Massively Scalable Platform for Data Farming Supporting Heterogeneous Infrastructure

Dariusz Król, Michał Wrzeszcz, Bartosz Kryza, Łukasz Dutka
AGH University of Science and Technology
Academic Computer Centre Cyfronet AGH
Krakow, Poland
{dkrol, wrzeszcz, bkryza, dutka}@agh.edu.pl

Jacek Kitowski
AGH University of Science and Technology
Department of Computer Science and Academic Computer Centre Cyfronet AGH
Krakow, Poland
kito@agh.edu.pl

Abstract — **Data farming is a scientific methodology, which heavily depends on technical advances in high throughput computing to generate large amounts of data with computer simulation to investigate studied phenomena. Unfortunately, the availability of versatile data farming systems is very limited and none of existing tool enables integration with novel Cloud solutions. This paper presents a flexible platform for conducting large-scale data farming experiments on heterogenous computational infrastructure including: clusters, Grids and Clouds. Another important feature of the presented platform is the support of interactive data farming experiments, which includes an online analysis of partial experiment results and experiment extending capabilities.**

Keywords - scalability; data farming; software platform; high throughput computing.

I. Introduction

In many disciplines of modern science, scientific discoveries are results of collecting and analyzing large amounts of data. In particular, an increasing popularity of conducting experiments, both physical and virtual, to understand studied phenomena leads to big data generation. A physical experiment often is too expensive to conduct it multiple times, e.g., when requires expensinve equipment such as airplane engines or battleships, thus computer simulations are performed instead. Technological advances in recent years have led to significant improvements in the computer simulation field, e.g., reduction of the required time to run a computer simulation and refinement of simulation models in regard to its complexity. One can now simulate complicated phenomena in minutes or hours instead of days or months, with an improvement of results quality and simulation complexity.

Based on this technological progress, new forms of scientific methodologies have emerged, which are based on data-intensive computation and analysis. One such a methodology is called "The Fourth Paradigm" [1], in which new scientific findings are discovered by analyzing big amount of data coming from various scientific experiments. A complementary approach, which is gaining more and more popularity in recent years, is Data Farming [2], whose main objective is to develop a better understanding of landscape of possibilities as well as outliers that may be discovered through simulation. This is especially important when concerning a decision-making process regarding complicated nature of scenarios involving security forces. The origin of the Data Farming methodology is in USA Marine Corps, where it was proposed to enhance military strategies. Though today, it is used in other disciplines of science [3-4]. The basic idea behind Data Farming is to grow significant amount of data by performing large number of simulations of a studied phenomena, each with a slightly different input values. Simulation results are described by a vector of parameters, called Measures of Effectiveness (MoE), which is used to evaluate each simulation. A result vector is treated as a single point of possible output landscape. After gathering a number of such points, a scientist can perform analysis of existing trends or anomalies, based on which, new insights into phenomena can be obtained.

A crucial requirement for conducting data farming experiments effectively is usage of high performance and throughput computer infrastructure. It is necessary to run a large number of simulations simultaneously and gathering output results. In addition, it is often required to integrate many heterogeneous computational infrastructures, when an experiment requires more computational power than a single computer centre can provide. Moreover, as new types of computational infrastructures are emerging, e.g., public Clouds, integration with existing infrastructures, e.g., Grid environments, becomes a major issue. Thus, a holistic platform, which will virtualize computational and storage resources, is required to conduct data farming experiment in an efficient way. In particular, it should automate all cumbersome technical aspects of infrastructure configuration and simulation running. Besides fulfilling functional requirements, such a platform should be scalable and adaptable to a changing state of knowledge about the studied phenomena, in order to be used in both small and large data farming experiments.

The rest of the paper is organized as follows. In Section II, we present existing tools, which can be utilized for conducting data farming experiments. Section III describes our platform, called Scalarm, its main design principles and objectives. Then, in Section IV, an experimental evaluation of the presented platform is depicted. We conclude this paper in Section V.

