Post-print of: Czarnul P., Kuchta J., Matuszek M., Proficz J., Rościszewski P., Szymański J., Wójcik M.: MERPSYS: An environment for simulation of parallel application execution on large scale HPC systems. SIMULATION MODELLING PRACTICE AND THEORY. Vol. 77, (2017), p. 124-140. DOI: 10.1016/j.simpat.2017.05.009

MERPSYS: An environment for simulation of parallel application execution on large scale HPC systems

Paweł Czarnul [a] Jarosław Kuchta [a], Mariusz Matuszek [a], Jerzy Proficz [b], Paweł Rościszewski [a], Michał Wójcik [a], Julian Szymański [a]

[a] *Department of Computer Architecture, Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, Gdańsk 80-233, Poland*

[b] *Academic Computer Centre, Narutowicza 11/12, Gdańsk 80-233, Poland*

a b s t r a c t

In this paper we present a new environment called MERPSYS that allows simulation of parallel application execution time on cluster-based systems. The environment offers a modeling application using the Java language extended with methods representing mes- sage passing type communication routines. It also offers a graphical interface for building a system model that incorporates various hardware components such as CPUs, GPUs, inter- connects and easily allows various formulas to model execution and communication times of particular blocks of code. A simulator engine within the MERPSYS environment simu- lates execution of the application that consists of processes with various codes, to which distinct labels are assigned. The simulator runs one Java thread per label and scales com- putations and communication times adequately. This approach allows fast coarse-grained simulation of large applications on large-scale systems. We have performed tests and veri- fication of results from the simulator for three real parallel applications implemented with C/MPI and run on real HPC clusters: a master-slave code computing similarity measures of points in a multidimensional space, a geometric single program multiple data parallel application with heat distribution and a divide-and-conquer application performing merge sort. In all cases the simulator gave results very similar to the real ones on configurations tested up to 1000 processes. Furthermore, it allowed us to make predictions of execution times on configurations beyond the hardware resources available to us.

Keywords: Parallel computing, Performance simulation, Simulation environment, Cluster systems

1. Introduction

Nowadays parallel systems have grown in size and complexity considerably. Parallel computing can be exploited within a single node using multicore CPUs, accelerators such as GPUs or coprocessors of a Intel Xeon Phi type. The number of cores and performance of such compute devices have been growing recently with several tens of cores in latest Intel® Xeon® CPUs [1], a few thousand of CUDA cores in latest NVIDIA GPUs [2]. Computer nodes that include such compute devices can then be interconnected with each other to form clusters. Such clusters usually incorporate communication interconnects such as Infiniband or Gbit Ethernet. The number of cores within a cluster today has exceeded 10 million in the case of

http://dx.doi.org/10.1016/j.simpat.2017.05.009

Sunway TaihuLight - Sunway MPP [3,4]. Additionally, volunteer computing has become a viable method for solving some large scale problems [5]. Furthermore, another level of parallelism involves coupling clusters or volunteer-based systems using wide area networks for grid type computing [6–8]. These architectures result in systems of an even larger scale.

For such systems, issues related to performance, reliability and power consumption arise. Specifically, speed-ups of a single application on large systems will be limited by dependencies within an algorithm, communication costs but also failures of computing devices. Additionally, power consumption is becoming one of the key concerns next to the computational power. Optimal utilization of such large-scale systems, especially in the face of rising heterogeneity of the hardware and applications, requires new versatile software frameworks which allow tuning, executing and monitoring of hybrid parallel programs. Our efforts in this area resulted in the KernelHive system [9] that allows to plug in scheduling and autotuning optimizers, focusing on particular goals, such as execution time and power consumption. Finding optimal values of application parameters and job assignments often requires low-cost estimation of the objective functions and thus, accurate models and simulations of real application executions.

In order to investigate these issues, we contribute by proposing a new environment called MERPSYS, available online [10] that allows modeling large scale cluster, volunteer and mixed systems, definition of parallel applications abstracting from well established APIs for parallel programming and simulation of execution of applications on such systems. The simulator returns execution time, on which we focus on in this paper, reliability of execution and power consumption, all using concrete parameters of compute devices and interconnects from a database. The environment allows us to conduct simulations of large scale systems that may be unavailable to the end-user at the given moment. Also, simulation of application execution is generally much faster than execution of a real-word application in the given environment. In this paper we focus on simulation of execution times of parallel applications within MERPSYS and its validation. It should be noted that the simulation of energy consumption using MERPSYS has been presented and validated in [11].

The outline of the paper is as follows. Section 2 discusses existing work related to systems and simulators for large scale computing systems and simulation of applications running on such systems. Section 3 includes description of the MERPSYS platform for simulation of parallel applications on large-scale clusters. Furthermore, Section 4 contains results of the experiments including description of parallel computing clusters as well as testbed applications such as computing similarity measures of points in a multidimensional space (master-slave paradigm), distribution of heat (single program multiple data/geometric parallelism), parallel sorting (divide and conquer). Within experiments we compared execution times of models of these applications in the MERPSYS environment against results obtained from real applications run in a real parallel large-scale environment. We validated correctness of the simulation environment. Section 5 contains discussion of the obtained results and based on it shapes future work.

2. Related work

This section contains descriptions of the most important simulation packages, toolkits and systems similar to the proposed MERPSYS. For the reader's convenience we present a synthetic summary in Table 1. It contains a comparison in terms of target simulated system types, means for system/resource modeling, APIs, assumptions for application modeling, simulation use cases and simulation goals. This summary, along with further discussion, highlights differences between the systems in terms of target simulation types and modeling in MERPSYS environment. Specifically, MERPSYS as the only platform supports all: performance, energy consumption and reliability metrics for simulated applications and environments. It also provides an API for automated runs of simulations with various configurations [12].

A discrete event Java-based simulation package SimJava, in which several layers are distinguished, is presented in [13]. Within the latter components are distinguished which can be used to create components in a higher layer. Several layers are described including Java, instrumentation, monitor, sampling, synchronization and scenario. The solution provides a flexible event-queuing system and allows to build other solutions based on it. In this respect, in contrast to SimJava, MERPSYS should be viewed as a simulator environment dedicated to cluster and volunteer environments with consideration of specific compute devices (CPUs, GPUs) and interconnects (network, PCI etc.) available in the provided database of components which allows to build and simulate nodes and clusters built out of those and obtain performance metrics.

OMNeT++, presented by Varga [14], is a C++ based discrete-event framework. It is mainly targeted at modeling networks including wireless and wired networks, protocols etc. A set of libraries includes many popular network protocols, but the programming model is suitable for other implementations (e.g. specific for HPC) as well. OMNeT++ is meant mainly for building network simulators, and as such in comparison to MERPSYS, does not allow for direct HPC simulation with consideration of various compute devices such as CPUs, GPUs, single and double precision performances, nor simulations considering power consumption of processors.

Paper [15] describes GridSim – a toolkit based on the SimJava library for modeling and simulation of various components and elements of grid systems including users, applications, resources, schedulers and brokers. GridSim was released under GPL license. In particular, in the system it is possible to analyze various types of resources with consideration of heterogeneity, static and dynamic scheduling, many jobs running in the environment, network speeds, advance reservation. Also, it can be used for optimization of the cost and time that can take into account possibility of failure. Compared to GridSim, MERPSYS considers also power consumption of particular compute devices and allows definition of models and simulation at a lower level of abstraction i.e. with consideration of compute devices within a node such as CPUs and GPUs, with single,

Table 1
Comparison of the simulators: summary.

	Target system	System/resource modeling	Application modeling, API	Simulations targeted at
SimJava	Computer architectures and parallel software systems, applicable to any network of communicating objects	Extending classes in Java,	Entity's behavior coded in Java	Execution time
OMNeT++	Computer networks and other distributed systems, mainly for building network simulators	Components (modules) exchanging messages, structure of the model expressed in the NED language	Simple modules developed in C++, simulation programs and kernel in C++	Network, performance
CloudSim	Cloud systems	Through components such as host (CPU, memory, storage parameters), VMs, allocation policy, cloud market, network topology in BRITE format	Tasks modeled through Cloudlets, dynamic workloads can be modeled through UtilizationModel	Resource provisioning, performance, cost, power consumption
GridSim	Cluster, grid, P2P	Modeling Processing Elements (with speeds) that form machines which form grid resources such as CPUs or clusters	Independent tasks modeled through Gridlet objects (computations, I/O, file sizes)	Performance, cost
MARS	HPC	Network (topology) model, pluggable models for network adapters and computing nodes	MPI applications – traces	Prediction of performance and application tuning, different network topologies, flexible routing schemes, arbitrary application task placement
GSSIM	grid, cloud	XML- based format descriptions: queues, resources, resource parameters	Standard Workload Format (SWF), Grid Workload Format (GWF)	Network modeling, execution time and power usage, modeling of network failures and security issues
SST/ macro	Large-scale parallel computer systems	Node, network etc. models specified in text files	Native C/C++/ Fortran and MPI	Performance oriented
SimGrid	Grid, cloud, HPC, P2P	XML	C, C++, Java, MPI	Performance, energy
MERPSYS	(Collection of) cluster and volunteer based systems	Graphical, built from components with WWW interface	Java extended with MPI-like API modeling communication	Performance, energy consumption, reliability

double precision performances (rather than MIPS ratings of machines in GridSim) connected with PCI Express, consideration of power consumption of a node based on the number of running threads [16] etc.

The MARS framework (MPI Application Replay network Simulator), described in [17], was created by IBM Research Laboratory. It allows to simulate high performance computing systems that include hundreds of thousands processors. The tool can be used for design of systems, prediction of performance and applications tuning. The environment, based on the OMNeT++ framework, offers various network topologies. It contains a model of a network and modules modeling particular components such as network and computing components. The simulator relies on replaying MPI [18] logs from real applications. MARS allows to model allocation of tasks to different nodes of a simulated system and collect statistics for visualization of simulation results. Simulation is based on replaying traces that contain MPI calls. Paper [17] claims to have simulated up to 65,536 nodes on an 32-way SMP cluster. Compared to this solution, MERPSYS uses Java extended with MPI like calls representing communication and, as such, allows to model an application at various levels of abstraction. Also, it allows to obtain simulated estimates of energy consumption [11] and probability of successful application execution in large-scale systems. To limit the size of the paper we skip detailed description of these functions of MERPSYS, focusing here on execution time simulations.

