Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

Volume Editors

Jesus Carretero
University Carlos III
Computer Architecture and Technology Area
Computer Science Department
Avda Universidad 30, 28911, Leganes, Spain
E-mail: jesus.carretero@uc3m.es

Javier Garcia Blas
University Carlos III
Computer Architecture and Technology Area
Computer Science Department
Avda Universidad 30, 28911, Leganes, Spain
E-mail: fjblas@arcos.inf.uc3m.es

Roman Wyrzykowski
Institute of Computer and Information Science
Czestochowa University of Technology
ul. Dąbrowskiego 73, 42-201 Częstochowa, Poland
E-mail: roman@icis.pcz.pl

Emmanuel Jeannot
Equipe Runtime
INRIA Bordeaux Sud-Ouest
200, Avenue de la Vielle Tour, 33405 Talence Cedex, France
E-mail: emmanuel_jeannot@inria.fr

Published by:
Computer Architecture,Communications, and Systems Group (ARCOS)
University Carlos III
Madrid, Spain
http://www.nesus.eu

ISBN: 978-84-608-2581-4

This document also is supported by:

Printed in Madrid — October 2015

Preface

Network for Sustainable Ultrascale Computing (NESUS)

We are very excited to present the proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015), a workshop created to reflect the research and cooperation activities made in the NESUS COST Action (IC1035) (www.nesus.eu), but open to all the research community working in large/ultra-scale computing systems. It was held in Krakow (Poland) on September 10-11, 2015.

The goal in scalable and sustainable technology today is to have on the one hand large parallel supercomputers, named Exascale computers, and on the other hand, to have very large data centers with hundreds of thousands of computers coordinating with distributed memory systems. Ultimately, NESUS idea is to have both architectures converge to solve problems in what we call ultrascale. Ultrascale systems combine the advantages of distributed and parallel computing systems. The former is a type of computing in which many tasks are executed at the same time coordinately to solve one problem, based on the principle that a big problem can be divided into many smaller ones that are simultaneously solved. The latter system, in both grid and cloud computing, uses a large number of computers organized into clusters in a distributed infrastructure, and can execute millions of tasks at the same time usually working on independent problems and big data. The applications of these systems and the benefits they can yield for society are enormous, according to the researchers, who note that this type of computing will help conduct studies about genomics, new materials, simulations of fluid dynamics used for atmospheric analysis and weather forecasts, and even the human brain and its behavior.

The goal of the NESUS Action is to establish an open European research network targeting sustainable solutions for ultrascale computing aiming at cross fertilization among HPC, large scale distributed systems, and big data management. Ultrascale systems are envisioned in NESUS as large-scale complex systems joining parallel and distributed computing systems that will be two to three orders of magnitude larger that today's systems. The EU is already funding large scale computing systems research, but it is not coordinated across researchers, leading to duplications and inefficiencies. The network will contribute to glue disparate researchers working across different areas and provide a meeting ground for researchers in these separate areas to exchange ideas, to identify synergies, and to pursue common activities in research topics such as sustainable software solutions (applications and system software stack), data management, energy efficiency, and resilience. Some of the most active research groups of the world in this area are members of this NESUS Action. This Action will increase the value of these groups at the European-level by reducing duplication of efforts and providing a more holistic view to all researchers, it will promote the leadership of Europe, and it will increase their impact on science, economy, and society.

The scientific objective of NESUS is to study the challenges presented by the next generation of ultrascale computing systems to enhance their sustainability. These systems, which will be characterized by their large size and great complexity, present significant challenges, from their construction to their exploitation and use. We try to analyze all the challenges there are and see how they can be studied holistically and integrated, to be able to provide a more sustainable system. The challenges that this type of computing poses affect aspects such as scalability, the programming models used, resilience to failures, energy management, the handling of large volume of data, etc. One of the NESUS goals is to find the way that all solutions that are proposed can be transmitted to user applications with the minimum possible redesign and reprogramming effort.

The project began last March with 29 European countries, but at present consists of 39 European countries and six countries from other continents. It now involves nearly 200 scientists, almost 40% of whom are young researchers, because

one essential goal of these Actions is to promote and create an ecosystem of scientists who can work on these matters in the European Union in the future.
This Action, which concludes in 2018, aims to produce a catalogue of open source applications that are being developed by the members of the Action and which will serve to demonstrate new ultrascale systems and take on their main challenges. In this way, anyone will be able to use these applications to test them in their systems and demonstrate their level of sustainability.

Prof. Jesus Carretero
University Carlos III of Madrid
NESUS Chair

October 2015

NESUS

Network for Sustainable Ultrascale Computing

COST Action IC1305 **2014 | 2018**

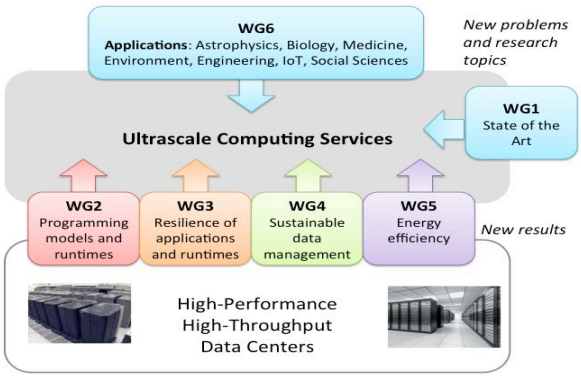
Aim

- Coordinate European efforts for proposing realistic solutions addressing major challenges of building sustainable Ultrascale Computing Systems (UCS) with a holistic approach.

To:

1. Increase EU research in the field of sustainable ultrascale computing.
2. Give coherence to the European ICT research agenda related to sustainability.
3. Build a multi-disciplinary forum for cross-fertilization of ideas for sustainable ultrascale computing.

Scientific Workplan



Topics

- WG1: New techniques to enhance sustainability holistically.
- WG2: Promoting new sustainable programming and execution models in the context of rapidly changing underlying computing architecture.
- WG3: Innovative techniques to deal with hardware and system software failures or intentional changes within the complex system environment.
- WG4: Study data management lifecycle on scalable architectures in a synergistic approach to pave the way towards sustainable UCS.
- WG5: Explore the design of metrics, analysis, frameworks and tools for putting energy awareness and energy efficiency at the next stage.
- WG6: Identify algorithms, applications, and services amenable to ultrascale systems and to study the impact of application requirements on the sustainable ultrascale system design.

Activities

- Research activities though WGs
- Set up collaborations through STSM and internships
- Training schools and PhD forum
- Meetings for WGs and MC
- Dissemination and cooperation with industry and stakeholders.
- Publications, conference organization, industry seminars, …

Information and Communication Technologies (ICT)

Participating countries: 45

EU COST countries: 33

AT, BA, BE, BG, BO, CH, CY, DE, DK, EE, EL, ES, FI, FR, HR, HU, IE, IL, IT, LT, LU, MK, MT, NL, NO, PL, PT, RO, SI, SK, SE, TR, UK

NNC countries: 6

AL, AM, MD, MO, RU, UA

Global Collaboration: 6

AU, CA, CO, IN, MX, US

,

Contact details

Chair of the Action
Jesus Carretero
jesus.carretero@uc3m.es

Website
www.nesus.eu

COST is supported by the EU RTD Framework Programme

TABLE OF CONTENTS

Second NESUS Workshop (NESUS 2015)

147 List of Authors

Parallel Processing For Gravity Inversion

Neki Frasheri

Polytechnic University of Tirana, Albania
nfrasheri@fti.edu.al

Abstract

In this paper results of recent updates of a simple algorithm for the inversion of gravity anomalies for 3D geosections in parallel computer systems are presented. A relaxation iterative principle was used updating step by step the geosection distribution of mass density. Selection of updates was done on basis of least squares error match of the update effect with the observed anomaly. Locally weighted least squares combined with the linear trend were used to obtain good inversion results for two-body geosections.

Keywords Parallel systems, Gravity inversion, Geophysics

I. Introduction

In the paper we present recent results obtained for the inversion of gravity anomalies in parallel computer systems. Inversion of geophysical anomalies is an old problem from the beginning of geophysics. For decades a multitude of manual and computer-based methods are developed. The problem is typically "ill-posed", from a mathematical point of view the traditional inversion implies mapping from a 2D array of measured ground surface values into a 3D array of voluminous physical parameters of the geosection [1][2].

Because of the physical and mathematical complexity of geophysical inversion [3], the problem is "attacked" through different methods simplifying the conditions. A typical traditionally used simplification has been reduction of dimensions through mapping a profile of 1D array of measured values into a 2D array representing a cross-section of geological structures, leading to a reduction of the volume of data and of calculations [4]. Other applied constraints include limitation to convex bodies [5], rectangular 3D prisms [6], stochastic methods [7] etc, just to mention few cases. The uncertainty character of the problem is considered by some scholars [8].

Despite decades of development, inversion remains problematic. A typical case of problematic 2D inversion of a two body geosection is given in [9] (Fig. 1).

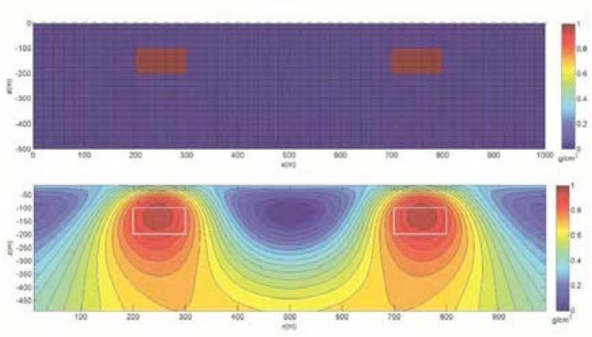


Figure 1: Typical inversion of two-body model.

Exploitation of parallel computer systems made easier the 3D inversion and different methods are experimented as in [10 - 13]. In our work we experimented a simple algorithm GIM (Geophysical Inversion and Modeling) for the 3D inversion of gravity anomalies aiming to compensate the simplicity of the algorithm with the increase of the volume of calculations made possible in parallel systems. The work started in framework of European FP7 project HP-SEE and first calculations were carried out in the HP Cluster Platform Express 7000 operated by the Institute of Information and Communication Technologies, Bulgarian Academy

of Sciences in Sofia, Bulgaria and the SGE system of the NIIFI Supercomputing Center at University of Pècs, Hungary. Recent results are obtained using the small parallel system in Faculty of Information Technology of Polytechnic University in Tirana.

II. The methodology of the work

The idea of algorithm CLEAN developed by Högbom [14] for the interpretation of radio-astronomy data was used. The algorithm is based on a simple relaxation principle - iterative approximation of geosection structures through small updates that offer the best approximation of the anomaly in each iteration. Gravity was considered as the simplest case of physical fields used in geophysics. The 3D geosection was modeled with a 3D array of elementary cuboids. In each iteration the cuboid that generates a gravity effect (elementary anomaly) which shape best approximates the observed anomaly is selected, and its mass density is modified with a predefined quantity. The effect of this increased quantity of mass density for the selected cuboid is subtracted from the observed anomaly and the whole process is repeated. In each iteration the 3D array of cuboids is scanned and for each cuboid the gravity effect in each point of the 2D array of observed anomaly is calculated (one elementary calculation for each couple cuboid - anomaly point). As result, with the supposition that dimensions of 2D and 3D arrays are N^2 and N^3, the complexity of the algorithm resulted O(N^8).

The parallelisation of the algorithm was done using both OpenMP and MPI techniques. For each iteration the scanning of the 3D array of cuboids was split in different threads and calculated in different computing cores. For the scalability of the algorithm the variations of runtime as function of the number of cores and the size of the 2D & 3D arrays. In order to reduce time delays from inter-process communication, only metadata for selected cuboids were exchanged between processes and each process had to repeat the subtraction of elementary anomalies from the observed one. Results of the work were presented in [15 - 20]. The least squares error was used as approximation metrics to evaluate how the shape of an elementary anomaly matches the observed anomaly. A modified least squares metrics was used for this purpose. In the first version of the algorithm the simple least squares error formula was used for each cuboid:

$$Err = \sum (G_{ij} - c * A_{ij} - d)^2 \qquad (1)$$

where: G_{ij} is observed anomaly value in the point (i,j) of the 2D array, A_{ij} is anomaly effect in the point (i,j) of the 2D array from the 3D cuboid with fictitious mass density of one unit. Two constants "c" and "d" were calculated to give the least error for each cuboid of the 3D geosection array, and the cuboid with the least error was selected modifying its mass density with a predefined quantity.

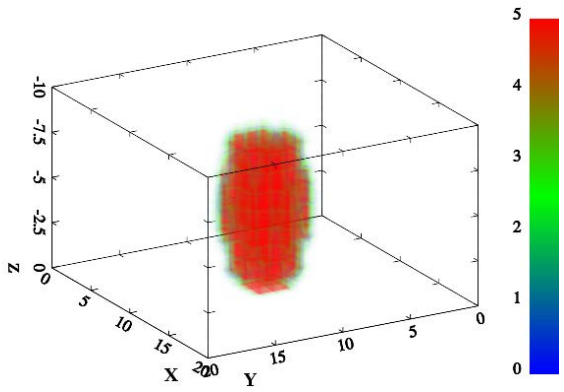


Figure 2: Inversion of single body model.

Experiments with the simple algorithm gave good results for the inversion of the gravity anomaly of a single vertical prismatic body (Fig. 2), while for multi-body geosections the results were characterized by the presence of a false depth central body instead of separate vertical prisms. The evolution of relaxation process of the algorithm was investigated joining together the central cross-section of the 3D geosection for each iteration, shaping a "carrot" that describes how the body was generated through iterations (shown in Fig. 3, see also Fig. 1):

It is visible from the Fig. 3 that the "focus" of the algorithm is the generation of a single depth body which anomaly approximates the observed one, and only remained anomaly is used for the generation of two shallow bodies that correspond with the tops of

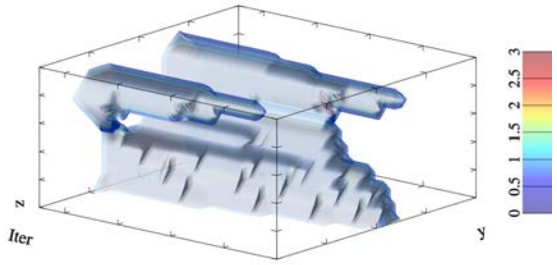


Figure 3: Evolution of inversion of the two-body model with the simple algorithm.

two original prismatic bodies (the same as in Fig. 1).

The modification of least squares error formula was done in three stages. First, the calculation of the error was limited in the part of the 2D array of observed anomaly points where the gravity effect of the cuboid is more significant. Second, weighted squares errors were used. Third, a linear local trend of the observed anomaly was considered:

$$Err = \sum W_{ij} * (G_{ij} - c * A_{ij} - d_i * i - d_j * j - e)^2 \quad (2)$$

where: W_{ij} is weights calculated on basis of A_{ij} values that $A_{ij} > L$; L is predefined threshold for definition of the calculation area; constants c, d_i, d_j and e were calculated to give the least error for each cuboid of the 3D geosection array. After the third modification of the least squares error formula, subtraction of the local linear trend, the evolution of inversion for the anomaly of two vertical prismatic bodies resulted in two realistic separate vertical "carrots".

III. Results of gravity anomaly inversion

The new algorithm was tested with synthetic anomalies generated in a 3D geosection with size 4,000x4,000x2,000 meters, and discretized with cuboids with edges 400m, 200m, 100m, 50m. The same mesh was used for the ground surface 2D array of observed anomaly. The same software was used to produce observed anomalies used as input for the inversion. Two cases were modeled, a single vertical prism 400x400x1,600m in depth -400m, and two vertical prisms of the same size in distance 1,600m. Scalability of the algorithm resulted similar for OpenMP (Fig. 4 and Fig. 5) and MPI.

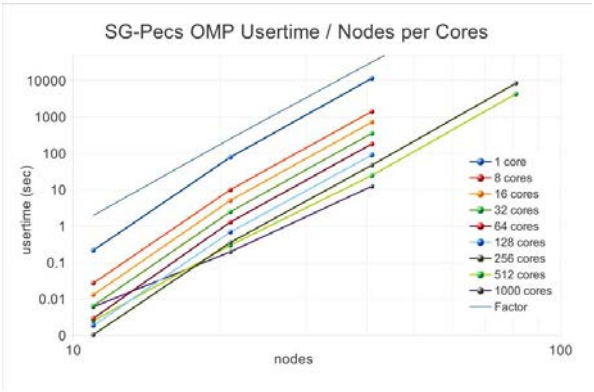


Figure 4: Scalability per model size.

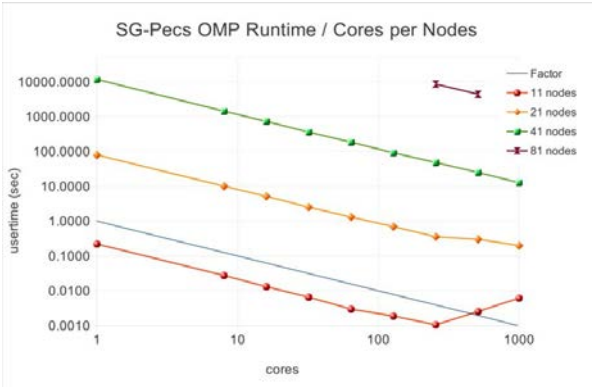


Figure 5: Scalability per number of cores.

Variation of runtime resulted the same as theoretically predicted. Runtime as function of model size (Fig. 4 is with factor O(N^8), while as function of number of cores C it is with factor O(C^{-1}). Only for small models parallelized with a great number of threads the scalability is spoiled due to the overhead of inter-process communication. In the case of the two body model, when the modified least squares formula was used, the evolution of geosection proceeded differently from the

case of Fig. 1, instead both bodies were developed in parallel as shown in Fig. 6:

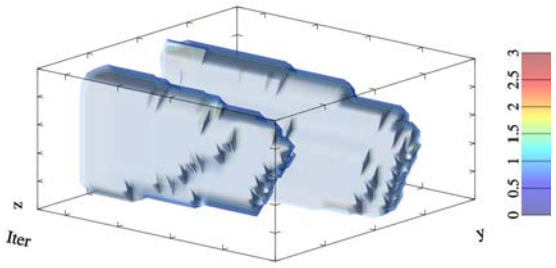


Figure 6: Inversion of two-body model with modified algorithm.

The final result of two body anomaly inversion is presented in Fig. 7. The difference in the volume of two inverted bodies is result of a small difference of the mass density of the original prisms.

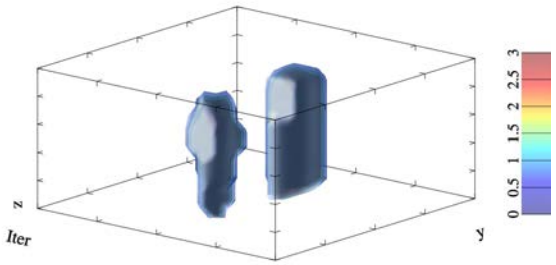


Figure 7: modified inversion of two body anomaly.

With the modified inversion algorithm there is clear contrast between the bodies and surrounding medium, and there is no in-depth bridging between two bodies as in the case of simple algorithm.

Scalability (Fig. 8) of the modified algorithm for the two body model was done in two ways. Digitized geosections with 200m and 100m sized cuboid were used, each with two cases of mass density step of 1.0 and 0.1 G/cm^3 for selected cuboids. Parallelization was done with 1 to 128 MPI processes using two schemes: a) running in a single computer node with 8 cores; and b) running in two computer node blocks interconnected with a 1Gbps Ethernet switch. The walltime experienced a jump when number of processes bypassed that of cores and increased in case of the small model while remaining almost constant for the medium-sized model. While when processes were distributed in all cores a small decrease of the trend was observed despite the low bandwidth of the interconnecting switch, an indication that the software may be run in multigrid environment.

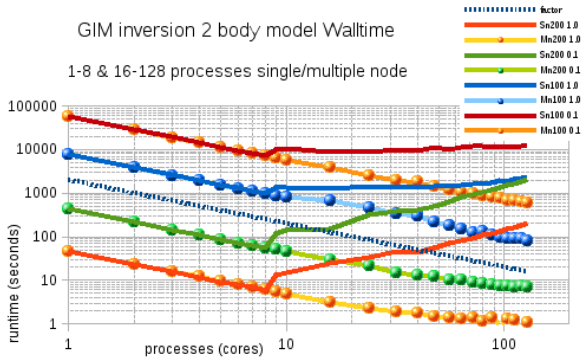


Figure 8: Scalability of two-body inversion.

IV. Conclusions

The principle of algorithm CLEAN of Högbom [14] was applied for the 3D inversion of gravity anomalies in parallel computer systems. The complexity of the algorithm resulted $O(N^8)$ for the ill-defined iterative process of mapping from a 2D array of surveyed anomaly values into a 3D array of mass density of the geosection. Parallelization was done using OpenMP and MPI. Calculations for a geosection 4,000x4,000x2,000m discretized with a step of 50m succeeded in 3 hours using 1,000 cores.

The relaxation iterative process was based on selection of the best cuboid which effect matched better the observed anomaly, using least squares error method. Experiments resulted successful for single body models while gave wrong three-lobe structures for two body models. In order to resolve this problem the best cuboid was selected calculating locally the weighted least squares error only for points around the cuboid,

subtracting the linear trend and using the shape of cuboid's anomaly as weights. This modification made possible to obtain good inversion approximation of the two-body model.

Complexity of algorithm is polynomial of 8^{-th} order that requires ultra scale parallel computer or multi-grid systems to obtain results for models with metric necessary for complex multi-method engineering works - the observed runtimes are for the inversion of gravity that is the simplest case (scalar field which scattering does not depend on environment heterogeneity) in the complex of geophysical methods. Inversion of anomalies for magnetic and electrical fields would require calculation of vectorial fields which scattering is dependent on environment heterogeneity and anisotropy.

References

[1] J. Hadamard, *Sur les Problemes aux Derivees Partielles et leur Signification Physique*, Bull Princeton University, 1902.

[2] W. Lowrie, *Fundamentals of Geophysics*, Cambridge University Press, 2007.

[3] M. Sen and P. Stoffa, *Global Optimization Methods in Geophysical Inversion*, Elsevier Science B.V., 1995.

[4] Z. Xiaobing, "Analytic solution of the gravity anomaly of irregular 2D masses with density contrast varying as a 2D polynomial function", *Geophysics*, v. 75; no. 2; p. I11-I19, March-April, 2010.

[5] J. Silva, W. E. Medeiros and V. C. F Barbosa, "Gravity inversion using convexity constraint", *Geophysics*, v. 65; no. 1; p. 102-112, January-February, 2000.

[6] P. Shamsipour, M. Chouteau, D. Marcotte and P. Keating, "3D stochastic inversion of borehole and surface gravity data using geostatistics", *EGM 2010 International Workshop, Adding new value to Electromagnetic, Gravity and Magnetic Methods for Exploration*, Capri, Italy, April 11-14, 2010.

[7] Z. Xiaobing, "3D vector gravity potential and line integrals for the gravity anomaly of a rectangular prism with 3D variable density contrast", *Geophysics*, v. 74; no. 6; p. I43-I53, November-December, 2009.

[8] F. J. Wellmann, F. G. Horowitz, E. Schill and K. Regenauer-Lieb, "Towards incorporating uncertainty of structural data in 3D geological inversion", *Elsevier Tectonophysics TECTO-124902*, 2010.

[9] M. S. Zhdanov, G. A. Wilson and L. Xiaojun, "3D imaging of subsurface structures using migration and regularized focusing inversion of gravity and gravity gradiometry data", in R. J. L. Lane (Ed), *Airborne Gravity - Abstracts from the ASEG-PESA Airborne Gravity Workshop*, Published jointly by Geoscience Australia and the Geological Survey of New South Wales, Geoscience Australia Record 2010/23, 2010.

[10] P. Rickwood and M. Sambridge, "Efficient parallel inversion using the neighborhood agorithm", *Geochemistry Geophysics Geosystems Electronic Journal of the Earth Sciences*, Volume 7, Number 11, 1 November, 2006.

[11] M. H. Loke and P. Wilkinson, "Rapid parallel computation of optimized arrays for electrical imaging surveys", *Near Surface 2009 - 15th European Meeting of Environmental and Engineering Geophysics*, Dublin, Ireland, 7-9 September, 2009.

[12] H. Zuzhi, H. Zhanxiang, W. Yongtao and S. Weibin, "Constrained inversion of magnetotelluric data using parallel simulated annealing algorithm and its application". *SEG Denver Annual Meeting - SEG Expanded Abstracts*, Vol. 29, EM P4 Modeling and Inversion, 2010.

[13] G. Wilson, M. Ĉuma and M. S. Zhdanov, "Massively parallel 3D inversion of gravity and gravity gradiometry data", *PREVIEW - The Magazine of the Australian Society of Exploration Geophysicists*, June, 2011.

[14] J. A. Högbom, "Aperture synthesis with a non-regular distribution of interferometer baselines", *Astr. Astrophys. Suppl.*, 15, 417, 1974.

[15] N. Frasheri and S. Bushati, "An algorithm for gravity anomaly inversion in HPC", *SCPE: Scalable*

Computing: Practice and Experience, vol. 13 no. 2, pp. 51-60, 2012.

[16] N.Frasheri and B. Cico, "Convergence of gravity inversion using OpenMP", *Information Technologies IT'12*, Zabljak - Montenegro, Feb 27 - March 02, 2012.

[17] N. Frasheri and B. Cico, "Analysis of the convergence of iterative gravity inversion in parallel systems", in L. Kocarev (Ed.), *Springer Advances in Intelligent and Soft Computing 150: ICT Innovations 2011*, Springer-Verlag, 2012.

[18] N. Frasheri and B. Cico, "Scalability of geophysical inversion with OpenMP and MPI in parallel processing", in S. Markovski and M. Gusev (Eds.), *Springer Advances in Intelligent Systems and Computing 207: ICT Innovations 2012: Secure and Intelligent Systems*, Springer-Verlag, 2013.

[19] N. Frasheri, S. Bushati and A. Frasheri, "A parallel processing algorithm for gravity inversion", *European Geosciences Union General Assembly EGU'2013*, Vienna 7-12 April, 2013.

[20] N. Frasheri, S. Bushati, A. Frasheri, and B. Cico, "Some results for 3D gravity inversion in parallel systems", *7th Congress of the BGS - Balkan Geophysical Society*, Tirana - Albania, 7-10 October, 2013.

Solution of Bi-objective Competitive Facility Location Problem Using Parallel Stochastic Search Algorithm

Algirdas Lančinskas†, Pilar Martínez Ortigosa*, Julius Žilinskas†

†Vilnius University, Lithuania
algirdas.lancinskas@mii.vu.lt
julius.zilinskas@mii.vu.lt

*University of Almeria, Spain
ortigosa@ual.es

Keywords Parallel Computing, Multi-Objective Optimization, Stochastic Search, Facility Location.

I. Introduction

The Facility Location (FL) deals the optimal placement of the facilities providing goods or services in a certain geographical area with respect to maximize the utility and/or minimize an undesirable effect of the facility being located. There is a variety of FL models proposed which varies on their ingredients such as location space, attractiveness of the facilities, or customers' behavior when choosing the most attractive facility [1, 2, 4, 5]. A lot of attention is paid for the Competitive Facility Location Problems (CFLPs) in which determination of the optimal location for the new facilities involves consideration of their possible competition for the market share with the preexisting facilities.

Real-world CFLPs usually require to simultaneously consider two or several objectives when locating the new facilities; e.g. maximize the market share of the new facilities while minimizing costs for their establishment or maintenance of the facility; minimize distance between facilities and customers in accordance with requirements for the minimal distance to urban areas (actual for semi-obnoxious facilities).

Our research is focused on the Competitive Facility Location Problem for Firm Expansion (CFLP/FE) where a firm already in the market is planning to establish a set of new facilities in order to increase its market share.

II. CFLP for Firm Expansion

Consider an expanding firm F_A having n_A preexisting facilities and its competitor – the firm F_B having n_B preexisting facilities – both servicing a discrete set I of *demand points* in a certain geographical area. The firm F_A is expected to locate a set X of n_X new facilities with respect to maximize the market share of the new facilities taking into account the competition with the facilities owned by F_B. Despite the attraction of new customers from the competitor F_B, the newly established facilities can also attract customers from the facilities already owned by the expanding firm F_A thus giving rise of the *effect of cannibalism*. Therefore the firm F_A faces a bi-objective optimization problem with the following objectives: (f_1) to maximize the market share of the facilities being located and (f_2) to minimize the loss of market share of the preexisting facilities of F_A (the effect of cannibalism).

Due to conflicting objectives usually it is impossible to find a single solution which would be the best by both objectives, but rather a set of compromising (non-dominated) solutions, called Pareto set; the corresponding set of the objective functions' values is called Pareto front. Determination of the exact Pareto front usually is a hard and time consuming task. On the other hand solution of practical CFLPs usually does not require to find the exact Pareto front, but rather its approximation by a set of non-dominated solutions.

III. Parallel Multi-Objective Stochastic Search

Multi-Objective Stochastic Search (MOSS) is a random search algorithm suitable for approximation of the Pareto front of a multi-objective optimization problem. MOSS is derived from its precursor Multi-Objective Single Agent Stochastic Search (MOSASS) algorithm proposed in [3].

The algorithm begins with an initial archive $\mathbb{A}$ of solutions which are non-dominated among themselves. The new solution $\mathbf{x}'$ is generated by applying slight modifications to the solution $\mathbf{x}$ randomly sampled from $\mathbb{A}$, where the strength of the modification depends on the repetitive successful and failed iterations; see [3] for details of the generation of the new solution. If the newly generated solution $\mathbf{x}'$ is not dominated by any one in the archive $\mathbb{A}$, then $\mathbb{A}$ is updated by including $\mathbf{x}'$ and removing all solution which are dominated by $\mathbf{x}'$, and the algorithm goes to the next iteration. Otherwise, if $\mathbf{x}'$ is dominated in $\mathbb{A}$, then a symmetric (in relation with $\mathbf{x}$) solution $\mathbf{x}''$ is evaluated in the same way as $\mathbf{x}'$. If the archive is updated either by $\mathbf{x}'$ or $\mathbf{x}''$, then iteration is assumed to be successful; otherwise, the iteration is assumed to be failed.

The main computational effort of the algorithm usually is devoted to the evaluation of objective function values. The evaluations of the objective values of different solutions can be considered as independent tasks thus giving availability to distribute the computational work among different processors. In such a distribution of tasks the information about all non-dominated solutions found so far (the archive $\mathbb{A}$) as well as values of other parameters of the algorithm must be accessed by all processors. Moreover if one of the processors is updating a parameter or the archive, access to it is blocked for any other processor in order to keep memory and data consistency.