II. STATE OF THE ART

Although, Data Farming is becoming a popular scientific methodology lately, the software supporting this methodology is rather limited. One of the very few examples of such tools is OldMcData [5], which supports only two parts of a data farming experiment. It can prepare input a set of input vectors based on possible range of parameter values and selected design of experiment methods. Afterwards, it can schedule simulations to run on available computational resources with the Condor software [6], which can be configured to work with distributed resources and is able to move simulations' output from distributed computational resources to a designated location. However, no method for data analysis is provided, which means that external tools have to be used. Moreover, running simulations is a batch-like process, i.e., a whole package of inputs is submitted to a scheduler at once. The user can proceed to data analysis after the whole experiment is finished. There is no information about any partial results and the user cannot modify the parameter space of an experiment once submitted. Condor supports heterogeneous infrastructure integration, but it lacks the scaling feature in regards to application managers, which means the infrastructure delegated to perform the experiment has to be set before starting simulations and cannot be changed during the runtime.

Although, data farming oriented tools are rather limited in number, there are several tools, which can support different phases of the data farming process independantly. One of the most important phases of the process is simulation execution with high throughput computational infrastructure. There are several tools available for this task as this is a generic problem in many computational disciplines. Distributed Infrastructure with Remote Agent Control (DIRAC) [7] is a platform supporting computations with heterogeneous resources including local clusters, Grids and Clouds. It was originally developed to provide a complete solution for using the distributed computing resources of the LHCb experiment at CERN for data production and analysis. DIRAC provides an additional abstraction layer between users and various compute resources to allow optimized, transparent and reliable usage. It exploits the concepts of Workload Management System with Pilot Jobs, which increase computations efficiency and reliability. DIRAC utilizes an agent-based architecture, where agents are deployed on the worker nodes, building a dynamic overlay network of readily available resources. These agents, being actually a representation of available computing resources, intend to reserve computational power to run actual tasks, which are distributed using a custom scheduling method. By using the Pilot jobs and Workload Management System concepts, DIRAC implements redundancy at the computational task level, i.e., DIRAC guarantees that tasks will be run, and in case of any failure it will be rescheduled. In addition, these concepts allow aggregating in a single system computing resources of a different nature, such as computational grids, clouds and clusters, transparently for the users. DIRAC provides the data management functionality, however it is related to data distribution in a reliable manner among computational resources. It does not provide functionality required to analyse job results. Also, it does not have design of experiment methods built in for sampling input parameter value space, based on which computational jobs should be generated and scheduled. Thus, it can be only used as a part of a complete data farming platform, rather than being a complete solution for its own.

Falkon, which stands for a "Fast and Light-weight tasK executiON framework", is a framework for rapid execution of many tasks on compute clusters [8]. Falkon focuses on efficient task dispatching, and delivers dispatching performance better than other systems, i.e., upto 440 tasks/sec. Furthermore, Falkon is highly scalable in terms of workers, which can be utilized to perform tasks, i.e., to over 54,000. Thus, applications end-to-end run time can be reduced in some cases up to 90% relative to versions that execute tasks via separate scheduler submissions. To achieve such performance and high scalability, Falkon utilizes a concept of multi-level scheduling to separate resource acquisition from task dispatch. Moreover, a streamlined dispatcher is used, which improves performance but eliminates support for features such as multiple queues, prorities, accounting, etc. Falkon consists of a dispatcher, a provisioner, and multiple executors. The dispatcher accepts tasks from clients and schedules subsequent tasks to next available executors. The provisioner is responsible for creating and destroying executors on available computational resources. Executors run tasks received from the dispatcher. Each new executor registers with the dispatcher. Components communicate via Web Services (WS) messages, except for notifications are performed via a custom TCP-based protocol. Although, Falkon provides high throughtput and executors' scalability, it lacks dispatchers' scalability, i.e., performance of Falkon is constrainted by capabilities of the server, which runs the dispatcher component. Moreover, whole Falkon functionality is limited only to dispatching, hence no functionality related to parameter space generation or results analysis is provided.

Since data farming is still a relatively uncharted territory none of existing tools provides functionality required for flexible running of various data farming experiments with different types of parameters and even simulation implementation technologies.