MERPSYS is focused on simulation of cluster-based systems and at this time it does not include features specific to cloud environments, that is provided by a toolkit such as CloudSim presented by Calheiros et al. [19]. The latter was designed to allow modeling and simulation of cloud systems and application provisioning environments. It provides modeling of virtual machines as well as consideration of provisioning methods. Furthermore, inter-connected clouds are supported for simulation. The environment allows space-shared and time-shared allocation of processing cores to virtualized services. Network is simulated using BRITE (Boston University Representative Internet Topology Generator) [20] that considers nodes that can correspond to data centers, brokers and hosts. It allows to simulate the network with latencies between nodes. The main feature of CloudSim is focus on virtualization technology in data centers. The simulator is able to consider many virtual machines, which may share same resources. It allows to link services with the cost per unit of memory, cost per unit of used bandwidth, cost per unit of storage, cost of ordered memory, cost of storage and transferred data. The simulator supports measurement of the cloud power consumption depending on system utilization.

Grid Scheduling simulator (GSSIM) [21] is an advanced tool that allows distributed computing simulations, and is primarily focused on scheduling. It was created in Poznań Supercomputing and Networking Center in Poland. GSSIM provides

means to simulate many components of a grid system including network and compute nodes. The authors claim, that GSSIM may be used for detecting bottlenecks in the network and perform energy aware simulations. It allows to assess power consumption depending on different workload types and scheduling policies. In GSSIM applications are described using Standard Workload Format (SWF) or Grid Workload Format (GWF) [22] files with extended metadata in the XML format. Applications can be described with consideration of execution time as well as time slots for execution important from the point of view of advanced reservations. Resource requirements may include the number of CPUs or memory that are necessary to complete the current task. The simulator provides a very extensive flow model of network with support for different network topologies. The network simulation utilizes a network path reservation and Dijkstra's algorithm to find the shortest route between the nodes and calculate the maximum flow between sites. GSSIM is available as a web application including an editor, experiment execution and experiment repository. It also provides a tool to analyze the results of a simulation. In contrast to GSSIM, MERPSYS is focused on cluster based systems and HPC applications modeled in the form of an abstracted parallel application in Java with MPI like extensions representing communication. MERPSYS also supports simulation of parallel application run reliability apart from performance and energy consumption simulation.

SST/macro, described by Adalsteinsson et al. [23], allows to model a parallel application execution using lightweight threads considering network interconnections. Specifically, threads create kernels that model computations or specific MPI communication routines. SST/macro was implemented in C++. Approaches of MERPSYS and SST to simulations are similar in the sense of focus on simulation of running a parallel application or several parallel applications on a system that is composed of certain computational (CPU, GPU, Xeon Phi) and network components. MERPSYS is more coarse-grained as it allows generalization of certain process or thread codes that in a real application can be executed by a large number of real processes or threads. Just a few process/thread codes are distinguished, each of which is marked with a label and a digit denoting the number of real processes/threads simulated. The MERPSYS approach is focused on a higher level of processing abstraction with emphasis on optimization of finding application configurations that give the best trade-off in terms of execution time, energy consumption and reliability when run on a large scale system. One of differences is that SST/macro uses an application model based on MPI, while MERPSYS employs a meta-language which is Java extended with methods representing computational blocks or communication (point-to-point or collective). MERPSYS allows to map many applications onto a complex system composed of multiple clusters with finding best sets of configurations for the applications. Simulations are intentionally coarse grained in order to be able to achieve results by running many simulations in reasonable time. The latter can be launched in parallel in a cluster.

SimGrid [24] is a framework that allows studying the operation of large-scale distributed computing systems. It supports grid, cloud, HPC or P2P systems. The software is free (GPL license) and can be downloaded from [25]. It is available for Linux, Windows, Mac OS X. SimGrid uses analytical models of a network with fast simulations and a reasonable accuracy level. SimGrid offers great performance and scalability. Authors claim that they were able to simulate with up to 2 million processors on a machine with 16 GB RAM. Times of simulation of the master-slave application for 1000 and 100,000 slave tasks are 0.7 s and 96 s respectively. The simulation was executed on a 2 GHz laptop with only 1GB of memory. Compared to SimGrid, in MERPSYS we also focus on simulations on cluster and volunteer computing paradigm such as presented in [26], also in terms of execution time, energy consumption – analyzed in [11]. MERPSYS allows to model systems at various levels i.e. connected systems, systems can be modeled as collections of machines connected with a given type of network and furthermore machine inside architectures can be modeled down to components such as CPUs and GPUs, PCI bus etc. In SimGrid, device parameters such as performance and power consumption are included in XML descriptions of hosts. In MERPSYS, we provide an extendable database of components such as concrete models of CPUs, GPUs, network interconnects that can be easily plugged into the model of the system and easily changed.

To support studies of scheduling strategies in Grid environments, a GangSim simulator has been developed as described by Dumitrescu and Foster [27]. The simulator allows to explore what site usage policies are appropriate in a grid environment, and how these policies impact achieved site and Virtual Organization performance.

The CloudReports, as described by Sa et al. [28], provides an open source simulation tool for energy-aware cloud computing environments. The detailed review of eleven simulation tools for cloud computing was presented by Pranggono et al. [29]. MERPSYS also offer analysis of energy consumption but it is dedicated to cluster, volunteer and mixed environments. A critical evaluation of the state-of-the-art modeling and simulation frameworks applicable to cloud computing systems has been presented by Bashar [30]. Another review focused on evaluation of performance and security issues was prepared by Malhotra and Jain [31]. Commonly used simulation tools for cloud computing security research were described in [32]. A review of major cloud simulators that analyze issues related to load balancing, power constraints, program offloading, cost modeling and security issues was presented by Ahmed and Sabyasachi [33].

3. Proposed solution – MERPSYS platform

In view of the aforementioned existing solutions and desired use cases presented in Section 1, the contribution of this paper is a new MERPSYS environment that allows fast simulation of parallel applications runs, in particular master-slave, geometric parallel and divide-and-conquer paradigms, on large-scale systems having thousands of processes or threads and assessment of execution time, reliability and power consumption of the particular run. Such a simulation is performed taking into account specific parameters of hardware computational components such as CPUs, GPUs (single, double precision floating point performance, power consumption, reliability) as well as network interconnections (startup time, bandwidth).

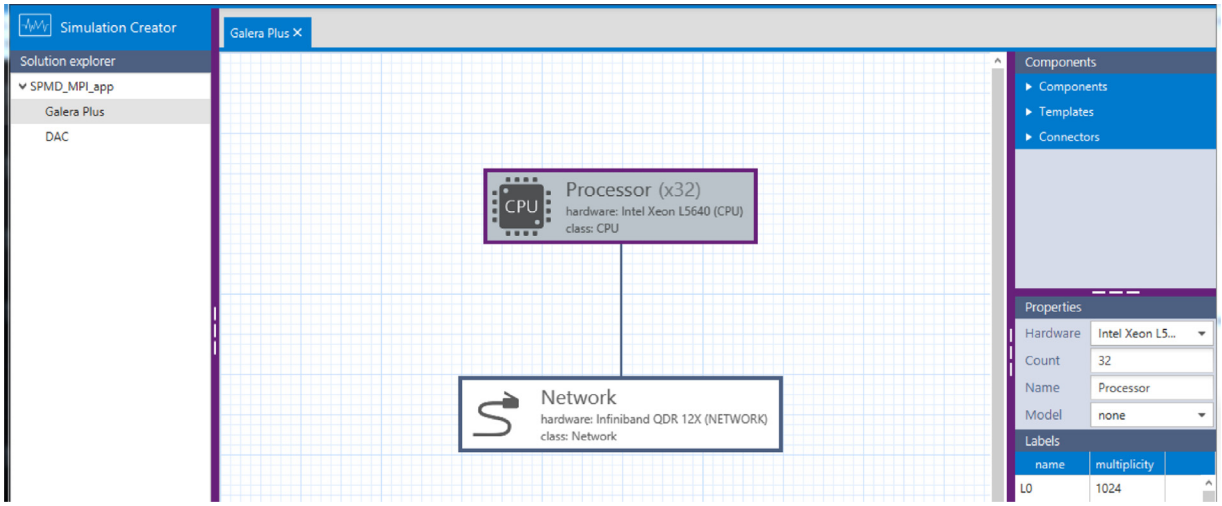

Fig. 1. MERPSYS system model.

MERPSYS includes an extendable database of real hardware components with these metrics. Components in the system model can be easily exchanged for others in order to obtain instant results for different hardware configurations. What is important the system allows us to identify dependencies of the execution time on the number of processes as well as various application parameters, which are demonstrated for real-life applications.

Some aspects of the MERPSYS environment have been already studied in their respective contexts. Specifically:

- In paper [34] we described validation of the presented environment simulating an application for parallel computation of similarities among large vectors on a collection of workstations connected with Gbit Ethernet. In this paper we use an improved version of this application run on a modern cluster for various numbers of slave processes.
- In paper [16] we presented models for particular computational and communication blocks benchmarked on selected clusters, including execution times, power consumption and functions describing reliability these were further introduced into the MERPSYS database.
- In work [11] we presented details of modeling energy consumption of SPMD and divide-and-conquer applications in MERPSYS. In contrast, in this paper we focus more on experiments with execution times.
- In paper [35] we introduced modeling of volunteer computing that further allows us to incorporate a component representing a group of volunteers (with a function describing its execution time based on a probabilistic model) into MERPSYS.

