC SX-Aurora TSUBASA) is shown in Fig. 7.

Table 7. The comparison of the sustained bandwidth(BW) and the execution time for different versions of the TCK ported to the NEC SX-Aurora TSUBASA architecture

lattice size	**initial version**	**vectorisation and parallelisation**	**improved memory access pattern**	**loops collapsed**	**multi-tasking**	**double to float precision**
32^3(BW) 32^3(time)	0.35 GB/s -	38 GB/s 3.33 s	63 GB/s 2.00 s	380 GB/s 0.33 s	391 GB/s 0.32 s	506 GB/s 0.25 s
64^3(BW) 64^3(time)	0.28 GB/s -	53 GB/s 18.9 s	129 GB/s 7.8 s	537 GB/s 1.8 s	576 GB/s 1.7 s	638 GB/s 1.5 s
128^3(BW) 128^3(time)	0.28 GB/s -	89 GB/s 91.2 s	249 GB/s 32.5 s	615 GB/s 13.2 s	625 GB/s 12.9 s	673 GB/s 12.0 s
256^3(BW) 256^3(time)	0.28 GB/s -	134 GB/s 482 s	395 GB/s 164 s	657 GB/s 98 s	690 GB/s 95 s	725 GB/s 90 s

5.4. Porting the Typical Kernel to Intel Xeon Architecture

Developing CPU version of the TCK is important, since the performance comparison between CPUs and GPUs (or NEC SX-Aurora TSUBASA) is interesting for many researches. Frequently, they need to understand if porting their programs or packages to new architectures is justified. Thus, readers of our paper may be able to obtain this knowledge, in the case when their programs are based on similar stencil schemes or mathematical algorithms, described in sections 2–4.

The performance of the developed CPU version was evaluated on the node of Lomonosov-2 supercomputer, which is equipped with Intel Xeon Gold 6126 processors.

Since OpenMP programming model can be efficiently used for modern Intel Xeon CPUs, the initial version of the typical computational kernel can be obtained in the same way as for the NEC SX-Aurora TSUBASA architecture, as described in section 5.3. Thus, when porting TCK to Intel Xeon, three-dimensional nested loop has been collapsed into one big loop, since such transformation allows achieving better parallelization and using AVX-512 vector instructions of Intel CPUs.

Next, we have studied what compiler shows better results for our application. We compared classical GNU gcc compiler to the Intel compiler (icpc), using the same flags in both cases. By choosing icpc, we managed to obtain a constant 3–4 % speedup, depending on a lattice size.

After that we studied the efficiency of OpenMP parallel instructions, applied to the investigated kernel. Using Intel VTune we have investigated that the whole CPU utilization of parallel version is 87 %, with the ratio of serial instructions below 1 %. That makes OpenMP directives quite relevant for investigated kernel.

Furthermore, we implemented a new memory access pattern, described above. The old version of a kernel had a small ratio of retired micro-ops (which shows the fraction of time processor was fully utilized by useful work), while ∼80 % of micro-ops were stalled waiting for data from DRAM (i.e., back-end bound stalls). New memory access pattern increased the ratio of retired instructions to 45 %, making the ratio of back-end bound micro-ops equal to 17 %. Finally, a vectorization of investigated kernel was studied. By placing *#pragma simd* directive and *-qoverride-limits* with *-qopt-zmm-usage=high* compiler flags, we managed to obtain 1.5 faster version of typical computational kernel.

Thus, the developed CPU version demonstrates high utilization of hardware resources, is efficiently parallelized and vectorized, and therefore in our opinion can be used for comparison with other two evaluated platforms.

6. Comparison of the Typical Kernel Performance on Different Architectures

After optimizing the TCK on different platforms, we cross-compared the behaviour of each optimization on each platform. Contributions of individual optimizations on each platform are given in Tabs. 5, 7, and sec. 5.4.

Multitasking did not lead to any significant acceleration on NEC SX-Aurora TSUBASA and Intel Xeon architectures, since they both have only a few cores (12 and 8 respectively). It can be efficiently occupied with calculations even by processing small lattices. Thus, multitasking of the studied program was found to be unnecessary on these platforms. At the same time, NVIDIA GPUs have much higher number of cores, which were hard to fully utilize without multitasking. Thus implementation of multitasking on NVIDIA GPUs resulted in a significant performance increase. Moreover, newer GPU microarchitecture demonstrated higher acceleration from multitasking optimization, since the resource of hardware parallelism is constantly growing with each generation of GPU.