III. SCALARM PLATFORM

Due to lack of versatile software for conducting data farming experiments, we developed a new system from scratch, called Scalarm [9], which stands for Massively Scalable Platform for Data Farming. Scalarm intends to fulfill the following requirements:

- support all phases of a data farming experiment, starting from a design of experiment phase, through simulation execution and progress monitoring, to statistical analysis of results,

- support different sizes of experiments from dozens to millions of simulations through massive scalability,
- support for heterogenous computational infrastructure including private clusters, Grids and Clouds.

A. Provided functionality

Scalarm functionality focuses on conducting experiments, which follows the Data Farming methodology. In addition, Scalarm introduces an exploratory approach to experiment conducting. In a batch-like experiment execution, the user submits an experiment as a single package, waits for all simulations to compute, and then analyze obtained results. Based on the result analysis, new experiments are conducted to investigate interesting cases in more details. This loop can be reapeated several times. On the other hand, the exploratory approach enables users to expand the parameter space of running experiments, based on an on-line analysis of already computed simulations, e.g., with regression trees and MoE histograms. Hence, the user can specify only small parameter space at first, and expand it on-line later on, which is a more natural way of conducting such experiments.

Supported use cases can be divided into three groups based on their expected results: experiment management, analysis and platform management. The first group includes activities related to preparation of new data farming experiments, their further monitoring and management, e.g., adding computational resources to execute simulations included in a concrete experiment. The second group, i.e., analysis, contains all actions, which intend to visualise and discover knowledge from simulations' results in form of various charts and graphs. Hence, they can be utilized to discover meaningful insight into studied phenomena. The last group, i.e., platform management, includes use cases, which are important for a multi-tenant environment to operate, but they do not support the data farming process directly, e.g., login.

B. Architecture of the platform

Selecting an appropriate architecture style for virtual platforms, which intend to be deployed at a large scale, is the basic problem of modern software engineering. At a high-level of abstraction, Scalarm follows the “master-worker” design pattern, i.e., one part of the platform is responsible for scheduling the actual work to the other part of the platform.

Scalarm’s architecture utilizes a service-oriented approach with an additional modification, which addresses the scalability requirement. To cope with the requirement, we do not operate on the level of components and services, which represent single instances only. Instead, we extended the meaning of an application's modularization unit to embrace the scalability feature. Thus, each Scalarm service can consist internally of a number of component’s instances, which provides exposed functionality, and a load balancer, which constitutes a single entry point to the service. An overview of the architecture is depicted in Fig. 1.