Two parallel algorithms ParMOSS/OMP and ParMOSS/MPI suitable for shared- and distributed-memory parallel computing systems, respectively, has been developed under considerations above.

The ParMOSS/OMP algorithm begins with the initialization of the parameters of the algorithm as well as the data and parameters of the optimization problem to be solved. This part of the algorithm is a single processor – the master. Further each of the slaves randomly selects an individuals from the archive $\mathbb{A}$ and evaluates its objective values as well as the dominance relation in $\mathbb{A}$ (as it is described above). If any of processors is accessing the archive or any other parameter of the algorithm, the access to that parameter or the archive is blocked for all other processors.

In distributed-memory computing systems information about solutions in $\mathbb{A}$ and values of algorithm parameters must be transfered by passing messages using Message Passing Interface (MPI). In order to guarantee consistent communication between processors, one of them is devoted for the management of the communication and overall process of the algorithm. Thus the parallel version of MOSS algorithm ParMOSS/MPI for distributed-memory parallel computing systems is developed following the master-slave strategy.

The master processor selects a random solution $\mathbf{x}_i$ from $\mathbb{A}$, generates a pair of new solutions $(\mathbf{x}'_i, \mathbf{x}''_i)$ (as it is described above), and sends it to the i-th processor (the slave) with the request to evaluate the first solution $\mathbf{x}'_i$. Here i varies from 1 to the number processors p thus ensuring that the pair will be generated for each processor. After all slaves are equipped by a pair of solutions, the master proceeds to the main loop and waits for the response from any of the slaves with an evaluated solution. Although all slaves are requested to evaluate $\mathbf{x}'_i$, some of them can also be requested to evaluate $\mathbf{x}''_i$ in the later stage of the algorithm. In general the master processor proceeds depending on whether evaluation of $\mathbf{x}'_i$ or $\mathbf{x}''_i$ is received and the fitness of the received solution with respect to solutions in the archive $\mathbb{A}$.

IV. Numerical Experiments

The developed parallel algorithms ParMOSS/OMP and ParMOSS/MPI have been experimentally investigated by solving different instances of CFLP/FE: 5000, 1000, 500, and 100 demand points for ParMOSS/OMP; 5000 and 1000 demand points – for ParMOSS/MPI. The Pareto front of a single instance has been approximated by 25000 function evaluations. The average duration of a single approximation by sequential MOSS was around 728 seconds using 5000 de-

mand points, around 145 seconds – using 1000 demand points, around 73 – using 500 demand points, and around 14 seconds – using 100 demand points.

The obtained results showed that the shared-memory algorithm ParMOSS/OMP has almost linear speed-up on up to 16 shared memory processors for all instances of the problem: 5000, 1000, 500, and 100 demand points; further reduction of the number of demand points is not reasonable in practical CFLPs.

Similar experiment has been performed for the distributed-memory algorithm ParMOSS/MPI. The Pareto front of CFLP/FE with 1000 demand points has been approximated using 2, 4, 8, and 16 processors. Results of the experimental investigation showed that speed-up of ParMOSS/MPI increases linearly with the increment of the number of processors. The speed-up of ParMOSS/MPI is lower than speed-up of ParMOSS/OMP exactly by one independent on the number of processors due to an idle time of the master processor which has no computational work. These results show that the shared-memory algorithm has notable advantage against the distributed-memory one. On the other hand the shared-memory computing systems have hardware limitations in the sense of number of shared-memory processors, whereas the distributed-memory algorithm can be executed on a significantly larger number of processors.

The performance of ParMOSS/MPI has been also investigated using from 32 to 192 processors. Results of the investigation showed that the approximation of the Pareto front of the problem with 5000 demand points has been performed with almost linear speed-up of the algorithm – the speed-up on 192 processors was around 186. The approximation of the Pareto front of the problem with 1000 demand points has been performed with notably lower speed-up, comparing with previous instance – the speed-up on 192 processors is around 155 which corresponds to 80% of effectiveness of the processors. On the other hand the speed-up of the algorithm on 96 processors is around 89 which corresponds to 93% of effectiveness of the processors when performing the computations; further increment of the number of processors is not reasonable for approximation of the Pareto front of a real-world CFLP as the approximation on 128 processors has been performed within 2 seconds.

Acknowledgment

The work has been partially supported by EU under the COST Action IC1305 “Network for Sustainable Ultrascale Computing (NESUS)”.

References

[1] R. Z. Farahani, S. Rezapour, T. Drezner, and S. Fallah. Competitive supply chain network design: An overview of classifications, models, solution techniques and applications. *Omega*, 45(0):92–118, 2014.

[2] T.L. Friesz, T. Miller, and R.L. Tobin. Competitive networks facility location models: a survey. *Papers in Regional Science*, 65:47–57, 1998.

[3] A. Lančinskas, P. M. Ortigosa, and J. Žilinskas. Multi-objective single agent stochastic search in non-dominated sorting genetic algorithm. *Nonlinear Analysis: Modelling and Control*, 18(3):293–313, 2013.

[4] Frank P. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129(3):461–470, 2001.

[5] C. S. ReVelle, H.A. Eiselt, and M .S. Daskin. A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research*, 184(3):817–848, 2008.

A Scheduler for Cloud Bursting of Map-Intensive Traffic Analysis Jobs

Ricardo Morla†, Pedro Gonçalves†, Jorge Barbosa‡

†INESC TEC and Faculty of Engineering, University of Porto
‡LIACC and Faculty of Engineering, University of Porto
Porto, Portugal
ricardo.morla@fe.up.pt

Abstract

Network traffic analysis is important for detecting intrusions and managing application traffic. Low cost, cluster-based traffic analysis solutions have been proposed for bulk processing of large blocks of traffic captures, scaling out the processing capability of a single network analysis node. Because of traffic intensity variations owing to the natural burstiness of network traffic, a network analysis cluster may have to be severely over-dimensioned to support 24/7 continuous packet block capture and processing. Bursting the analysis of some of the packet blocks to the cloud may attenuate the need for over-dimensioning the local cluster. In fact, existing solutions for network traffic analysis in the cloud are already providing the traditional benefits of cloud-based services to network traffic analysts and opening the door to cloud-based Elastic MapReduce-style traffic analysis solutions. In this paper we propose a scheduler of packet block network analysis jobs that chooses between sending the job to a local cluster versus sending it to a network analysis service on the cloud. We focus on map-intensive jobs such as string matching-based virus and malware detection. We present an architecture for an Hadoop-based network analysis solution including our scheduler, report on using this approach in a small cluster, and show scheduling performance results obtained through simulation. We achieve up to more than 50% reduction on the amount of network traffic we need to burst out using our scheduler compared to simple traffic threshold scheduler and full resource availability scheduler. Finally we discuss scaling out issues for our network analysis solution.

Keywords Packet Network Traffic Analysis, Hadoop, Cloud Bursting

I. Introduction

Many companies invest in their private IT data centers to meet most of their needs. However, these private data centers might not cope with workload peaks, leading to delays and lower throughput. Using extra computing power to handle the unpredictable and infrequent peak data workloads is expensive and inefficient, since a portion of the resources will be inactive most of the time. Migrating the whole application to a cloud infrastructure, though possibly cheaper than investing in the private data center because servers are only rented when needed, is still expensive and could compromise private and important data. A hybrid model offers the best solution to address the needs for elasticity when peak workloads appear, while providing local privacy if needed. In this model, the local IT data center of an enterprise is optimized to fulfill the vast majority of its needs, resorting to additional resources in the cloud when the local data center resources become scarce [1]. This technique is called Cloud Bursting and enables an enterprise to scale out their local infrastructure by bursting their applications to a third-party public cloud seamlessly, if the need arises [2].

Cloud Bursting could be of use to network managers and security experts. In fact, their need for more computational power that can perform more

complex network intrusion detection and application traffic management is hand in hand with the variability of network traffic and of traffic analysis workload. Big data style analysis software and services using the open source YARN and Hadoop platform[1] are available both in the research community [3, 4] and commercially[2]. Although [3] in particular provides a range of network analysis jobs from low level packet and flow statistics to intrusion detection, their performance evaluation focuses mainly on the throughput of the analysis system with different file sizes. In particular, Map and Reduce run time distributions that could be helpful for large scale evaluation have not been characterized.

Three types of network traffic analysis are typical [5]: 1) real-time analysis, where continuous streams of data are analyzed and processed as they arrive; 2) batched analysis, where data is aggregated in batches and analyzed periodically; and 3) forensics analysis are analysis that only occur when special events are triggered, for example, when a major intrusion has been detected and requires detailed analysis of logged traffic. This work focuses on 24/7 continuous batch analyses of consecutive captures of network packet traffic and on the natural variability that characterizes such traffic. Because intrusion detection results for each batch must be delivered as soon as possible to allow for early detection, traffic variability in a continuous batch processing means either over-dimensioning the computing cluster or waiting longer to get detection results. Our proposal is to use Cloud Bursting to burst some jobs and achieve lower waiting times. We present an integrated solution in section II that we have implemented in our networking laboratory. An important part of this solution is the scheduler that decides whether to burst a job based on the size of the file that needs to be analyzed and on the resource usage in the cluster. We present our Map-intensive job, job model, scheduler rationale, and cluster capacity estimation in section III. In section IV we show results of running our network analysis job in a small Hadoop cluster. The goal here is to provide an example of how the scheduler works and to obtain an empirical distribution for the Map run time to be used in our simulator in section V. In section V we describe the simulator, provide a distribution for link load, and characterize run times of single and continuously arriving jobs. In section VI we show job delay and burst count for our scheduler and compare it with baseline schedulers. We conclude with scale out analysis in section VII, related work analysis in section VIII, and final remarks in section IX.

[1] http://hadoop.apache.org
[2] http://www.pravail.com

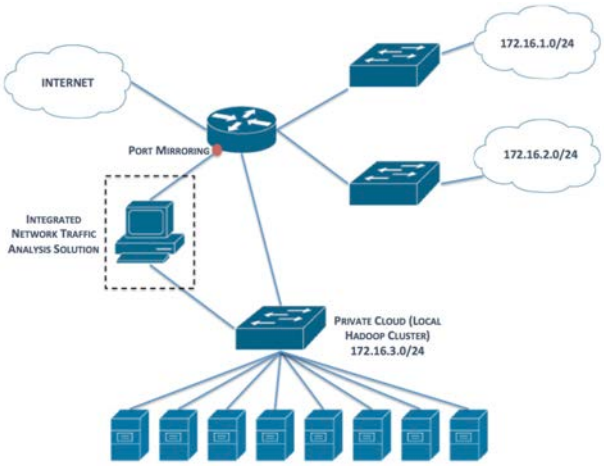


Figure 1: Our network topology with port mirroring, PCAP traffic file capture, cluster for PCAP file processing, and our control node (INTAS).

II. Cloud Bursting Architecture

Figure 1 shows the topology of our network. Outbound traffic from the 172.16.1.0/24 and 172.16.2.0/24 networks is captured via port mirroring and the tcpdump application in our Integrated Network Traffic Analysis Solution (INTAS). INTAS will decide whether to send each PCAP file created by tcpdump to our local private cloud or to burst the file to a public cloud, reachable through the Internet. In this paper we assume the public cloud is provisioned such that an adequate job performance is achieved.

Figure 2 shows the architecture and modules of our solution. The Network Traffic Gatherer module uses tcpdump to capture batches of network traffic from a pre-specified network interface and generate PCAP files. Once ready, the PCAP files are copied to a folder

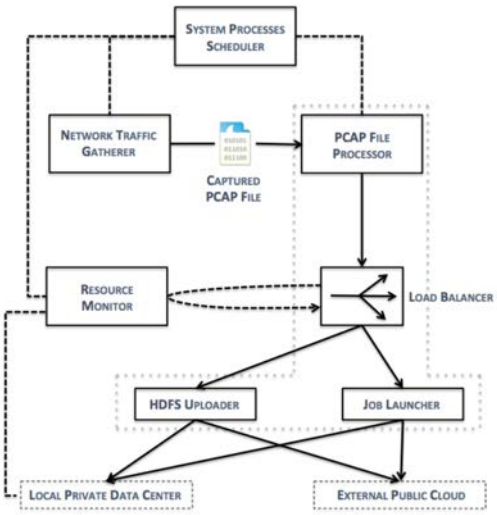


Figure 2: The architecture of INTAS.

where the PCAP File Processor module determines whether or not the PCAP file is new before delivering it to the Scheduler/Load Balancer. Using the cluster resource utilization information provided by the Resource Monitor, the Scheduler decides whether to launch the job locally or to burst it. In either case, the PCAP file is first uploaded to HDFS on the destination cluster through the HDFS Uploader module. After the upload is complete, the Job Launcher connects to the Hadoop cluster to launch the map-intensive network analysis job.

III. Scheduler

III.1 Map-Intensive Network Analysis Jobs

Map-intensive network analysis jobs such as traffic classification and virus or malware detection using string matching can have the following two-phase design. The Map phase checks packet payload for the presence of signature keys. This outputs a list of which applications or viruses were identified on which packets. The reduce phase merges these results and writes them to the job's output file.

The Map phase of this kind of network analysis job is computationally expensive. Simply put, all possible string sequences in the packet's transport layer payload must be compared to the signature keys of the different viruses, malware, or specific applications we want to find. For a signature key with b_{key} bytes and a string comparison factor $k^{key}[second/byte]$, the total processing time of a packet is $t_{packet} = k^{key}(b_{packet} - b_{key})$, with $b_{packet} \geq b_{key}$. For small keys and relatively large packets, we can have a close approximation of t_{packet} by using its upper bound $k\,b_{packet}$, where k is the sum of all string comparison factors and which can also be expressed as $k = nk_{avg}$ with n keys and k_{avg} average string comparison factor. This upper bound is proportional to the size of the packet. We find an estimate of k by benchmarking the target cluster as reported later in the paper.

Under the Map-Reduce paradigm, each traffic capture file is split into a number of data blocks and each block is processed by a Map process. Although the exact number of packets in each block can change according to the nature of the network traffic, the size in bytes of almost all data blocks is pre-configured and independent of the nature of the traffic. Our estimate for the Map processing time of a block with b_{block} bytes is $t_{block} = k\,b_{block}$. We assume the switching time between processing of packets is much smaller than the actual processing.

The Reduce phase of this kind of network analysis jobs is much less computationally demanding than the Map phase. We assume the Reduce processing time to be zero.

III.2 Job Completion Time Estimate

The number of map tasks that a network analysis job launches depends on the size of the input traffic capture file and on the Map-Reduce data block size. A block size of 128 MB is typical, which for a slightly under 900 MB traffic capture file size b_{file} would yield 7 data blocks on which 7 Map processes would need to run. More generically, we compute $m = b_{file}/b_{block}$ and estimate the number of Map processes to be $M = \lceil m \rceil$. We disable speculative execution to improve this estimate.

The container is the unit of resource allocation in Hadoop YARN. Typically, two containers run per host.

The mapping to containers sets the limit to the number of Maps that a network analysis task can run simultaneously. One Map process runs on a container. Each Map-Reduce job is managed by an ApplicationMaster that requires an additional host throughout the complete lifetime of the job. Although Reduces also require containers, their impact on resource usage in this kind of network analysis job is much smaller than that of Maps and we do not consider them.

For a YARN cluster with H hosts, the number of Maps that can run simultaneously is $M_s = 2(H-1) - 1$. Comparing M_s and M takes us to the concept of Map waves. Each wave of M_s Maps will run simultaneously for approximately t_{block}, after which a new wave of M_s (or fewer) Maps will start. This repeats until all M Maps have run. We compute $n = M/M_s$ and estimate the number of waves as $N = \lceil n \rceil$. Our estimate for the job completion time is $t_{Mjob} = (N-1) * t_{block} + k(m - \lfloor m \rfloor)b_{block}$ if the last wave has a single Map task (in which case $M - NM_s = 1$), and $t_{Mjob} = N * t_{block}$ otherwise, which is the more frequent case.

In addition to T_{Mjob}, the scheduler needs have an estimate of the time it takes the system to upload the captured traffic file from the capture node to HDFS. We use a simple estimate $T_{upload} = b_{file}/r$ with r in $[bytes/second]$. The total job completion time estimate is $T_{job} = T_{Mjob} + T_{upload}$.

At any moment in time after the job has been scheduled, the estimate for the job completion time $T_{job}^{remaining}$ can be updated as follows:

- $T_{Mjob} + (b_{file} - b_{file}^{uploaded})/r$, if the upload not finished yet. $b_{file}^{uploaded}$ is the number of bytes that have been uploaded so far.
- $T_{Mjob} - N^{done}\, t_{block} - t_{wave}$, if the upload finished. N^{done} is the number of completed waves and t_{wave} is how long the current wave has been running.

This estimate is valid for a single job running in the cluster.

III.3 Scheduling Concurrent Jobs

An incoming traffic analysis job is scheduled regardless of the size of the capture file when no job is running on the cluster. When traffic peaks, it is possible that a new chunk of traffic is captured and ready for analysis before the traffic analysis job of the previous chunk is over. In this case where a job is running in the local cluster when a new job arrives, the scheduler will have to decide whether to send the incoming job to the local cluster or to the cloud.

Our baseline scheduler verifies if there are enough containers for the job in the local cluster. If there are, the job is ran locally; otherwise the job is sent to the public cloud. We assume the connection to the public cloud and the public cloud cluster itself are provisioned such that an adequate response time is achieved. We do not explore public cloud performance in this paper.

The approach of the baseline scheduler is over simplifying and can send more jobs to the public cloud than needed: 1) the time it takes to upload the capture file to the local HDFS can be enough for the current job to complete and the incoming job to be processed locally; 2) the last wave of a job may use fewer resources than the previous waves, which can be used by the incoming job. Our proposed scheduler takes advantage of the job completion time estimate and wave model presented in the previous section to try to reduce the number of uploads to the public cloud and reducing the impact on job completion time.

Our proposed scheduler schedules incoming candidate job locally if:

- **S0.** Local cluster is empty or enough containers are available to run the incoming job.
- **S1.** Local cluster has single current job and current job finishes before or up to Th_1 seconds after incoming job upload time to local cluster HDFS. For this we use $T_{job}^{remaining}$ of the current job and T_{upload} of the incoming candidate job. Th_1 provides some tolerance to the decision and can be chosen to be a few percent of the wave duration.
- **S2.** Local cluster has single current job and: 1) the next to the last wave of current job finishes before or up to Th_1 seconds after incoming job upload time to local cluster HDFS and 2) the number of containers of the last wave of the candidate incoming job is smaller than or equal to the number of

containers not used by last wave of the current job.

III.4 Benchmarking and Estimating Capacity

Our scheduler requires an estimate of the block execution time t_{block} and of the upload to HDFS rate r. Our approach for estimating these values is to run the network analysis job we are interested in benchmarking on an sample data set prior to running the job on the target traffic capture files. This provides samples for measure Map completion time and file upload time, which can be averaged or maxed and used as estimate for t_{block} and r.

Estimating the capacity of a cluster for this kind of network analysis is important for: 1) defining the traffic throughput that a given cluster can support for analysis and 2) specifying how many and what kind of nodes there should be in the cluster for a given traffic throughput that needs to be analyzed. We estimate the maximum throughput of a traffic capture that a cluster can support as $T_{hput} = M_s * b_{block}/t_{block} = (2(H-1)-1)/k$. Using this estimate, a cluster with 10 nodes, 128 MB block size, and $t_{block} = 5min\,(k = 2.23\,\mu s/byte)$ would support approximately 60 Mbit/s traffic captures, which yields a maximum 10 minute traffic capture file of 4.5 GB.

IV. String Matching Job in Small Physical Clusters

We ran a Map-intensive string matching job with 200 signature keys on H=5 node low-end YARN clusters. To more easily launch our Hadoop 2.3.0 cluster we use a private cloud based on Openstack IceHouse[3] and the Sahara Openstack module[4] that provisions data-intensive application clusters like YARN. The bare-metal operating system is Ubuntu 14.04. Each bare metal runs a single virtual machine YARN node.

Due to time and lab resource constraints we ran the two experiments of this section in different hardware. The scheduling experiment was run on a heterogeneous cluster with the following bare-metals: Intel Core i7-3770 3.40GHz, two Intel Core i7 950 3.07GHz, Intel E5504 2.00GHz. The map runtime experiment was run on a homogeneous cluster in which the bare-metal CPUs are older, 2008 Intel Core 2 Quad Q9300 running at 2.50GHz. CPU names are as reported by /proc/cpuinfo.

We captured two PCAP files locally on a 100 Mbit/s link for 10 minutes each. One file (A) is 1.22 GB and the other (B) is 2.01 GB, corresponding to 17 Mbit/s and 26 Mbit/s of traffic going out of our local network. One file has 10 128 MB data blocks and the other 16.

IV.1 Scheduling

For our scheduling experiments we replayed the two PCAP files A and B into our system every 10 minutes according to this sequence: "ABBBBA-ABABAA-AAAAAA". File A has the cluster running slightly below capacity whereas file B has the cluster running above capacity. This means that during the first phase "ABBBBA" the cluster will be running well above capacity, during the second phase "ABABAA" the cluster will be just slightly over capacity, and during the third phase "AAAAAA" the cluster will be running below capacity.

Figures 3 and 4 show an example of processing the first phase "ABBBBA". To produce analysis results as soon as possible, the Maps for job 6A should start immediately after the upload is completed, i.e. on the right side of the small box near 6A. Notice that without scheduler (figure 3, job id 54), the Maps have to wait until the previous job completes – which is a considerable amount of time. In figure 4 with the scheduler and by bursting job 5B, job 6A can start immediately after upload.

Figure 5 shows how the job sequence was burst or delayed using the three different scheduling approaches: no scheduler, simple, and advanced. Because we want to have an idea of the impact of the scheduler on each job individually, we can compare the run time of a job T_{job} with the sum of the job's Map run times divided by the number M_s of Maps that can run simultaneously. We define this metric as $D = T_{job}/T_{ideal} - 1$ with $T_{ideal} = 1/M_s \sum_i t^i_{block}$ and where t^i_{block} is the run time of the i^{th} Map of the job. The results confirm our intuition that by bursting some jobs the job delay can

[3]https://www.openstack.org/software/icehouse/
[4]http://docs.openstack.org/developer/sahara/

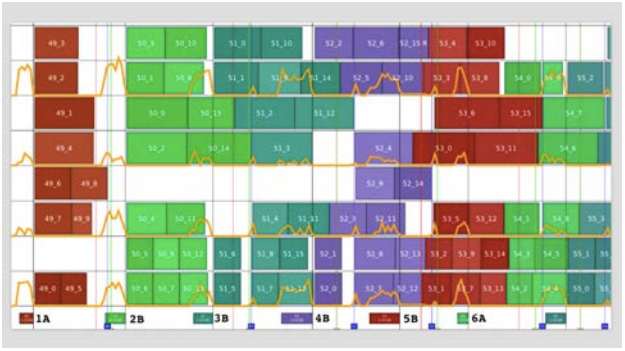


Figure 3: Example of processing phase "ABBBBA" without bursting. Horizontal lines are the placeholder for container utilization through time. "X_Y" labeled boxes represent Map run Y of job X. Smaller boxes with file sizes and e.g. "1A" legend represent PCAP file upload to HDFS. Plot lines on each pair of containers represent traffic in and out of the network interface in the bare metal where the containers run.

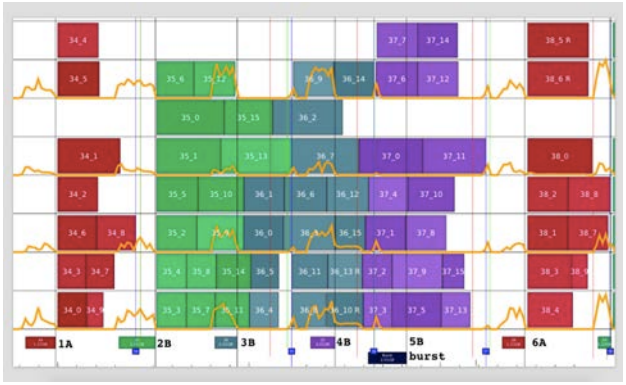


Figure 4: Example of processing phase "ABBBBA" with bursting.

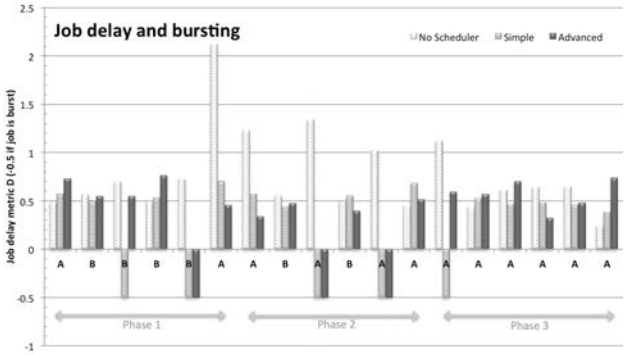


Figure 5: Job delay and bursting for different scheduling options on the example sequence of network analysis jobs.

be reduced. The advanced scheduler only burst 3 jobs while not causing distinguishable delays compared to the simple scheduler that burst 5 jobs. Interestingly, the results for job delay without scheduler suggest some spillover from phase 1 to phase 2 that seems to make job delay in phase 2 larger than in phase 1 when in fact phase 1 is more demanding than phase 2. This could also be due to the inherent randomness of this system. This is one of the reasons why we now go to simulation and better understand the performance of the bursting approaches.

IV.2 Map run time distribution

Figure 6 shows the distribution of the run time and throughput of 762 128MB-block Maps that were used to analyze files A and B. We ran the analysis job on file A 48 times and on file B 24 times. The Map run time 50^{th} percentile is 6 minutes and 3 seconds. We can use this value to estimate the capacity of our cluster as follows: $k = 363s/128MB = 2.70\mu s/byte$ and thus $T_{hput} = 20.7Mbit/s$. With 10 minute captures this yields a file size of 1.44 GB. We use this value in our scheduler. A more conservative approach would choose the 90^{th} percentile and a resource utilization factor of 2/3 yielding $k = 494s/128MB/\frac{2}{3} = 5.52\mu s/byte$ and $T_{hput} = 10.64Mbit/s$.

V. Simulator

V.1 Design

We built an event-driven simulator in python to simulate map-intensive traffic analysis jobs. The simulator keeps a list of jobs to be scheduled *job_launch_schedule*, including their randomly generated file sizes and the times at which they are be launched in the cluster. This corresponds at the times at which a PCAP file is ready to be processed in the YARN cluster. For this paper it's every 10 minutes.

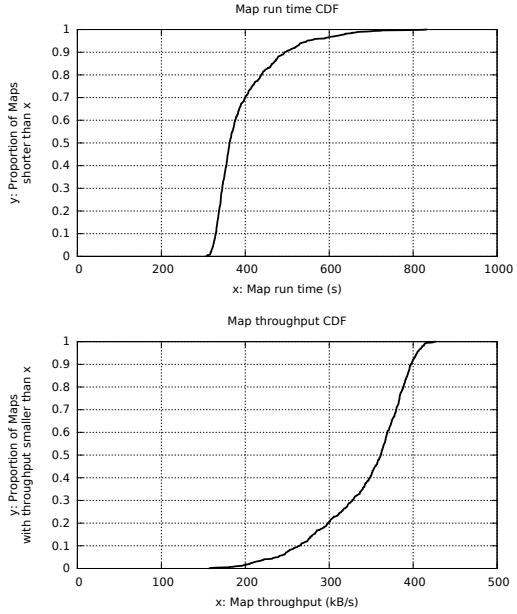


Figure 6: Empirical CDFs of run time and throughput of 762 Maps of our network analysis job.

This list is populated before the simulation starts and jobs are popped out and launched as the simulation progresses. YARN can be configured to allow a maximum of simultaneously running applications in the cluster. To account for this, a job that is popped out of the *job_launch_schedule* list will go to a *job_waiting* list. If the number of currently running applications is below the maximum, the oldest job is removed from the *job_waiting* list and included in the *job_running* list. Jobs in the *job_runnig* list will compete for available cluster containers in round-robin. Only the job at the top of the list will get a container to run a Map. For every container a running job gets to run a Map it will be pushed to the back of the list. Container release events are scheduled according to the randomly generated Map run time distribution and processed by the simulation together with job launch events. The simulation finishes when there are no more events to process or maximum simulation time is reached. Block size is 128MB.

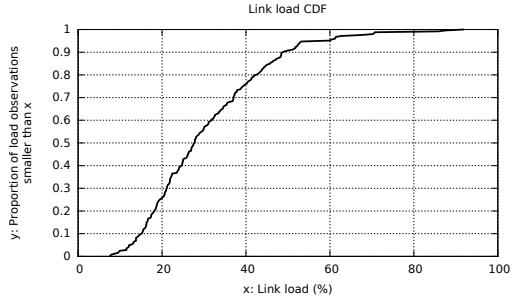


Figure 7: 10 Gbps link load empirical CDF.

V.2 Workload Distributions

Synthetic workload generation requires two components. The first is a distribution for the Map run time, for which we use the Map run time empirical distribution obtained from our 5 node clusters and shown in figure 6. The second is the PCAP file size distribution that determines the number of data blocks and Map tasks that the job needs to process. As this is directly related to the amount of traffic through a packet network, we build on publicly available data from the CAIDA Center for Applied Internet Data Analysis in San Diego, CA[5]. We use every month, hour-long average bitrate on both directions of their 10 Gbps San Diego links to Chicago and San Jose to build a 247 point empirical link load distribution. We show the CDF of this distribution in figure 7. Average link load is 31%.

Figure 8 shows the Q-Q plots for our synthetic data and the empirical distributions. Notice how close the Q-Q points are to the $y = x$ line. Because we have fewer observations in the high link load region the points there are not as close to the $y = x$ line as the other points.

Simulation results in the rest of the paper are obtained by running 10^4 jobs for each link capacity. In continuously arriving jobs including when the schedulers are used we run each set of 10^4 jobs 20 times to get final results.

[5] The CAIDA UCSD http://www.caida.org/data/passive/trace_stats/

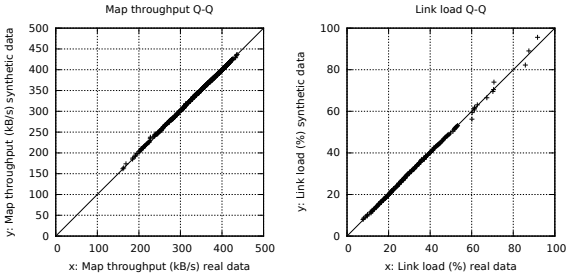


Figure 8: Q-Q plots comparing 10k synthetic data points and the empirical distributions of Map throughput and link load from figures 6 and 7.