In view of these papers that described particular elements of the whole approach, this work contributes by presenting MERPSYS as a whole simulation platform and contains the following:

1. The architecture of the whole platform including components and their interactions.
2. Methodology including screenshots of the modeling tool.
3. Modeling, benchmarking, simulation in MERPSYS and demonstration of matching benchmarking and simulation results with low error for applications falling into three various parallel programming paradigms i.e. master-slave, Single Program Multiple Data and divide-and-conquer.
4. Validation of simulated results (execution times) against real results for up to over 1000 processes.
5. Comparison of the MERPSYS solution to solutions such as SimJava, OMNeT++, CloudSim, GridSim, MARS, GSSIM, SST/macro and SimGrid.

From a user's point of view the MERPSYS environment provides panels for definition of the following:

1. Hardware components – such as CPUs, GPUs, accelerators and network interconnects along with their parameters. It can be defined via a Web page on the MERPSYS server [10].
2. Computational model – formulas that define execution times considering hardware parameters, the number of instructions of a given type, the number of threads for computational components such as CPUs or GPUs as well as communication times for point-to-point and collective operations. Such models may be different for various clusters using various interconnects. A computational model can be defined via a Web page on the MERPSYS server. We defined specific formulas for three clusters located at Gdansk University of Technology and described these in paper [16].
3. System model – the structure of a system is defined by dragging computational components and connecting them using communication components on a canvas as shown in Fig. 1. The system model can be hierarchical and include multiple levels: WAN, LAN, cluster and computer (CPUs, GPUs, accelerators, PCI etc.).

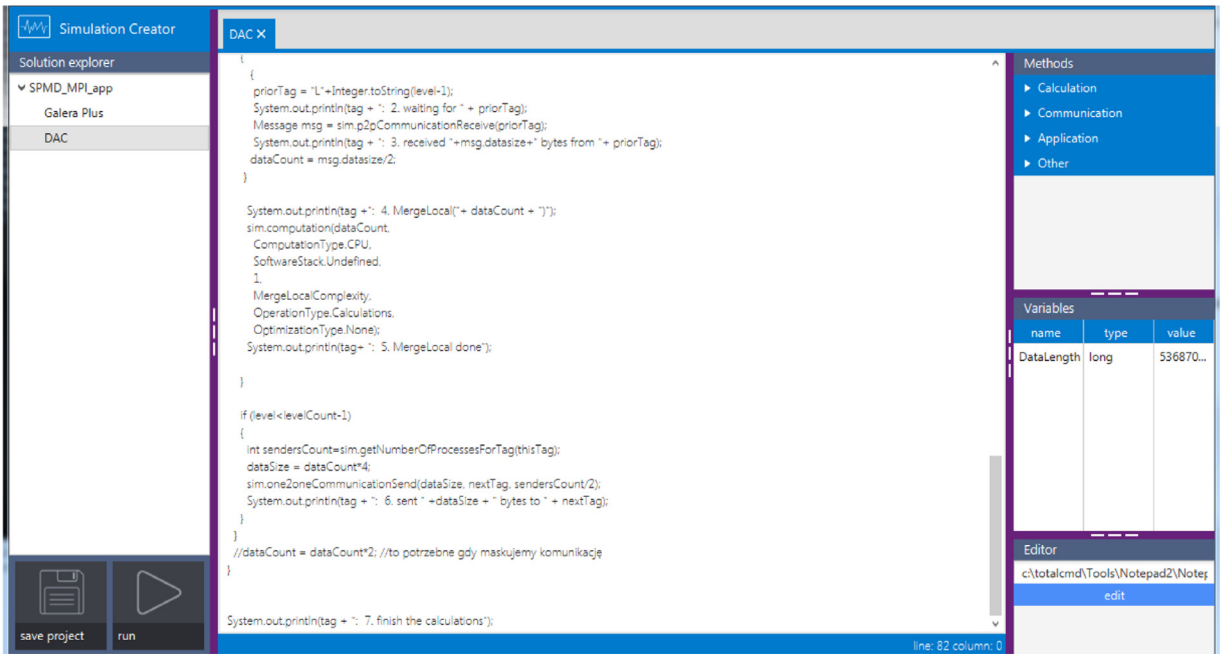

Fig. 2. MERPSYS application editor.

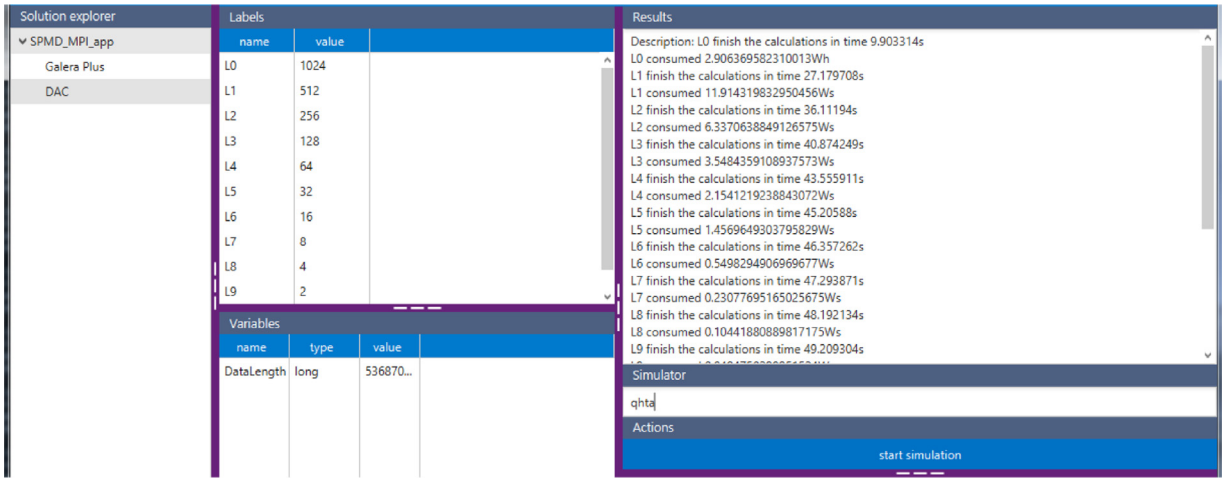

Fig. 3. MERPSYS simulation panel.

4. Application model – presents code of an application which could be launched in a real-life environment. As shown in Fig. 2, the application is coded using a meta language which is Java extension with a library of functions representing computational blocks and a variety of communication functions: point-to-point and collective ones. The application consists of codes for one or more processes or threads each of which is marked with a distinct label.
5. Experiments – a panel allows to define an experiment using the previously created models. Experiments use a specific configuration including the number of processes or threads with particular labels as well as input parameters. At launching an simulation it is necessary to specify how many processes or threads of each label are to be run. This starts a multithreaded simulator that launches one thread for each label with proper scaling of data sizes, computational and communication times. The MERPSYS simulation panel is shown in Fig. 3.

The system model, application model and experiments are defined and managed within a client-side Java application that connects to a MERPSYS server.

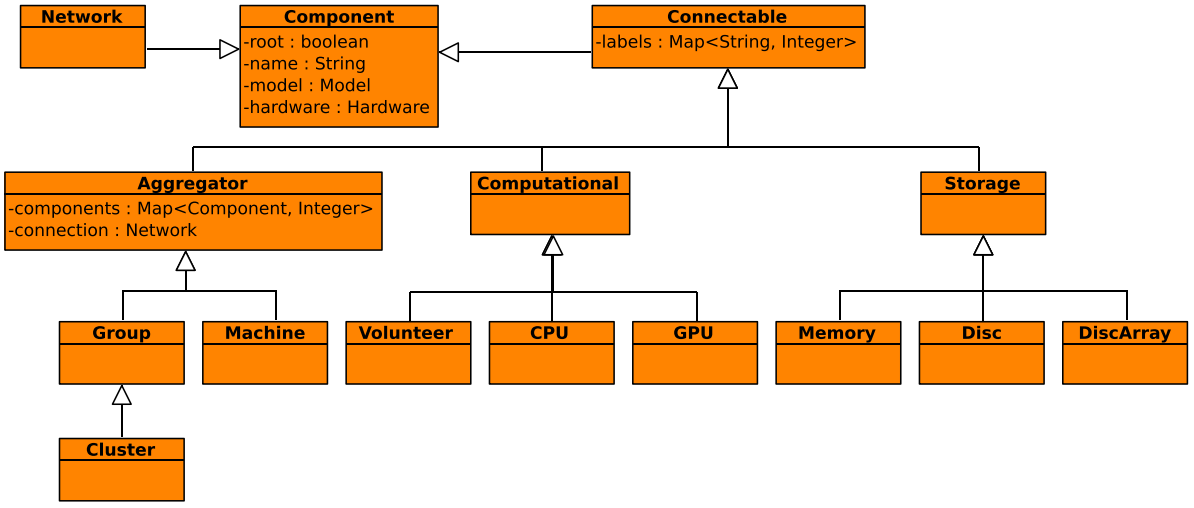

Fig. 4. System model entities.

3.1. Use cases for the MERPSYS platform

Consequently, the system allows a variety of practical use cases such as:

1. **Simulation of running a given application on a system with various numbers of compute devices for assessment of the best configuration taking into account execution time (including probability of failure) and power consumption e.g. performance/power consumption**. This may help to assess best hardware configurations that need to be purchased.
2. **Simulation of speed-ups for various input data sizes.**
3. **Identification of bottlenecks in code** through testing of existing (parts of) parallel codes on large scale systems. The process of developing a MERPSYS application model can help to analyze and understand the application better. For example, while identifying and tuning the communication and computation blocks for the model in [36], we noticed a bottleneck due to imbalanced data. Then, simulations in MERPSYS allowed us to assess the application execution time after elimination of this bottleneck. This case shows the simulator is useful for real-life problems and helps to decide if it is worth to spend time on certain code improvement.
4. **Testing various hardware components in a system model**: testing performance and power consumption on a system model with a certain structure but various hardware components. The system allows easy substitution of e.g. computational components such as CPUs for assessment of performance and power consumption.
5. **Assessment of reliability of computations and execution times** taking into account failures in a large scale system. This allows assessment of the shortest execution times considering failures and need for rerunning the application in case of failure.
6. **Scheduling of several applications on a large-scale system** that is composed of cluster and volunteer components for global optimization of performance, reliability and power consumption. Specifically, in MERPSYS several independent parallel applications can be modeled, where processes of each application never communicate with processes of other application. Processing resources are shared at the simulator level taking into account multiple processes assigned to a compute device along with its capabilities including the number of cores. Network resource sharing can be included in the model by the user, as shown for the master-slave application in Section 4.2.1. Consequently, we can model such a set of applications as a single parallel application in MERPSYS.
7. **Teaching parallel programming and performance evaluation**.