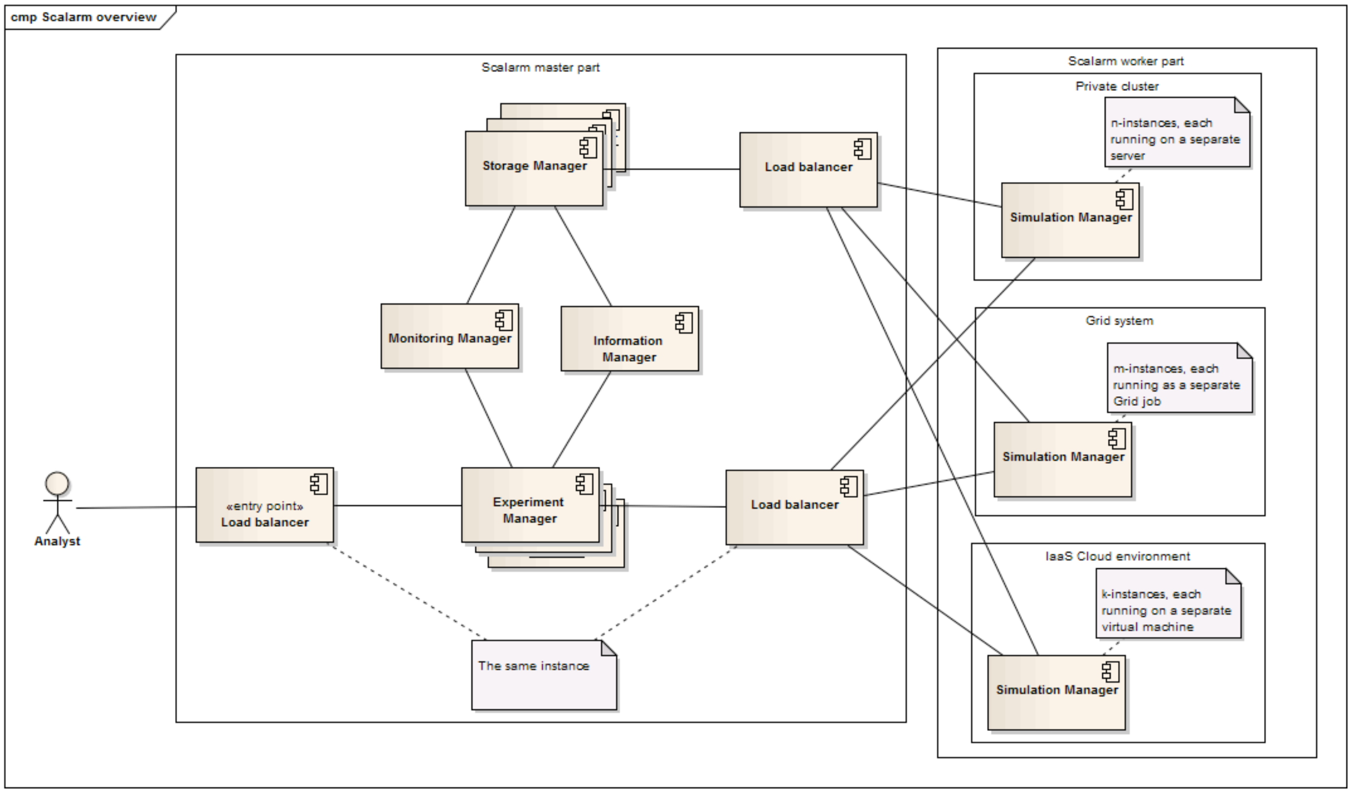

Figure 1. A component diagram of Scalarm.

The Scalarm platform includes the following components:

- Experiment Manager, which handles all interaction between the platform and actual users via a Graphical User Interface. On the one hand, it constitutes a gateway to the platform for analysts, i.e., provides a coherent view of information about all running and completed data farming experiments, and enables analysts to create new experiments or to conduct statistical analysis on existing ones. On the other hand, Experiment Manager is responsible for scheduling simulations to Simulation Managers.
- Storage Manager is an equivalent of the persistence layer concept but in a form of a separate service. Other components, mainly Experiment and Simulation Managers use this service to store different types of data: structural information about each executed simulation and experiment, and actual results of simulations, which may be either binary or text data. By utilizing a built-in load balancer, Storage Manager can be treated as a virtually centralized but physically distributed single point of data storage, which facilitates the client side while preserving performance and scalability.
- Simulation Manager is an intelligent wrapper for actual simulations, which can be deployed on various computational infrastructures, e.g., private cluster, Grids or Clouds. It can be treated as an implementation of the Pilot job concept, i.e., a special application that intends to acquire computational resources to run actual applications. However, while the Pilot job concept was created for Grid environments only, Simulation manager is infrastructure independent. The wrapper is responsible for preparing whole environment for a simulation, i.e., download necessary code dependencies and input parameter values. After a simulation is finished, Simulation Manager uploads results to the "master" part, i.e., log files and other binary outputs are sent to Storage Manager, while MoE values are sent to Experiment Manager along with information about simulation completion. As it can operate in a highly dynamic and unreliable environment, Simulation Manager supports fault tolerance for Experiment and Storage Managers failures as well as network connectivity issues. Moreover, to maximize resource utilization, Simulation Manager starts multiple simulations in parallel based on actual computational resource capabilities, i.e., additional simulations are started if it will not significantly decrease performance of already started simulations.
- Information Manager is an implementation of the Service locator pattern, known from SOA-based systems. It is a "well-known" place for each component in the system, which stores information about other components' locations.
- Monitoring Manager constitutes a distributed monitoring system for the Scalarm platform. It contains two separated elements: sensors, which periodically sent monitoring data and a service, which stores this information. Sensors are built directly into each Experiment, Storage and Simulation Managers. It collects information about workload of Scalarm components, using operating system metrics, e.g., CPU and RAM memory utilization, as well as component specific metrics, e.g., response time of various requests.