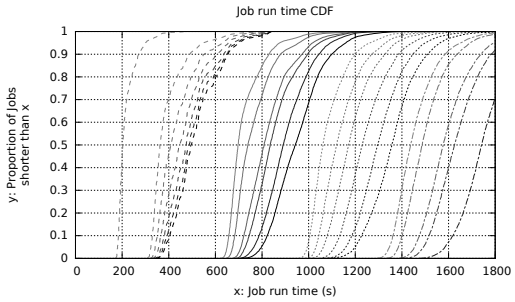


Figure 9: CDFs of the run time of fixed workload jobs on an $M_s = 7$ cluster. From left to right the link throughput values in Mbit/s are: (1, 2, 5, 7, 10, 12, 12.53), (15, 17, 20, 22, 24.5, 25.05), (27, 30, 32, 35, 37, 37.58), (40, 42, 45, 47, 50, 100). The 100 Mbit/s line is out of range. The CDFs for each link throughput form groups according to the number of waves that our model indicates (1:long dash, 2: solid, 3:short dash, 4:long-short dash). Line width in each group increases with link throughput.

V.3 Single Job

Understanding the performance of a cluster under a single job workload helps validate the simulator and can provide insights into the performance analysis of our scheduler under a continuous workload. Figure 9 shows the run time CDF of a single job for a fixed link bitrate that we vary from 1 Mbps to 40 Mbps on our H=5 simulated cluster. The bitrate grows from left CDF to right CDF. We can notice groups of CDFs that correspond to waves in our job model. The first group goes up to 12.53 Mbit/s which yields a 7 128MB block PCAP file requiring exactly the number of Maps (7) that can run simultaneously in a wave. We call this the wave boundary bit rate. With a slight increase to 15 Mbit/s, file size of 1073 MB, and 9 Maps (8 128 blocks and 1 48 MB block), there is a significant increase in the 50^{th} percentile run time from 495 s at 12.53 Mbit/s to 698 s at 15 Mbit/s, as this file size requires two waves of Maps. As the link traffic increases to include multiple waves this difference seems to decrease. Unless stated otherwise, link throughput values and their linestyle mapping in figure 9 are used throughout the paper.

As discussed in section V.2, link throughput is not constant and this has an impact on job runtime and resource usage CDFs. Figure 10 shows run time for links with capacity equal to the fixed bitrate values used in the previous section. Run times are generally smaller than those in figure 9 as the random workload distribution is applied to the link capacity and on average results in smaller bitrates and smaller PCAP files to process. Notice that the wave phenomenon is no longer obvious to observe except for the first two link capacity values.

V.4 Continuously Arriving Jobs

We now consider the case that we are most interested in: PCAP files freshly captured and uploaded to be processed by our network analysis job every 10 minutes. Bit rate and Map processing time distributions will make some jobs last more than 10 minutes, in

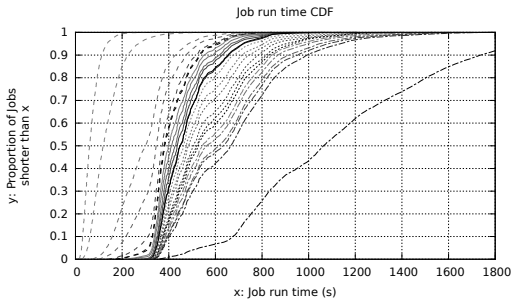


Figure 10: CDFs of the run time of random workload jobs on an $M_s = 7$ cluster following our random workload distribution.

which case the next job will start later. Because we are interested in obtaining the results of the analysis as soon as possible after PCAP file capture and upload to the cluster, we include this delay in the job run time. Job run time results for this case are shown in figure 11 and the Q-Q plot comparison with the single job case is shown in figure 12. The effect of the current job delaying the next is not noticeable in the lower capacity links. Above 27 Mbit/s the Q-Q plots get noticeably away from the $y = x$ line, especially for job run times higher than 400 s. This means that with these link capacities a significant number of jobs will be starting to experience delay because the previous job did not finish on time. For the 100 Mbit/s link the job run time will eventually increase monotonically with each incoming job at which point the cluster is unable to keep up. For the range of link capacities that we are studying in our $M_s = 7$ cluster, our bursting approach and scheduler will likely only be useful above 27 and below 100 Mbit/s where congestion exists but is moderate.

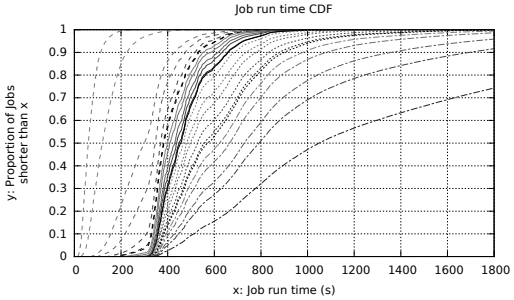


Figure 11: CDFs of the run time of random workload jobs on an $M_s = 7$ cluster following our random workload distribution but now with continuously arriving jobs. The 100 Mbit/s line is out of range.

VI. Scheduler Performance Results

In this section we compare four approaches: no scheduler, a simple scheduler using S_0 from section III.3, our proposed advanced scheduler using S_0, S_1, and S_2, and a traffic threshold-based scheduler that bursts all jobs with more than 50% link load. We compare these approaches on two aspects for each link bandwidth: 1) Q-Q job run time distributions with respect to singe

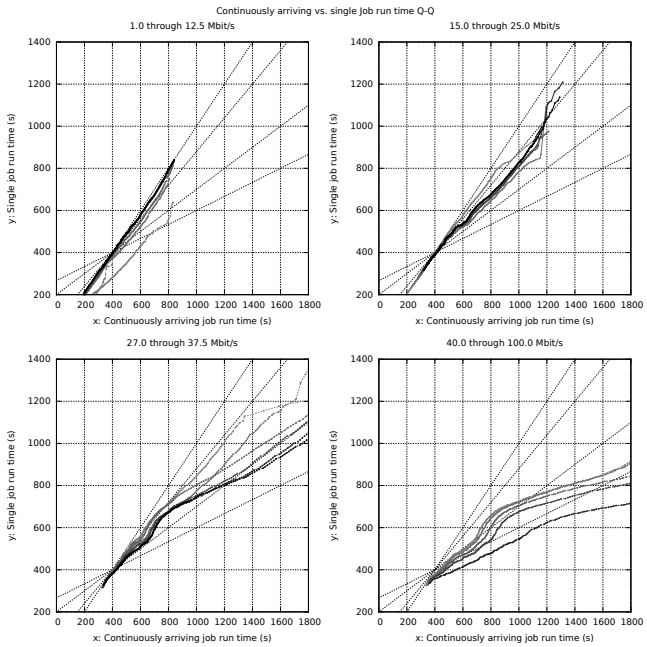


Figure 12: Q-Q plot comparing the job run time distributions of each link capacity with a single job (y) and continuously arriving jobs (x).

job as in figure 12 and 2) proportion of traffic that is burst. We include the run time of burst traffic in our Q-Q analysis by considering that it will be processed as a single job on the cloud. Noting that the Q-Q plots for the simple, advanced, and threshold approaches are visually similar to each other and to that of the approach without scheduler shown in figure 12, we use the following metric T to quantify their differences. Consider y and x as the Q-Q run time values for the run time distribution of single jobs (y) and the run time distribution for a given scheduler approach (x). We define $t = y/x - 1$, that is negative for each point where the run time value of the scheduler approach is larger than that of the single job. The scheduler performance metric we define is $T = 1/n \sum t$ for an n point Q-Q plot. Figure 13 shows metric T and the proportion of burst traffic for our set of link capacities. The four approaches have similar run times just slightly worse than single job up to 25 Mbit/s link capacity. From that value the run time T metric of the approach without scheduler degrades progressively whereas the

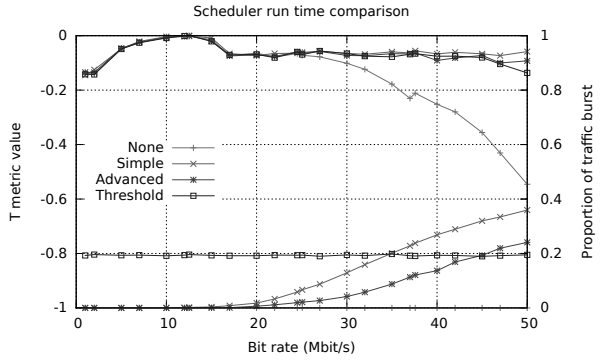


Figure 13: T metric and proportion of traffic burst for different scheduler approaches. T metric values are closer to the top of the graph, traffic burst values closer to the bottom.

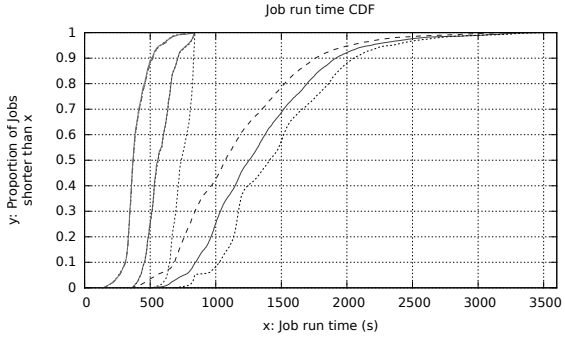


Figure 14: CDFs of the run time of random workload jobs for typical link bitrates on clusters of different sizes. Long dash: 10 Mbit/s and 100 Mbit/s on an M_s=7 cluster. Solid line: 10 Mbit/s, 100 Mbit/s, and 1 Gbit/s on an M_s = 70 cluster. Short dash: 10 Mbit/s, 100 Mbit/s, 1 Gbit/s, and 10 Gbit/s on an M_s = 700 cluster.

other three approaches are reasonably similar and not much worse than lower link capacities or single job. Looking at burst traffic, the simple approach bursts almost twice as much traffic as the advanced approach. The approach with less burst traffic is the advanced scheduler up to 45 Mbit/s. At 35 Mbit/s the advanced scheduler bursts 56% less traffic than the other two schedulers. Thus for the range of values where bursting could be useful, i.e. from 27 Mbit/s and below 100 Mbit/s, the advanced scheduler yields both the smallest amount of burst traffic and single-job comparable run times.

VII. Scaling Out

12 Mbit/s and the other bit rate values used in figure 9 are not link capacities that can be found in typical network links. Four typical link capacities are 10 Mbit/s, 100 Mbit/s, 1Gbit/s, and 10Gbit/s. The conservative capacity estimation approach in section IV.2 puts our $M_s = 7$ cluster at approximately 11 Mbit/s, which is not enough for the four typical link capacities. With that in mind, in figure 14 we show job run time for the four typical link capacities on clusters of 3 different sizes: our $M_s = 7$ initial cluster and $M_s = 70$ and $M_s = 700$ clusters.

VIII. Related Work

In the introduction we provided arguments for the relevance of our approach in the context of the analysis of network traffic. In summary, cloud bursting of batched Map-intensive network analysis jobs and the scheduling of such bursts has not been proposed before in the traffic analysis literature. In this section we take a look at different general-purpose cloud bursting approaches and argue that they are not adequate to our batched Map-intensive network analysis jobs.

Existing open-source virtualized data center management tools such as OpenStack and OpenNebula already support cloud bursting. Their initial focus was to provide an abstraction layer for the low level details of transitioning VMs (Virtual Machines) between data centers [1]. Throughout recent years, continuous improvements have been made to the amount of provided features and configurable parameters to better suit the users' needs. Today, many available solutions including Seagull [1] and the OPTIMIS Project [2] offer scheduling policies that determine if and when to burst to the cloud. However, these solutions are typically geared towards web applications and do not adequately support applications that need to process large amounts of data.

As the world embraces the ever-growing paradigm

of Big Data, Cloud Bursting can also be used in the context where cloud resources are used to store additional data if local resources become scarce. In fact, the use of Cloud Bursting for data-intensive computing has not only been proved feasible but scalable as well. [6] presents a middleware that supports a custom MapReduce type API and enables data-intensive computing with Cloud Bursting in a scenario where the data is split across a local private data center and a public cloud data center. Data is processed using computing power from both the local and public cloud data centers. Furthermore, data on one end can be processed with computing power from the other end, albeit lowering the overall job execution time. BStream [7] is a Cloud Bursting implementation that uses Hadoop YARN with MapReduce in the local data center and the Storm stream processing engine in the cloud to process MapReduce jobs instead of using YARN. The use of Storm allows the output of Maps to be streamed to Reduces in the cloud. Both [6] and [7] are better suited for forensic jobs for which a large data set must be analyzed and outputs from local Maps need to be processed on the cloud, than for batched analysis of smaller yet still computationally demanding data sets. In our case jobs can fit both in the local data center or the public cloud and the challenge is to process continuously arriving jobs.

IX. Conclusion

We have set out to explore cloud bursting for Map-intensive network traffic analysis jobs. Using our proposed architecture for collecting, cloud bursting, and processing traffic on Hadoop clusters, we characterized the run times of Maps on our physical clusters and used it to drive a simulation assessment of our job model and cloud bursting scheduler. Our scheduler bursts out up to more than 50% less traffic than other schedulers we compared. We plan to extend the traffic analysis modeling to other types of network traffic analyses and on platforms such as Spark using GPUs and to understand heterogeneity and energy consumption issues of scaling out the traffic analysis.

References

[1] T. Guo, U. Sharma, P. Shenoy, T. Wood, and S. Sahu. *Cost-Aware Cloud Bursting for Enterprise Applications*, ACM Transactions on Internet Technology, 13(3):1-24, 2014.

[2] S. K. Nair, et al. *Towards Secure Cloud Bursting, Brokerage and Aggregation*. Proceedings of the 8th IEEE European Conference on Web Services, ECOWS 2010, pages 189-196, 2010.

[3] Y. Lee and Y Lee. *Toward scalable internet traffic measurement and analysis with Hadoop*, ACM SIGCOMM Computer Communication Review, 43(1):5-13, 2012.

[4] RIPE. *Large-scale PCAP Data Analysis Using Apache Hadoop*, `https://github.com/RIPE-NCC/hadoop-pcap`, 2012.

[5] A. Pallavi and P. Hemlata. *Network Traffic Analysis Using Packet Sniffer*, International Journal of Engineering Research and Applications, 2(3):854-856, 2012.

[6] T. Bicer, D. Chiu, and G. Agrawal. *A Framework for Data-Intensive Computing with Cloud Bursting*. 2011 IEEE International Conference on Cluster Computing, pages 169âĂŞ177, September 2011.

[7] S. Kailasam, P. Dhawalia, S. J. Balaji, G. Iyer, and J. Dharanipragada. *Extending MapReduce across Clouds with BStream*. IEEE Transactions on Cloud Computing, 2(3):362-376, July 2014.

Distributed Parallel Computing for Visual Cryptography Algorithms

Raimondas Čiegis†, Vadimas Starikovičius†, Natalija Tumanova†,
Minvydas Ragulskis*, Rita Palivonaitė*

† Vilnius Gediminas Technical University, Lithuania, * Kaunas Technological University, Lithuania
rc@vgtu.lt

Abstract

The recent activities to construct exascale and ultrascale distributed computational systems are opening a possibility to apply parallel and distributed computing techniques for applied problems which previously were considered as not solvable with the standard computational resources. In this paper we consider one global optimization problem where a set of feasible solutions is discrete and very large. There is no possibility to apply some apriori estimation techniques to exclude an essential part of these elements from the computational analysis, e.g. applying branch and bound type methods. Thus a full search is required in order to solve such global optimization problems. The considered problem describes visual cryptography algorithms. The main goal is to find optimal perfect gratings, which can guarantee high quality and security of the visual cryptography method. The full search parallel algorithm is based on master-slave paradigm. We present a library of C++ templates that allow the developer to implement parallel master-slave algorithms for his application without any parallel programming and knowledge of parallel programming API. These templates automatically give parallel solvers tailored for clusters of computers using MPI API and distributed computing applications using BOINC API. Results of some computational experiments are presented.

Keywords Visual Cryptography, Parallel Algorithm, BOINC, Parallel Templates

I. Introduction

The recent activities to construct exascale and ultrascale distributed computational systems are opening a possibility to apply parallel and distributed computing techniques for applied problems which previously were considered as not solvable with standard computational resources. In this paper we consider a global optimization problem, where a set of feasible solutions is discrete and very large. There is no possibility to apply some apriori estimation techniques to exclude an essential part of these elements from the computational analysis, e.g. applying branch and bound type methods [1]. Thus a full search is required in order to solve such global optimization problems.

The given problem describes visual cryptography algorithms [2]. The main goal of this paper is to find optimal perfect gratings, which can guarantee high quality and security of the visual cryptography method. The full search parallel algorithm is based on master-slave paradigm [3]. We present a library of C++ templates that allow the developer to implement parallel master-slave algorithms for his application without any parallel programming and knowledge of parallel programming API. These templates automatically give parallel solvers tailored for clusters of computers using MPI API and distributed computing applications using BOINC API [4]. For application of such MPI templates see [5].

The rest of this paper is organized as follows. In Section II, the discrete global optimization problem is formulated. The optimality criterion for finding the optimal perfect grating function is defined and the set of feasible solutions is described. The paral-

lel master-slave type algorithm is presented in Section III. Here we also describe a genetic evolutionary algorithm. Heuristics as an alternative for full search algorithms are recommended for such type of deterministic global optimization problems. Our aim is to investigate the efficiency of such algorithms in the case when the set of feasible solutions is described by quite complicated non-local requirements. In Section IV, templates for master-slave type algorithms are described. They allow developers to implement parallel master-slave algorithms for their applications without any parallel programming on clusters of computers using MPI and distributed computing systems using BOINC technology. Results of computational experiments are given in Section V. They illustrate the theoretical scalability results. Some final conclusions are presented in Section VI.

II. Problem Formulation

Here we define the most important details on advanced dynamic visual cryptography algorithms. The image hiding method is based on time-averaging moire gratings [6]. The method generates only one picture, which is used as a plaintext. The secret image can be seen by the human visual system only when the original encoded image is oscillated in a predefined direction at a strictly defined amplitude.

Function $F(x)$ defines a greyscale grating if the following requirements are satisfied

(i) The grating is a periodic function $F(x+\lambda) = F(x)$, here λ is the pitch of grating, and $0 \le F(x) \le 1$;

(ii) m-pixels n-level greyscale function $F_{mn}(x)$ is defined as:

$$F_{mn}(x) = \frac{y_k}{n}, \quad \frac{(k-1)\lambda}{m} \le x \le \frac{k\lambda}{m}, \quad k = 1, \dots, m,$$

where $k = 1, \dots, m$, $0 \le y_k \le n$.

We will consider a subset P of perfect greyscale step functions, they satisfy the following additional requirements:

(1) The grating spans through the whole greyscale interval

$$\min y_k = 0, \quad \max y_k = n.$$

(2) The average greyscale level in a pitch of the grating equals

$$\gamma := \frac{1}{m}\sum_{k=1}^{m} y_k = \frac{n}{2}.$$

(3) The "norm" of the greyscale grating function must be at least equal to the half of the norm of the harmonic grating

$$\|F\| \ge \|\widetilde{F}\| = \frac{1}{2\pi}, \quad \|F\| := \frac{1}{\lambda}\int_0^{\lambda} \left| F(x) - \frac{1}{2} \right| dx.$$

(4) The pitch of the grating λ must be strongly identifiable, the main peak of the discrete Fourier amplitude must be at least two times larger compared to all other Fourier modes

$$\sqrt{a_1^2 + b_1^2} \ge 2\sqrt{a_j^2 + b_j^2}, \quad j = 2, 3, \dots, m-1,$$

where the function F is expanded into the Fourier truncated series

$$F(x) = \frac{a_0}{2} + \sum_{j=1}^{m-1} \left(a_j \cos\frac{2\pi k x}{\lambda} + b_j \sin\frac{2\pi k x}{\lambda} \right).$$

Now we formulate the optimality criterion for finding the optimal perfect grating function

$$\delta(F_{mn}^0) = \max_{F_{mn} \in P} \min_{s \in S_1} \left(\sigma\big(H_s(F_{mn}, \widetilde{\xi}_s)\big) \right), \qquad (1)$$

where the standard deviation of a grayscale step grating function oscillated harmonically is given by (s is the oscillation amplitude):

$$\sigma\big(H_s(F_{mn}, \widetilde{\xi}_s)\big) = \frac{\sqrt{2}}{2}\sqrt{\sum_{j=1}^{m-1} (a_j^2 + b_j^2) J_0^2\,(2\pi j s/\lambda)},$$

where J_0 is the Bessel function of the first type.

III. Parallel Algorithm

The determination of the optimal perfect grating (1) requires to test a full set D of gratings in two steps. For each given grating the following algorithm is applied:

1. First, the testing of grating is done to check if all conditions of perfect gratings are satisfied.

2. Second, for a perfect grating the value of the standard deviation of the grating function is computed and the optimal value is updated.

This algorithm is fully parallel and it can be implemented by using the master and slaves paradigm [5]. Thus it is well suited for application of distributed computing technologies, including BOINC technology.

The complexity of the full search algorithm is of order

$$W = \mathcal{O}(m^2 n^m).$$

Here the factor m^2 arises due to application of simple Fourier summing algorithm instead of FFT algorithm. For small values of m this approach is more robust and flexible.

For industrial applications gratings with $12 \leq m \leq 25$ and $15 \leq n \leq 127$ are considered. In order to reduce the size of set D two specific modifications are applied.

1. Due to the first condition of the perfect gratings we can fix $y_m = 0$.

2. The periodicity condition and the mirror transformation are applied to exclude the gratings, which were tested in earlier stages of the full search algorithm.

We note, that such modifications still not change the asymptotic of the complexity of the full search algorithm.

The presented parallel full search algorithm requires very big computational resources. As an alternative heuristic methods can be considered. A natural selection is to use genetic evolutionary algorithms. We have applied a modification of the standard genetic method [7].

- Every chromosome represents one period of a grayscale function $F_{mn}(x)$. The initial population comprises of N randomly generated chromosomes (perfect gratings) with values of genes uniformly distributed over the interval $[0, n-1]$. We note, that only perfect gratings are included into the population.

- The crossover between two chromosomes is done by using the random roullete method. The chance that the chromosome will be selected to the mating is proportional to its fitness value. The main difficulty of this step is that after crossover between two perfect gratings in most cases we obtain non-perfect new grating. Thus this step is continued while the specified number M of new chromosomes are obtained. It is allowed to include into the new population more than one copy of the same chromosome.

- A mutation procedure is used with a slight modification that if the value of one gene is reduced by δ_k, then the value of the other randomly selected gene is increased by the same amount. Again, only perfect new chromosomes are included into the updated population.

In computational experiments we have tested the quality of solutions obtained by using this heuristic based on the genetic evolutionary algorithm.

IV. MPI and BOINC Templates

We obtain a parallel solver for the considered problem using our C++ templates for distributed computing applications. These templates were designed for easy and quick development of parallel applications based on master-slave parallelization paradigm [3]. In master-slave parallel algorithm, master process reads the problem input, generates and distributes jobs to the slave processes. Slave processes receive jobs from the master, solve them and return back the obtained results. Finally, master process receives the results from the slaves and generates a new set of jobs if necessary.

Our C++ templates allow the developer to implement the parallel master-slave algorithm for his application without any parallel programming and knowledge of parallel programing API. The developer needs only to implement the application-specific parts of the code for the reading of the problem input, consecutive generation of single jobs, solving of the single job, processing or merging of obtained results. These application-specific tasks need to be implemented and placed in appropriate virtual functions of our C++

templates. The workflow of the whole master-slave algorithm (including communication between the master and slave processes) is provided by the basic classes of our C++ templates.

Let us now formulate the special features of our C++ templates for distributed computing applications:

- The templates are built as a hierarchy of C++ classes. Basic classes implement the basic functionality of master-slave algorithm and specify the pure virtual functions, which need to be implemented in descendant classes to obtain application-specific parallel solvers.
- Input parameters and results of the job are exchanged between the master and slave using input and output files.
- Design of the templates allows to build a parallel application using MPI API [8] or distributed computing application using BOINC API [4] applying the same C/C++ code with implementation of application-specific tasks.

The usage of technology based on input and output files is not as efficient as a direct message passing between processes. However, the performance overhead is negligible for the coarse grained jobs. This is it the case for our problem. In turn, such an approach significantly simplifies the template and allows the communication of input and output data between the master and slaves without application-specific parallel programming.

Such an approach also allows the implementation of distributed computing applications. Currently, our programing tool allows easy and quick development of distributed application for volunteer computing project based on the Berkeley Open Infrastructure for Network Computing (BOINC) [4], which is the most popular middleware for volunteer computing. Using our C++ templates, application for BOINC project can be developed without any knowledge of BOINC API. Moreover, MPI version of application solver is very useful in testing and debugging implementations of application-specific tasks.

Application-specific tasks are separated and implemented in different classes:

- *WorkGenerator* class. It reads the problem input in the constructor, generates and writes to the file input for the next job by calling application-specific function *GenerateInputForNewJob*(FILE *jobInputFile), which must be provided by the application developer.
- *ClientApplication* class. It reads the input file, solves the job, and writes the results to output file by calling application-specific function *SolveSingleJob*(const char* inputFileName, const char* outputFileName), which must be provided by the application developer.
- *ResultsAssimilator* class. It is processing results of the single job and merging them with previous results by calling application-specific function *AssimilateResults*(FILE *jobResultsFile), which must be provided by the application developer.

For our problem we don't need it, but for BOINC project application, developer needs also to provide a separate class for validation of the obtained results.

V. Computational Experiments

Parallel numerical tests were performed on the computer cluster "Vilkas" at the Laboratory of Parallel Computing of Vilnius Gediminas technical university. We have used up to eight nodes with Intel® Core™ i7-860 processors with 4 cores (2.80 GHz) and 4 GB of RAM per node. computational nodes are interconnected via Gigabit Smart Switch.

V.1 Parallel search algorithm

First, we have solved a small benchmark problem in order to show a very good scalability of such type of parallel algorithms. They can be implemented efficiently on very large distributed heterogeneous systems, including BOINC technology.

We have solved the optimization problem for $m = 10$, $n = 10$. In table 1, we present the total wall time $T_{s,p\times c}$ in seconds, when parallel computations are done on a cluster with p nodes and c cores per node, and s slaves have solved computational tasks.

The master is responsible for generation and distribution of job set and accumulation of results from slaves, a separate core is used to run this part of the parallel algorithm. Also, we present the values of parallel algorithmic speed-up

$$S_s = \frac{T_{s,p\times c}}{s}.$$

	1, 2x1	2, 3x1	3, 4x1	7, 2x4	11, 3x4
$T_{s,p\times c}$	478.0	242.2	160.2	78.2	50.1
S_s	1	1.97	2.98	6.11	9.54

Table 1: The total wall time $T_{s,p\times c}$ and speed-up S_s values for solving the grading optimization problem with $m = 10$, $n = 10$.

The degradation of the efficiency of the parallel algorithm for $s = 7$ and $s = 11$ slaves is explained by the well-known fact, that in the case of more cores per node the shared-memory structure becomes a bottleneck when too many cores try to access the global memory of a node simultaneously [9]. This conclusion is confirmed also by results of more computational experiments with different configurations of nodes and clusters:

$$T_{2,1\times 3} = 275.5, \quad T_{3,2\times 2} = 166.6, \quad T_{3,1\times 4} = 183.0.$$

The presented estimate of the complexity of the parallel search algorithm gives quite accurate estimate from above for the total computation time. For example, using results of previous computational experiments we get prediction that a problem with $m = 11$, $n = 12$ on 4×4 cluster will be solved in $T = 1195$ seconds. The result of computational experiments is $T_{15,4\times 4} = 939$ seconds. Again, we can note that the scalability of the parallel search algorithm is very good, the same problem is solved on 5×4 cluster in $T_{15,4\times 4} = 746$ seconds, this CPU time is very close to the prediction from the scalability analysis.

V.2 Genetic search algorithm

The full search algorithm requires very big computational resources and leads to a big challenge even for ultrascale distributed computational systems. Thus alternatives based on heuristic global optimization algorithms also should be investigated. Next we present results of computational experiments for the heuristic search algorithm which is based on genetic evolutionary algorithm.

In table 2, we present the standard deviation values for optimal gratings and gratings computed by using the genetic algorithm.

m	n	δ_{optim}	$\delta_{genetic}$
8	13	0.06178	0.06178
9	13	0.06310	0.06267
10	13	0.05984	0.05717
11	13	0.06162	0.05808

Table 2: The standard deviation values for optimal gratings and gratings computed by using the genetic algorithm.

The presented results of computational experiments show that the classical genetic algorithm is not efficient for this type of problems. Such a behaviour of the given heuristic is connected to the fact that for perfect gratings the mutation of two high-quality gratings mostly will not produce a new perfect grating. But exactly this step is most important for obtaining efficient genetic algorithms for solving discrete global optimization problems.

VI. Conclusions

In this paper we have described a library of templates for implementation of parallel master-slave type algorithms. These C++ templates allow to build a parallel solver automatically from the sequential version of the algorithm. The parallel solvers for clusters using MPI API or distributed computing applications using BOINC API are generated using the same C/C++ code. Only application-specific tasks must be provided by users. These templates are used to generate a parallel solver for applied problem of visual cryptography. The provided results of computational experiments have confirmed theoretical scalability estimates of the parallel algorithm.

The complexity of the given discrete global op-

timization is very big even for modern distributed computational systems. Thus as an alternative some heuristics can be considered. Results of application of classical genetic heuristic algorithms are showing that such standard algorithms are not efficient for this type of problems. In the future paper we will investigate hybrid genetic algorithm. In these memetic algorithms the approximations obtained by genetic method are also subject to local improvement phases. For such local optimization the modifications of the full-search algorithm described above can be used.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.

References

[1] R. Horst, P.M. Pardalos and N.V. Thoai, *Introduction to Global Optimization, Second Edition*. Kluwer Academic Publishers, 2000.

[2] P.S. Revenkar, A. Anjum and W.Z. Gandhare. "Survey of visual cryptography schemes," *Intern. Journal of Security and Its Applications*, vol. 4, no. 2, pp. 56-70, 2010.

[3] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, 1994.

[4] D. P. Anderson, "Boinc: a system for public resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 1-7.

[5] M. Baravykaitė and R. Čiegis, "An implementation of a parallel generalized branch and bound template", *Mathematical Modelling and Analysis*, vol. 12, no. 3, pp. 277–289, 2007.