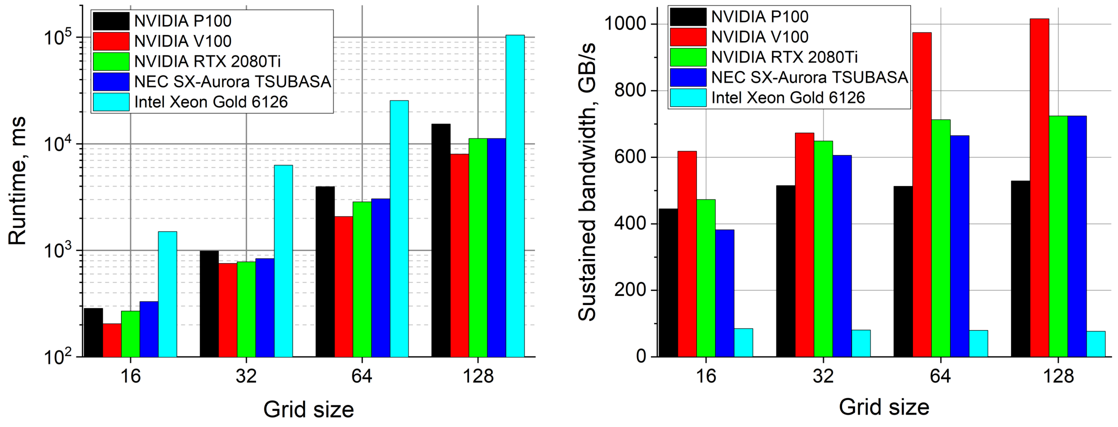
Improving memory access pattern (eliminating arrays of structures and splitting the lattice) was found to be absolutely crucial for the NEC SX-Aurora TSUBASA architecture, since using LOAD and STORE instructions leads to 3–4 times acceleration. The same optimization is not determinative for NVIDIA GPUs, because the initial memory access pattern provides higher utilization of L1 cache. Still, the new memory access pattern reduces runtime of the TCK by 1.5–2 times. For Intel Xeon processors, the new memory access pattern increases the number of retired micro-ops and thus reduced a kernel runtime by 1.5 times.

Changing precision demonstrates the highest acceleration (1.35 times) on NVIDIA GPUs, while practically does not speed up calculations on NEC SX-Aurora TSUBASA architecture due to the implementation details of LOAD instructions in it.

Figure 4 shows the overall comparison of the TCK performance on target platforms. These data provide the runtime and the sustained bandwidth values for most optimized versions of the TCK on each architecture. The sustained bandwidth values for all target architectures are proportional to the theoretical peak bandwidth values. The highest bandwidth (and therefore lower runtime) is achieved on NVIDIA V100 GPU architecture. NVIDIA RTX 2080 Ti GPU performs slightly better than NEC on small and medium grids, but on large grids their bandwidths and runtimes are equal. NVIDIA P100 GPU demonstrated results 15 %–40 % below NVIDIA RTX 2080 Ti GPU and NEC SX-Aurora TSUBASA. In comparison to these architectures, Intel Xeon Gold 6126 bandwidth are strikingly lower, and runtime is larger, accordingly.

Conclusions

In this paper, we investigated and improved the computational efficiency of simulations software for liquid crystals on various modern supercomputing architectures. The computational problem in this software belongs to the class of stochastic optimization of a functional defined on finite space cubic lattice. Namely, the solver is based on Markov chain Monte Carlo with Metropolis algorithm, paralleled by sparse checkerboard decomposition. The following plat-



(a) (b)

Figure 4. The comparison of (a) runtime (in logarithmic scale) and (b) sustained bandwidth values of the TCK on different platforms

forms were used: NVIDIA GPU (Pascal, Volta and Turing microarchitectures), NEC SX-Aurora TSUBASA vector engines, and Intel Xeon Gold 6126 processors.

We studied and compared the efficiency of multiple optimization strategies for this software on each platform. These included the usage of more suitable memory access patterns, implementation of multitasking for efficient utilization of massive parallelism of the platforms, improving cache hit-rates, parallel workload balancing and several others.

As a result of the provided research, evaluated platforms can be ranked by the ability to efficiently solve this discussed class of problems as follows: (1) NVIDIA V100 GPUs, (2) NEC SX-Aurora TSUBASA, (3) NVIDIA RTX 2080 Ti GPUs, (4) NVIDIA P100 GPUs, (5) Intel Xeon Gold 6126 CPUs. It should be noted that NVIDIA GPUs and NEC SX-Aurora TSUBASA showed roughly equal efficiency and performance for this type of computational problems, while Intel Xeon processors demonstrated significantly lower performance for it.