C. *Supported applications*

Scalarm was originally evaluated in a multi-agent simulation area, with a goal of supporting a training process of security forces. A sample simulation scenario involved controlling the access of civilians to a military base camp during elections in a mission abroad. In this scenario, civilians were waiting in front of a camp entrance to an operation base with an intention to start a skirmish. From the security forces point of view, the goal of this scenario was to prevent the escalation of agression by effectivie negotiations. However, civilians may act differently, depending on input parameter values, hence actions performed by security forces should be adjusted to a concrete behaviour. A goal of a data farming experiment, which used this simulation scenario, was to find out how to minimize the number of injured civilians in such a scenario, regardless their behaviour.

Scalarm facilitated the experiment at the following phases:

- A design of experiment phase, whose result is a specification of the input parameter space. Scalarm provides a set of views, where an analyst specifies types of parametrization for each input parameter and design of experiment methods, which should be used.
- Simulation execution on heterogenous computational infrastructure. Scalarm supports different types of computational infrastructures, i.e., common computational clusters available via SSH, Grid environments accessible via the gLite middleware [10], and public clouds supporting Amazon EC2 API.
- Statistical analysis of results. Scalarm provides a set of built-in graphs, which can be created based on completed simulation results: histograms, regression trees and bivariate graphs.

For more details about conducted data farming experiments regarding security forces, please refer to [11]. Though, Scalarm was evaluated with a particular type of simulations, it can be used in any other science discipline, where the Data Farming methodology can be utilized, e.g., materials science or life-science.

IV. Experimental Evaluation

To evaluate Scalarm, we conducted both functional tests of supporting different computational infrastructures and performance tests to measure the platform's scalability.

Functional tests concerned simulating a scenario described in Section III. Two standard HP ProLiant worker nodes (described in a following section) were used to run one instance of Experiment and Storage Managers. To run Simulation Managers, we used:

- 9 worker nodes from a private cluster,
- 50 Grid jobs scheduled to a Grid infrastructure,
- 50 High-CPU Extra Large instances from Amazon EC2.

In the experiment design phase, 14 from 92 of simulation input parameters (describing initial emotional state and other attributes of simulated entities) were set to the "Range" parametrization with 2^k method applied, which generated 16 386 different cases to simulate. The utilized set of resources enabled us to execute more than 620 simulations simultaneously with more then 140 simulations complete in each minute.

An output of each simulation included: a text file with less than 7 MB of simulation logs, and 44 different MoEs describing aggregated emotional states of different entity groups and statistic regarding the simulated scenario. Compressed logs were sent to Storage Managers, which had a disk array connected with 6 TB of total capacity.

After several minutes of computations, an analysis of gathered results was conducted using histograms and regression trees. Based on this analysis, the experiment was extended with additional parameter values. A whole test was recorded and can be found online at [12].

The second set of tests concerned the platform's scalability. We intended to evaluate scalability of the master part, which includes Experiment and Storage Managers, since running many independent workers is trivial. We used production infrastructure, however an empty simulation was actually performed to minimize the number of Simulation Managers required to saturate platform's throughput, which was measured with completed simulations registered by Experiment Managers in a period of time.

A. Testing scenarios

Our testing scenarios focused on evaluating how Scalarm handles experiments of various sizes with different amount of computational resources. The main measured parameter was the total execution time of each experiment. Scalarm has three main components, namely Experiment, Storage and Simulation Managers, which can be scaled. In presented tests, the number of Simulation Managers was experimentally selected to saturate platform's throughtput. Hence, only numbers of Experiment and Storage Managers were used as parameters of performed tests.

Regarding experiment sizes, i.e., the number of simulations within an experiment, we used the following set of values to present full capabilities of the platform: 100 000, 200 000, 500 000, 1 000 000, 2 000 000, 5 000 000.

Concerning computational resources, the parameter depicted the number of servers dedicated to run Experiment and Storage Managers. Our tests included the following values of this parameter: 1, 2, 4 and 8. However, each component run on a separate set of servers, which means that in each test, the total number of servers was doubled.