What is important is that the simulation runs in a considerably shorter execution time than a real code. Obviously, this depends on how detailed the model of an application is. If large computational parts of the code can be expressed by computational blocks then execution times can be very short. Furthermore, the system allows to make simulations on various hardware configurations that may be unavailable to the user in reality at the given moment but e.g. may be considered for purchase.

3.2. System model

Fig. 4 presents the system model class inheritance diagram. The *Component* class is a base for all the entities and the *Connectable* class is a base for all the entities which can be connected to some type of a connection medium (e.g. network,

Key	Value	Delete
nCores	8.0	
performanceS	287.55	
performanceD	250.0	
performanceSource	3.0	
powerConsumptionLoad	158.67	

Fig. 5. Selected CPU parameters.

system bus, etc.). It provides information concerning name, used computational model and specified hardware. A particular system can be defined with the following entities:

- *Network* — represents connections between components within one aggregator, it can be WAN/LAN network connection between machines or bus connection between computational components;
- *Aggregator* — a logical group of a selected components connected using the same connection type:
  - *Group* — machines inside one local network, e.g. a computational cluster,
  - *Machine* — a workstation or a server;
- *Computational* — components capable of performing computations:
  - *CPU, GPU* — different processing units,
  - *Volunteer* — a volunteer system (set of volunteers);
- *Storage* — various type of storage like RAM memory, HDD/SSD drives or a cluster disc array.

This hierarchical structure allows us to define various systems. Those can be very top generalized systems made out of a number of clusters and volunteer systems connected by WAN as well as a very detailed lower level system with specified RAM, CPU/GPU and HDD/SSDs. This structure does not give any details about the hardware, only the basic types. Detailed information is stored within *Hardware*s which can be defined independently of the system model. An example with selected CPU parameters is shown in Fig. 5, along with the number of cores, single, double precision floating point performance, power consumption under stress, another being power consumption when idle.

As it can be seen, all the components can be assigned to the *Model* instance independently. Usually there is only a need for assignment of a model to the root component and the rest of child components will use the same one. In more complex systems which are built with different clusters and volunteer systems at the same time, usage of different computational models is essential.

The simulation process requires a system, which minimally consists of one aggregator with at least one component, a network and a simulated application code. As a result of the simulation process a tuple containing execution time, consumed energy and fault probability is returned. In order to perform a simulation, a specific computational model is required (*Model* entity). Those can be defined globally in a form of functions and assigned to the system root component. Functions in a computational model can make use of hardware dependent attributes stored in a *Hardware* entity. Moreover, some models can be tuned for particular hardwares (*HardwareModel* entity). Tuning allows changing model functions to be more accurate for a particular hardware instance.

In the computational model, functions that define execution time of a block use the number of threads, number of operations and hardware parameters of a compute device including performance. The aforementioned function can account for slowing down through cache use by multiple threads or e.g. incorporation of a stochastic model can be implemented using probability values modeling e.g. duration of a computational block, if needed. This is possible through incorporation of certain coefficients measured experimentally by the user for the given code. A similar approach can be used for modeling contention employing proper coefficients and functions in the computational model reflecting increased average communication times due to contention for various operations such as point-to-point, scatter and gather [16]. While this requires user involvement in additional measurements, it provides a flexible and accurate method to reflect the given case using MERPSYS. Actually, in the tests in this paper we adopted this very approach when using a different function for modeling communication times with contention through decreased bandwidth per communication for the master-slave application presented in Section 4.2.1. Additionally, computing power consumption within a node based on the number of active threads is done this way, using the function presented in [16].

3.3. Parallel paradigms available in the platform

The MERPSYS platform supports programming any type of a parallel message passing application. The application model uses a meta language that is an extension of Java with several additional functions modeling computations and communication through:

computational blocks: *void computation(double dataSize, ComputationType type, SoftwareStack stack, int numberOfThreads/Processes, String complexityFunction, OperationType operationType, OptimizationType optimizationType)* –

models a computational block that performs the number of operations returned by *complexityFunction* using data size parameter with optional specification of target computational devices, software stack, operation types and optimizations. Computations are to be performed on a given number of processes or threads. For instance, when a sorting computational block is used within a more complex algorithm, the complexity of the former would be e.g. $O(d \cdot \log(d))$ where d is the number of elements to be sorted. Function *computation(...)* allows to incorporate such complexity but also additional coefficients to represent the actual number of operations.

communication functions – most commonly used functions include:

- *void p2pCommunicationSend(double dataSize, String receiverTag)* – used for modeling a peer to peer communication send, pairwise communication between all pairs of processes with a given tag of the sender and the receiver tag is considered,
- *Message p2pCommunicationReceive (String senderTag)* – a matching receive to *void p2pCommunicationSend(double dataSize, String receiverTag)*,
- *void one2oneCommunicationSend(double dataSize, String receiverTag, int nCommunications)* – used for modeling *nCommunications* number of one to one communications,
- *Message one2oneCommunicationReceive (String senderTag)* – a matching receive to *void one2oneCommunicationSend(double dataSize, String receiverTag, int nCommunications)*,
- *void scatterSend(double dataSize, List<String> receiverTags)* – used for modeling a scatter operation to processes with the specified tags,
- *Message scatterReceive(String senderTag)* – a matching receive to *void scatterSend(double dataSize, List<String> receiverTags)*.

Any legal Java constructs are allowed in MERPSYS including data types, sequences, loops or conditional instructions. It should be noted that the aforementioned functions allow specification of additional labels that specify languages being modeled (e.g. C, C++, etc.), communication APIs (e.g. MPI, Hadoop etc.). This allows consideration of coefficients describing overheads of particular software stacks and is also related to calibration described next. Consequently, this solution is as general as the modern Java language and allows consideration of specifics of particular APIs if provided.

In practice, developing the process implementations requires knowledge about the computation and communication parts which the application consists of, as well as dependencies between them. Such knowledge can be developed from scratch, derived from the code of an existing application or logs from previous executions. Granularity of the implementation depends on the decision of the modeling researcher, facing a trade-off between accuracy and execution time of the model. For example, in [36] we modeled a parallel deep neural network training application, consisting of two processes: *master* and *slave*. The chosen granularity level assumed two communication blocks for each process and one computation block for the *slaves*. The meta-language code accounted only for the high-level parallelization aspects and did not concern the computation internals. This allowed fast evaluation of the execution time and power consumption of the application. Namely, simulating a set of computations with various parameter configurations, which in a real system would last over 335 h, while using MERPSYS took over 2 h on an Intel Core i7-4712HQ CPU. These encompassed 16 simulations, with a total of 50,928 calls to either computational or communication blocks. The accuracy of such a coarse-grained model was high enough to provide valuable insights and help optimize the application.

In particular, the following typical parallel programming paradigms are offered within the platform through ready-to-use templates that can be extended:

1. master-slave in which a master distributes data chunks among available slaves (can be combined into a hierarchical pattern),
2. geometric parallelism (SPMD) suitable for simulations of a variety of phenomena such as in medicine – [37], electromagnetics – [38], weather prediction – [39] etc.,
3. divide-and-conquer in which a problem is recursively divided into subproblems until a certain size is reached for which a direct solution is applied [40].

These patterns are representative of a variety of practical applications. Examples of these are demonstrated in practical scenarios as shown in Section 4.

3.4. Usage methodology for the MERPSYS system

In MERPSYS the methodology shown in Fig. 6 should be applied step by step:

1. System modeling.
2. Application modeling.
3. Model calibration against selected known execution times in a (small scale) system.
4. Performing simulations for possibly large sizes of input data, number of processes or threads and the number of nodes/cores including previously untested configurations.

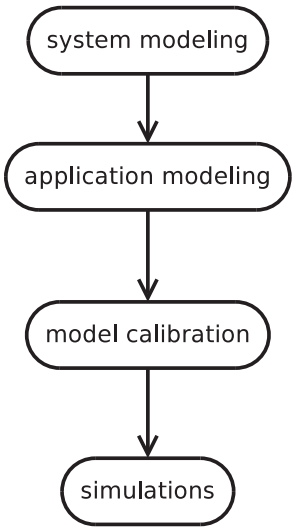

Fig. 6. Methodology for the MERPSYS platform.

It should be noted that the system model uses various coefficients related to concrete hardware devices i.e. performance in GFlop/s, power consumption etc. The system model contains functions for modeling execution time of computational blocks defined in the application model. While preparing these functions, one can take into account various parameters connected with the computing device characteristics and computational block parameters. Possible computing device characteristics depend on the hardware model defined in the MERPSYS database. The computational block parameters are defined in the application model and passed through the computational block API.

Apart from these parameters, these functions can use certain coefficients that should be calibrated based on knowledge about former application executions. Calibration of the model means setting such coefficients so that simulated execution times do correspond to real execution times measured in a possibly small scale system. Once such coefficients have been found, the user can launch a variety of simulations even for system sizes not available at the moment.

For example, for a neural network training application in [36], the used computing device characteristic was the number of floating point operations per second and the used computational block parameter was data size, i.e. size in bytes of an archive with training examples. The function for execution time was a linear function of data size divided by the device performance. Two coefficients of this linear function were calibrated based on real results from training execution on one GPU, using the ordinary least squares method. The model calibrated in such way appeared to achieve mean percentage error between 1.5% and 2.7% for different configurations utilizing up to 8 GPUs.