[6] M. Ragulskis and A. Aleksa, "Image hiding based on time-averaging moire", *Optics Communications*, vol. 282, no. 14, 2752-2759, 2009.

[7] D. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers, 2002.

[8] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," www.mpi-forum.org, Version 1.1, 1995.

[9] N. Tumanova and R. Čiegis, "Parallel algorithms for parabolic problems on graphs", in *High-Performance Computing on Complex Environments*, Chapter 4, pp. 51-71, John Wiley & Sons, Inc, 2014.

On Autonomic HPC Clouds

Dana Petcu

Institute e-Austria Timişoara and West University of Timişoara, Romania
petcu@info.uvt.ro

Abstract

The long tail of science using HPC facilities is looking nowadays to instant available HPC Clouds as a viable alternative to the long waiting queues of supercomputing centers. While the name of HPC Cloud is suggesting a Cloud service, the current HPC-as-a-Service is mainly an offer of bar metal, better named cluster-on-demand. The elasticity and virtualization benefits of the Clouds are not exploited by HPC-as-a-Service. In this paper we discuss how the HPC Cloud offer can be improved from a particular point of view, of automation. After a reminder of the characteristics of the Autonomic Cloud, we project the requirements and expectations to what we name Autonomic HPC Clouds. Finally, we point towards the expected results of the latest research and development activities related to the topics that were identified.

Keywords HPC Clouds, Autonomic Computing

I. Introduction

The main characteristics of Cloud computing that have make it a market success for distributed applications are the elasticity in resource usage and on-demand self-service. However, the availability of a large number of computing resources has attract also various parallel computing applications that are ready to can make a compromise in performance in favor of scalability opportunities and instant availability of resources. While the supercomputing center have long queues that are serving resource-greedy batch parallel applications already tuned to match the architecture of the supercomputer, the HPC resources offered as Cloud services are nowadays seen as alternative for just-in-time parallel applications, real-time or interactive parallel applications, or even to support special environmental settings that are hard to be set in a supercomputing center through the interaction with admins. However, most of the current parallel applications are expected to run in the Cloud environment in the same manner as they are running on supercomputers (especially for production phases of an application), despite the multiple warnings from the literature about the lost of performance relative to the case of a supercomputing usage. In such context, the Cloud environment is expected to behave as a Grid environment, with the main difference that the user has more control on the software stack that is supporting the application execution (a thin difference as the Grid environments are allowing to simulate Cloud environments on top of Grid services). While such perspective is useful for the users who do not have access to the Grid environments that are mainly targeting academic users, the Cloud computing paradigm can offer more to HPC applications than the grid-like perspective.

Despite the fact that even the Cloud computing community has not taken up yet the full benefit of the rich results that have been obtained in the field of autonomic computing, we discuss in this paper a particular approach, on autonomic computing in HPC Clouds.

The paper is organized as follows. Firstly, we discuss the concept of Autonomic Cloud and the techniques that can make it a reality. Then we introduce the concept of Autonomic HPC Cloud and we look how the Autonomic Cloud concepts and techniques are projected in this particular case. Finally, we investigate the prospects of new results in the field by pointing towards various initiatives to provide partic-

ular solutions for Autonomic Cloud, respectively Autonomic HPC Clouds.

II. Autonomic Clouds

II.1 Autonomic computing

The automation of resource management is referred nowadays under the name of autonomic computing [1]. Autonomic computing refers also to information systems capable to self-manage following the goals set by human administrators. It is an inter-disciplinary field involving knowledge from several well-known branches of computer science, like distributed systems, artificial intelligence, bio-inspired computing, software engineering and control systems.

An autonomic system is expected to take decisions on its own using certain policies. It should also check and optimize its status and automatically adapt itself to changing conditions. The concept of autonomic computing is inspired from biology, in an analogy with the autonomic nervous system which takes care of low-level functions of the human body such as temperature regulation without any conscience action.

We consider in this paper that that the Autonomic Computing refers to the self-managing characteristics of a computing system.

II.2 The concept of Autonomic Cloud

Autonomic techniques are applied in Cloud environments, as rapid scalability of the pool of resources is needed to support unpredictable number of demands without a human intervention, or fast adaptation is requested to avoid the failures of hardware resources.

In what follows, we consider that the main characteristics of an Autonomic Cloud are the followings, as argued in [2]:

- involves computing and software services which instance number varies by adapting to unpredictable changes;
- follows a contextual behavior through methods of self-management, self-tuning, self-configuration, self-diagnosis, and self-healing;
- easy to manage and operate the services and deployments by using techniques for the design, build, deployment and management of resources with minimal human involvement;
- presents itself as a robust and fault tolerant system.

II.3 Architectures

The authors of [3] proposed a framework for evaluating the degree of adaptability supported by an architectural style and classified the most known architectural styles for distributed applications (e.g., pipe and filter, publish/subscribe, SOA, peer-to-peer) according to this framework. The most adaptable architectural styles for Autonomic Clouds can follow such adaptability evaluation.

The authors of [4] classified the architectural contributions to Cloud computing; to match the classification requirements at platform-as-a-service layer, for example, the programming environment (i.e., the tools for the development of applications) need to be ignored in order to focus on the execution environment with the goal to find an optimum deployment of application components on Cloud resources. At the software-as-a-service level the main goal is to make sure that services respect their QoS (e.g., response time).

An autonomic computing framework is naturally implemented using multi-agent systems, like in [5]. However, also other artificial intelligence techniques, like genetic algorithms, neural networks, or multi-objective and combinational optimization heuristics, can be successfully applied.

II.4 Techniques

Auto-scaling and load balancing. In order to take advantage of the elasticity characteristic of the Cloud, the deployed applications need to be automatically scaled in a way that makes the most efficient use of resources. Several frameworks have been introduced in the last years to support the application development taking into account the need of scalability. For example, SmartScale [6] and AutoJuju [7] are automated

scaling frameworks that uses a combination of vertical (adding more resources to existing VM instances) and horizontal (adding more VM instances) scaling to ensure that the application is scaled in a manner that optimizes both resource usage and the reconfiguration cost incurred due to scaling. Such scaling strategies are encountered also in [8] where different scalability patterns are used in combination with performance monitoring to allow automatic scalability management.

The auto-scaling of application components is useless without techniques for load balancing the requests among the scaled components. Usually load balancing is considered among physical machines, like in [9]. As Cloud applications are built from components, a load balancing among software components or services need to be considered. An energy-aware load balancer is proposed in [10].

Scheduling. The scheduling problem is as a multi-objective problem where the transfer, deployment and energy consumption costs need to be simultaneously minimized. Several approaches used in heterogeneous environments as presented in [11, 12, 13, 14] can be applied.

The problem of finding the mapping which minimizes the cost is NP complete and as a result scheduling (meta-)heuristics should be used to find sub-optimal solutions. Meta-heuristics such as those based on neural networks or evolutionary algorithms or linear programming proved their efficiency in solving cost problems found in scheduling problems [15].

Adaptive resource provisioning. The problem of adaptive virtualized CPU provisioning has received a lot of attention (for example, in [16, 17, 18]). However, an automated adaptive resource provisioning system as proposed on [19], based on feedback controllers (customer add-on outside of the Cloud service itself), was not reported yet. Current adaptive systems are based on real-time monitoring, like in [20].

In [21] an automated framework for resource allocation is presented: it can adapt the parameters to meet specific accuracy goal, and then dynamically converge to near-optimal resource allocation (optimal in terms of minimum costs). The proposed solution can handle unexpected changes in the data distribution characteristics and/or rates of the streaming application. Resource allocation for streaming processing was considered also in [22] which proposed an elastic scaling of data parallel operators.

A current challenge for Cloud providers is to automate the management of virtual servers while taking into account both high-level quality of service requirements of hosted applications as well as the resource management costs. In this context, the paper [23] proposes an autonomic resource manager to control the virtualized environment which decouples the provisioning of resources from the dynamic placement of virtual machines and which aims to optimize a global utility function which integrates both the degree of SLA fulfillment and the operating costs (a constraint programming approach to formulate and solve the optimization problem).

Probabilistic models were extensively used to assess the reliability of software systems at the architectural level, like in [24, 25], and these should to applied also to Cloud systems. The idea of [26] to reason at runtime about the non-functional attributes of the system and to perform accordingly some adaptations is particularly interesting in the context of Autonomic Clouds.

Service selection. Due to the tremendous number of Cloud services and the lack of a standard for their specification, manual service selection is a time costly task. Automatic methods for matching the user needs with the offers are therefore needed. In [27] for example a method for finding semantically equal SLA elements from differing SLAs by utilizing several machine learning algorithms is presented, together with a framework for automatic SLA management. Themis [28] is an implementation of a proportional-share auction that maximizes resource utilization while considering virtual machine migration costs; it uses a set of feedback-based control policies to adapt the application bid and resource demand to fluctuations in price. A decision support system based on risk analysis was proposed more recently in [29].

Service composition. Cloud service description languages should allow the automatically composition of

Cloud service to achieve a certain goal. The paper [30] formalizes the issue of automatic combination of Cloud services. Moreover, a proof-of-the-concept implementation is revealed to leverage a batch process for automatically constructing possible combinations of Cloud services, followed by a search for the best fit solution.

In [31] is presented an approach, named Café (Composite Application Framework), to describe configurable composite service-oriented applications and to automatically provision them across different providers. Components can be internal or external to the application and can be deployed in any of the delivery models present in the Cloud. The components are annotated with requirements for the infrastructure they later need to be run on. A component graph is used to capture the dependencies between components and to match against the infrastructures offered by different providers.

Service discovery. Software agents have been successfully used in the recent years for service discovery, brokering or composition. Cloudle [32] is such an agent-based search engine for Cloud service discovery proving that agent-based cooperative problem-solving techniques can be effectively adopted for automating Cloud service composition. As reported in [33], agents can be used also in the mechanism for service migration between Clouds.

Self-configuration. An autonomic computing application is expected to be composed of autonomic components interacting with each other. An autonomic component can be modeled in terms of a local and a global control loops with sensors for self-monitoring, effectors for self-adjustment, knowledge and planner or adapter for exploiting policies based on self- and environment awareness.

The authors of [34] proposed an automated line for deploying distributed application composed of a set of virtual appliances, which includes a decentralized protocol for self-configuring the virtual application machines. The solution named VAMP (Virtual Applications Management Platform) relies upon a formalism for describing an application as a set of interconnected virtual machines and on an engine for interpreting this formalism and automating the application deployment using infrastructure services. The formalism offers a global view of the application to be deployed in terms of components with the associated configuration- and interconnection constraints and with their distribution within virtual machines; it extends OVF language, dedicated to virtual machines description, with an description language for distributed application software architecture.

The above described approach is complemented by the one from [35] where is exposed a mechanism that requires zero manual intervention during the configurations on the IP addresses of the resources from a Cloud center.

An automated approach to deploy pre-configured and ready-to-run virtual appliances on the most suitable Cloud infrastructure is still missing. However, in [36] is proposed an architectural approach using ontology-based discovery to provide QoS aware deployment of appliances on Cloud service providers.

The paper [37], that exposes the design of an adaptive framework for optimizing the execution of scientific applications in the Cloud, uses a MAPE-K loop, well known in autonomic computing, and relies on the concept of utility for optimizing the configuration of the Cloud stack on a per-job basis.

Self-healing. The policy-based, goal-based, or utility-based approaches are reactive techniques that enable the Autonomic Cloud to respond to problems only when they occur [38]. For example, in [39] a policy based management is used to evaluate the state of the system against predefined rules and actuate self-healing to return the system to the desired state – focus is put on investigating the capability of the system to recognize a fault and react to it. Alternatively, proactive techniques like the ones proposed in [40, 41] predict a set of environmental conditions before they actually happen.

Reactive and proactive techniques are usually implemented using multi-agent systems. In the context of Autonomic Clouds we can distinguish between two types of solutions. The first type of solutions exploits the idea of building multi-agent controllers for autonomic systems, which are capable to manage the system and themselves, as in [42]. The second idea is

No. Characteristic	Target	Proposed approach
1. Resources/services variability	Auto-scaling	Combine vertical with horizontal scaling
		Use scaling patterns
		Performance monitoring
	Load-balancing	At the physical machines level
		At the software service level
		At the application components level
	Scheduling	Optimize transfer costs
		Optimize deployment costs
		Optimize energy consumption
	Adaptive resource provisioning	Control-policy to adapt to price fluctuations
		Use feedback controllers
3. Contextual behaviour	Self-healing	Policy-based approach to evaluate state against rules
		Multi-agent controllers
		VMs/services acting as agents
	Self-configuration	Use descriptors of the interconnected components
		Component dynamic identification
		Use pre-configurated appliances
	Automatic pricing	Self-adjust price according application requirements
3. Easy deployment & management	Optimal deployment	Use patterns of architectural styles
	Selection	Find semantically equal SLA
		Auctions for mazimize utilization & reduce migration costs
	Composition	Description language to support composition
		Automatically construct combinations & search for best fit
		Dependency graphs for application components and their annotation with infrastructure requirements
	Discovery	Agent-based search engine
4. Robust and fault tolerant	Reliability	Decouple resource provisioning from dynamic placement
		Reason at run-time about the non-functional attributes
		Use probabilistic models
	Migration	Agent-based mechanism

Table 1: Characteristics, targets and approaches for Autonomic Clouds (after [2])

to allow virtual machines and services to behave like agents and to make decisions based on local policies and local knowledge as in [43, 44].

Autonomic pricing mechanism. Most of the Cloud service providers charge their clients for metered usage based on fixed prices. In [45] were exposed pros and cons of charging fixed prices as compared to variable prices. Deploying an autonomic pricing mechanism that self-adjusts pricing parameters to consider application and service requirements of users is shown to achieve higher revenue than various other common fixed and variable pricing mechanisms.

Summarize. Table 1 summarizes the main characteristics and targets in Autonomic Cloud as well as the existing approaches.

III. HPC Clouds

The Cloud paradigm is attractive for the HPC applications due to the promise of: (a) instant availability

instead the long waiting queues of supercomputers; (b) software stack that can be selected by the application owner instead the need of a match with the installed OS or libraries; (c) larger capacity than offered by the local or regional HPC centers; (d) abstractization of the hardware resources through virtualization and which provide a certain level of portability, instead a strong dependence on the hardware architecture; (e) service level agreements instead best effort.

However, there are several HPC requirements that are not matching the Cloud concept and are hindering its adoption in the HPC communities: (a) the need to be close to the bar metal to obtain the highest possible performance from the underlying machines instead the use of virtualized environments; (b) the need to use optimized HPC libraries that are hardware dependent instead general purpose software; (c) the need of tuned hardware for particular computations, including combination of CPUs, GPUs and FPGAs, instead commodity hardware; (d) the need for a large number of resources for well-defined time instead a small number of resources for an indefinite time; (e) the need for high throughput interconnections based on userspace communication pathways requiring wired data transfer instead virtualized networks transfers; (f) the need for parallel file-systems instead distributed file-systems.

The HPC applications of which performance is not strongly affected by the differences between the Cloud infrastructure services and the supercomputing center services are the ones which do not have special requirements in high-throughput interconnection and storage. For example, parametric studies in which a certain program is executed multiple times with different parameters (or bags of independent tasks) can run in Cloud environment with a relative low lost in performance compared with supercomputing environments. Moreover, new opportunities are offered by the instant availability for parallel streaming applications or real time parallel applications.

The HPC-as-a-service (HPCaaS) concept makes a compromise between the common understanding of distributed Cloud services and the HPC requirements which where enumerated above. It refers to a bare-metal compute model similar to an in-house cluster (therefore, named here cluster-on-demand). The HPC cluster of Amazon EC2[1], Gompute[2], Penguin-on-Demand (POD)[3], R-HPC[4], Cyclone[5], Salbacore[6] are few example of HPCaaSs. Dozens of popular open source applications frequently used by the HPC community, including schedulers and load balancers, are ready to be selected and executed in the acquired environments. GPGPUs are available at request. Data transfer inside the clusters are not charged. High throughput networks are available.

However, the HPCaaS offers are hardware dependent and a HPC application owner still need to evaluate the services before the deployment for production purposes. The scale-up and scale-out options are rarely encountered (a counterexample is in the offer of Cyclone). As the offers are quite different, the migration of parallel applications between various HPCaaS provider sites remains an issue. Despite the adoption of concepts of service level agreement adoption, the fault tolerance remains an application provider problem. Self-configuration is far to be achieved and the application provider that intends to consume a HPCaaS needs to have parallel programming knowledge. A holistic vision of the evolution of HPC Clouds towards a tight integration betweeb programming models, runtime middleware and virtualization infrastructure was presented recently in [46].

IV. Towards Autonomic HPC Clouds

In this section we discuss the requirements and expectations of future HPCaaSs compliant with the concepts and techniques of autonomic computing.

IV.1 The concept of Autonomic HPC Clouds

In most cases, the scalability of an HPC application is predictable. This fact allows the supercomputing centers to request as input parameter for the job submission descriptor the number of resources that will

[1] https://aws.amazon.com/hpc/
[2] http://www.gompute.com/
[3] https://pod.penguincomputing.com/
[4] http://www.r-hpc.com/
[5] http://www.sgi.com/products/hpc_cloud/cyclone/
[6] http://www.sabalcore.com/

be allocated, as well as a time interval. While this hypothesis is maintained by the current HPCaaSs, it is not the case for most of other Cloud services that are designed to support scale-ups and -outs and not defined time intervals for resource allocations. The design of Autonomic HPC Clouds should foreseen that a variable number of resources are possible, especially to enable the Big Data or streaming data parallel processing.

The hardware failures were considered until now a minor problem in the HPC computing world. However, with the advent of the race for achieving the petascale levels in HPC, the hardware and software failure tolerance increased in importance. The changes in the programming paradigms and languages that are foreseen for petascale systems to coupe with failure tolerance are of high interest for the HPC Cloud providers in conjunction with the pressure to fulfill high quality service level agreements. Self-* procedures are welcomed in this context to be combined with the classical tools for HPC support.

The deployment and the control of the HPC applications and its resources is still far from being hardware-independent. Moreover, few of the HPCaaS providers are measuring the costs based on job submission rather on being based on the allocated hardware.

Consequently, we consider that the four main characteristics of Autonomic Clouds are relevant in the definition of an Autonomic HPC Cloud. An HPC Cloud is autonomic if: (1) involves computing and software services which instance number varies; (2) follows a contextual behavior through methods of self-management, self-tuning, self-configuration, self-diagnosis, and self-healing; (3) easy to manage and operate the services and deployments; (4) presents itself as a robust and fault tolerant system.

IV.2 Requirements related to the architecture

A similar framework with the one proposed in [3] to evaluate the degree of adaptability supported by an architectural style of distributed systems should be adopted for the architectural styles of parallel applications (e.g. master-slave, bag of tasks, parametric studies).

Currently, the targeted architecture HPCaaS is of IaaS type. No offers exists yet for HPC oriented PaaS or HPC based SaaS. Moreover, the variety of the HPCaaS services in terms of hardware support opens the problem of portability of the HPC Cloud applications and the interoperability in case of building HPC Clouds Federations or Multi-HPC-Clouds.

An abstract representation of the HPCaaS which allow an automatic deployment is needed to be defined, standardized and adopted by HPCaaS providers is urgently needed.

IV.3 Autonomic computing techniques compliance with HPC Cloud

Auto-scaling and load balancing In order to take advantage of the elasticity characteristic of the Cloud, the deployed applications need to be scale up and scale down during their execution. However, the current deployment style of most HPC applications is supposing the allocation of a fixed number of hardware and software resources. The introduction of automatic scaling mechanisms in the parallel programming paradigms and languages is necessary to be pursued.

The classic load balancers used in HPC environments are also assuming by default an estimation of the number of requested resources and execution time. In order to coupe with a variable number of resources during execution or the undefined execution time, or even with resource usage optimizations or energy-aware resource consumption, the current HPC load balancers should be revised and a good starting point are the new proposals for Cloud specific load balancers.

While horizontal and vertical scaling based on the number of virtual machines (more identical VMs, respectively more identical resources to existing VMs) are a custom in classical Cloud services, the scaling of HPC applications in heterogeneous environments composed of various computing, storage and network devices is a challenge problem not yet investigated. Different scalability patterns used in combination with performance monitoring can also help in HPC case the automatic scalability management.

Scheduling The scheduling problem was intensively studied by the HPC provider communities and beyond, and has been escalated with the Grid computing advent. In the context of the current trend to allow the portability of HPC applications from CPUs to GPGPUs and viceversa via abstractions at the programming level, or to take into consideration the energy consumption costs of interest for both the HPC Cloud provider and the HPC application provider, the existing scheduling techniques should be redefined.

A proof-of-concept framework was proposed in [47] for self-adaptive resource scheduling ensuring a trade off between performance and energy at real time on multi-core or many-core based heterogeneous Clouds.

Adaptive resource provisioning Finding the an optimum quantity of HPC resources can be critical for the automation in HPC Clouds. Being unaware of the power of the available resources, the HPC application provider is usually not able to estimate the application needs of resource. Automatic frameworks for resource provisioning and sizing for HPC applications in HPC Clouds are therefore needed. Such framework should use performance models of the application, performance estimators according to the available resources as well as performance monitoring date of previous executed applications to build some performance estimators. Feedback controllers (customer add-on outside of the Cloud service itself) are also welcomed if the application is not running for the first time in the HPC Cloud environment. Beyond the performance estimators, the automated framework for resource allocation should follow policies and rules (cost minimization, energy consumption minimization, communication minimization volume, specific SLAs, or a combination of these criteria).

A proposal for an autonomous resource management system with self-properties for HPC Cloud was discussed in [48].

Service selection HPC applications are often using special libraries. Few HPCaaS offers are including nowadays the access to pre-installed HPC specific software that enable the execution of particular applications (e.g. OpenFOAM provided by Cyclone). A market place of such appliances can improve the quality of the HPCaaSs.

The diversity of the HPCaaS due to the variation of the underlying hardware poses the problem of the right service for a certain HPC application. Moreover, there is a lack of a standard for their specification. Autonomic methods for matching the HPC application needs with the offers are therefore needed.

Switching between computational resources from different providers or inside a Heterogenous Cloud in order to select the most suitable computational environment for an HPC application execution is a desirable feature to avoid hardware or vendor lock-in. In the case of HPC applications, the migration is hindered by hardware incompatibility and the need to tune the software for specific hardware to achieve high performance. A proof-of-concept tool named ADAPT [49] is trying to provide a solution by an autonomic execution monitor that exploits feedback from the runtime and resolves missing dependencies.

Service composition Often the HPC applications are build from multiple programs depended on specific libraries and which are executed in a specific order according to a particular workflow. This split in various components offers the opportunities to play with various configuration settings and to find an 'optimal' combination (from a certain point of view, e.g. response time, costs, energy etc). The approaches that were proposed for the Autonomic Cloud are well suited to be applied also in the case of the Autonomic HPC Cloud.

Service discovery The number of the available HPCaaS is limited and their offer is still easy to be followed by a human agent. The management tools behind the HPCaaS are currently proprietary. If open-source tools will be developed, it is possible that the number of the HPCaaS will increase exponentially, providing incomes to small computing centers. Then the discovery of the new services will become a problem also for the Autonomic HPC Cloud.

Self-configuration An automated approach to deploy pre-configured and ready-to-run virtual appliances on the most suitable Cloud infrastructure is still

missing. This fact is valid also for HPC Clouds. Moreover, the lack of virtualization layer (bar metal offers in HPCaaS) is complicating the application deployer task. The efforts to provide virtualization layers also for GPUs (available nowadays only for a particular hardware architecture) and to reduce the performance impact of the virtualization software are further steps towards an easy to manage configuration process.

An automatic I/O configurator for HPC applications was proposed in [50]; it utilizes machine learning models to perform black-box performance/cost predictions.

Self-healing The HPC Cloud offer should provide a certain quality stated in a service level agreement. E.q. hardware and software services should be available 99.99% of the time. Faulty services should be detected just in time and automatic reactive mechanisms should be in place. The performance monitoring of the services is therefore essential and current techniques should be improved both in terms of the number of sensors, but also in what concerns non-intrusiveness and policies. Proactive techniques were not approached yet in HPC Cloud context.

Autonomic pricing mechanism HPCaaS billings are usually based on hardware allocation time, rather than on job as expected in a supercomputing center (with one counterexample: R-HPC is able to provide job-based bills). Therefore, most of the HPCaaS providers charge their clients based on fixed prices. Variable prices can be conceived based on the urgency of the HPC tasks versus the current load of the provider resources and the supplementary sofware stack that is requested beyond the bar metal.

V. Expectations from on-going research activities

Panacea, HARNESS, MIKELANGELO and CloudLightning are four European research and development on-going initiatives that are working with Autonomic Clouds, respectively Autonomic HPC Clouds. Their progress will influence the future of these fields.

PANACEA[7] proposes innovative solutions for a proactive autonomic management of cloud resources, based on a set of advanced machine learning techniques and virtualization. PANACEA supports the following properties of the Autonomic Cloud: (a) self-healing against anomalies by recovering from multiple node and link failures and using proactive rejuvenation of applications and servers for preventing crashes and increasing the availability, predicting the threshold violation of response time of servers; (b) self-configuring by efficiently mapping user's requirements onto distributed clouds and configuring on-the-fly in the presence of anomalies, self-optimizing using proactive migration of virtual machines from one cloud resource to another, maintaining the quality of service of end-to-end flows; (c) self-protecting using proactive reconfiguration of overlay networks to protect against security attacks.

HARNESS[8] integrates heterogeneous hardware and network technologies into data centre platforms, to increase performance, reduce energy consumption, and lower cost profiles for Cloud applications. It develops an enhanced PaaS software stack that brings new degrees of freedom to cloud resource allocation and optimisation. Technologies such as FPGAs, GPGPUs, programmable network routers, and solid-state disks promise increased performance, reduced energy consumption, and lower cost profiles. Specialised technologies are virtualised into resources that can be managed and accessed at the platform level. The Cloud platform has access to a variety of resources to which it can map the components. A flexible application may potentially be deployed in many different ways over these resources, each option having its own cost, performance, and usage characteristics.

MIKELANGELO[9] is addressing the issues of HPC Clouds and targets a core bottleneck, the virtual I/O. The work is concentrated on improvement of virtual I/O in KVM.

CloudLightning[10] proposes a new way of provisioning heterogeneous cloud resources to deliver services, specified by the user, using a particular service de-

[7] http://projects.laas.fr/panacea-cloud/
[8] http://www.harness-project.eu/
[9] http://www.mikelangelo-project.eu/
[10] http://cloudlightning.eu/

scription language. Due to the evolving complexity of modern heterogeneous clouds, it proposes to build our system based on principles of self-management and self-organisation. The goal is to address energy inefficiencies particularly in the use of resources and consequently to deliver savings to the cloud provider and the cloud consumer in terms of reduced power consumption and improved service delivery.

VI. Conclusions

We defined the main characteristics of the Autonomic HPC Clouds and we identified the key elements of a roadmap to evolve HPC-as-a-Service offers to take advantage of the Cloud and Autonomic Computing concepts. The problems to be solved are complex and the solutions are not straightforward. Partial solutions are foreseen to be provided by ongoing research and development initiatives.

Acknowledgment

The work related to Autonomic HPC Clouds is supported by the European Commission under grant agreement H2020-6643946 (CloudLightning). The CLoudLightning project proposal was prepared by eight partner institutions, three of them as earlier partners in the COST Action IC1305 NESUS, benefiting from its inputs for the proposal. The section related to Autonomic Clouds is supported by the Romanian UEFISCDI under grant agreement PN-II-ID-PCE-2011-3-0260 (AMICAS).

References

[1] J.O. Kephart, D.M. Chess, "The vision of autonomic computing", *Computer* 36, pp. 41-50, 2003.

[2] D. Petcu, "Building automatic clouds with an open-source and deployable platform-as-a-service", in *Advances in Parallel Computing* 23, IOS Press, pp. 3-19, 2014.

[3] R.N. Taylor, N. Medvidovic, P. Oreizy, "Architectural styles for runtime software adaptation", in *Proceedings of WICSA/ECSA*, 2009, pp. 171-180.

[4] M. Litoiu, M. Woodside, J. Wong, J. Ng, G. Iszlai, "A business driven cloud optimization architecture", in *Proceedings of ACM SAC*, 2010, pp. 380-385.

[5] S. Venticinque, R. Aversa, B. Di Martino, D. Petcu, "Agent-based cloud provisioning and management. Design and prototypal implementation", in *Proceedings CLOSER*, 2011, 184-191

[6] S. Dutta, S. Gera, A. Verma, B. Viswanathan, "Smartscale: automatic application scaling in enterprise clouds", in *Proceedings of IEEE CLOUD*, 2012, pp.221-228.

[7] B. Karakostas, "Towards autonomic cloud configuration and deployment environments", in *Proceedings of ICCAC*, 2014, pp. 93-96.

[8] M. Ughetti, "Scalability patterns for platform-as-a-service", in *Proceedings of IEEE CLOUD*, 2012, pp. 718-725.

[9] C. Adam, R. Stadler, "A middleware design for large-scale clusters offering multiple services", *IEEE Transactions on Network and Service Management* 3:1, 2006, pp. 1-12.

[10] A. Paya, D. Marinescu, "Energy-aware load balancing and application scaling for the cloud ecosystem", *IEEE Transactions on Cloud Computing*, 2015, doi: 10.1109/TCC.2015.2396059.

[11] M.E. Frincu, N. M. Villegas, D. Petcu, H. Muller, R. Rouvoy, "Self-healing distributed scheduling platform", in *Proceedings of CCGRID*, 2011, pp. 225-234.

[12] M.E. Frincu, C. Craciun, "Dynamic and adaptive rule-based workflow engine for scientific problems in distributed environments", in *Cloud Computing and Software Services: Theory and Techniques*, CRC Press, 2010, pp. 227-251.

[13] M.E. Frincu, "Dynamic scheduling algorithm for heterogeneous environments with regular task input from multiple requests", in *Lecture Notes in Computer Science* 5529, 2009, pp. 199-210.

[14] F. Micota, M. Frincu, D. Zaharie, "Population-based metaheuristics for tasks scheduling in heterogeneous distributed systems", in *Lecture Notes in Computer Science* 6046, 2011, pp. 321-328.

[15] A. Benoit, L. Marchal, J.F. Pineau, Y. Robert, F. Vivien, "Scheduling concurrent bag-of-tasks applications on heterogeneous platforms", in *IEEE Transactions on Computers* 59:2, 2010, pp. 202-217

[16] P. Padala, K.G. Shin, X. Zhu, M .Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, "Adaptive control of virtualized resources in utility computing environments", in *Proceedings of Eurosys*, 2007, pp. 289-302.