B. Testing environment

In case of performance tests, we used a computing cluster to run Experiment and Storage Managers to minimize the network latency. Simulation Managers were scheduled to a part of PL-Grid infrastructure located within the same site.

To run each component, we used standard HP ProLiant worker nodes, connected with each other through a 10 GbE network switch, while connection between a worker node and switch was 1 GbE link. Each worker node has the following parameters:

- 2x Intel Xeon CPU L5420 @ 2.50GHz
- 16 GB RAM
- 120 GB hard drive (5400 RPM)

C. Evaluation results

Aggregated test results are depicted in Fig. 2. Each line on the chart denotes a separate configuration of Scalarm used in tests, i.e., numbers of servers running Experiment and Storage Managers represented as a pair (<experiment managers count>, <storage managers count>). For each configuration, we measured total execution time in seconds for experiments of different sizes.

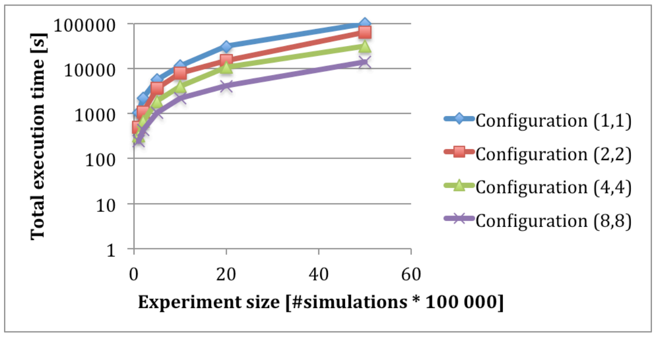

Figure 2. Experiments' execution time for different Scalarm configurations.

There are a few things worth noticing. First of all, the more resources Scalarm has, the better performance it provides. The performance gain varied depending on actual experiment size. Let's compare configurations (1,1) and (2,2). Execution time decreases by 53% for experiment size 100 000, but only by, 31% for experiment size 1 000 000.

The second notice concerns the execution time of experiments with an increasing size using the same configuration. Regardless the configuration, the execution time of subsequent experiments with an increasing number of simulations rises more than linearly. It is caused by an increasing effort of simulation information management. Each simulation is represented in Scalarm by a row in a non-relational database. Performance of such databases depends on the IO subsystem, especially when concerning millions of rows. Hence, after exceeding some thresholds of a database size (about a million of rows on a single server), database operations tend to take more than expected.

Based on obtained results, we calculated speedup (1) and efficiency (2) metric using classical formulas.

$$\text{Speedup}(N) = T(1) / T(N) \quad (1)$$

$$\text{Efficiency}(N) = \text{Speedup}(N) / N \quad (2)$$

Efficiency of Scalarm (depicted in Fig. 3) is greater than 0.7 in most cases, which is a good result, especially when concerning a wide range of tested configurations. Furthermore, for some experiment sizes and the configuration consisted of 2 servers for Experiment and Storage Managers respectively, efficiency is greater than 1, which could be have been caused by data sharding between instances of a database on seperated servers, which enabled having all data in memory instead of using local disk.

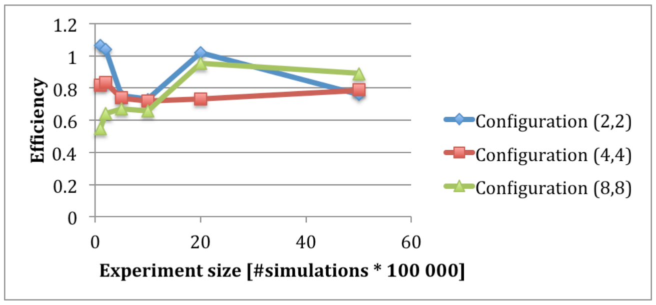

Figure 3. Scalarm efficiency for configurations including more than 1 server per component.