3.5. Architecture, design and implementation issues of the proposed platform

Fig. 7 presents all components of the MERPSYS platform, their placement and connections between them. The platform was built with following modules:

- Application — an Enterprise application,
- Web — a web module,
- Engine — an EJB business module,
- Editor — a desktop application allowing for application modeling,
- Simulator — a multi-threaded application performing simulations of parallel applications,
- HTML Pages — dynamic HTML pages served to a web client,
- Database — a platform database.

The Application module is a central element of the whole platform. It is an Enterprise type application developed using the Java EE 1.7 technology [41]. This module requires an execution environment, which can be any Java EE Full Profile [42] certified application server. Currently GlassFish Open Source Edition 4.0 [43] is used.

A persistent container utilizes the PostgreSQL Server [44]. There we store information about platform users, hardware details, distributed system models and simulations with their results.

Machines on which the application and the database servers run are connected to the same local Ethernet network and follow database connectivity patterns outlined in [45].

The Web module, similarly to the EJB module is a part of the Application module. The Web module does not implement any business methods but provides only user interface and acts as a proxy to the EJB module.

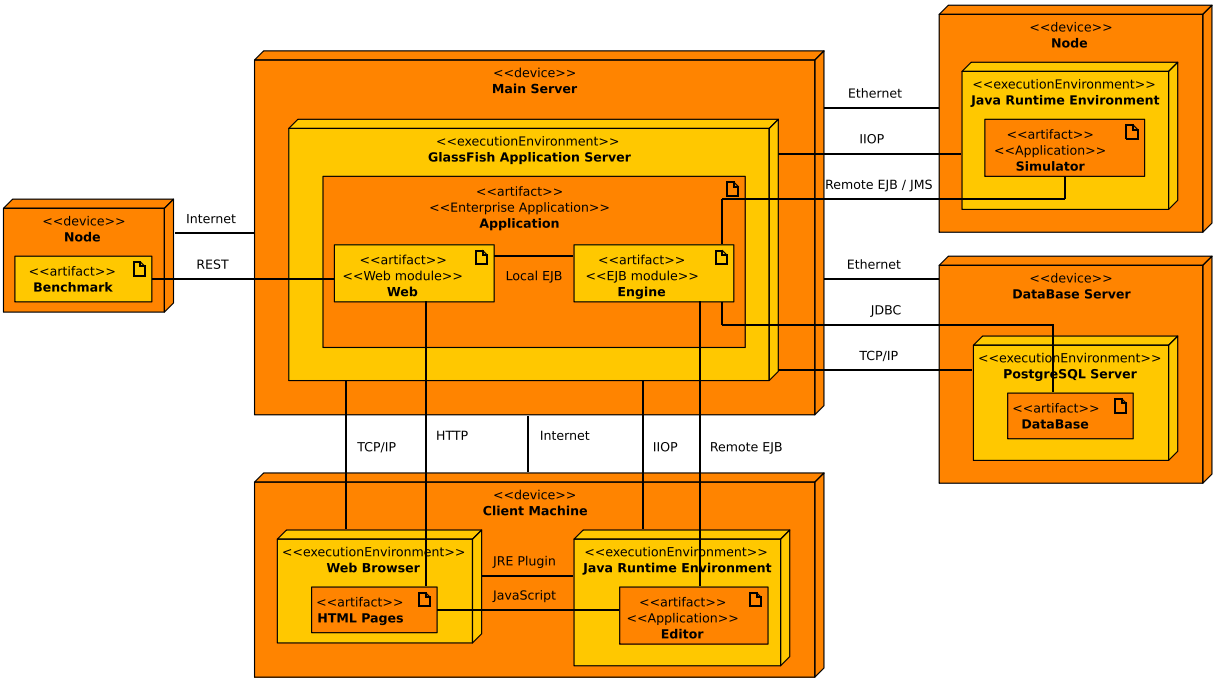

Fig. 7. Infrastructure of the MERPSYS platform.

Dynamic HTML pages run inside a user's web browser launched on the user's machine. Because communication between the user's machine and the platform server is done through the Internet, both machines do not need to be connected to the same local network. The whole interaction between the user and the Web module is done using the HTTPS protocol. The user can view simulation results and create/edit/delete mathematical models and hardwares. Moreover, a system administrator can manage platform users. The system and application editor was prepared in a form of the Java SE desktop application. In order to enable starting the Editor shown in Figs. 1–3 from a WWW page, the Java Web Start technology [46] was used.

A coarse-grained simulation process is performed by a Simulator application prepared using the Java SE technology. The simulator itself is a parallel, multithreaded Java application. For each distinct label in an application model, a separate thread is launched – irrespective of the number of processes with a given label in an application model. The simulator scales computations properly, taking into account label multiplicities. It uses a discrete event-based model for simulation progression. For communication components, a queue is used that allows to insert sent messages and fetch received messages. Contention can be dealt with using proper communication cost functions where necessary.

For the sake of parallelization of simulations (of the same or different distributed systems), many instances of the Simulator can be started. The simulation process requires database access so as in the Editor application the EJB remote interfaces are used. A process of commissioning simulations is implemented as a queuing system. Simulation tasks are sent from the EJB module to one central queue realized in the JMS (Java Message Service) technology [47]. All Simulator instances are connected to this queue in order to receive simulation tasks. The JMS server provides that a particular message will be consumed by only one Simulator instance. This allows introducing load balancing between the Simulator instances based on the fact that a particular Simulator will check for the simulation tasks only if it is not overloaded. As a JMS server, the GlassFish application server was used with its internal Open MQ [48] JMS server instance.

4. Experiments and results

For the three applications chosen as representative in terms of parallel processing paradigms – master-slave, geometric parallelism and divide-and-conquer, we performed the following steps within experiments:

1. Implementation of a real parallel application for each of the chosen paradigms – details are provided in Section 4.2.
2. Running application performance tests on a real cluster – details of testbed systems are provided in Section 4.1.
3. Coarse-grained modeling of the application and system in MERPSYS.
4. Simulations and calibration of cost functions based on selected results from real runs.
5. Simulation for other configurations (input data size, the number of processes etc.)
6. Comparison of results.

In order to validate quality of simulations in MERPSYS, for the following tests we calculated a maximum (dmax) and average (davg) errors between simulated (s) and real (r) measurement values, defined as:

$$d_i = \frac{|r_i - s_i|}{r_i} \cdot 100\%$$
$$dmax = max_i\{d_i\}$$
$$davg = \frac{\sum_{i=1}^{n} d_i}{n}$$

4.1. Testbed environment(s)

Testbed 1 (for the master-slave application) The physical environment for testing the master-slave application consisted of a cluster (located at Gdansk University of Technology, Poland) with 104 identical nodes, each with two Intel(R) Xeon(R) CPU E5345 2.33 GHz processors with 4 physical processing cores, 8 MB cache running Linux kernel version 2.6.32. Each node had 16GB of RAM. All nodes were interconnected with a Gbit Ethernet, as well as an Infiniband switches.

Testbed 2 (for geometric SPMD and divide-and-conquer applications) The real environment where experiments with a geometric and divide-and-conquer applications were performed was cluster Galera+ (located at Academic Computer Centre, Gdansk, Poland) [49] consisting of 192 nodes, each controlled by an autonomous operating system (Science Linux) and interconnected by fast Infiniband network (QDR: 40 Gbps) used for processing synchronization and data delivery as well as by Gbit Ethernet used for management purposes.

Each cluster node contains two Intel Xeon 2.27 GHz multicore processor units, with 6 physical and 12 logical (HyperThreading) computation cores, 16GB RAM memory and proper network interface cards. Finally the whole cluster features a 500TB disk array exposed to the nodes using Glustre remote file system, which can be used for data storage and exchange.

For data population, exchange and synchronization of the applications deployed on the cluster can use a number of MPI implementations including MVAPICH and OpenMPI. The experiments were performed using the former implementation and for their execution a set of nodes (up to 32) was exclusively used i.e. up to 384 physical cores and up to 768 including HT. A model of this testbed in the MERPSYS environment is shown in Fig. 1.

4.2. Testbed applications and results

In order to verify the concept for real applications, we have developed three applications representative in terms of parallel programming paradigms, ran the applications on large modern HPC platforms and verified simulated execution times for various sets of parameters obtained in MERPSYS against real values. In terms of computations to communication ratios, exemplary values include 1.74 for 8 processes for the master-slave application for testbed environment 1 and 4.22 for 32 processes for the divide-and-conquer application for testbed environment 2.

4.2.1. Master-slave

We developed a parallel master-slave application (240 lines of code) that computes similarity of points in a multidimensional space in parallel. An input to the application is a set of N-dimensional points $\vec{p_i}$. The parallel application, developed with C and MPI, partitions input data into batches which are distributed by a master process among a group of slave processes. Upon sending back results a slave is sent another batch for processing. Assuming that the number of batches is considerably larger than the number of nodes in a cluster, the scheme can balance load even if processors have various speeds. In this particular case, a batch consists of two groups of points such that each point from the first group is compared to each point from the second group. Parameters of the application input include: PC – the number of points, DS – the number of dimensions. An additional parameter K defines the size of the batch i.e. the number of points in a data packet. This code is also representative in terms of potential applications. Computing similarity is a basic operation in a wide range of data mining tasks as described by Witten and Frank [50]. Thus we find it a very representative task and a good example to evaluate our system for. Data mining methods based on distances usually require to compute a proximity matrix that for large datasets may be computationally expensive, typically it is $O(PC^2)$. A good example for practical usage of similarities computations for machine learning is a K-NN classifier as described by Cover and Hart [51] or Du and Chen [52] for text categorization, or unsupervised learning approach based on K-means as discussed in [53].