[17] E. Kalyvianaki, T. Charalambous, S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters", in *Proceedings of ICAC*, 2009, pp. 117-126.

[18] S.M.Park, M.Humphrey, "Feedback-controlled resource sharing for predictable escience", in *Proceedings of SC*, 2008, article 13.

[19] H.C. Lim, S. Babu, J.S. Chase, S.S. Parekh, "Automated control in cloud computing: challenges and opportunities", in *Proceedings of ACDC*, 2009, pp. 13-18.

[20] F. Fargo, C. Tunc,Y. Al-Nashif, A. Akoglu, S. Hariri, "Autonomic Workload and Resources Management of Cloud Computing Services," in *Proceedings of ICCAC*, 2014, pp.101-110.

[21] S. Vijayakumar, Q. Zhu, G. Agrawal, "Automated and dynamic application accuracy management and resource provisioning in a cloud environment", in *Proceedings of GRID*, 2010, pp. 33-40.

[22] S. Schneider, H. Andrade, B. Gedik, A. Biem, K.L. Wu, "Elastic scaling of data parallel operators in stream processing", in *Proceedings of IPDPS*, 2009, pp. 1-12.

[23] H.N. Van, F. D. Tran, J.M. Menaud, "SLA-aware virtual resource management for cloud infrastructures", in *Proceedings of CIT*, 2009, pp. 357-362

[24] A. Filieri,C. Ghezzi, V. Grassi, R. Mirandola, "Reliability analysis of component-based systems with multiple failure modes", in *Lecture Notes in Computer Science* 6092, 2010, pp. 1-20.

[25] I. Krka, L. Golubchik, N. Medvidovic, "Probabilistic automata for architecture-based reliability assessment", in *Proceedings of QUOVADIS*, 2010, pp. 17-24.

[26] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, "Model evolution by run-time parameter adaptation", in *Proceedings of ICSE*, 2009, pp. 111-121.

[27] C. Redl, I. Breskovic, I. Brandic, S. Dustdar, "Automatic sla matching and provider selection in grid and cloud computing markets", in *Proceedings of GRID*, 2012, pp. 85-94.

[28] S. Costache, N. Parlavantzas, C. Morin, S. Kortas, "Themis: economy-based automatic resource scaling for cloud systems", in *Proceedings of HPCC*, 2012, pp. 367-374.

[29] S. Gupta, V. Muntes-Mulero, P. Matthews, J. Dominiak, A. Omerovic, J. Aranda, S. Seycek, "Risk-driven framework for decision support in cloud service selection", in *Proceedings of CCGRID*, 2015, 545-554.

[30] D.K. Nguyen, F. Lelli, M.P. Papazoglou, W.J. Van den Heuvel, "Issue in automatic combination of cloud services", in *Proceedings of ISPA*, 2012, pp. 487-493.

[31] R. Mietzner, T. Unger, F. Leymann, "Cafe: a generic configurable customizable composite cloud application framework", in *Lecture Notes in Computer Science* 5870, 2009, pp. 357-364.

[32] K. M. Sim, "Agent-based cloud computing", *IEEE Transactions on Services Computing* 99, 2011, pp. 1.

[33] C.T. Fan, W.J. Wang, Y.S. Chang, "Agent-based service migration framework in hybrid cloud", in *Proceedings of HPCC*, 2011, pp. 887-882.

[34] X. Etchevers, T. Coupaye, F. Boyer, N. De Palma, "Self-configuration of distributed applications in

the cloud", in *Proceedings of CLOUD*, 2011, pp. 668-675.

[35] C. Hu, M. Yang, K. Zheng, K. Chen, X. Zhang, B. Liu, "Automatically configuring the network layer of data centers for cloud computing", *IBM Journal of Research and Development* 55:6, 2011, 3:1–3:10.

[36] A.V. Dastjerdi, S.G.H. Tabatabaei, R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery", in *Proceedings of CCGrid*, 2010, pp. 104-112.

[37] M. Koehler, S. Benkner, "Design of an adaptive framework for utility-based optimization of scientific applications in the cloud", in *Proceedings of UCC*, 2012, pp. 303-308.

[38] J.O. Kephart, W.E. Walsh, "An artificial intelligence perspective on autonomic computing policies", in *Proceedings of POLICY*, 2004, pp. 3-10.

[39] T. Lorimer, R. Sterritt, "Autonomic management of cloud neighborhoods through pulse monitoring", in *Proceedings of UCC*, 2012, pp. 295-302.

[40] V. Casola, E.P. Mancini, N. Mazzocca, M. Rak, U. Villano. "Self-optimization of secure web services", *Computer Communications* 31, 2008, pp. 4312-4323.

[41] M. Litoiu, "A performance analysis method for autonomic computing systems", *ACM Transactions on Autonomous and Adaptive Systems* 2:1, 2007, article 3.

[42] B.A. Caprarescu, D. Petcu, "A self-organizing feedback loop for autonomic computing", in *Proceedings of COMPUTATIONWORLD*, 2009, pp. 126-131.

[43] K. Begnum, N.A. Lartey, L. Xing, "Cloud-oriented virtual machine management with MLN", in *Lecture Notes in Computer Science* 5931, 2009, pp. 266-277.

[44] B.A. Caprarescu, N.M. Calcavecchia, E. Di Nitto, D.J. Dubois, "SOS cloud: self-organizing services in the cloud", in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering* 87, 2012, pp. 48-55.

[45] C.S. Yeo, S. Venugopal, X. Chu, R. Buyya, "Autonomic metered pricing for a utility computing service", *Future Generation Computer Systems* 26 (8), 2010, pp. 1368-1380.

[46] D. Petcu, H. Gonzalez-Velez, B. Nicolae, J.M. Garia-Gomez, E. Fuster-Garcia, C. Sheridan, "Next generation HPC clouds: a view for large-scale scientific and data-intensive applications", *Lecture Notes in Computer Science* 8806, 2014, pp. 26-37.

[47] J. Wood, B. Romoser, I. Zecena, Z. Zong, "B-MAPS: a self-adaptive resource scheduling framework for heterogeneous cloud systems", in *Proceedings of CAC*, 2013, Article 19.

[48] M.E. Frincu, D. Petcu, "Resource management for HPC on the cloud", in *High-Performance Computing on Complex Environments*, 2014, pp. 303-323.

[49] J. Slawinski, V. Sunderam, "Autonomic multi-target deployment of science and engineering HPC applications," in *Proceedings of ICCAC*, 2014, pp.180-186.

[50] L. Mingliang, J. Ye, Z. Jidong, Z. Yan, S. Qianqian, M. Xiaosong, C. Wenguang, "ACIC: Automatic cloud I/O configurator for HPC applications," in *Proceedings of SC*, 2013, pp. 1-12.

Labeling connected componets in binary images based on cellular automata

Biljana Stamatovic

University of Donja Gorica, Montenegro
biljana.stamatovic@udg.edu.me

Abstract

This short paper introduce an algorithm for labeling connected components in n-dimensional binary images based on cellular automata, , $n \geq 2$. Here is presented tree-dimensional binary images algorithm. The algorithm code was implemented in NetLogo programming environment. The algorithm is local and can be efficiently implemented on data-flow parallel platforms with an asymptotic complexity of $O(L)$ on an $L \times L \times L$ bynary image.

Keywords labeling connected components, cellular automata, data-flow

I. Introduction

Labeling of connected components has been intensively studied. This is one of the most fundamental operations in pattern analysis, pattern recognition, computer (robot) vision, and machine intelligence. Many algorithms have been proposed for addressing this issue [1, 2, 3, 4, 5, 6, 7, 8]. The labeling algorithms transform a binary image in an image in which groups of connected cells (pixels) are gruoped in disjoint components with unique labels. So, the analysis of the image can be performed on a higher level than the pixel level.

The cellular automata (CA) can be considered as an alternative way of computation based on local data flow principles. Also, the CA can be seen as a computing machine in the sense that it is able to transform an input configuration, embedded in its initial state, to an output configuration. Here, we supose that initial data, pixels of image, are loaded inti CA cells. A CA can be informally represented as a set of regularly and locally connected identical elements. The elements can be only in a finite set of states. The CA evolves in discrete time steps, changing the states of elements according to a local rule, which is the same for all elements. The new state of each element depends on its previous state and on the state of its neighbourhood. The characteristic properties of CAs are therefore locality, discreetness and synchrony. More definitions about connectivity and CAs can be found in [9], [10].

II. Definitions and algorithm

We consider a CA as a 3D lattice network of unite cubes (cells) whose centres are in integer lattice. For simplicity, we suppose that the lattice has $N = L \times L \times L$ cells $c_{i,j,k}$ with positions determined by indices $i, j, k = 1, ..., L$ in x, y and z directions, respectively, with $L \geq 3$. Each cell can exist in a finite number of states marked by colors. The cells can change their states at the end of time-steps that are discrete moments in time after the computing of time-steps is completed. The state of the cell $c_{i,j,k}$ in a time-step t is denoted by $c_{i,j,k}(t)$ and the state of all lattice cells by C_t, $t \geq 0$.

The initial configuration C_0 is represented by a 3D grid of cube cells and each cell can exist in two different states denoted by white or black colors. Boundaries of the grid are black. All the remaining cells of the grid, are coloured (labeled) white with the probability p and black with probability $1 - p$. The probabilities are independent for each cell. Example of an initial configuration is in Figure 1.

Two cells $c_{i,j,k}$ and $c_{l,m,n}$ are $6 - neighbours$ if $|i - l| + |j - m| + |k - n| = 1$. An ordered sequence of cells $c_1, ..., c_n$ is called $6 - path$ if each cell c_i is a 6-neighbour

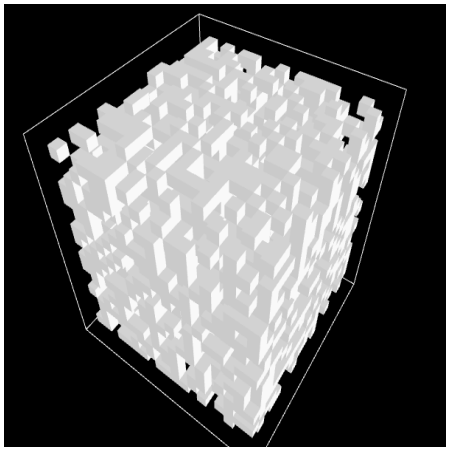


Figure 1: Initial grid configuration generated with $p = 0.4$.

of the next cell c_{i+1} in the sequence, $i = 1, ..., n-1$, $n \geq 2$. Set of cells is $6-$*connected*, if for any two cells there exists a 6-path of cells, within the set, between them. A maximum 6-connected set of white cells is called a *component*. In [7] can be find definitions for 18 and 26 connectivity in 3D. For short notation, the relative positions of the cells $c_{i+1,j,k}$, $c_{i-1,j,k}$, $c_{i,j+1,k}$, $c_{i,j-1,k}$, $c_{i,j,k+1}$, $c_{i,j,k-1}$ to the cell $c_{i,j,k}$ are denoted by E, W, N, S, U, D, respectively.

The goal of our algorithm is to assign a unique label to each of the components. The algorithm is similar for 18 and 26 connectivities, for 2D labeling connected components (4 and 8 - connectivity) and also can be generalized for n-dimension, $n > 3$.

III. Algorithm and experimental results

Let $i, j, k \in \{0, 1, 2, ..., L-1\}$ and $t \in N, t \geq 0$. Denote by $c_{(i,j,k)}(t)$ the state of a cell (i, j, k) in time step t and by $c(t)$ an argument of a local transition function, which is an ordered collections of 6-neighbours cell's states in time step t.

We use the terminology of colors. A cell is in state "m" if it is coloured by color m. 0-color is white, 1-color is black, m-color is a color with code m, $m > 1$, e.g. in RGB implementation.

Let C_0 be an initial configuration. We will define two CAs A_i by their local transition functions φ_{A_i}, $i = 1, 2$.

Step 1: Each component has the lowest down, right and south cell. The CA A_1 makes unique labeling, by changing states of white cells whose E, S, D 6-neighbours are black. The unique colors can be determined from the cells' positions and must be different from white and black. The color we denoted by $col(i, j, k)$ for the cell (i, j, k). On this way all the lowest down, right and south cells in all components are labeled by unique colors. Also, some other cells change their states. The local transition function is defined by $\varphi_{A_1}\left(c_{i,j,k}(0)\right) = c_{i,j,k}(1)$, where

$$c_{i,j,k}(1) = \begin{cases} col(i,j,k),\ c_{i,j,k}(0) = 0 \wedge c_{i+1,j,k}(0) = \\ \quad = c_{i,j-1,k}(0) = c_{i,j,k-1}(0) = 1 \\ c_{i,j,k}(0), otherwise \end{cases}$$

For the labeling the CA A_1 makes one step. On figure 2 is the configuration on the end of the step for the initial configuration from figure 1.

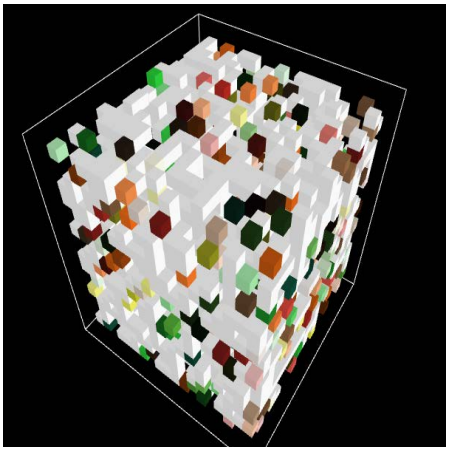


Figure 2: Grid configuration from Figure 1 after first step of the algorithm.

Step 2: Local transition function of the CA A_2 is defined by $\varphi_{A_2}\left(c_{i,j,k}(t)\right) = c_{i,j,k}(t+1)$, $t \geq 0$, where

$$c_{i,j,k}(t+1) = \begin{cases} max\left(c(t)\right),\ c_{i,j,k}(t) \neq 1 \\ c_{i,j,k}(t), otherwise \end{cases}$$

CA A_2 will color every white component with different color (not black or white) using iteration, until the state of the lattice becomes constant.

Using the described CAs we have algorithm 1 for the labeling connected components.

Data: Initial configuration C_0
Result: Connected componets labeling for C_0
φ_{A_1};
while *exist cell with change* **do**
  φ_{A_2};
end
Algorithm 1: Algorithm for labeling 6-connected components

On Figure 3 is the configuration on the end of the algorithm for the initial configuration from Figure 1.

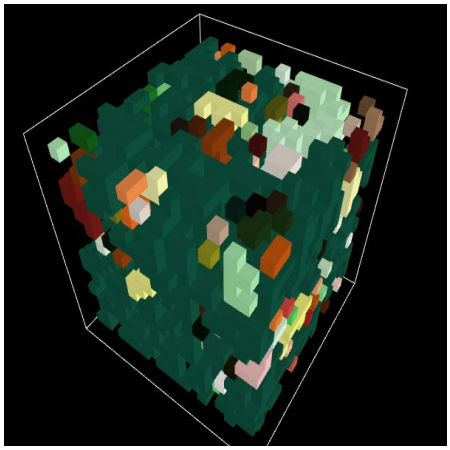


Figure 3: Grid configuration from Figure 1 on the end of the algorithm.

The implementation of the algorithm is made in NetLogo 5.0.4. The algorithm was extensively evaluated on various test cases for different size of grids and probabilities, i.e. densities of white cells in initial configuration. We calculated the average time steps required for 100 initial configurations of $31 \times 31 \times 31$ grid for the different probabilities (Table 1). However, the current implementation of the algorithm with NetLogo is limited with grid sizes and no possibility to use variables in RGB notation.

Probability	Mean value
0.2	18.11
0.25	30.25
0.3	69.92
0.35	131.44
0.4	87.13
0.45	75.55
0.5	68.82
0.55	68.43
0.6	68.16
0.65	67.35

Table 1: Average number of time steps on the $31 \times 31 \times 31$ *grid in 100 iterations with different probability of initial cell coloring*

IV. Conclusion and future investigations

The proposed CA algorithm has several advantages, e.g. it is not limited by the number of cells, its evolution is inherently parallel, and it has a strong resemblance to the important approaches in the nature like principles of cells or elementary particles. Complexity of the algorithm depends on the component shapes but an asymptotic complexity is $O(L)$ on an $L \times L \times L$ binary image. The algorithm use iterative function and can resolve the problem of stack overflow that could appear in recursive labeling which is studied in [11]. Drawbacks of the algorithm are in using global variables for stopping CA's work. Lack of global communication, implies problems related to global synchronization, data manipulation and inability for calculation of complex mathematical operations, however, these difficulties can be resolved by dedicated hardware resources.

The heterogeneous computing, supported today with data-flow approaches, FPGAs, SoCs, GPUs and manycore systems, are promising platforms for the implementation of the efficient CA based algorithms.

References

[1] Lifeng He, Yuyan Chao and Kenji Suzuki, "A New Two-Scan Algorithm for Labeling Connected Components in Binary Images," in *Proceedings of the World Congress on Engineering* , London, U.K., Vol II, July 2012, p. 1141-1146.

[2] Lifeng He, Yuyan Chao, Kenji Suzuki and Kesheng Wu, "Fast connected-component labeling," *Pattern Recognition*, vol. 42, 2009, 1977 - 1987.

[3] T. Y. Kong and A. Rosenffeld,, *Toplplogical algorithms for digital image processing*, Elsevier Science, Amsterdam, Netherlands, 1996.

[4] R.E. Cypher, L. Snyder and J. L. C. Sanz, "Practical Algorithms for Image Component Labeling on SIMD Mesh Connected Computers," *IEEE Transactions on Computers*, vol. 39, 1990, 276 - 281.

[5] Fei Zhao, Huan zhang Lu and Zhi yong Zhang, "Real-time single-pass connected components analysis algorithm," *EURASIP Journal on Image and Video Processing*, vol 2013, 2013.

[6] Kenji Suzuki, Isao Horiba, and Noboru Sugie, "Linear-time connected-component labeling based on sequential local operations," *Computer Vision and Image Understanding* , Vol 89, 2003, p. 1-23.

[7] Qingmao Hu, Guoyu Qian and Wieslaw L. Nowinski, "Fast connected-component labelling in three-dimensional binary images based on iterative recursion," *Computer Vision and Image Understanding* , Vol 99, 2005, p. 414-434.

[8] Christopher T. Johnston and Donald G. Bailey, "FPGA implementation of a Single Pass Connected Components Algorithm," in *4th IEEE Interrnational Symposium on Electroniic Design, Test and Appilcatoins* , DELTA 2008, Hong Kong, January 23-25, 2008, p. 228-231

[9] Biljana Stamatovic, Gregor Kosec, Roman Trobec, Xiao Xuan and Sinisa Stamatovic, "Cellular Automata Supporting n-Connectivity," *Mathematical Problems in Engineering*, Vol 2014, 2014.

[10] Biljana Stamatovic and Roman Trobec, "Data parallel algorithm in finding 2-D site percolation backbones," in *Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)* , Porto, Portugal, 2014, p. 65-70.

[11] G. Borgefors, I. Nystrom and G.S.D. Baja, Connected components in 3D neighbourhoods," in *Proceedings of the 10th Scandinavian Conference on Image Analysis*, 1997, p. 567-572.

Nature-Inspired Algorithm for Solving NP-Complete Problems

Atanas Hristov

University of Information Science and Technology, Ohrid, Macedonia
atanas.hristov@uist.edu.mk

Abstract

High-Performance Computing has become an essential tool in numerous natural sciences. The modern high-performance computing systems are composed of hundreds of thousands of computational nodes, as well as deep memory hierarchies and complex interconnect topologies. Existing high performance algorithms and tools already require courageous programming and optimization efforts to achieve high efficiency on current supercomputers. On the other hand, these efforts are platform-specific and non-portable. A core challenge while solving NP-complete problems is the need to process these data with highly effective algorithms and tools where the computational costs grow exponentially. This paper investigates the efficiency of Nature-Inspired optimization algorithm for solving NP-complete problems, based on Artificial Bee Colony (ABC) metaheuristic. Parallel version of the algorithm have been proposed based on the flat parallel programming model with message passing for communication between the computational nodes in the platform and parallel programming model with multithreading for communication between the cores inside the computational node. Parallel communications profiling is made and parallel performance parameters are evaluated on the basis of experimental results.

Keywords Artificial Bee Colony, High-Performance Computing, Parallel Algorithm, NP-complete problems, message passing, multithreading

I. Introduction

Accelerating the development and deployment of advanced computing systems and cloud computing platforms will require a comprehensive strategy integrating efforts from invention to deployment. The modern high-performance computing systems (HPCS) are composed of hundreds of thousands of computational nodes, as well as deep memory hierarchies and complex interconnect topologies. Existing high performance algorithms and tools already require courageous programming and optimization efforts to achieve high efficiency on current supercomputers. On the other hand, these efforts are platform-specific and non-portable. Currently, most of the HPCS are based on convectional sequential programming languages like C, C++, FORTRAN, etc. In order to achieve better parallel performance the flat parallel programing model with message passing in distributed memory systems, supported by the MPI standard and parallel programming model with multithreading in shared memory systems using the OpenMP programming interface have been included as a template libraries. The main disadvantages of the parallel programming based on conventional programming languages are: process synchronization, deadlocks, workload balancing, and thread concurrency. In order to improve this situation, Intel provides a range of tools specifically designed to help developers parallelize their applications. Three sets of complementary models for multithreading programming in shared memory systems are supported by Intel: Intel Cilk Plus, Intel Threading Building Blocks (Intel TBB) and Intel Array Building Blocks (Intel ArBB). The main purpose of those models is to increase the reliability, portability, scalability and the parallel performance of the application during the

multithreading execution [1, 2, 3].

The complexity class of decision problems NP-complete can be used as a pattern for benchmarking and parallel performance evaluation of HPCS. This paper investigates the efficiency of Nature-Inspired optimization algorithm for solving NP-complete problems, based on Artificial Bee Colony (ABC) metaheuristic. Highly parallel version of the well-known N-queens problem has been proposed based on ABC optimization. The parallel version of the algorithm have been proposed based on the flat parallel programming model with message passing for communication between the computational nodes in the platform and parallel programming model with multithreading for communication between the cores inside the computational node. The Intel Threading Building Blocks (TBB) programming model has been chosen as a standard for multithreading computations in shared memory systems. The Message Passing Interface (MPI) has been chosen as standard for communication in distributed memory systems.

II. Background

The complexity class of decision problems NP-complete can be used as a pattern for benchmarking and parallel performance evaluation of HPCS. The main idea behind using NP-complete problems for evaluation of the overall parallel performance of the HPCS is that those problems cannot be solved in polynomial time in any known way which require high computational power and time. The N-queens problem belongs to the class of NP-complete problems, requiring a brute-force algorithm for finding all possible solutions. The N-queens problem is formulated as solving the task to place N queens on N x N chessboard in such way that no queens attack each other, i.e. on every row, column or diagonal, there is only one queen. The complexity of this algorithm is O(N!), which comes from the fact that there are (N2!)/(N!*(N2-N)!) possible solutions to place the queens on the board [4, 5, 6]. In order to provide efficient solutions for this problem and to minimize the time and space complexity, many various optimization techniques have been proposed. In this paper we provide the experimental results gained by solving N-Queens problem, using Artificial Bee Colony (ABC) optimization. The ABC algorithm belongs to the class of nature-inspired algorithms, which simulate the behavior of the honey bee swarms in the nature. The main advantage of ABC algorithm is that uses only common control parameters such as colony size and maximum cycle number. The ABC algorithm is very powerful optimization tool which provide a population based search procedure. The ABC algorithm also combine local search methods, by using artificial bees which fly around multidimensional search space, with global search methods, by using another kind of artificial bees which fly and choose the food source randomly without any experience and memorize the new position if it is a better than the one that is already in their memory. Thus lead to balancing of the exploration and exploitation process. The main idea behind using the ABC algorithm for solving the N-Queens problem is that ABC is very effective optimization algorithm for finding the best optimized solution, which is declared according to the position of the food source, and the amount of nectar found in that solution. [7, 8, 9].

III. Resource planning while solving NP-complete problems

During the resource planning process while solving NP-complete problems two strategies can be applied: static and dynamic [10]. In the static resource planning process, each node executes only one part of the search tree. On Figure 1, an example of static resource planning on octal-core platform is given.

The main problem with static resource planning strategy is that the spatial search trees are highly unbalanced. This can lead to significantly reduce of the parallel performance of the application. This strategy can be improved by simultaneously scanning of larger number of subtrees in order to balance the load of computational nodes.

Dynamic resource planning strategies for solving NP-complete problems, allow scanning a large number of subtrees simultaneously, providing better load balancing and higher parallel efficiency. In this paper two dynamic strategies have been proposed. The first strategy proposed a model in which the main process searching in depth the spatial search tree and the gener-

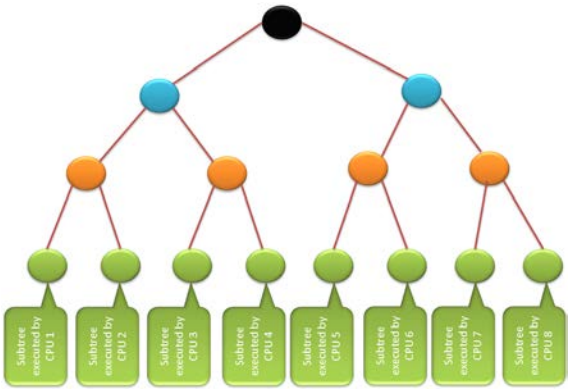


Figure 1: Static resource planning while solving NP-complete problems.

ated subtrees from this process are distributed among all available processes of the system.

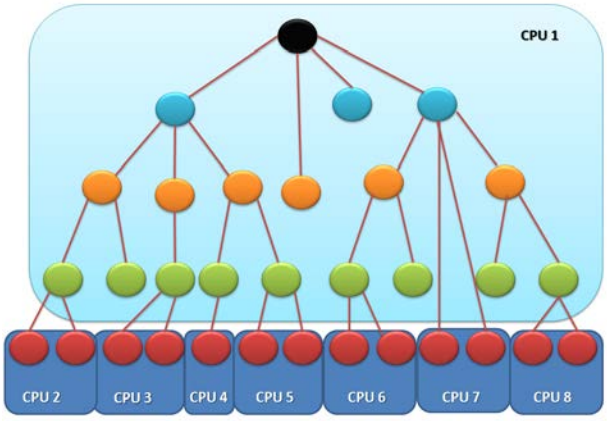


Figure 2: Depth search dynamic resource planning strategy while solving NP-complete problems.

On Figure 2, an example of dynamic resource planning by using depth search strategy on octal-core platform is presented. Thus the probability of an unbalanced load significantly reduced, while potential parallel performance of the system dramatically increases.

Another strategy for dynamic resource planning is by maintaining a list of nodes of the spatial search tree. All problems and subproblems from the spatial search tree are placed in the list and each process takes the last unprocessed knot.

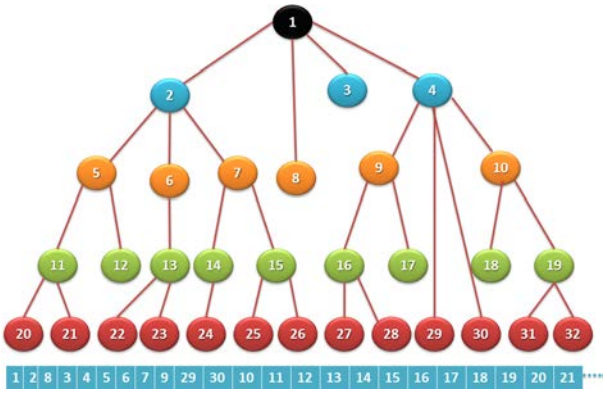


Figure 3: Maintaining a list of nodes dynamic resource planning strategy while solving NP-complete problems.

On Figure 3, an example of dynamic resource planning by using maintaining a list of nodes dynamic strategy is presented. This strategy provides an efficient way to allocate the workload among computational nodes of the system even in highly unbalanced trees of the search space. Thus the efficiency of the parallel algorithm, as well as the potential parallel performance of the system does not depend on the balance of the tree. The main disadvantage of this strategy is that it requires additional system resources for maintenance and searching in the list of nodes.

IV. Parallel implementation of ABC algorithm

An effective resource utilization of the modern high performance computing platforms is a subject for many scientific research investigations. The resource management optimization for those platforms is an essential part for optimal resource allocation while solving NP hard problems. The proposed algorithm for solving NP-Complete problems is based on Artificial Bee Colony (ABC) metaheuristic. In this paper we will present a nature-inspired approach for solving the N-queens problem which belongs to the class of

NP-complete problems. The ABC simulates the collective behavior of the honeybees in nature. The basic approach during implementation process is building a computer model which will simulate the collective behavior of the bees while collecting nectar. Parallel computing model for solving N-queens problem based on Artificial Bee Colony (ABC) metaheuristic is present on Figure 4.

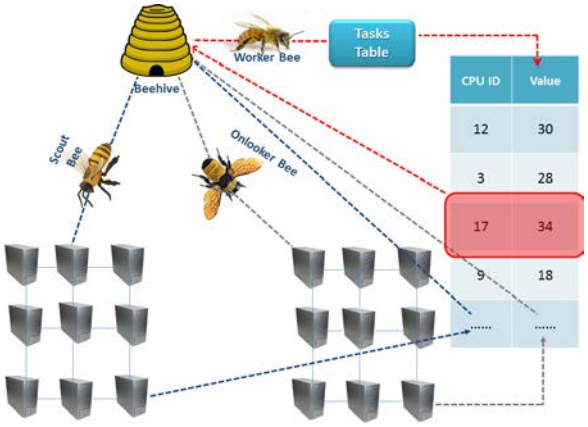


Figure 4: Parallel computing model for for solving N-queens problem based on Artificial Bee Colony metaheuristic.

In the proposed parallel model the number of all possible solutions should be determinate, which technically is the number of employed bees. Initially, a random population is generated, followed by repeating cycles of searching for employed, scout, and onlooker bees. The employed bee makes a modification of its initial food source in her memory and therefore finds a new food source. If the amount and quality of the nectar from the second position is better than, the employed bee forgets its initial food source. When the employed bees finish the search process, they pass their information to the onlookers located at the base. Then each onlooker makes a calculation on the received information and determines the food source. In order to implement the parallel algorithm for solving the N-Queens problem, the IntelTBB programming model and ABC metaheuristic have been used. The proposed algorithm used dynamic resource allocation. The activities of each beehive simulate one processor, while the actions of bees simulate threads. The number of bees that simulate each thread depends on the architecture of the target platform. The algorithm supports two types of global data: table of available resources and a table of unfinished tasks. The tables should be visible to all bees as bees carry out direct access to the data founded in the tables.