An average throughput for the Configuration (1, 1) was about 4776 simulations per minute. We estimated the number of Simulation Managers, which would be required to saturate Scalarm when running actual simulations by comparing to the throughput of running actual simulations and the throughput with an empty simulation. In the case of our simulation, we should have more than 120 000 of Simulation Managers running simultanously. This was the main reason why the scalability evaluation was performed with an empty simulation.

V. Conclusions and Future Work

In this paper, we presented a versatile system for running large scale data farming experiments involving processing of a parameter space where custom design of experiment methods and interactive fine tuning of processed parameter space are required. The system is currently being evaluated for military mission planning support in order to improve behavior models for agent-based simulation component and to allow drawing conclusions regarding selected Measures of Effectiveness for higher echelons.

The future work will include application of the platform in a metallurgy scenario [13], with focus on distributed semantic-based Virtual Organization collaborations [14].

Acknowledgment

The authors are very grateful to Łukasz Flis, Marek Magryś and Patryk Lasoń from Cyfronet for help in preparing the testing environment based on the PL-Grid infrastructure. The research is partially supported by the POIG.02.03.00-00-096/10 “PL-Grid PLUS” project. D. Król thanks to the National Science Centre grant no. 2012/05/N/ST6/03461 for support. AGH-UST grant no. 11.11.120.865 is also acknowledged.

References

[1] T. Hey, S. Tansley, and K. Tolle, “The Fourth Paradigm: Data-Intensive Scientific Discovery”, Eds., Redmond, VA: Microsoft Research, 2009, ISBN 978-0-9825442-0-4.

[2] A. Brandstein and G. Horne, “Data Farming: A Meta-Technique for Research in the 21st Century”, in Maneuver Warfare Science 1998, Marine Corps Marine Corps Combat Development Command Publication, Quantico, Virginia, 1998.

[3] D. Moses, “Data farming helps hospital keep nurses at bedside”, HealthCareITNews, [online: http://www.healthcareitnews.com/news/data-farming-helps-hospital-keep-nurses-bedside as of January 14, 2013]

[4] T. Beach, et al., “Application of Design of Experiments & Data Farming Techniques for Planning Tests in a Joint Mission Environment”, International Data Farming Workshop 15, November 2007.

[5] S. Upton, “Users Guide: OldMcData, the Data Farmer”, Version 1.1, United States Marine Corps Project Albert. Quantico, Virginia, 2010.

[6] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, “Condor: a distributed job scheduler”, Beowulf Cluster Computing with Windows, MIT Press Cambridge, MA, USA, 2002, pp. 307 – 350, ISBN:0-262-69275-9.

[7] J. Saborido, F. Gomez-Folgar, J. L. Cacheiro, and R. G. Diaz, “DIRAC Integration with Cloud Stack”, Proc. IEEE Third International Conference on Cloud Computing Technology and Science, 2011, pp. 537-541. ISBN: 978-0-7695- 4622-3.

[8] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, “Falkon: a Fast and Light-weight tasK executiON framework”, Proc. ACM/IEEE conference on Supercomputing, 2007, pp. 1-12, ISBN: 978-1-59593-764-3.

[9] B. Kryza, D. Król, M. Wrzeszcz, Ł. Dutka, and J. Kitowski, „Interactive Cloud Data Farming Environment for Military Mission Planning Support”, Computer Science Journal, vol 13(4), 2012, pp. 89-100.

[10] gLite – Lightweigt Middleware for Grid Computing website, [online: http://glite.cern.ch/ as of January 14, 2013]

[11] S. Dlugolinsky, et al., “Using parallelization for simulation of human behaviour”. 7th International Workshop on Grid Computing for Complex Problems, Bratislava, 2011, pp. 258-265, ISBN 978-80-970145-5-1.

[12] Scalarm website – overview section, [online: http://www.scalarm.com/overview.html as of January 14, 2013]

[13] J. Kitowski and B. Kryza, “Dynamic virtual organization management framework supporting distributed industrial collaboration”, Computer Methods in Materials Science, vol. 11(4), 2011, pp. 514-523.

[14] A. Mylka, A. Mylka, B. Kryza, and J. Kitowski, “Integration of Heterogenous Data Sources in an Ontological Knowledge Base”, Computing and Informatics, vol. 31(1), 2012, pp. 189–223.