We have performed several tests that extend previous results. In paper [34] we performed tests on a collection of workstations in a LAN, for various parameters such as the number of points, the dimension size and the number of points in a data packet. In this paper we present results for an improved version of code compared to [34] which sends two groups of points to a slave such that each point from the first group is compared to each point from the second group. This code was run on a cluster described in Section 4.1 – testbed 1. Furthermore, in this paper we present execution times (Fig. 8) and speed-up (Fig. 9) for a selected case of 5000 points, 1000 dimension size and 100 points in a point group.

We have performed very detailed profiling of selected real executions that revealed both processing and communication times as well as network contention. We discovered that while using Ethernet, contention started to become visible for 8+

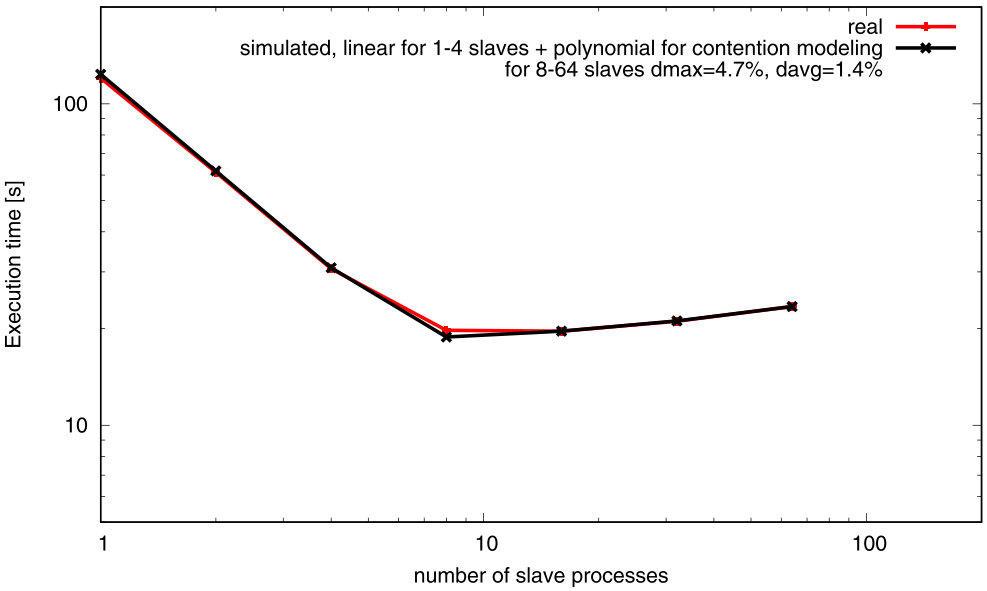

Fig. 8. Master-slave application – Execution time [s] vs. number of processes.

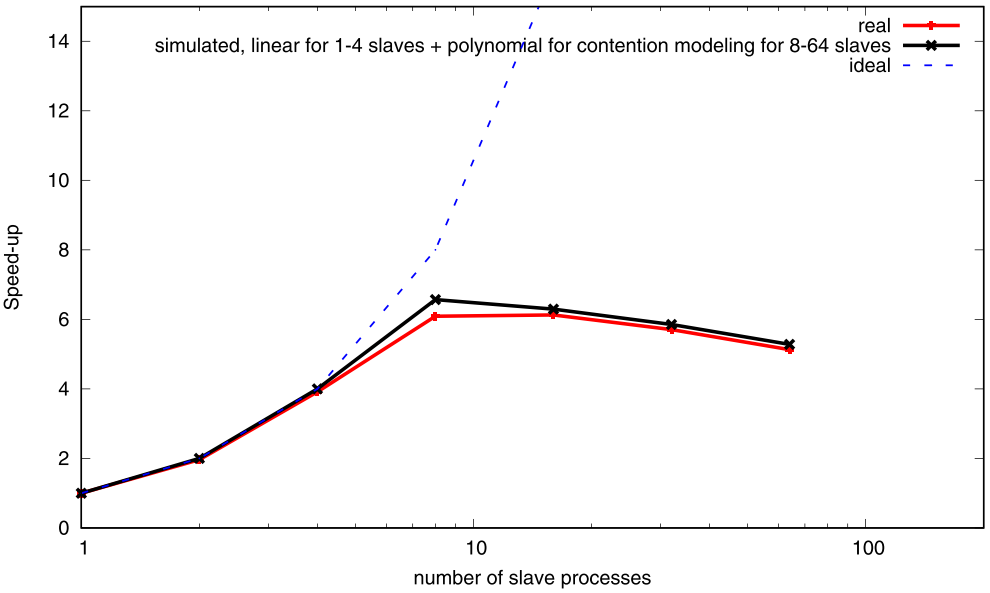

Fig. 9. Master-slave application – Speed-up vs. number of processes.

slaves when run on distinct nodes. Consequently, in MERPSYS we exploited its feature to model point-to-point communication as functions. Specifically, for up to 4 processes we used a linear function that considers startup time, message data size and bandwidth, independently from other communications. Starting with 8 processes though, we used a polynomial function of the second degree with proper coefficients that matched real runs for 8, 16, 32 and 64 processes with a low degree of error, as can be seen from the charts. The 8 process scenario is a borderline case in which neither linear nor polynomial estimate is optimal which results in the largest error for this number of slaves. Simulation of a run with 64 slaves modeled as codes with individual labels took 8 s on an Intel i5-7200U CPU 2.50 GHz. Within the simulation there were either 40 or 42 communication calls and either 20 or 21 computation calls per slave.

4.2.2. Geometric SPMD

We implemented a parallel geometric Single Program Multiple Data application (260 lines of code). From the point of view of application structure, dependencies and communication, this code represents solving a variety of physical phenomena in applications e.g.: weather prediction – [39], heat distribution, simulations in medicine such as phenomena in

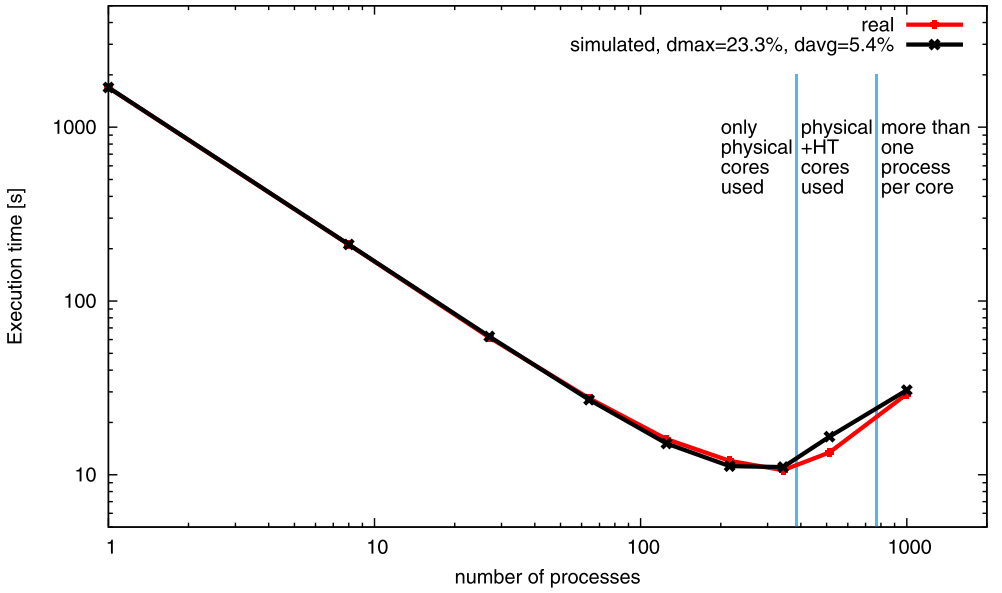

Fig. 10. Geometric SPMD application – Execution time [s] vs. number of processes.

myocardium – [37], electromagnetics – [38] etc. Problems in these domains are often modeled as (sets of) differential equations which are then transformed and solved in successive, discrete time steps. In every time step, a domain represented by a grid of cells needs to be updated. Specifically, values associated with each cell need to be updated typically using values from the previous time step associated with this very cell and its neighbors in the grid. In a parallel implementation, this requires partitioning of the domain into disjoint subdomains each of which is processed by a distinct process or thread. The aforementioned dependencies in update equations require communication between domains before computations proceed to the next time step.

For the tests we have implemented a parallel application, programmed in C with MPI, that simulates heat distribution across space. The application can work in 2D or 3D domains. The domain can be partitioned into rectangles or cuboids respectively. In the code communication is performed using MPI_Send/ MPI_Recv in right/left directions in all dimensions. In terms of processing and computation/communication ratio this application can be thought as a template that is applicable to any of the aforementioned examples of SPMD applications. What they differ in is the data associated with cells and update equations. Results for the testbed application for a 3D domain of size $2000 \times 2000 \times 2000$ cells were obtained in a cluster described in Section 4.1 – testbed 2. Fig. 10 presents execution times of the real runs and execution times obtained in the MERPSYS simulation environment. Fig. 11 presents speed-ups resulting from the real runs and those stemming from simulation results. Speed-up values were directly derived from execution times presented in Fig. 10, therefore no additional error values were calculated for this metric. Simulations in the MERPSYS environment on a mobile quad core i7 CPU took less than 0.35 s each. For these particular simulations it was enough to simulate 1 iteration with communication calls in 3 dimensions (12 calls) and 1 call to a computational function. It can be seen that execution times and speed-ups measured from the simulator are very close to those measured in the real system. What is more, it is true for 1–343 processes each running on a separate physical core, 512 and 729 processes running on physical and HyperThreading cores as well as on 1000 processes that needed to time share available cores.