In the proposed algorithm, the bees are located in beehive, so call beehive of the scout bees and beehive of the onlooker and worker bees. Initially, the main problem is divided in several sub-problems, which are stored to the table of outstanding tasks. When the algorithm is started, the beehive generates N number of scout bees, where N represents the number of processors in the system. Each scout bee checks whether a processor is free or busy by execution of specific task on it. If a free resource is found the scout bee record the ID of the processor into the table of available resources. Also by executing specific piece of code, the scout bee determinate the value of the processor, which basically evaluate the suitability of the processor to execute specific tasks. Depending of the suitability of the processor, the scout bee gains a value to the processor. Once these operations are done, the scout bee returns to the beehive, where it is terminates. On the next step, the beehive generates M number of onlooker bees, where M is the optimal number of parallel threads. After generation, the onlooker bees search in to table of available resources. If the onlooker bee finds a free resource, it takes the ID of the processor and removes it from the table. The priority is given to the processor with highest value from the table. If the onlooker bee didn't find any free resource in the table, the bee will return to the beehive and will be terminate. Once the onlooker bee takes the available resource it starts to behave as a worker bee. Thus obtained K number of worker bees initially turned to the table of outstanding tasks where they taking certain sub-problem, remove it from the table and submit it for execution by the processor which ID has been taken from the table of available resources. Once the processor solves a sub-problem, it provides the solution to a worker bee. The worker bee with the current solution returns to beehive 2, where it is terminated.

V. Experimental Results

The proposed algorithm for parallel solving of the N-queens problem, based on metaheuristic ABC, is verified and its effectiveness has been studied experimentally based on multithreaded implementation using Intel Threading Building Blocks (TBB) programming model. The target multiprocessor platform for conducting experimental results is IBM Blade HS22 server with two quad-core processors Xeon Quad Core 2.00GHz, 6GB RAM, operating system Windows Server 2008. For implementation of the parallel algorithm and TBB programming model, Intel Parallel Studio 2010 program environment have been used. Experimental results were conducted for a different workload, i.e. for different size of the chessboard: 8x8, 12x12, 14x14, and 16x16. Also, two parallel versions of N-Queens problem have been tested: the first one based on the proposed ABC algorithm and the second one based on well-known backtracking algorithm. On Figure 5 the executional time while solving the N-Queens problem using sequential, ABC, and backtracking algorithm is given.

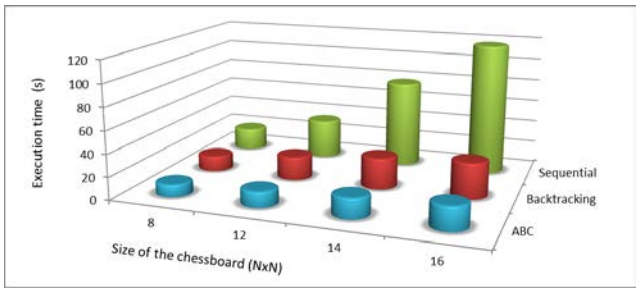


Figure 5: Executional time while solving the N-Queens problem using sequential, ABC, and backtracking algorithm.

From the chart shown on Figure 2 can be concluded that for size of the chessboard up to 10x10, the executional times of serial and parallel implementation of the program are relatively close due to the very short calculation time of the problem. The overall calculation time for the mention size of the chessboard is in the range of few milliseconds up to few seconds. On the other hand, when the size of the chessboard increases, the number of possible optimal and suboptimal solutions growths exponentially. The execution time of the sequential algorithm also growths exponentially, but the potential parallelism of the application increases for a given factor. This leads to very high executional time for sequential algorithm, and slightly increases in the parallel execution of the program.

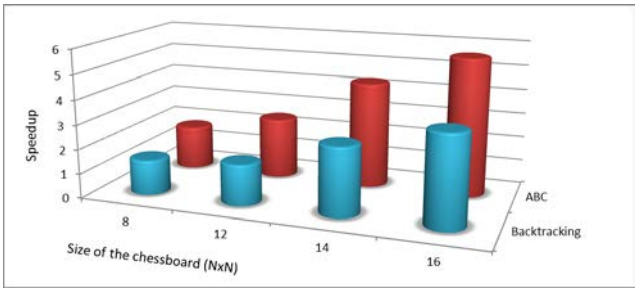


Figure 6: Executional time while solving the N-Queens problem using sequential, ABC, and backtracking algorithm.

On Figure 6 the speedup gained while solving the N-Queens problem using ABC and backtracking algorithm is given. When the size of the board increases to 16x16, the speedup gained by program implementation on octal-core platform using backtracking search algorithm is 3.68, while using ABC algorithm 5.2. The main reason behind this is that the ABC algorithm has better workload balance and better data structure in the search space through creating tables with tasks and adequate resources allocation of the relevant subproblems implementation.

VI. Conclusion and future work

An effective resource utilization of the modern high performance computing (HPC) systems is a subject for many scientific research investigations. The resource management for those platforms is an essential part for optimal resource allocation while solving NP complete problems. An effective parallel algorithm strongly determines the overall parallel performance of the high-performance computing system. This paper suggests an innovative algorithm for solving N-queens problem on multi-processor platforms based on parallel metaheuristic "Artificial Bee Colony" (ABC) optimization. The efficiency of the proposed algorithm was evaluated

on the basis of the software tools of Intel Array Building Blocks build-in Intel Parallel Studio. The proposed parallel implementation was developed on the basis of Message Passing Interface (MPI) and Intel Threading Building Blocks (TBB) programming models. Finally, we applied the proposed algorithm on IBM Blade HS22 server with two quad-core processors. This allows us to observe the behavior of the cluster while solving the N-queens problem. From the experimental results we conclude that the speedup gained by program implementation on octal-core platform using backtracking search algorithm is 3.68, while using ABC algorithm 5.2. Future objectives of this research include implementation of our algorithm on very large-scale systems and on the new generation of ExaScale machines.

Acknowledgment

The results reported in this paper are part of the research project, Center of excellence "Supercomputing Applications" - DCVP 02/1, supported by the National Science Fund, Bulgarian Ministry of Education and Science.

References

[1] Kristof P., Hongtao Yu, Zhiyuan Li, and Tian X., "Performance Study of SIMD Programming Models on Intel Multicore Processors," in *26th International Symposium in Parallel and Distributed Processing*, Shanghai, China, 21-25 May 2012, pp. 2423-2432.

[2] Wooyoung Kim and Voss M., "Multicore Desktop Programming with Intel Threading Building Blocks," *IEEE Software journal* , vol. 28, no. 1, pp. 23-31, 2011.

[3] Newburn C.J., Byoungro So, Zhenying Liu, McCool M., Ghuloum A., and Toit S.D., "Intel's Array Building Blocks: A retargetable, dynamic compiler and embedded language," in *9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, Chamonix, France, 02-06 April 2011, pp. 224-235.

[4] Khademzadeh A., Sharbaf M.A., and Bayati A., "An Optimized MPI-based Approach for Solving the N-Queens Problem.," in *7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Victoria, BC,Canada, 12-14 November 2012, pp. 119-124.

[5] P. Panwar, V. P. Saxena, A. Sharma, and V. Sharma, " Load Balancing using N-Queens Problem," *International Journal of Engineering Research Technology* , vol. 2, no. 1, 2012.

[6] Ayala A., Osman H., Shapiro D., and Desmarais J.M., "Accelerating N-queens problem using OpenMP," in *6th IEEE International Symposium on Applied Computational Intelligence and Informatics*, Timisoara, Romania, 19-21 May 2011, pp. 535-539.

[7] Teodorovic D., Lucic P., and Markovic, G., "Bee Colony Optimization: Principles and Applications," in *8th Seminar on Neural Network Applications in Electrical Engineering*, Belgrade, Serbia, 25-27 September 2006, pp. 151-15.

[8] Banharnsakun A., Achalakul T., and Sirinaovakul B., "Artificial bee colony algorithm on distributed environments," in *Second World Congress on Nature and Biologically Inspired Computing*, Fukuoka, Japan, 15-17 December 2010, pp. 13-18.

[9] Marinakis Y., Marinaki M., and Matsatsinis N., "A hybrid discrete Artificial Bee Colony - GRASP algorithm for clustering," in *International Conference on Computers Industrial Engineering*, Troyes, France, 6-9 July 2009, pp. 548-553.

[10] Xiaozhong G.,Gaochao Xu, and Yuan Z., "Dynamic Load Balancing Scheduling Model Based on Multi-core Processor," in *Fifth International Conference on Frontier of Computer Science and Technology*, Changchun, Jilin Province, 18-22 August 2010, pp. 398-403.

Log File Analysis in Cloud with Apache Hadoop and Apache Spark

Ilias Mavridis
Aristotle University of Thessaloniki, Greece
imavridis@csd.auth.gr

Eleni Karatza
Aristotle University of Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

Log files are a very important set of data that can lead to useful information through proper analysis. Due to the high production rate and the number of devices and software that generate logs, the use of cloud services for log analysis is almost necessary. This paper reviews the cloud computational framework Apache™ Hadoop®, highlights the differences and similarities between Hadoop MapReduce and Apache Spark™ and evaluates the performance of them. Log file analysis applications were developed in both frameworks and performed SQL-type queries in real Apache Web Server log files. Various measurements were taken for each application and query with different parameters in order to extract safe conclusions about the performance of the two frameworks.

Keywords Log analysis, Cloud, Apache Hadoop, Apache Spark, Performance evaluation

I. Introduction

The log files are a rich source of information that can be used for various purposes. However, the high production rate and the diversity between the logs makes it difficult to analyze. The log production rate can reach to several TeraBytes (TB) or PetaBytes (PB) per day, for example Facebook dealt with 130 TB of logs every day [1] in 2010 and in 2014 they have stored 300 PB of logs [2]. For these reasons conventional database solutions can't be used for the analysis, but cloud or even interconnected cloud systems [3] required to achieve scalability and elasticity. Many big companies like Facebook, Amazon, ebay, etc. use cloud computing to analyze logs. Also from academia there are many papers which investigate cloud computing (mainly Hadoop) to analyze logs [4] - [14].

Hadoop is the framework that has mainly been used to store and analyze data. Hadoop was designed for batch processing providing scalability and fault tolerance but not fast performance [15]. It enables applications to run in thousands of nodes with Petabytes of data. Hadoop responds to the large amount of logs by breaking up log files into blocks and distribute them to the nodes of the Hadoop cluster. It follows a similar strategy for computing by breaking jobs into a number of smaller tasks that will be executed in nodes of the cluster.

However, Hadoop's performance is not suitable for real-time applications [16] because it frequently writes and reads data from the disk. Spark solves this problem by minimizing these data transfers from and to disk by using effectively the main memory and performing in-memory computations. Also it provides a new set of high-level tools for SQL queries, stream processing, machine learning and graph processing [17].

Our work complements existing research by investigating and comparing log file analysis in Hadoop and Spark. The rest of the paper is organized as follows. Section II provides an overview of related research. Section III describes briefly what is log file and log file analysis in cloud. Section IV outlines the two open source computing frameworks Hadoop and Spark. Section V describes the setting of our experiments. Section VI presents the experimental results. Finally Section

VII concludes this paper.

II. Related Work

There are many papers whose authors investigate and propose the use of cloud computing to log file analysis. Paper [4] discusses the differences between the traditional relational database and big data. The authors claim that log files were produced in higher rate than traditional systems can serve and show experimental log file analysis through Hadoop cluster.

Paper [5] presents a weblog analysis system based on the Hadoop HDFS, Hadoop MapReduce and Pig Latin Language. The system aims to assist administrator to quickly analyze data and take business decisions. It provides an administrators monitoring system, problem identification and system's future trend prediction.

Also in [6] the authors discuss a Hadoop based system with Pig for web log applications. A web application has been created to distributed store log files on the Hadoop cluster, run MapReduce jobs and display results in graphical formats like bar charts. They have conclude that by this way there is a significant response time improvement and that MapReduce can successfully and efficiently process large datasets.

In line with [5] and [6], paper [7] proposes a mass log data processing and data mining method based on Hadoop to achieve scalability and high performance. To achieve scalability and reliability the log data are stored in HDFS and it is used Hadoop's MapReduce for high performance. In this case also, the experimental results show that the Hadoop based processing improves the performance of querying.

The paper [8] presents a scalable platform named Analysis Farm, for network log analysis, fast aggregation and agile query. To achieve storage scale-out, computation scale-out and agile query, OpenStack has been used for resource provisioning, and MongoDB for log storage and analysis. In experiments with Analysis Farm prototype with 10 MongoDB servers, the system managed to aggregate about 3 million log records in a 10-minute interval time and effectively query more than 400 million records per day.

A Hadoop based flow logs analyzing system has been proposed in paper [9]. This system uses for log analysis a new script language called Log-QL, which is a SQL-like language. After experiments they concluded that their distributed system is faster than the centralized system.

Paper [10] presents a cloud platform for batch log data analysis with Hadoop and Spark. The authors propose a cloud platform with batch processing and in-memory computing capabilities by combining Hadoop, Spark and Hive/Shark. The proposed system manage to analyze logs with higher stability, availability and efficiency than standalone Hadoop-based log analysis tools.

In paper [11] has been implemented a Hadoop MapReduce-based framework to analyze logs for anomaly detection. First they collect the system logs from each node of the monitored cluster to the analysis cluster. Then, they apply the K-means clustering algorithm to integrate the collected logs. After that, they execute a MapReduce-Based algorithm to parse these clustered log files. By this way, they can monitor the distributed cluster status and detect its anomalies.

Log file analysis can also be used for system threats and problem identification. Paper [12] presents a new approach which uses a MapReduce algorithm for log analysis to provide appropriate security alerts or warnings. They achieve a significant improvement in response time for large log file analysis and as a result to a faster reaction by the administrator.

In [13] the authors describe an approach which uses log file analysis for intrusion detection. The objective of the paper is to enhance the throughput and scalability by using Hadoop MapReduce and cloud computing infrastructure. They describe the architecture and implement performance analysis of an intrusion detection system based on Cloud Computing. From the experiments they conclude that the system fulfills the scalability, fault tolerant and reliability expectations which is designed for.

Finally [14] presents SAFAL, a Spatio-temporal Analyzer of FTP Access Logs collected by UNAVCO's data center. These logs contain massive amounts of data like borehole seismic, strainmeter, meteorological, and digital imagery data. The system was developed using MapReduce/Hadoop in order to identify trends in GPS data usage. They conducted several experiments and found that SAFAL was able to analyze very efficiently millions of lines of FTP access logs. Also

the authors conclude that it could be possible to create near real time maps by the analysis of the logs.

III. Log File Analysis

Log file analysis is the analysis of log data in order to extract some useful information [17]. As the log data come from many different systems in a variety of forms, a proper analysis requires a good knowledge of the system. It must be clear what is good and bad for a specific system and what is suspicious or not. Worth noting that a same value maybe is suspicious for a system but completely normal for another one.

III.1 Log Files

Each working computer system collects information about various operations. This information is stored in specific files which called log files [19]. Log files consist of log messages or simply log(s). A log message is what a computer system, device, software, etc. generates in response to some sort of stimuli [18]. The information that pulled out of a log message and declares why the log message generated is called log data [18].

A common log message includes the timestamp, the source, and the data. The timestamp indicates the time at which the log message was created. The source is the system that created the log message and the data is the essence of the log message. Unfortunately this format is not a standard and so the log message can be significantly different from system to system.

```
1 192.168.100.252 - - [31/Jul/2014:09:30:23 +0700] "GET /p4p/report_detail_edit.php?name_t=3 HTTP/1.1" 200 8013
2 110.168.229.112 - - [31/Jul/2014:09:30:25 +0700] "GET /p4p/detail.php HTTP/1.1" 200 5433
3 110.168.229.112 - - [31/Jul/2014:09:30:25 +0700] "GET /p4p/jquery/jquery.calendars.persian.js HTTP/1.1" 404 324
```

Figure 1: Apache web access log.

One of the most widespread types of log files in the web is the log files that were produced by the Apache HTTP Server [20]. Figure 1 shows the first three lines of a real Apache web access log file.

As shown in Figure 1, the first element of every row is the ip address of the client (e.g. 192.168.100.252) or may be the name of the node which made the request to the server. Then there are two dashes (- -) which means that there is no value for this two fields [20]. The first dash stands for the identity of the client specified in RFC 1413 [21], and the second dash represents the user id of the person requesting the server. The fourth element in each of these log messages indicates the date and time that the client's request had been served by the server and in the same brackets there is the server's time zone (e.g. +0700). Next in double quotes is the request line from the client. The request line first contains the method that has been used by the client (e.g. GET), then there is the requested source (e.g./`p4p/report_detail_edit.php?name_t=3`) and finally the used protocol (e.g. HTTP/1.1). At the end of each row there are two numbers that follows the request line.The first number is the status code that the server returns to the client (e.g. 200) and the last number indicates the size of the object returned to the client and is usually expressed in bytes (e.g. 8013).

III.2 Log Analysis in the Cloud

The rise of cloud computing and the growing requirement for processing and storing capabilities for log analysis, resulted in the combination of cloud computing and log analysis. It has emerged a new term, the Logging as a Service (LaaS). There are some cloud service providers that undertake to analyze the logs for one of their clients [22]. Users of such services can collect logs from various devices and software and submit them to the cloud for processing, as shown in Figure 2. The LaaS is a quite new cloud service but there are already providers like [23] and [24] that offer different service products. There are some features and capabilities common to all and some others that vary from provider to provider.

The main elements that LaaS has is the file uploading from the user to the cloud, indexing of data (for fast search, etc.), long-term storage and a user interface to search and review the data [18]. Most providers also supports various types of log formats and have their own API [23] [24]. Moreover, the majority of providers charge their services with the model "pay as you go", where the user is charged depending on the use of services has made.

On the other hand, there are differences in the way that each provider has developed its system [18]. Some providers have built their services to another

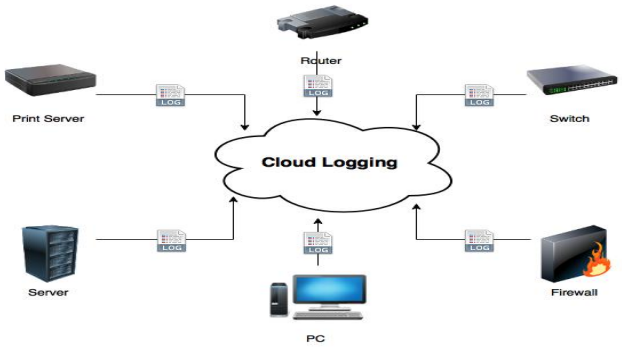


Figure 2: Logging as a Service (LaaS).

provider's cloud, while others in their own cloud infrastructure. Also, although all LaaS providers offer the possibility of long-term storage of data, the charges are not the same and the highest possible storage time differs also. Furthermore as it's expected the charges vary from provider to provider.

IV. The Computational Frameworks

The cloud computing frameworks that have been reviewed are the Hadoop and Spark. Hadoop is a well-established framework that has been used for storing, managing and processing large data volumes for many years by companies like Facebook, Yahoo, Adobe, Twitter, ebay, IBM, Linkedin and Spotify [25]. Hadoop is based on MapReduce programming model which is developed for batch processing. However the need for real-time data analysis leads to a new general engine for large-scale data processing, Spark. Spark was developed by AMPLab [26] of UC Berkeley and unlike the Hadoop's MapReduce, it uses the main memory, achieving up to 100 times higher performance for certain applications compared to Hadoop MapReduce [27].

IV.1 Apache Hadoop

Hadoop [28] is a framework for running applications on large clusters built of commodity hardware. It comes from Apache Nutch [29], which is an open source search engine. Key element to develop Hadoop were two Google papers, the first one was published in 2003 and describes the Google Distributed Filesystem -GFS [30] and the second one was published in 2004, and describes the MapReduce [31]. In February 2006 a part of Nutch became independent and created Hadoop. In 2010 a team from Yahoo began to design the next generation of Hadoop, the Hadoop YARN (Yet Another Resource Negotiator) or MapReduce2 [32].

The YARN changed the resource management to overcome the problems that had arisen, and also made the Hadoop capable of supporting a wide range of new applications with new features. The YARN is more general than the MapReduce (Figure 3), in fact the MapReduce is a YARN application. There are other YARN applications like Spark [33], which can run parallel to the MapReduce, under the same resource manager.

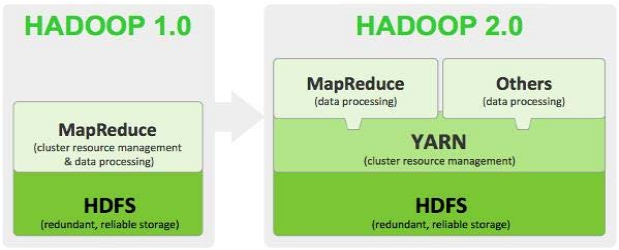


Figure 3: Hadoop 1.0 to Hadoop 2.0 [33].

Hadoop at its core lies at the HDFS [34] and the MapReduce computational model. However the term is also used for a set of related programs that used for distributed processing and processing of large-scale data, such as $Hive^{TM}$ [35], $Mahout^{TM}$ [36], $Zookeeper^{TM}$ [37] and others.

IV.1.1 Hadoop Distributed File System

The Hadoop distributed file system (HDFS) is created as a file system with blocks. As shown in Figure 4, the files are separated into blocks of a fixed size and stored at different nodes of Hadoop cluster [34]. Because the files are divided into smaller blocks, HDFS can store files much bigger than the disk capacity of each node. The stored files follows the write-once, read-many approach and can not be modified. On the other

hand there are the metadata files that describe the system and are changeable. There is a dedicated node called NameNode that stores all system's metadata and ensures that is always up to date.

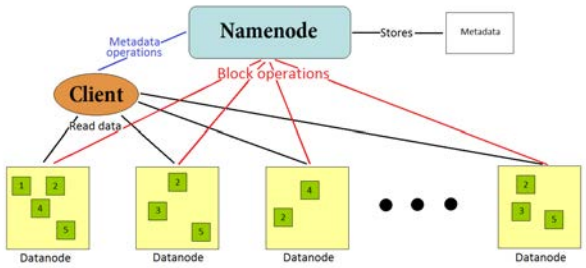


Figure 4: HDFS architecture.

The HDFS is implemented with the master/slave model. The NameNode is the master that manages the file system namespace and determines the client's access to the files. The slaves are called DataNodes and are responsible for storing the data and do anything that NameNode dictates them.

For fault-tolerance, HDFS replicates each block of a DataNode to other nodes [38]. To prevent disaster from Namenode failure, there is a secondary NameNode and replicas of the NameNode's metadata. Also worth noting that HDFS tries to respond to a read request with the closer copy to the reader (Rack Awareness) [34] in order to minimize the total bandwidth utilization and the reading time.

IV.1.2 MapReduce

MapReduce is a batch-based, distributed computing framework presented by Google's paper [31]. A MapReduce program consist of the Map Phase and the Reduce Phase [39]. Initially the data are processed by the map function and produce an intermediate result in the form of <Key, Value>. There can be many values with the same key. After that follows the reduce function. The reduce function performs a summary operation that process the intermediate results and generates the final result.

In the original version of the Hadoop MapReduce there are two types of nodes, the JobTracker (master) and TaskTrackers (slaves). In each MapReduce cluster there is a JobTracker that is responsible for resource management, job scheduling and monitoring [40]. The TaskTrackers run processes that were assigned to them by the JobTracker.

With Hadoop Yarn the execution model became more scalable and generic than the earlier version. The new Hadoop Yarn can run applications that do not follow the MapReduce model. With YARN, there is no longer a single JobTracker that does all the resource management, instead the ResourceManager and the NodeManager manage the applications. The ResourceManager is allocating resources to the different applications of the cluster. The ApplicationMaster negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor the component tasks [32].

IV.2 Apache Spark

Spark was developed in 2009 by AMPLab of UC Berkeley, and became an open source project in 2010 [41]. In 2013, the program was donated to the Apache software foundation and in February 2014 the Spark was a high-level program in the same foundation [42]. In November 2014, the engineering team at Databricks set a new record in large-scale sorting using Spark [43].

Spark extends the popular MapReduce model, supports more types of data processing and the combination of them, such as SQL-type queries and data flow processing. For ease of use, Spark has Python, Java, Scala and SQL APIs, and many embedded libraries.

One of the main features of Spark is the exploitation of main memory [44]. It may accelerate an application to one hundred times using memory and ten times using only the disc compared to Hadoop MapReduce cluster [41].

IV.2.1 Spark Ecosystem

Spark is a general purpose engine that supports higher-level items specialized to a particular kind of processing [41]. These components are designed to operate close to the core, and can be used as libraries during the development of a program.

The components of the Spark ecosystem are [41]:

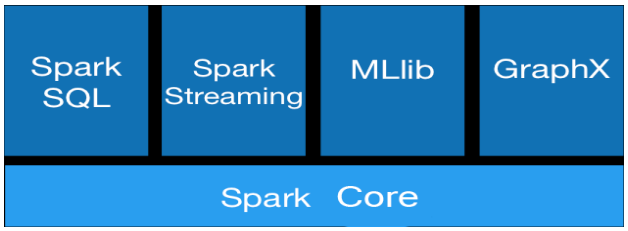


Figure 5: Spark ecosystem.

- Spark Core: Is the general execution engine for the Spark platform and every other functionality is built on top of it.
- Spark SQL: Is a Spark module for structured data processing. It can act as a distributed SQL query engine.
- Spark Streaming: Enables interactive and analytical applications across both streaming and historical data.
- MLlib: Is a scalable machine learning library.
- GraphX: Is a graph computation engine that enables users to manipulate and perform parallel processing in graphs.

IV.2.2 Resilient Distributed Dataset

Spark uses a new parallel and fault-tolerant data structure called Resilient Distributed Dataset (RDD) [45]. Spark automatically distributes the RDD data in the cluster and performs parallel processing on them. The RDDs can contain any object or class of Python, Java or Scala.

The RDDs supports two types of operations, transformations which generate a new dataset from an existing one, and actions which return a value after running a computation on a dataset [46]. For example, map is a transformation that passes each element of a RDD to a function and results to a new RDD with the computed values. On the contrary, reduce is an action that passes each element of a RDD to a function and returns a single value as a result.

To achieve efficiency Spark's transformations are "lazy" [47], which means that the computation of a new RDD is not executed immediately after the command is given. Instead, the transformations run only when an action needs the transformation result. On the other hand actions run immediately.

One of the most important capabilities of Spark is persisting or caching a dataset in main memory [47]. By maintaining a RDD in main memory, each node can perform much faster future computations in this dataset, often more than ten times faster. Also the cached RDD is fault-tolerant, which means that if a partition of RDD is damaged, then it will automatically recalculated with the proper transformations and will be replaced.

V. Experimental Setup

We have conducted a series of tests to experimentally evaluate the performance of the two frameworks. For this purpose has been developed a cluster with virtualized computing resources of Okeanos [48]. Okeanos is an IaaS (Infrastructure as a Service) Service, developed by the Greek Research and Technology Network [49]. It is offered to the Greek Research and Academic community and provides access to Virtual Machines, Virtual Ethernets, Virtual Disks, and Virtual Firewalls, over a web-based UI.

In these experiments have been used 5 nodes, 4 slaves and 1 master. The slaves have configured with 2 cpu units, 6GB memory, 40GB disk space and the master with 8 cpu units, 8GB memory and 40GB disk space.

The log file that has been used to testing is a real world log file. This file is an Apache HTTP Server log which is accessible through internet and was found after a relevant search. The log messages of this file have the form shown in Figure 1. To perform the experiments the two frameworks were installed to the cluster in the same nodes. The programs were developed in the same language (java) for both frameworks and the log files were saved in HDFS.

VI. Experimental Results

For each program, measurements were taken related to the execution time and the number of active slave nodes, the size of the input file and the type of program.

VI.1 Hadoop Expirements

We conducted different tests with different input file sizes, number of active nodes and programs. As shown in Figure 6 and Figure 7 the increment of the input file size results in the increment of the program's execution time. We also observe an increment in execution time when the number of active nodes is reduced, with a particularly big increment when remains only one slave node. These observations is completely reasonable because by these ways the processing volume for each node has been increased and as a result the processing time.

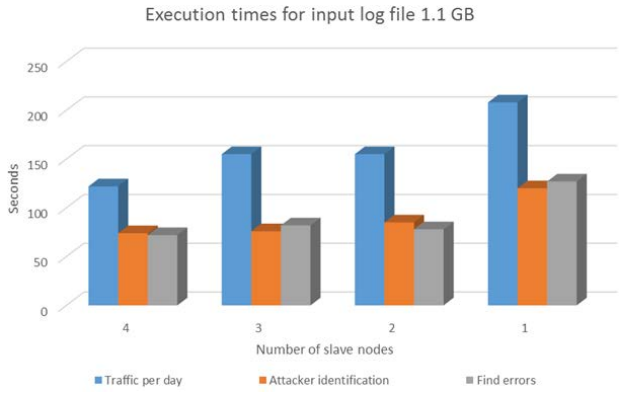


Figure 6: Execution times of Hadoop programs with 1.1GB input file.

Furthermore we see that the first program (blue) takes considerably more time to run. This is due to the nature of the program, because its Reduce phase requires much more processing work than the Reduce phase of the other two programs. And while Map processes have almost the same execution times, the big difference in Reduce process makes a difference in the end.

Moreover we observe that there is a slight difference in execution times between two or four nodes for the smaller file. This makes sense because the file is relatively small and two nodes have enough computing power to execute the required processes. On the other hand for the same reason we see that for the largest file each additional node makes a difference in execution time.

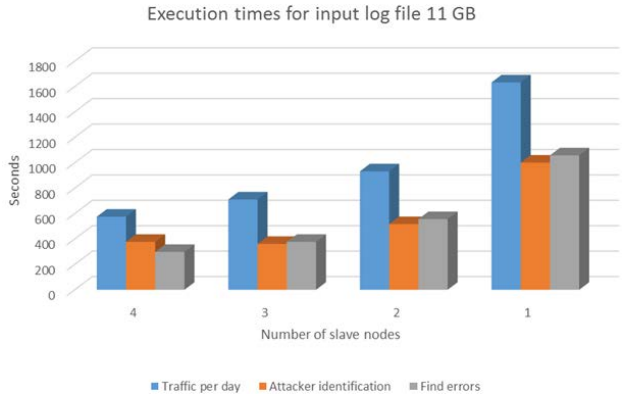


Figure 7: Execution times of Hadoop programs with 11GB input file.