The optimization techniques demonstrated above are useful not only to the particular program of liquid crystals simulations software. Instead, we believe it can be applied to a wide range of computational problems dealing with Markov chain Monte Carlo simulations on finite space cubic lattice, including ensemble simulations, aim search, stochastic optimization and other techniques, aimed to solve problems in mathematics, computational physics, chemistry and biology, economics and multidisciplinary studies.

Acknowledgments

The reported study was funded by the Russian Foundation for Basic Research, project number 20-37-70036. The work presented in section 5.3 is supported by the Russian Ministry of Science and Higher Education, agreement No. 075-15-2019-1621. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University.

References

1. Afanasyev, I.V., Voevodin, V.V., Voevodin, V.V., et al.: Analysis of Relationship Between SIMD-Processing Features Used in NVIDIA GPUs and NEC SX-Aurora TSUBASA Vector Processors. In: Parallel Computing Technologies - 15th International Conference, PaCT 2019, Proceedings. Lecture Notes in Computer Science, vol. 11657, pp. 125–139. Springer (2019). https://doi.org/10.1007/978-3-030-25636-4_10

2. Barbosa, C.H., Kunstmann, L.N., Silva, R.M., et al.: A workflow for seismic imaging with quantified uncertainty. Computers & Geosciences 145, 104615 (2020). https://doi.org/10.1016/j.cageo.2020.104615

3. Baxter, R.: The inversion relation method for some two-dimensional exactly solved models in lattice statistics. Journal of Statistical Physics 28(1), 1–41 (1982). https://doi.org/10.1007/BF01011621

4. Bergstrom, L.: Measuring NUMA effects with the STREAM benchmark. CoRR abs/1103.3225 (2011), http://arxiv.org/abs/1103.3225

5. Block, B., Virnau, P., Preis, T.: Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model. Computer Physics Communications 181(9), 1549–1556 (2010). https://doi.org/10.1016/j.cpc.2010.05.005

6. Boroni, G., Dottori, J., Rinaldi, P.: Full GPU implementation of Lattice-Boltzmann methods with Immersed Boundary Conditions for Fast Fluid Simulations. The International Journal of Multiphysics 11(1), 1–14 (2017). https://doi.org/10.21152/1750-9548.11.1.1

7. Dudzin-acuteski, M., Sznajd, J.: Suzuki-Trotter decomposition and renormalization of a transverse-field Ising model in two dimensions. Phys. Rev. B 55(22), 14948–14952 (1997). https://doi.org/10.1103/PhysRevB.55.14948

8. Egawa, R., Komatsu, K., Momose, S., et al.: Potential of a modern vector supercomputer for practical applications: performance evaluation of SX-ACE. The Journal of Supercomputing 73(9), 3948–3976 (2017). https://doi.org/10.1007/s11227-017-1993-y

9. Fang, Y., Feng, S., Tam, K.M., et al.: Parallel tempering simulation of the three-dimensional Edwards–Anderson model with compact asynchronous multispin coding on GPU. Computer Physics Communications 185(10), 2467–2478 (2014). https://doi.org/10.1016/j.cpc.2014.05.020

10. Floudas, C.A., Pardalos, P.M.: Encyclopedia of Optimization. Springer Science+Buisiness Media, LLC. (2009), https://www.springer.com/gp/book/9780387747583

11. Geng, Y., Noh, J., Drevensek-Olenik, I., et al.: High-fidelity spherical cholesteric liquid crystal Bragg reflectors generating unclonable patterns for secure authentication. Scientific Reports 6(1) (2016). https://doi.org/10.1038/srep26840

12. Goodby, J.W., Tschierske, C., Raynes, P., et al. (eds.): Handbook of Liquid Crystals. Wiley-VCH Verlag GmbH & Co. KGaA (2014). https://doi.org/10.1002/9783527671403

13. Gourgoulias, K., Katsoulakis, M.A., Rey-Bellet, L.: Information criteria for quantifying loss of reversibility in parallelized KMC. Journal of Computational Physics 328, 438–454 (2017). https://doi.org/10.1016/j.jcp.2016.10.031

14. Gourgoulias, K., Katsoulakis, M.A., Rey-Bellet, L.: Information metrics for long-time errors in splitting schemes for stochastic dynamics and parallel kinetic Monte Carlo. SIAM Journal on Scientific Computing 38(6), A3808–A3832 (2016). https://doi.org/10.1137/15m1047271