4.2.3. Divide-and-conquer

We also implemented a parallel divide-and-conquer application, programmed with C and MPI (200 lines of code). A part of the code for the application in the MERPSYS environment is shown in Fig. 2. The input data is partitioned into at least two data packets. Then, the same procedure is applied on these data packets in parallel. Partitioning is repeated until a minimum data size is reached. Then a data packet is processed using a specified algorithm. This scheme results in a divide-and-conquer tree which can have significant depths, potentially various node degrees and computation/communication ratio. Branches of the tree are assigned to various MPI processes as long as these are available since we used the number of processes equal to the number of processing cores in a cluster. We used the merge sort algorithm but the code can be used as a template applicable to any other divide-and-conquer algorithm by exchanging partitioning/processing/merging parts of code e.g.: discrete Fourier transform, multiplication of complex numbers etc. Results for the testbed application for a vector size of 536870912 elements were obtained in a cluster described in Section 4.1 – testbed 2. Fig. 12 presents execution times of real runs vs. execution times obtained in the MERPSYS simulation environment. Fig. 13 presents speed-ups resulting from the real runs and those stemming from simulation results. Simulations in the MERPSYS environment on a mobile quad core

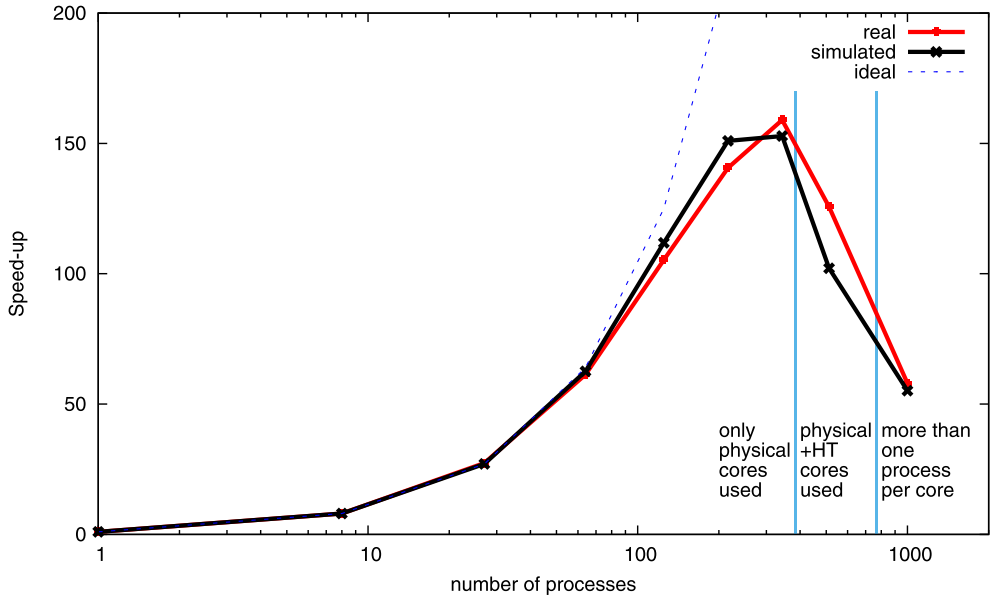

Fig. 11. Geometric SPMD application – Speed-up vs. number of processes.

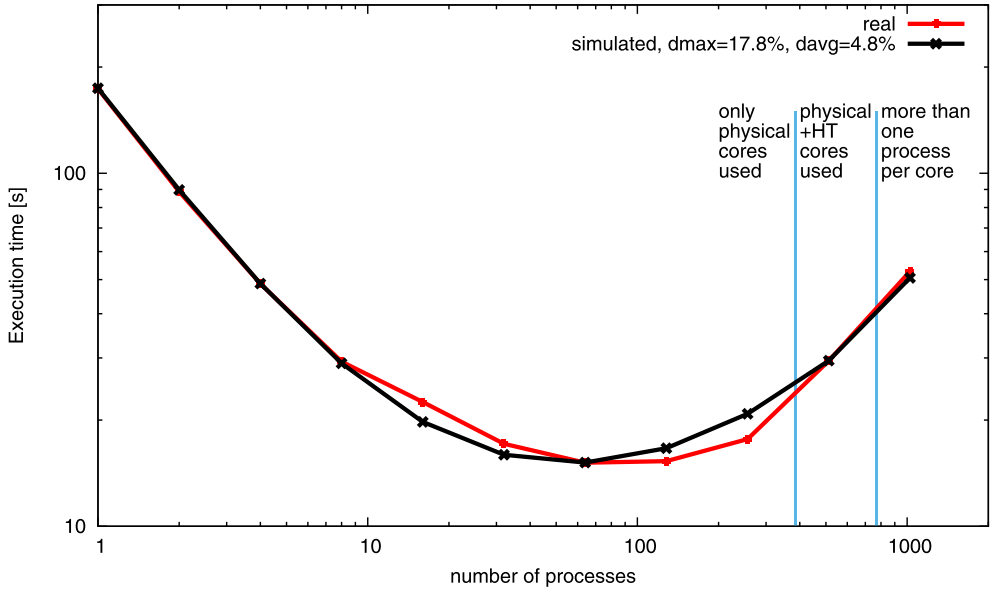

Fig. 12. Divide-and-conquer application – Execution time [s] vs. number of processes.

i7 CPU lasted below 1.2 s each. For these particular simulations, for 1024 processes, there were 10 communication calls and 11 calls to computational functions. It can be seen that execution times and speed-ups measured from the simulator are very close to those measured in the real system. What is more, it is true for 1–256 processes each running on a separate physical core, 512 processes running on physical and HyperThreading cores as well as on 1024 processes that needed to time share available cores. In Figs. 11 and 13 a transition from overpessimistic to overoptimistic estimates can be observed. Our working hypothesis assumes that specific bindings of processes to cores might play a role and influence real execution time, especially taking communication between processes into account. Even though the differences are small, we are planning to explore this in our future research.

5. Conclusions and future work

In this paper we proposed a new MERPSYS environment that supports modeling and fast application execution simulation for various parallel programming paradigms (including master-slave, geometric parallel and divide-and-conquer) through the

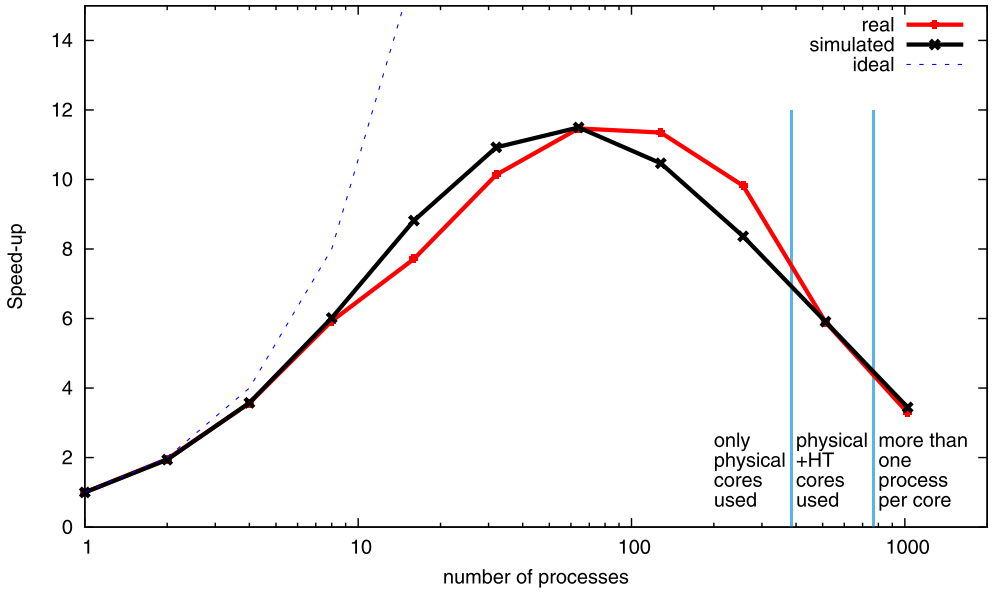

Fig. 13. Divide-and-conquer application – Speed-up vs. number of processes.

well known Java language with extensions in the form of MPI like routines. Execution times predicted by MERPSYS simulations are very close to measured execution times of real applications (see individual experiment error values). We were very satisfied to observe average errors between 1.4 and 7.8% for all conducted simulations. Not only do simulation results match real values but also demonstrate local minimas for application execution time versus the number of processes. What is also important is that such matching was obtained for applications representative of three distinct parallel processing paradigms: master-slave, geometric single program multiple data and divide-and-conquer applications that differ in computation/communication ratio, communication and synchronization methods.

Consequently, the models could also be used for finding expected execution times for other configurations, either application or system. In particular, specific components in the system model such as CPU, GPU or network switch models can be instantly replaced with others from the MERPSYS database in order to obtain corresponding results and e.g. find upgrade paths for hardware.

In the future, we plan to extend evaluation of MERPSYS with NAS Parallel Benchmarks (NPB) [54]. Furthermore, other goals related to MERPSYS include building an automatic calibration tool that, based on results from selected real runs and regression analysis, would match coefficients corresponding to hardware components such as those not yet in the database. Additionally, we are working on an automated tool embedded in MERPSYS that would suggest best architectures and components for any given code including execution time and power consumption.

Acknowledgments

The work was partially performed within grant “Modeling efficiency, reliability and power consumption of multilevel parallel HPC systems using CPUs and GPUs” sponsored by and covered by funds from the National Science Centre of Poland based on decision no DEC-2012/07/B/ST6/01516. The work has been supported partially by the Polish Ministry of Science and Higher Education.