Finally in Figure 7 all four nodes were better exploited due to the large input file. At this case the doubling of the active nodes leads to almost halve of the execution time (differs for program to program).

VI.2 Spark Expirements

In correspondence to Hadoop's experiments, relevant experiments carried out in Spark. Spark programs can run as a standalone Spark applications or executed on YARN. For the following tests the programs run as standalone Spark applications (both are supported from the developed system).

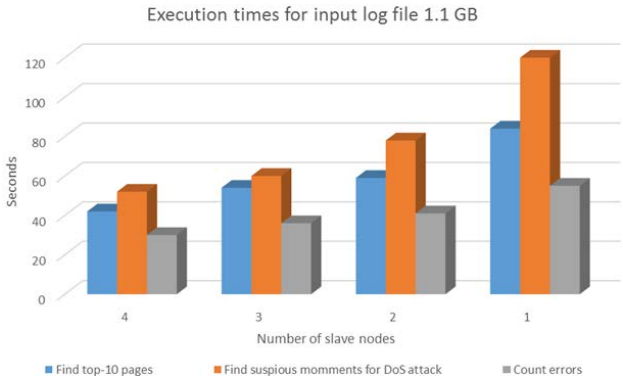


Figure 8: Execution times of Spark programs with 1.1GB input file.

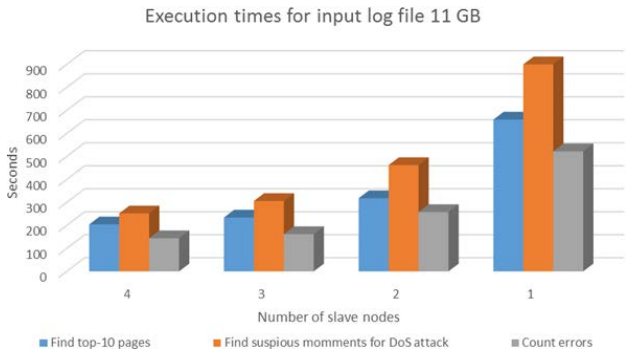


Figure 9: Execution times of Spark programs with 11GB input file.

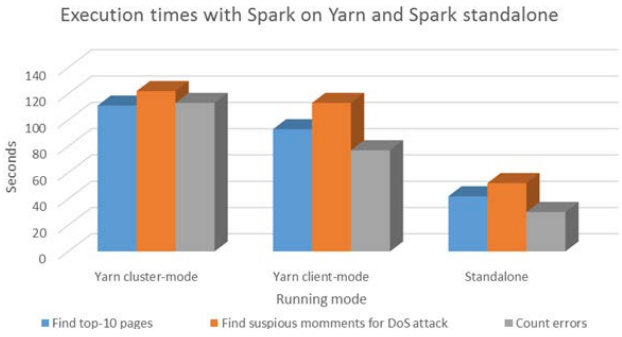


Figure 10: Spark on yarn and Spark standalone.

In these tests we generally observed that Spark's behavior is same as Hadoop. The increment of the size of the input file or the reduce of active slaves increase the program execution time especially when reamains active only one node. This is reasonable because -as explained previously- with these two ways the processing volume for each node has been increased and hence the processing time.

Furthermore in Figure 8 and Figure 9 we see that the three programs that were tested have different execution times, and even maintain the finish order. The third program (gray) has the less execution time, follows the first (blue) and finally the second (orange). This is because the programs require a different number and type of calculations, so the simpler program finishes first.

Also we observe that for input file of 1.1 GB there is a small difference in execution time with two to four nodes, because the file is relatively small and the process required can be carried out with two nodes. Same as Hadoop, for large file of 11 GB, each additional node makes a difference by contributing to the execution process.

In addition these programs are executed with the same input file in the same cluster but in a different way. Figure 10 shows the difference in the execution time according to whether the programs run on YARN or standalone. The execution of Spark programs on YARN offers additional features such as monitoring, dynamic resource management of the cluster, security through Kerberos protocol, possibility of parallel execution of various programs (e.g. MapReduce, Hive) and other features that are not supported by the standalone mode. However, as shown in Figure 10 the execution of programs on YARN is quite slower than standalone, that is because YARN has a quite complex resource management and scheduling compared to the Spark standalone and as a result there is a difference in execution time.

As shown in Figure 10, there are two types of yarn modes. In cluster-mode the driver runs in a process of the master who manages YARN. On the contrary in client-mode driver runs on client's process [50].

VI.3 SQL-type Queries Experiments

Figure 11 presents the performance comparison of Hadoop Hive and Spark SQL which are used for sql-type queries. For the experiments that we have contacted the Spark SQL runs in standalone mode and the executed queries are about error counting. The execution time of Spark SQL improved significantly when the table saved in main memory with the command CACHE TABLE tableName. As shown in Figure 11 the performance of Spark SQL is better than Hive. This happens because Spark SQL has a set of techniques to prevent reads and writes to disk storage, caching of tables in memory and optimizing efficiency.

VI.4 Overall Results

Various measurements have shown how systems react to changes in the size of the input file, the type of

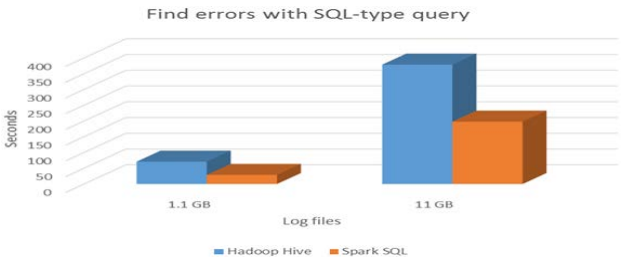


Figure 11: Execution times of SQL-type queries.

executed program and available nodes in the cluster. The two frameworks are highly configurable and their performance may vary depending on their settings. However by mainly maintaining presets and executing the same program with the same input data we can draw a conclusion.

To directly compare the two frameworks were executed on both the same programs. The first program is about error counting and the second is about errors finding. Figure 12 shows the performance of the frameworks for the first program, for input files of 1.1 GB and of 11 GB. The Spark presents the best performance, follows the Spark SQL, and then Hadoop MapReduce and Hadoop Hive with big difference in execution times. These results are in complete agreement with what has been previously described and confirm that the performance of Spark in most cases is better than Hadoop. The same conclusion comes from Figure 13 showing the performance of the second program for both frameworks.

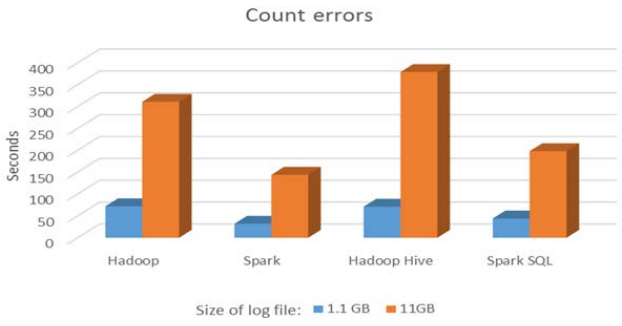


Figure 12: Execution times of count errors programs and SQL-type queries.

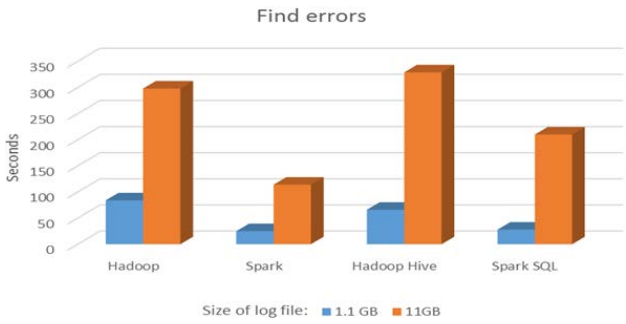


Figure 13: Execution times of find errors programs and SQL-type queries.

VII. Conclusions

This work aims to investigate the analysis of log files with the two most widespread computing frameworks in cloud computing, the well-established Hadoop and rising Spark. In the two frameworks have developed, executed and evaluated realistic programs for analyzing logs.

The two frameworks have the common goal of parallel processes execution on distributed files or other input files. The Hadoop is one of the first frameworks for cloud computing, is widely used and is one of the most active projects of the Apache foundation. Over the years Hadoop evolved and improved in order to meet the new era needs. These new needs led also to the creation of Spark.

Spark is different from Hadoop's MapReduce to two key points, which give Spark better performance and flexibility. The first is that Spark saves intermediate results in memory instead of the disk, thus it dramatically reduces the execution time. Secondly, Spark except of MapReduce functions, supports a wide range of new capabilities that can be combined to generate new powerful programs.

The various experiments that have been carried out show Spark's best performance. However, the programs were implemented in such a way to make possible the comparison between the two frameworks. As future work could be implemented programs that make full use of Spark capabilities in order to evaluate the performance of the framework for more complex

log analysis programs.

Acknowledgment

The authors would like to thank Okeanos the GRNET's cloud service for the valuable resources.

References

[1] https://www.facebook.com/notes/facebook-engineering/scaling-facebook-to-500-million-users-and-beyond/409881258919

[2] https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

[3] I.A. Moschakis and H.D. Karatza, "A meta-heuristic optimization approach to the scheduling of Bag-of-Tasks applications on heterogeneous Clouds with multi-level arrivals and critical jobs," *Simulation Modelling Practice and Theory*, Elsevier, vol. 57, pp. 1-25, 2015.

[4] B. Kotiyal, A. Kumar, B. Pant and R. Goudar, "Big Data: Mining of Log File through Hadoop," in *IEEE International Conference on Human Computer Interactions (ICHCI'13)*, Chennai, India, August 2013, pp. 1-7.

[5] C. Wang, C. Tsai, C. Fan, Sh. Yuan, "A Hadoop based Weblog Analysis System," in *7th International Conference on Ubi-Media Computing and Workshops (U-MEDIA 2014)*, Ulaanbaatar, Mongolia, July 2014, pp. 72-77.

[6] S. Narkhede and T. Baraskar, "HMR log analyzer: Analyze web application logs over Hadoop MapReduce," *International Journal of UbiComp (IJU)*, vol.4, No.3, pp. 41-51, 2013.

[7] H. Yu and D.i Wang, "Mass Log Data Processing and Mining Based on Hadoop and Cloud Computing," in *7th International Conference on Computer Science and Education (ICCSE 2012)*,Melbourne, Australia, July 2012, pp. 197.

[8] J. Wei, Y. Zhao, K. Jiang, R. Xie and Y. Jin, "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis," in *International Conference on Cloud and Service Computing (CSC)*, Hong Kong, China, December 2011, pp. 354-359.

[9] J. Yang, Y. Zhang, S. Zhang and Dazhong He, "Mass flow logs analysis system based on Hadoop," in *5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Guilin, China, November 2013, pp. 115-118.

[10] X. LIN, P. WANG and B. WU, "Log analysis in cloud computing environment with Hadoop and Spark," in *5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT 2013)*, Guilin, China, November 2013, pp. 273-276.

[11] Y. Liu, W. Pan, N. Cao and G. Qiao, "System Anomaly Detection in Distributed Systems through MapReduce-Based Log Analysis," in *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, Chengdu, China, August 2010, pp. V6-410 - V6-413 .

[12] S. Vernekar and A. Buchade, "MapReduce based Log File Analysis for System Threats and Problem Identification," in *Advance Computing Conference (IACC), 2013 IEEE 3rd Internationa*l, Patiala, India, February 2013, pp. 831-835.

[13] M. Kumar and Dr. M. Hanumanthappa, "Scalable Intrusion Detection Systems Log Analysis using Cloud Computing Infrastructure," in *2013 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, Tamilnadu, India, December 2013, pp.1-4.

[14] H. Kathleen and R. Abdelmounaam, "SAFAL: A MapReduce Spatio-temporal Analyzer for UNAVCO FTP Logs," in *IEEE 16th International Conference on Computational Science and Engineering (CSE)*, Sydney, Australia, December 2013, pp. 1083-1090.

[15] http://wiki.apache.org/hadoop/

[16] G.L. Stavrinides, H.D. Karatza, "A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds," in *3rd International Conference on Future Internet of*

Things and Cloud (FiCloud'15), Rome, Italy, August 2015, pp. 231-239.

[17] `https://databricks.com/spark`

[18] Dr. A. A. Chuvakin, K. J. Schmidt, Chr. Phillips, P. Moulder, *Logging and Log Management: the authoritative guide to understanding the concepts surrounding logging and log management*, Elsevier Inc. Waltham, 2013.

[19] J. Pinto Leite, "Analysis of Log Files as a Security Aid," in *6th Iberian Conference on Information Systems and Technologies (CISTI)*, Lousada, Portugal, June 2011, pp. 1-6.

[20] `http://httpd.apache.org/docs/1.3/logs.html`

[21] `http://www.rfc-base.org/rfc-1413.html`

[22] D. Jayathilake, "Towards Structured Log Analysis," in *9th International Joint Conference on Computer Science and Software Engineering (JCSSE 2012)*, Bangkok, Thailand, May âĂŞ June 2012, pp. 259 264.

[23] `www.loggly.com`

[24] `www.splunkstorm.com`

[25] `http://wiki.apache.org/hadoop/PoweredBy`

[26] `https://amplab.cs.berkeley.edu/software`

[27] R. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker and I. Stoica, "Shark: SQL and Rich Analytics at Scale," in *SIGMOD 2013*, New York, USA, June 2013, pp. 13-24.

[28] `http://wiki.apache.org/hadoop`

[29] `http://nutch.apache.org`

[30] G. Sanjay, G. Howard and L. Shun-Tak, "The Google File System," in *19th ACM Symposium on Operating Systems Principles*, Lake George, NY, October 2003.

[31] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, December 2004.

[32] `http://hortonworks.com/hadoop/yarn/`

[33] `http://wiki.apache.org/hadoop/PoweredByYarn`

[34] `http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html`

[35] `http://hive.apache.org/`

[36] `http://mahout.apache.org/`

[37] `http://zookeeper.apache.org/`

[38] `http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

[39] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters, " *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[40] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen and D. Chen, " G-Hadoop: MapReduce across distributed data centers for data-intensive computing, " *Future Generation Computer Systems*, vol. 29, no 3, pp. 739-750, 2013.

[41] `https://databricks.com/spark`

[42] `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50`

[43] `http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`

[44] `https://spark.apache.org/`

[45] M. Zaharia, M. Chowdhury, T. Das, Ank. Dave, J. Ma, M. McCauley, M. J. Franklin, Sc. Shenker and Ion Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *9th USENIX conference on Networked*

Systems Design and Implementation, CA, USA, 2012, pp. 2-2.

[46] http://spark.apache.org/docs/latest/programming-guide.html

[47] M. Zaharia, M. Chowdhury, M. J. Franklin, Sc. Shenker and Ion Stoica, "Spark: Cluster Computing with Working Sets," in *2nd USENIX conference on Hot topics in cloud computing*, CA, USA, 2010, pp.10-12.

[48] https://okeanos.grnet.gr

[49] Ev. Koukis and P. Louridas, "okeanos IaaS," in *EGI Community Forum 2012 / EMI Second Technical Conference*, Munich, Germany, March 2012.

[50] http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.3/bk_using-apache-hadoop/content/yarn_overview.html

NekBone with Optimized OpenACC directives

Jing Gong, Stefano Markidis, Michael Schliephake, Erwin Laure
PDC Center for High Performance Computing,
KTH Royal Institute of Technology, Sweden, and
the Swedish e-Science Research Centre (SeRC)
(gongjing,markidis,michs,erwinl)@pdc.kth.se

Luis Cebamanos
The University of Edinburgh, UK
l.cebamanos@epcc.ed.ac.uk

Alistair Hart
Cray Exascale Research Initiative Europe, UK
ahart@cray.com

Misun Min, Paul Fischer
Argonne National Laboratory, U.S.A
mmin,fischer@mcs.anl.gov

Abstract

Accelerators and, in particular, Graphics Processing Units (GPUs) have emerged as promising computing technologies which may be suitable for the future Exascale systems. Here, we present performance results of NekBone, a benchmark of the Nek5000 code, implemented with optimized OpenACC directives and GPUDirect communications. Nek5000 is a computational fluid dynamics code based on the spectral element method used for the simulation of incompressible flow. Results of an optimized NekBone version lead to 78 Gflops performance on a single node. In addition, a performance result of 609 Tflops has been reached on 16,384 *GPUs of the Titan supercomputer at Oak Ridge National Laboratory.*

Keywords NekBone/Nek5000, OpenACC, Spectral element method, GPUDirect

I. Introduction

There is a long history to employ GPUs to accelerate Computational Fluid Dynamic (CFD) codes [1, 2, 3]. However, most implementations use the Nvidia parallel programming and computing platform; CUDA. This means that developers need to rewrite their original applications in order to obtain a substantial performance improvement [4].

OpenACC [5] is a directive-based HPC parallel programming model, using host-directed execution with an attached accelerator device. In addition, GPUDirect for communication enables a direct path for data exchange between GPUs bypassing CPU host memory. In the paper, we extend the initial results [6, 7, 8] on porting Nek5000 to GPU systems, to enhance and optimize the performance on massively parallel hybrid CPU/GPU systems. In this implementation, the large-scale parallelism is handled by MPI, while OpenACC deal with the fine-grained parallelism of matrix-matrix multiplication.

The paper is organized as follows. In Section II we give an overview of the Nek5000 and NekBone code. In Section III we discuss in details regarding

optimized matrix-matrix multiplications and gather-scatter operators. The performance results are provided in Section IV. Finally, we summarize the results and further works.

II. Nek5000 and its NekBone benchmark

Nek5000 [9] is an open-source code for simulating incompressible flows using MPI for parallel communication. The code is widely used in a broad range of applications. The Nek5000 discretization scheme is based on the spectral-element method [10, 11]. In this approach, the incompressible Navier-Stokes equations are discretized in space by using high-order weighted residual techniques employing tensor-product polynomial bases.

In Nek5000, the derivatives in physical space can be calculated using the chain rule [12],

$$\frac{\partial \mathbf{U}}{\partial x_l} = \sum_{m=1}^{3} \frac{\partial \mathbf{U}}{\partial r_m} \frac{\partial r_m}{\partial x_l}. \quad (1)$$

Typically, this equation (1) is evaluated on the GLL points of N. In the case of three-dimensional with number of elements E, it creates an additional $9n$ memory references and $36n$ operations, where $n = E \cdot N^3$ is the total number of gridpoints. The tensor-product-based operator evaluation can be implemented as matrix-matrix products. This implementation of (1) makes possible to port the most time-consuming parts of the code into a GPU-accelerated system.

NekBone [13] is configured with the basic structure and user interface of the extensive Nek5000 software. NekBone solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient solver and exposes the principal computational kernel to reveal the essential elements of the algorithmic-architectural coupling that is pertinent to Nek5000. Consequently, the results from investigating the performance and profiling of NekBone can be directly applied to Nek5000.

III. Optimized OpenACC implementations

III.1 Matrix-Matrix Multiplications

The matrix-matrix multiplications are performed through the direct operator evaluation based on Equation (1). Algorithm 1 shows the pseudo-code of the `local_grad_acc` subroutine which computes the derivatives of U using CCE compiler.

Algorithm 1 CCE version for the final optimized derivative operations.

```
local_grad_acc
!$ACC DATA PRESENT(w,u,gxyz,ur,us,ut,wk,dxm1,dxtm1)
!$ACC PARALLEL LOOP COLLAPSE(4) GANG WORKER VECTOR
!$ACC& VECTOR_LENGTH(128)  PRIVATE(wr,ws,wt)
      do e = 1,nelt
         do k=1,nz1
         do j=1,ny1
         do i=1,nx1
            wr = 0
            ws = 0
            wt = 0
!$ACC LOOP SEQ
            do l=1,nx1    ! serial loop, no reduction needed
               wr = wr + dxm1(i,l)*u(l,j,k,e)
               ws = ws + dxm1(j,l)*u(i,l,k,e)
               wt = wt + dxm1(k,l)*u(i,j,l,e)
            enddo
            ur(i,j,k,e) = gxyz(i,j,k,1,e)*wr
     $                  + gxyz(i,j,k,2,e)*ws
     $                  + gxyz(i,j,k,3,e)*wt
         enddo
         enddo
         enddo
      enddo
!$ACC END PARALLEL LOOP
...
```

Algorithm 2 illustrates the use of OpenACC directives with PGI compiler. Here, the OpenACC directives `KERNELS` and `LOOP VECTOR` are used for an optimized performances with the PGI compiler.

The other optimized implementation evaluated in this paper is to call CUDA device functions from within OpenACC kernels. This can be done using the OpenACC directive `!$acc host_data use_device`. The OpenACC construct `host_data` indicates the address of device data available on the host, then the arrays are listed in the `use_device` clause within the `host_data` region. The compiler will generate code to use the device copy of the arrays, instead of the host copy. The interface implemented between OpenACC and CUDA functions is provided below.

```
!$acc host_data use_device(w,u,ur,us,ut,gxyz,dxm1,dxtm1)
   if (nx1.eq.8) then
     call ax_cuf8<<<nelt,dim3(nx1,ny1,nz1)>>>(w,u,
  $                ur,us,ut,gxyz,dxm1,dxtm1)
...
   else if (nx1.eq.14) then
     call ax_cuf14<<<nelt,dim3(nx1,ny1,nz1/2)>>>(w,u,
  $                ur,us,ut,gxyz,dxm1,dxtm1)
   else
     call ax_cuf16<<<nelt,dim3(nx1,ny1,nz1/4)>>>(w,u,
```

Algorithm 2 PGI version for the final optimized derivative operations.

```
local_grad_acc
 !$ACC DATA PRESENT(w,u,gxyz,ur,us,ut,wk,dxm1,dxtm1)
 !$ACC KERNELS
 !$ACC& GANG
      do e = 1,nelt
 !$ACC& LOOP VECTOR(NZ1)
         do k=1,nz1
 !$ACC& LOOP VECTOR(NY1)
         do j=1,ny1
 !$ACC& LOOP VECTOR(NX1)
         do i=1,nx1
            wr = 0
            ws = 0
            wt = 0
 !$ACC LOOP SEQ
            do l=1,nx1    ! serial loop, no reduction needed
               wr = wr + dxm1(i,l)*u(l,j,k,e)
               ws = ws + dxm1(j,l)*u(i,l,k,e)
               wt = wt + dxm1(k,l)*u(i,j,l,e)
            enddo
            ur(i,j,k,e) = gxyz(i,j,k,1,e)*wr
     $                  + gxyz(i,j,k,2,e)*ws
     $                  + gxyz(i,j,k,3,e)*wt
         enddo
         enddo
         enddo
      enddo
!$ACC END KERNELS
...
```

```
  $                  ur,us,ut,gxyz,dxm1,dxtm1)
   endif
   istat = cudaDeviceSynchronize()
!$acc end host_data
```

The utilization of shared memory in GPUs is very important for writing optimized CUDA code since access to shared memory is much faster than to global memory. Shared memory is allocated per thread block, therefore all threads in a block have access to the same shared memory. On NVIDIA Kepler GPUs with compute capability 3.x, shared memory has 32 banks, with each bank having a bandwidth of 64-bits per clock cycle. Kepler GPUs are configurable where either successive 32-bit words or 64-bit words are assigned to successive banks. This is particularly important for cases with higher polynomial degree. Considering that our matrix-matrix operations use three temporary arrays of size N^3 a total of 64KB shared memory is required for the case $N = 14$ with double precision. An example of such implementation in CUDA FORTRAN for polynomial degree 14 can be seen in Algorithm 3.

Algorithm 3 CUDA FORTRAN version for the final tuned derivative operations

```
subroutine local_grad_cuf14
     real, intent(out) :: w(lx1,ly1,lz1,lelt)
     real, intent(in)  :: u(lx1,ly1,lz1,lelt)

     real gxyz(lx1,ly1,lz1,2*ldim,lelt)

     real, intent(in) :: dxm1(lx1,lx1)
     real, intent(in) :: dxtm1(lx1,lx1)

     real rtmp,stmp,ttmp,wijk1e,wijk2e
     real, shared :: shdxm1(lx1,lx1)
     real, shared :: shdxtm1(lx1,lx1)
     real, shared :: shur(lx1,ly1,lz1)
     real, shared :: shus(lx1,ly1,lz1)
     real, shared :: shut(lx1,ly1,lz1)
     integer e,i,j,k,l

     e = blockIdx%x
     k = threadIdx%z
     j = threadIdx%y
     i = threadIdx%x

      if (k.eq.1) then
         shdxm1(i,j) = dxm1(i,j)
         shdxtm1(i,j) = dxtm1(i,j)
      end if
      call syncthreads()

      rtmp = 0.0
      stmp = 0.0
      ttmp = 0.0
      do l=1,lx1
        rtmp = rtmp+shdxm1(i,l)*u(l,j,k,e)
        stmp = stmp+shdxm1(j,l)*u(i,l,k,e)
        ttmp = ttmp+shdxm1(k,l)*u(i,j,l,e)
      enddo
          shur(i,j,k) = gxyz(i,j,k,1,e)*rtmp
     $                + gxyz(i,j,k,2,e)*stmp
     $                + gxyz(i,j,k,3,e)*ttmp
      rtmp = 0.0
      stmp = 0.0
      ttmp = 0.0
      do l=1,lx1
        rtmp = rtmp+shdxm1(i,l)*u(l,j,k+7,e)
        stmp = stmp+shdxm1(j,l)*u(i,l,k+7,e)
        ttmp = ttmp+shdxm1(k+7,l)*u(i,j,l,e)
      enddo
         shur(i,j,k+7) = gxyz(i,j,k+7,1,e)*rtmp
     $                 + gxyz(i,j,k+7,2,e)*stmp
     $                 + gxyz(i,j,k+7,3,e)*ttmp

     call syncthreads()
...
```

III.2 GPUDirect Gather-Scatter operator

The Gather-Scatter operator is implemented by `gs_op` routine in NekBone. Notice that we have already split the `gs_op` routine with local gather and scatter operations on GPUs in [7]. In the implementation only the non-local data need to be transferred between GPU and CPU to conduct MPI communication. The non-local data is exchanged with standard MPI subroutines `MPI_Irecv()`, `MPI_Isend()`, `MPI_Waitall()` with combination of `MPI_Waitall`. The modified `gs_op` operator with local gather and scatter is described in Algorithm 4.

Algorithm 4 Modified Gather-scatter operator (adapted from [7]).

```
unew_l = u_l
! u_g = Q u_l Local Gather on GPU
 !$ACC PARALLEL LOOP
 u_g = 0
 do i = 1, nl
    li = lgl(1,i)
    gi = lgl(2,i)
    u_g(gi) = u_g(gi)+u_l(li)
 enddo

gs_op(u_g,1,1,0) ! MPI communication between CPUs

! u_l = Q^T u_g Local Scatter on GPU
!$ACC PARALLEL LOOP
do i = 1, nl
   li = lgl(1,i)
   gi = lgl(2,i)
   unew_l(li) = u_g(gi)
enddo
```

In [8] a new version of Gather-scatter operator with GPUDirect is developed. The new version accelerates all four parts of the gather-scatter routine: local-gather, global-scatter, global-gather, and local-scatter, whereas previous versions only accelerated the local parts. Accelerating the global loops allows us to use the GPUDirect pragmas, as the buffers are prepared on the GPU. The local-gather and local-scatter loops above are included into the tuned `fgs_fields_acc`. In a similar manner, the global-scatter and global-gather loops are also accelerated. In this new version, data communication between CPUs is not necessary since the GPU would efficiently perform the local additions and does not need any information from other nodes.

IV. Performance Results

IV.1 Systems and compiler environments

We performed our simulations on few supercomputing systems. *Titan*, a Cray XK7 system at the Oak Ridge Leadership Computing Facility (OLCF), consists of 18,688 AMD Opteron 6274 16-core CPUs and 18,688 Nvidia Tesla K20X GPU computing accelerator with 6GB of GDDR5 memory each. Titan has a hybrid architecture with peak performance of 27 Pflops and 40 PB of Lustre storage. The pair of nodes shares a Gemini high-speed interconnect router in a 3D torus topology. *Curie* is a PRACE Tier-0 system, installed at CEA in France. The Curie system has total 144 hybrid nodes. Each hybrid node has two 4-cores Westmere-EP@2.67GHz CPUs and two Nvidia M2090. The compute nodes are connected through a QDR InfiniBand network and the topology of this InfiniBand network is a full fat tree. *Raven* system at Cray has 8 XK7 compute nodes with 8 NVIDIA Telsa K20 GPUs. A compute node of Raven has one Opteron processor with a total of 16 processor cores and 2 NUMA nodes with a total of 16 GB of DDR3-1600 main memory. The Opteron processors run at 2.1 GHz. Each GPU has 6 GB of GDDR5 memory.

Raven and Titan support GPU-Direct with both Cray CCE and PGI compilers. The Curie system supports only the PGI compiler without GPU-Direct features.

IV.2 Single GPU Performance tests

We optimized our GPU enabled code with CCE and PGI compilers for the compute intensive matrix-matrix multiplications routines, as discussed in a previous section using Algorithms 1–3. Figures 1–4 show the single GPU performance tests on different platforms. For all cases, the performance critically depends on the computational workload on the GPU. The more calculations are completed on the GPU, the performance is higher. the performance increases as the number of elements (E) increases and the performance significantly increases with the polynomial order (N). In addition, different compilers and versions also affect the performance of NekBone.

On the Curie hybrid nodes the performance increases around 5-10% with the optimized OpenACC directives compared to the original case, see Figures 1 and 2. The maximum performance achieved is 43.6 Gflops with elements $E = 4096$ and polynomial order

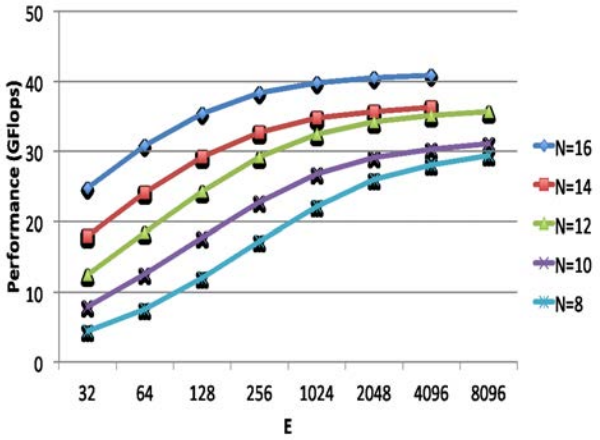


Figure 1: The performance with the total number of grid points $n = E \cdot N^3$ for $E = 32, 64, \ldots 8096$ and $N = 8, 10, 12, 14, 16$ on single node of Curie using PGI compilers. Original OpenACC directives.