15. Harju, A., Siro, T., Canova, F.F., et al.: Computational physics on graphics processing units. In: Applied Parallel and Scientific Computing - 11th International Conference, PARA 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7782, pp. 3–26. Springer (2012). https://doi.org/10.1007/978-3-642-36803-5_1

16. Huth, B., Meyer, N., Wettig, T.: Lattice QCD on a novel vector architecture. CoRR abs/2001.07557 (2020), https://arxiv.org/abs/2001.07557

17. Kapitan, V.Y., Nefedev, K.V.: High performance calculation of magnetic properties and simulation of nonequilibrium phenomena in nanofilms. In: Modeling, Simulation and Optimization of Complex Processes - HPSC 2012, pp. 95–107. Springer (2014). https://doi.org/10.1007/978-3-319-09063-4_8

18. Khan, M., Li, W., Mao, S., et al.: Real-time imaging of ammonia release from single live cells via liquid crystal droplets immobilized on the cell membrane. Advanced Science 6(20), 1900778 (2019). https://doi.org/10.1002/advs.201900778

19. Komatsu, K., Momose, S., Isobe, Y., et al.: Performance evaluation of a vector supercomputer SX-Aurora TSUBASA. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis. pp. 54:1–54:12. SC ’18, IEEE, Piscataway, NJ, USA (2018). https://doi.org/10.1109/SC.2018.00057

20. Krakhalev, M.N., Rudyak, V.Yu., Prishchepa, O.O., et al.: Orientational structures in cholesteric droplets with homeotropic surface anchoring. Soft Matter 15(28), 5554–5561 (2019). https://doi.org/10.1039/c9sm00384c

21. Lagerwall, J.P., Scalia, G.: A new era for liquid crystal research: Applications of liquid crystals in soft matter nano-, bio- and microtechnology. Current Applied Physics 12(6), 1387–1412 (2012). https://doi.org/10.1016/j.cap.2012.03.019

22. Larsen, T., Bjarklev, A., Hermann, D., Broeng, J.: Optical devices based on liquid crystal photonic bandgap fibres. Optics Express 11(20), 2589 (2003). https://doi.org/10.1364/oe.11.002589

23. Li, W., Fan, Z., Wei, X., Kaufman, A.: GPU-Based flow simulation with complex boundaries. Tech. rep. (2003)

24. Maltseva, D., Zablotskiy, S., Martemyanova, J., et al.: Diagrams of states of single flexible-semiflexible multi-block copolymer chains: A flat-histogram Monte Carlo study. Polymers 11(5) (2019). https://doi.org/10.3390/polym11050757

25. Maruyama, N., Nomura, T., Sato, K., Matsuoka, S.: Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In: Conference on High Performance Computing Networking, Storage and Analysis, SC 2011. pp. 11:1–11:12. ACM (2011). https://doi.org/10.1145/2063384.2063398

26. Otsuka, Y., Seo, H., Motome, Y., Kato, T.: Finite-temperature phase diagram of quasi-one-dimensional molecular conductors: Quantum Monte Carlo study. Journal of the Physical Society of Japan 77(11), 113705 (2008). https://doi.org/10.1143/jpsj.77.113705

27. Peng, B., Li, J., Akkas, S., et al.: Rank position forecasting in car racing. In: 35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021, Portland, OR, USA, May 17-21, 2021. pp. 724–733. IEEE (2021). https://doi.org/10.1109/IPDPS49936.2021.00082

28. Phillips, J.C., Hardy, D.J., Maia, J.D.C., et al.: Scalable molecular dynamics on CPU and GPU architectures with NAMD. The Journal of Chemical Physics 153(4), 044130 (2020). https://doi.org/10.1063/5.0014475

29. Raedt, H.D., Lagendijk, A.: Monte Carlo simulation of quantum statistical lattice models. Physics Reports 127(4), 233–307 (1985). https://doi.org/10.1016/0370-1573(85)90044-4

30. Rao, S.S.: Engineering Optimization: Theory and Practice. John Wiley & Sons, Inc. (2009), https://www.wiley.com/en-us/Engineering+Optimization%3A+Theory+and+Practice%2C+5th+Edition-p-9781119454793

31. Rathore, N., de Pablo, J.J.: Monte Carlo simulation of proteins through a random walk in energy space. The Journal of Chemical Physics 116(16), 7225–7230 (2002). https://doi.org/10.1063/1.1463059