References

[1] Intel® Xeon® CPUs, http://ark.intel.com/#@Processors, online; accessed 12-April-2017.
[2] NVIDIA GPUs, https://www.top500.org/system/178764, online; accessed 12-April-2017.
[3] Sunway taihulight, https://www.top500.org/system/178764, online; accessed 12-April-2017.
[4] H. Fu, J. Liao, J. Yang, L. Wang, Z. Song, X. Huang, C. Yang, W. Xue, F. Liu, F. Qiao, et al., The sunway taihulight supercomputer: system and applications, Sci. China Inf. Sci. 59 (7) (2016) 072001.
[5] BOINC, http://boinc.berkeley.edu/, online; accessed 12-April-2017.
[6] Globus toolkit, http://toolkit.globus.org/toolkit/, online; accessed 12-April-2017.
[7] UNICORE, http://www.unicore.eu/documentation/manuals/unicore/files/client_intro.pdf, online; accessed 12-April-2017.
[8] Gridbus, http://gridbus.cs.mu.oz.au/middleware/, online; accessed 12-April-2017.
[9] P. Rosciszewski, P. Czarnul, R. Lewandowski, M. Schally-Kacprzak, Kernelhive: a new workflow-based framework for multilevel high performance computing using clusters and workstations with CPUs and GPUs, Concurrency Comput. 28 (9) (2016) 2586–2607. http://dx.doi.org/10.1002/cpe.3719.
[10] MERPSYS server, http://merpsys.eti.pg.gda.pl/portal, online; accessed 12-April-2017.
[11] P. Czarnul, J. Kuchta, P. Rościszewski, J. Proficz, Modeling energy consumption of parallel applications, in: 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), 2016, pp. 855–864.

[12] P. Rosciszewski, Executing multiple simulations in the MERPSYS environment, in: Modeling Large-Scale Computing Systems. Practical Approaches in MERPSYS, Gdansk University of Technology, 2016, pp. 123–133. 978-83-938367-2-7, https://repository.os.niwa.gda.pl/handle/niwa_item/138.
[13] W. Kreutzer, J. Hopkins, M. van Mierlo, Simjava – a framework for modeling queueing networks in java, in: Proceedings of the 29th Conference on Winter Simulation, WSC '97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 483–488. http://dx.doi.org/10.1145/268437.268548.
[14] A. Varga, OMNet++, in: Modeling and Tools for Network Simulation, Springer Berlin Heidelberg, 2010, pp. 35–59, doi:10.1007/978-3-642-12331-3_3.
[15] R. Buyya, M. Murshed, Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency Comput. 14 (13–15) (2002) 1175–1220, doi:10.1002/cpe.710.
[16] J. Proficz, P. Czarnul, Performance and Power-Aware Modeling of MPI Applications for Cluster Computing, Springer International Publishing, Cham, 2016, pp. 199–209. http://dx.doi.org/10.1007/978-3-319-32152-3_19.
[17] W.E. Denzel, J. Li, P. Walker, Y. Jin, A framework for end-to-end simulation of high-performance computing systems, in: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, Simutools '08, ICST, vol. 21, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, ICST, Brussels, Belgium, Belgium, 2008. pp. 21:1–21:10. http://dl.acm.org/citation.cfm?id=1416222.1416248.
[18] Message passing interface forum, 2015, MPI : A Message-Passing Interface Standard, Version 3.1.
[19] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. Pract. Exper. 41 (1) (2011) 23–50. http://dx.doi.org/10.1002/spe.995.
[20] A. Medina, A. Lakhina, I. Matta, J. Byers, Brite: Boston University representative internet topology generator, 2001,
[21] S. Bak, M. Krystek, K. Kurowski, A. Oleksiak, W. Piatek, J. Weglarz, GSSIM - A tool for distributed computing experiments, Sci. Program. 19 (4) (2011) 231–251. http://dx.doi.org/10.3233/SPR-2011-0332.
[22] Grid workload format, http://gwa.ewi.tudelft.nl/pmwiki/, online; accessed 12-April-2017.
[23] H. Adalsteinsson, S. Cranford, D.A. Evensky, J.P. Kenny, J. Mayo, A. Pinar, C.L. Janssen, A simulator for large-scale parallel computer architectures, Int. J. Distrib. Syst. Technol. 1 (2) (2010) 57–73, doi:10.4018/jdst.2010040104.
[24] H. Casanova, A. Legrand, M. Quinson, Simgrid: a generic framework for large-scale distributed experiments, in: Proceedings of the Tenth International Conference on Computer Modeling and Simulation, UKSIM '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 126–131. http://dx.doi.org/10.1109/UKSIM.2008.28.
[25] Simgrid website, http://simgrid.gforge.inria.fr/, online; accessed 12-April-2017.
[26] B. Donassolo, H. Casanova, A. Legrand, P. Velho, Fast and scalable simulation of volunteer computing systems using simgrid, in: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 605–612. http://dx.doi.org/10.1145/1851476.1851565.
[27] C.L. Dumitrescu, I. Foster, Gangsim: a simulator for grid scheduling studies, in: Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, vol. 2, IEEE, 2005, pp. 1151–1158.
[28] T.T. Sa, R. Calheiros, D. Gomes, Cloudreports: an extensible simulation tool for energy-aware cloud computing environments, in: Cloud Computing, Springer International Publishing, 2014, pp. 127–142. ISBN 978-3-319-10529-1.
[29] B. Pranggono, D. Alboaneen, H. Tianfield, 11 Simulation Tools for Cloud Computing, CRC Press, 2014.
[30] A. Bashar, Modeling and simulation frameworks for cloud computing environment: a critical evaluation, Int. J. Comput. Inf. Eng. 1(9) 1−6, http://www.pmu.edu.sa/kcfinder/upload/files/ICCCSS2014_Abul_Bashar.pdf.
[31] R. Malhotra, P. Jain, Study and comparison of cloudsim simulators in the cloud computing, SIJ Trans. Comput. Sci. Eng. Appl., 1(4) 111−115.
[32] M. Kaleem, P. Khan, Commonly used simulation tools for cloud computing research, in: Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on, 2015, pp. 1104–1111.
[33] A. Ahmed, A.S. Sabyasachi, Cloud computing simulators: a detailed survey and future direction, in: Advance Computing Conference (IACC), 2014 IEEE International, IEEE, 2014, pp. 866–872.
[34] P. Czarnul, P. Rosciszewski, M.R. Matuszek, J. Szymanski, Simulation of parallel similarity measure computations for large data sets, in: 2nd IEEE International Conference on Cybernetics, CYBCONF 2015, Gdynia, Poland, June 24–26,2015, IEEE, 2015, pp. 472–477. http://dx.doi.org/10.1109/CYBConf.2015.7175980.
[35] P. Czarnul, M. Matuszek, Performance modeling and prediction of real application workload in a volunteer-based system, in: Applications of Information Systems in Engineering and Bioscience, Proceedings of 13th International Conference on Software Engineering, Parallel and Distributed Systems conference (SEPADS), WSEAS, Gdansk, Poland, 2014, pp. 37–45. ISBN: 978-960-474-381-0, http://www.wseas.us/e-library/conferences/2014/Gdansk/SEBIO/SEBIO-03.pdf.
[36] P. Rosciszewski, Modeling and simulation for exploring power/time trade-off of parallel deep neural network training, in: Proceedings of ICCS 2017 Conference, Zurich, Switzerland, Procedia Computer Science, 2017. In press.
[37] P. Czarnul, K. Grzeda, Parallel simulations of electrophysiological phenomena in myocardium on large 32 and 64-bit linux clusters, in: P. Kacsuk, J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19–22, 2004, Proceedings, Vol. 3241 of Lecture Notes in Computer Science, Springer, 2004, pp. 234–241. http://dx.doi.org/10.1007/978-3-540-30218-6_35.
[38] K. Key, J. Ovall, A parallel goal-oriented adaptive finite element method for 2.5-d electromagnetic modelling, Geophys. J. Int. 186 (1) (2011) 137–154. http://dx.doi.org/10.1111/j.1365-246X.2011.05025.x.
[39] S. Buckeridge, R. Scheichl, Parallel geometric multigrid for global weather prediction, Numer. Linear Algebra Appl. 17 (2–3) (2010) 325–342. http://dx.doi.org/10.1002/nla.699.
[40] P. Czarnul, Parallelization of divide-and-conquer applications on intel xeon phi with an openmp based framework, in: J. Swiatek, L. Borzemski, A. Grzech, Z. Wilimowska (Eds.), Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology - ISAT 2015 - Part III, Karpacz, Poland, September 20–22,2015, Vol. 431 of Advances in Intelligent Systems and Computing, Springer, 2015, pp. 99–111. http://dx.doi.org/10.1007/978-3-319-28564-1_9.
[41] Java EE 1.7, http://www.oracle.com/technetwork/java/javaee/tech/index.html, online; accessed 12-April-2017.
[42] Java EE full profile, http://jcp.org/aboutJava/communityprocess/final/jsr342/index.html, online; accessed 12-April-2017.
[43] Glassfish open source edition, http://glassfish.java.net/docs/, online; accessed 12-April-2017.
[44] PostgreSQL Server, http://www.postgresql.org/docs/, online; accessed 12-April-2017.
[45] Oracle, Java DataBase Connectivity Tutorial, http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html.
[46] Java web start technology, http://jcp.org/aboutJava/communityprocess/final/jsr056/index.html, online; accessed 12-April-2017.
[47] Java message service, http://jcp.org/aboutJava/communityprocess/final/jsr914/index.html, online; accessed 12-April-2017.
[48] Open message queue, http://docs.oracle.com/cd/E19798-01/, online; accessed 12-April-2017.
[49] Galera+ cluster, http://task.gda.pl/kdm/sprzet/gplus/, online; accessed 12-April-2017.
[50] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2005.
[51] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, Inf. Theory IEEE Trans. 13 (1) (1967) 21–27.
[52] M. Du, X.-s. Chen, Accelerated k-nearest neighbors algorithm based on principal component analysis for text categorization, J. Zhejiang Univ. SCI. C 14 (6) (2013) 407–416. http://dx.doi.org/10.1631/jzus.C1200303.
[53] J.A. Hartigan, M.A. Wong, Algorithm AS 136: a k-means clustering algorithm, Appl. Stat. 28 (1978) 100–108.
[54] R.F.V.d. Wijngaart, Nas Parallel Benchmarks Version 2.4, Technical Report NAS Technical Report NAS-02-007, NASA Advanced Supercomputing (NAS) Division, 2002. https://www.nas.nasa.gov/assets/pdf/techreports/2002/nas-02-007.pdf.