32. Resch, M.M., Kovalenko, Y., Bez, W., et al. (eds.): Sustained Simulation Performance 2018 and 2019. Springer (2020). https://doi.org/10.1007/978-3-030-39181-2

33. Romero, J., Bisson, M., Fatica, M., Bernaschi, M.: High performance implementations of the 2D Ising model on GPUs. Computer Physics Communications 256, 107473 (2020). https://doi.org/10.1016/j.cpc.2020.107473

34. Rudyak, V.Yu., Emelyanenko, A.V., Loiko, V.A.: Structure transitions in oblate nematic droplets. Physical Review E 88(5) (2013). https://doi.org/10.1103/physreve.88.052501

35. Rudyak, V.Y., Krakhalev, M.N., Sutormin, V.S., et al.: Electrically induced structure transition in nematic liquid crystal droplets with conical boundary conditions. Physical Review E 96(5) (2017). https://doi.org/10.1103/physreve.96.052701

36. Shakirov, T., Zablotskiy, S., Boeker, A., et al.: Comparison of Boltzmann and Gibbs entropies for the analysis of single-chain phase transitions. The European Physical Journal H 226, 705–723 (2017). https://doi.org/10.1140/epjst/e2016-60326-1

37. Shao, W., Guo, G.: Multiple-try simulated annealing algorithm for global optimization. Mathematical Problems in Engineering 2018, 1–11 (2018). https://doi.org/10.1155/2018/9248318

38. Shvetsov, S.A., Emelyanenko, A.V., Boiko, N.I., et al.: Communication: Orientational structure manipulation in nematic liquid crystal droplets induced by light excitation of azodendrimer dopant. The Journal of Chemical Physics 146(21), 211104 (2017). https://doi.org/10.1063/1.4984984

39. Shvetsov, S.A., Rudyak, V.Yu., Emelyanenko, A.V., et al.: Photoinduced orientational structures of nematic liquid crystal droplets in contact with polyimide coated surface. Journal of Molecular Liquids 267, 222–228 (2018). https://doi.org/10.1016/j.molliq.2018.01.054

40. Sivakumar, S., Wark, K.L., Gupta, J.K., et al.: Liquid crystal emulsions as the basis of biological sensors for the optical detection of bacteria and viruses. Advanced Functional Materials 19(14), 2260–2265 (2009). https://doi.org/10.1002/adfm.200900399

41. Tian, Z., Yokoyama, H., Araki, T.: Parallel latent dirichlet allocation using vector processors. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). pp. 1548–1555. IEEE (2019). https://doi.org/10.1109/hpcc/smartcity/dss.2019.00213

42. Tran, N.P., Lee, M., Hong, S.: Performance optimization of 3D lattice Boltzmann flow solver on a GPU. Scientific Programming 2017, 1–16 (2017). https://doi.org/10.1155/2017/1205892

43. Tu, J., Yeoh, G.H., Liu, C.: Chapter 9 - some advanced topics in CFD. In: Tu, J., Yeoh, G.H., Liu, C. (eds.) Computational Fluid Dynamics (Third Edition), pp. 369–417. Butterworth-Heinemann, third edition edn. (2018). https://doi.org/10.1016/B978-0-08-101127-0.00009-X

44. Weigel, M.: Simulating spin models on GPU. Computer Physics Communications 182(9), 1833–1836 (2011). https://doi.org/10.1016/j.cpc.2010.10.031

45. Weigel, M.: Monte Carlo methods for massively parallel computers. In: Order, Disorder and Criticality, pp. 271–340. World Scientific (2017). https://doi.org/10.1142/9789813232105_0006

46. Yamada, Y., Momose, S.: Vector engine processor of NEC Brand-New supercomputer SX-Aurora TSUBASA. In: Intenational symposium on High Performance Chips (Hot Chips2018) (2018)

47. Yusoff, M.N.S., Jaafar, M.S.: Performance of CUDA GPU in Monte Carlo simulation of light-skin diffuse reflectance spectra. In: 2012 IEEE-EMBS Conference on Biomedical Engineering and Sciences. pp. 264–269. IEEE (2012). https://doi.org/10.1109/iecbes.2012.6498056

48. Zablotskiy, S.V., Martemyanova, J.A., Ivanov, V.A., Paul, W.: Diagram of states and morphologies of flexible-semiflexible copolymer chains: A Monte Carlo simulation. Journal of Chemical Physics 144(24), 244903 (2016). https://doi.org/10.1063/1.4946035