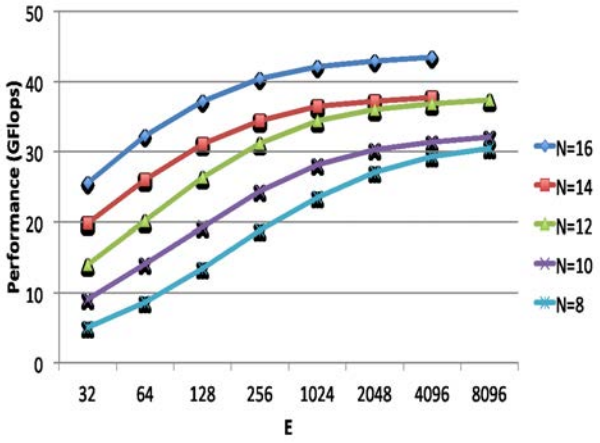


Figure 2: The performance with the total number of grid points $n = E \cdot N^3$ for $E = 32, 64, \ldots 8096$ and $N = 8, 10, 12, 14, 16$ on single node of Curie using PGI compilers. Optimized OpenACC directives.

$N = 16$.

Figure 4 shows the performance on Titan with Cray CCE, PGI, and PGI CUDA FORTRAN compilers. The CUDA FORTRAN code is around 10% faster than the OpenACC code. However, it is also important to highlight the little effort required to port an application like NekBone to GPU systems using OpenACC compared to the CUDA porting process. Furthermore, the small number of additional lines of code required to port an application to OpenACC is not comparable with the addition of CUDA kernels code. Such is the case that is necessary to rewrite the CUDA code for each polynomial order (N) case.

IV.3 Multi GPU Performance tests

From Figures 1 and 2, even without MPI communication we can identify that the performance of the matrix-matrix multiplication kernels highly depends on the order of polynomial (N) and number of elements (E). Larger values of N give better performance. This effect could be cause due to the amount of work per thread (which is proportional to N) is greater, which either leads to better kernel efficiency or assists to offset the latency cost of launching kernels. Also, the MPI communication overlaps the less workload of GPUs. Consequently, the degradation performance with the increase of the number of GPUs for the strong scalability is expected. The performance of OpenACC version is quite different from the original MPI ver-

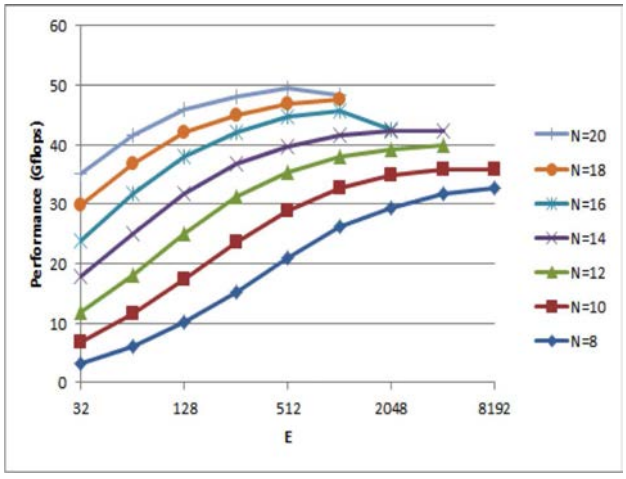


Figure 3: The performance with the total number of grid points $n = E \cdot N^3$ for $E = 32, 64, \ldots 8096$ and $N = 8, 10, 12, 14, 16$ on single node of Raven using CCE compilers.

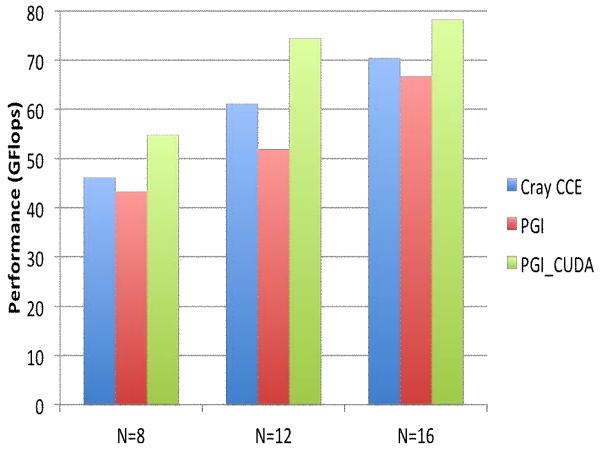


Figure 4: The performance with the total number of grid points $n = E \cdot N^3$ for $E = 512$ and $N = 8, 12, 16$ on single node of Titan using CCE and PGI compilers.

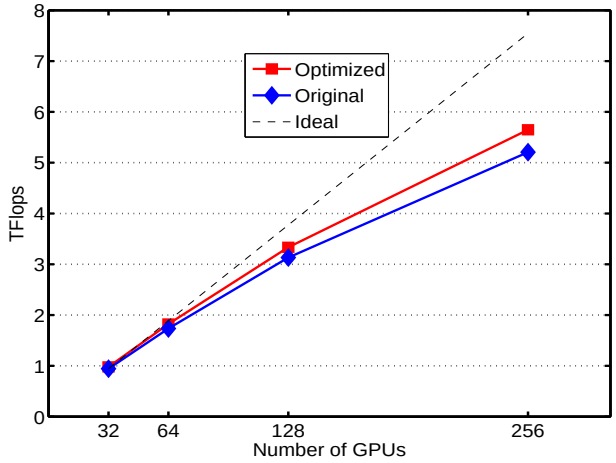


Figure 5: Strong scaling results with total number points are $n = 1024 \cdot 16^3 \cdot 32$ on Curie using PGI compilers.

sion where parallel efficiency 0.6 has been measured for strong scalability between 32768 and 1048576 MPI ranks [14].

Figure 5 shows the NekBone strong scaling performance, measured in Tflops, with up to 256 GPUs as a solid line, while the black dashed line represents the ideal strong scaling on Curie. The parallel efficiency with 256 GPUs was 69% compared when using 32 GPUs. In order to get better performance we should use as many elements per node as we can fit into GPU memory (6GB for the Tesla M2090 card).

Each Curie hybrid node has two sockets and each GPU is bound to a socket. By the default both processes are running on only one GPU. As a result, the `CUDA_VISIBLE_DEVICES` variable should be set to 0 or 1 depending on the rank of the process. In addition, it is necessary to bind the processes to the GPU to make sure they run on the same socket that hosts the desired GPU. However the binding of the processes still slows down the application since both processes share some resources. For instance, I/O operations are slower or the MPI communications may behave differently. Figures 6 and 7 show the weak scaling results on Curie where we can see how the curve representing the optimized version is growing apart as more GPUs are used in the simulation.

For the test on Titan, 1024 elements and 16th-order polynomials were used for a total of 4,194,304 points per GPU. Figure 8 shows the NekBone weak scaling performance, measured in TFlops, with up to 16,384

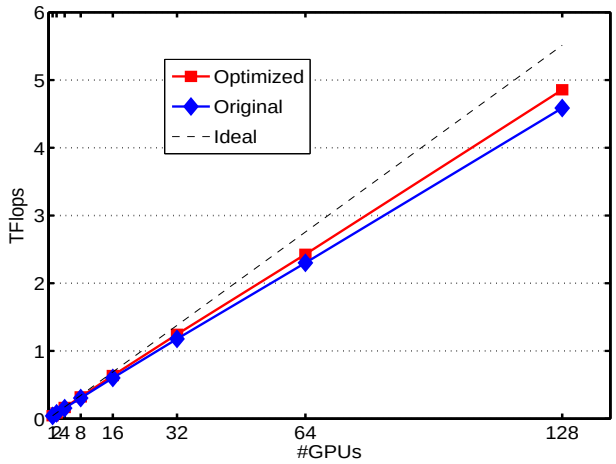


Figure 6: Weak scaling results with $n = 1024 \cdot 16^3$ per GPU on Curie using PGI compilers.

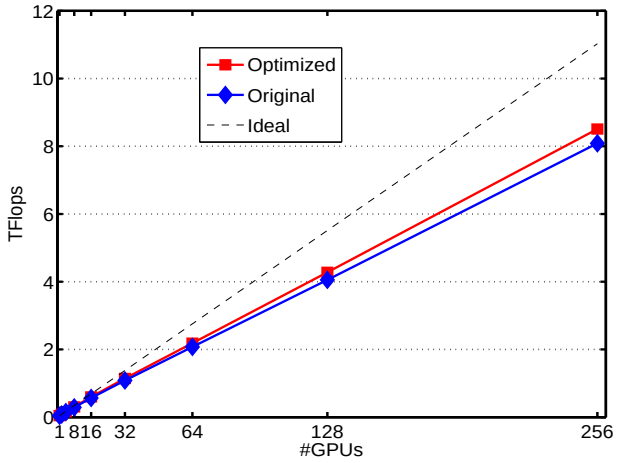


Figure 7: Weak scaling results with $n = 1024 \cdot 16^3$ per GPU on Curie using PGI compilers.

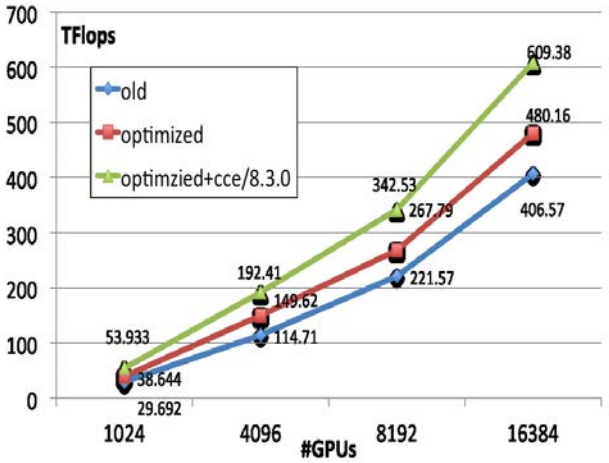


Figure 8: Weak scaling performance with $n = 1024 \cdot 16^3$ per GPU on Titan using CCE compilers.

GPUs. The parallel efficiency on 16,384 GPUs was 52.8% compared with single GPU and the maximum performance obtained is 609.8 Tflops with optimized OpenACC code. This good scaling results is achieved by using the proper construction of the global communication and the code simplicity.

IV.4 GPUDirect tests

The Cray performance analysis tool `CrayPat` is used to conduct the profiling analysis for the data communication. The tests are conducted on the Raven system with 8 GPUs and the number of grid points is $n = 1024 \cdot 16^3$ per GPU. The total time on the MPI communication is 2.52 sec with the modified gather-scatter Algorithm 4, see Profiling by Function Group Table using CrayPat below.

```
 Table 1:  Profile by Function Group and Function

  Time% |      Time |     Imb. |  Imb. |   Calls |Group
        |           |     Time | Time% |         | Function
 100.0% | 11.864466 |       -- |    -- | 34739.0 |Total
|-----------------------------------------------------------
|  94.7% | 11.235358 |       -- |    -- | 30332.2 |USER
||----------------------------------------------------------
||  19.9% |  2.366802 | 0.000701 |  0.0% |   200.0 |dssum_acc_.ACC_
...
||   1.3% |  0.152252 | 0.000371 |  0.3% |   200.0 |dssum_acc_.ACC_
...
```

With the Algorithm developed in [8], the total time is reduced to 2.36 sec. This can be seen in the next Table.

```
Table 1:  Profile by Function Group and Function

  Time% |      Time |     Imb. |  Imb. |   Calls |Group
        |           |     Time | Time% |         | Function
 100.0% | 11.806398 |       -- |    -- | 38739.0 |Total
|-----------------------------------------------------------
|  94.8% | 11.193856 |       -- |    -- | 34332.2 |USER
||----------------------------------------------------------
||  20.0% |  2.357107 | 0.000715 |  0.0% |   200.0 |fgs_fields_acc
...
```

V. Conclusions

A hybrid Nek5000 version was created to exploit the processing power of multi-GPU systems by using OpenACC compiler directives. This work focused on advance GPU optimizing and tuning of the most time-consuming parts of Nek5000, namely the matrix-matrix multiplication operations and the preconditioned linear solve operation. Furthermore, the gather-scatter kernel used with MPI operations has been redesigned in order to decrease the amount of data transferred between the host and the accelerator. The speed-up achieved using OpenACC directives is 1.30 with a 16th order polynomial on 16,384 GPUs of the Cray XK7 supercomputer when compared to 16,384 full CPU nodes having 262,144 CPU cores in total.

We have been able to compare performance results of NekBone versions running with CUDA FORTRAN and OpenACC. Although OpenACC has proven to be a simpler solution for porting applications to GPUs, our results demonstrate that CUDA is still more efficient and that in OpenACC there is still room for improvement.

With a multi-GPU setup, the gather-scatter operator and the associated MPI communication can be improved. The original gather-scatter operator was split

into two parts. First a local gather on the GPU is performed, followed by the transfer of the boundary values at the interfaces of the domain. Then the boundary values need to be copied to a local CPU memory, communicated via network to the memory of another CPU, and then transferred back a memory of a remote GPU to finally carry out a local scatter on the GPU. This approach allows a considerable reduction in the amount of data to be moved from the GPU and CPU memory and vice versa.

In spite of the reduction in the amount of data transferred, the additional transfers between the host and accelerator have an effect on the achievable performance. In the future, we will employ the techniques such as overlapping of GPU kernels with host-accelerator memory transfers to further increase the performance of the OpenACC version of Nek5000.

Acknowledgments

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS) and the Swedish e-Science Research Center (SeRC). We acknowledge PRACE for awarding us access to resource CURIE based in France at CEA as well as the computing time on the Raven system at Cray and the Titan supercomputer at Oak Ridge National Laboratory. We would also like to thank Brent Leback for the CUDA FORTRAN code used in the paper.

References

[1] J. H. Chen, A. Adhere, B. De Supinski, M. DeVries, E. Hawkes, S. Klasky, W. Liao, K. Ma, J. Mellor-Crummey, N. Podhorszki, et al., "Terascale direct numerical simulations of turbulent combustion using S3d", *Computational Science & Discovery* vol. 2, no. 1, 2009.

[2] D. C. Jespersen, "Acceleration of a CFD code with a GPU", *Scientific Programming*, vol. 18, no. 3-4, pp. 193-201, 2010

[3] C. K. Aidun and J. R. Clausen, "Lattice Boltzmann method for complex flows", *Annual Review of Fluid Mechanics*, vol. 42, pp. 439-472, 2010

[4] K. Niemeyer and C. Sung, "Recent progress and challenges in exploiting graphics processors in computational fluid dynamics", *The Journal of Supercomputing*, vol. 67, no. 2, pp. 528-564, 2014.

[5] OpenACC standard, `http://www.openacc-standard.org`

[6] J. Gong, S. Markidis, M. Schliephake, E. Laure, D. Henningson, P. Schlatter, A. Peplinski, A. Hart, J. Doleschal, D. Henty, and P. Fischer, Nek5000 with OpenACC, in *Solving Software Challenges for Exascale, the International Conference on Exascale Applications and Software, EASC 2014 Stockholm, Sweden, April 20-23, 2014*, Stefano Markidis, Erwin Laure (Eds.), Springer LNCS8759, 2015.

[7] S. Markidis, J. Gong, M. Schliephake, E. Laure, A. Hart, D. Henty, K. Heisey, and P. Fischer, "OpenACC acceleration of the Nek5000 spectral element code", *International Journal of High Performance Computing Applications*, vol. 29, pp. 311-319, 2015.

[8] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, and M. Min "An MPI/OpenACC Implementation of a High Order Electromagneticcs Solver with GPUDirect Communication", accepted in *International Journal of High Performance Computing Applications*.

[9] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier, Nek5000 web page, Web page: http://nek5000.mcs. anl. gov.

[10] A. T. Patera, "A spectral element method for fluid dynamics: laminar flow in a channel expansion", *Journal of Computational Physics*, vol. 54, No. 3, pp. 68-488, 1984

[11] H. M. Tufo and P. F. Fischer, "Terascale spectral element algorithms and implementations", in *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, ACM, 1999, p. 68.

[12] M. Deville, P. Fischer, and E. Mund, *High-order methods for incompressible fluid flow*, Cambridge University Press, 2002.

[13] NekBone: Proxy-Apps for Thermal Hydraulics, `https://cesar.mcs.anl.gov/content/software/thermal_hydraulics`

[14] Nek5000 strong scaling tests to over one million processes. `http://nek5000.mcs.anl.gov/index.php/Scaling`

Scheduler hierarchies to aid peta-scale cloud simulations with DISSECT-CF

Gabor Kecskemeti

Laboratory of Parallel and Distributed Systems at the Institute for Computer Science and Control of the Hungarian Academy of Sciences (MTA SZTAKI), Kende u. 13-17, Budapest 1111, Hungary
kecskemeti.gabor@sztaki.mta.hu

Abstract

IaaS cloud simulators are frequently used for evaluating new scheduling practices. Unfortunately, most of these simulators scarcely allow the evaluation of larger-scale cloud infrastructures (i.e., with physical machine counts over a few thousand). Thus, they are seldom applicable for evaluating infrastructures available in commercial cloud settings (e.g., users mostly do not wait for simulations to complete in such settings). DISSECT-CF was shown to be better scaling than several other simulators, but peta-scale infrastructures with often millions of CPU cores were out of scope for DISSECT-CF as well. This paper reveals a hierarchical scheduler extension of DISSECT-CF that not only allows its users to evaluate peta-scale infrastructure behaviour, but also opens possibilities for analysing new multi-cloud scheduling techniques. The paper then analyses the performance of the extended simulator through large-scale synthetic workloads and compares its performance to DISSECT-CF's past behaviour. Based on the analysis, the paper concludes with recommended simulation setups that will allow the evaluation of new schedulers for peta-scale clouds in a timely fashion (e.g., within minutes).

Keywords Cloud computing, simulator, petascale, hierachical scheduling

I. Introduction

Cloud computing infrastructures [1] have rapidly developed into a commodity. Based on virtualisation technologies, they offer simple and straightforward management capabilities for virtual infrastructures. As a result, users of Infrastructure as a Service (IaaS) clouds can more rapidly respond to their ever changing demand patterns, while they do not have to face the everyday issues that would arise with the maintenance of physical infrastructures. Because of the many benefits of IaaS clouds, their adoption has became widespread.

Unfortunately, this widespread use limits research on the internal behaviour of IaaS clouds. To overcome these limits, researchers often turn to cloud simulators to analyse their new ideas [2]. These simulators allow rapid evaluation of many new scenarios; however, they frequently have scaling issues of their own. Thus, they restrain those scenarios that can be evaluated with them. And even in cases when they scale well for the larger-scale simulation needs of recent cloud infrastructures, they are too specialised for general research (e.g., they do not allow simultaneous evaluation of both cloud internals and user side behaviour).

DISSECT-CF (DIScrete event baSed Energy Consumption simulaTor for Clouds and Federations [3]) was proposed as a general purpose, compact, highly customisable open source cloud simulator with special focus on the internal organisation and behaviour of IaaS systems. Compared to other state-of-the-art cloud simulators its performance is already amongst the best. However, even DISSECT-CF has scaling issues when it needs to simulate such large-scale computing infrastructures as the front entries in the top500 supercomputers list[1].

This paper analyses past DISSECT-CF behaviour when simulating infrastructures similar to the ones listed amongst the top500 supercomputers. Based on

[1] http://top500.org

the analysis, the paper concludes that the large amount of physical machines (handled by a single virtual machine – VM – placement technique) cause the scaling issues in the past simulator. Therefore, this paper proposes a generic technique to organise scheduler hierarchies in DISSECT-CF. These hierarchies allow the reduction of the number of machines handled by a single VM placement technique. In order to overcome the inefficiencies that could be caused by the newly introduced hierarchies, the simulator now introduces several ways for interacting between the various levels of the scheduler hierarchy: (i) automated high-level VM request revocation, (ii) VM request rejection, (iii) automated hierarchy setup and (iv) VM request propagation though cloud boundaries.

Although the introduced hierarchies are good to increase the scalability of the simulator and allow the evaluation of larger-scale systems, the newly proposed technique is still limited by several factors: (i) it cannot support nodes with mixed accelerator-CPU constructs – accelerator-CPU interactions cannot be handled with the new hierarchical model because the new model is limited to a hierarchy of a single kind of resource (because of a limitation in DISSECT-CF)–, (ii) similarly, inhomogeneous multi-cloud systems are still out of scope, (iii) the automated hierarchy setup is dependent on the kind of simulated workload – with improper hierarchy setup, the simulation might still face scalability issues–, and finally (iv) the relation between the actual layout of the simulated infrastructure and the real one can become very detached (the automatically introduced hierarchies usually have different layout than the actual racks, clusters and data centres in an IaaS).

The paper concludes with the analysis of the new hierarchical scheduling. Using synthetic traces a comparison is shown between the past and current simulator with infrastructures ranging from a few thousand to almost two million CPU cores. The behaviour of the extended simulator is also compared to CloudSim [4], revealing that DISSECT-CF has a performance advantage between 5-136$\times$ over CloudSim. Even compared to its past self, the new DISSECT-CF performs significantly better and its hierarchical scheduling mechanism could provide up to four-fold performance increase in smaller-scale infrastructure simulations, and a performance improvement of over 92$\times$ is observable for large-scale simulations.

The rest of the paper is structured as follows: the paper continues with studying state-of-the-art simulators to reveal their problems. Then, in Section III, the paper introduces a new hierarchical VM scheduling technique for DISSECT-CF. Next, the paper presents a performance evaluation for the improved simulator in Section IV. Finally, the paper provides its conclusive thoughts in Section V.

II. Related Works

CloudSim [4] is amongst the most popular IaaS cloud simulators. It introduces the simulation of virtualised data centres mostly focusing on computational intensive tasks and data interchanges between data centres. An extension called NetworkCloudSim [5], improved its support for in-data-centre network communications. There is also an extension that simulates the energy consumption of the physical machines in a data centre based on specpower benchmarks [6]. CloudSim also ignited an ecosystem around it adding performance improvements, inter-cloud operations and GUI wrappers for teaching [7, 8, 9, 10]. Despite its widespread use and its healthy ecosystem, research done with CloudSim is mostly limited to clouds with a few thousand CPU cores. This limitation severely affects the applicability of the results of CloudSim based simulations.

The SimGrid framework [11] is another widely used simulator for analysing distributed systems (e.g., grids, peer-to-peer systems). This simulator is not focused on clouds and only includes constructs to support virtualisation (e.g., hypervisors and live migration – [12]). Unfortunately, the lack of higher-level cloud related constructs reduces the applicability of SimGrid in most cloud simulation scenarios. Its users would need significant expertise in every cloud management issue so they can build and evaluate complete cloud-like scenarios.

Next, an analysis of GroudSim, which is a simulator developed at the University of Innsbruck [13, 14], was performed. This simulator aims at runtime performance, while it also integrates with the ASKALON workflow system. Until recently this simulator fol-

lowed a black box model (i.e., it did not simulate any internal details of the cloud management behaviour). Nowadays, GroudSim incorporates the DISSECT-CF simulator to enable the simulation of internal IaaS behaviour [15]. However, the complex cross-simulation synchronisation and workflow orientation of GroudSim makes it less scalable than DISSECT-CF alone.

While the previously mentioned simulators were heavily influenced by past simulators of grids and/or distributed systems, for performance reasons, they also made compromises on the simulation of networking functionalities. Such issues are resolved by simulators like iCanCloud [16] and GreenCloud [17]. These are built on network simulators (e.g., OMNeT++ or NS2) to most accurately simulate network communications in cloud systems. Other than their networking improvements, GreenCloud also offers precise energy estimates for networking and computing components, while iCanCloud is also user oriented and thus offers support in the decision making regarding the use of IaaS systems [18]. As these simulators are network oriented, their use cases are different from the rest of the simulators discussed in this section (e.g., they are mostly used to evaluate localised phenomena thus their scaling capabilities are not relevant).

Finally, there are some specialised simulators like xSim [19] and SimMatrix [20]. These simulators are proven to perform well for large-scale systems, but their scope is limited. For example xSim is focusing on the analysis of MPI workloads in exa-scale systems, while SimMatrix is focused on many task computing. Because of their over-specialisation these simulators are not suitable for analysing general problems in large-scale systems.

II.1 Problem Statement

After analysing the prior art, it can be concluded that existing simulators have many drawbacks for those planning to investigate scheduling in large-scale IaaS systems (e.g., they do not provide foundations for constructing scheduling hierarchies instead they expect their users to construct the hierarchies on their own). To fulfil the needs of such scheduling scenarios, the rest of this paper reveals a new hierarchical virtual machine scheduling technique to be applied by the DISSECT-CF simulator. With the use of this technique researchers will have better insights on infrastructure behaviour even if significantly larger-scale systems are simulated than it was previously possible by past simulators.

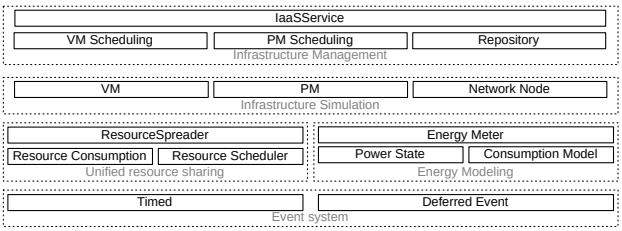


Figure 1: The architecture of DISSECT-CF

III. Generic Hierarchical Scheduling

III.1 Overview

DISSECT-CF [3] is a compact, customisable open-source simulator with focus on the internal organisation and behaviour of IaaS systems. Figure 1 presents its architecture. The figure groups the major components with dashed lines into subsystems. Each subsystem is implemented as independently from the others as possible. There are five major subsystems each responsible for a particular aspect of internal IaaS functionality: (i) event system – for a primary time reference; (ii) unified resource sharing – to resolve low-level resource bottleneck situations; (iii) energy modelling – for the analysis of energy-usage patterns of individual resources (e.g., network links, CPUs) or their aggregations; (iv) infrastructure simulation – to model physical and virtual machines as well as networked entities; and finally (v) infrastructure management – to provide a real life cloud like API and encapsulate cloud level scheduling (the target of this paper's improvements).

III.2 Scaling Bottleneck

Although the simulator was designed from the beginning with high scalability in mind, the performance of its VM placement mechanisms is dependent on the number of physical machines registered at an IaaS service. Thus, to reduce this bottleneck, but to allow

simulator developers to still design simple schedulers, a new hierarchical solution was needed. This new solution must allow the old scheduling mechanisms to work without even knowing that they only play a small role in a large-scale simulation, but the simulator's users should also have a chance to alter the hierarchy and the way higher-level schedulers interact with the original scheduling mechanisms.

III.3 Hierarchical Scheduling

III.3.1 Generic Scheduling

The new hierarchical scheduler concept of DISSECT-CF is built around the following functionalities: (i) schedulers should be able to propagate VM requests amongst each others, (ii) higher-level schedulers should be able to cancel requests, (iii) independently from the level of the scheduler, the same scheduling interface must be used (allowing to implement even cross-cloud scheduling mechanisms). In the following paragraphs, these functionalities are detailed.

VM propagation The hierarchical scheduler interface is expected to be implemented by every entity in the system who is planning to accept VM requests (ranging from Physical Machines, to low-level VM schedulers to even IaaS systems). This interface contains the following operations: (i) VM request, (ii) VM termination, (iii) VM migration, (iv) VM request cancellation and (v) VM resource reallocation. With the "standardised" interface it is not only possible to migrate VMs across various VM managers but also possible to create federations of multiple IaaS systems. In the new hierarchy, VM requests are propagated until they reach a physical machine that can serve them. If there are no physical machine that can serve a request in a low-level scheduler (one that directly interacts with physical machines), then the scheduler is allowed to queue the VM request. If the VM request is handled by an entity that does not directly interact with physical machines, then based on a scheduling policy it must decide to which lower-level scheduler it propagates the VM request (it cannot queue the request on its own). The selection of the lower-level scheduler can be accomplished by a range of techniques. The simulator currently delivers a random, a round robin and a weighted probabilistic scheduler selection technique (where VM requests are propagated to lower level schedulers that are less likely to queue them).

Automated request cancellation and resubmission As with many hierarchical schedulers in the past, it could frequently happen that one of the low-level schedulers gets overloaded with VM requests while others are left with very little work to do (this is a likely scenario with weighted probabilistic techniques). In order to automatically avoid bottlenecks, DISSECT-CF contains an automated request cancellation and resubmission technique. Upon submitting a VM request, higher-level schedulers will tag the request with an expiration time. This tag will be a timestamp after which time the lower-level scheduler is not allowed to serve the VM request, instead the request should be sent back to the higher-level scheduler who sent it. This tagging mechanism allows the low-level schedulers to prioritise the almost expired requests (as a measure of keeping service level objectives). Also, the high-level schedulers could penalise those lower-level schedulers that did not succeed in completing their designated VM requests within the expected period of time. In order to refrain VM requests from getting cancelled too frequently, the high-level schedulers analyse the VM throughput of each of their underlying schedulers and they set up VM request termination times so requests will expire with a small likelihood (in the current simulation setup, the resubmission rate is set up to be around a single request out of every thousand).

VM request rejection As VM requests received by lower-level schedulers are tagged with their expiration times, the low-level schedulers can decide if they are willing to queue such requests. When a tagged request arrives, the scheduler will automatically reject the request if its queue is significantly longer than the queues of other schedulers with similarly sized managed infrastructures.

III.3.2 Automated Hierarchy Formation

The simulator allows the precise definition of the hierarchy of the schedulers during the construction of

IaaSServices, ensuring that it matches the real life hierarchies set up in large scale cloud systems. However, to allow investigations about the effect of different hierarchies, this definition is not obligatory. If the user prefers, the hierarchy can be automatically constructed. The automatic hierarchies can even evolve during the entire runtime of a simulation (allowing the user to investigate several hierarchy adaptation scenarios and their effect on runtime). Also, the simulator can save an automatically determined hierarchy for later use, so users will have a chance to reuse previously efficiently working hierarchies (this option also allows users to utilise the auto constructed hierarchies outside of the simulated world and check the correctness of their simulations in real life).

The simulator allows the creation of the following kinds of schedulers (all these schedulers are also exemplified in Figure 2):

Regular schedulers. These schedulers have an IaaSService as their parents and they manage physical machines directly. These are the original kinds of schedulers possible in the simulator.

Low-level schedulers. They manage physical machines directly, but their parents are not IaaSServices. Instead they are operating with a high-level scheduler.

High-level schedulers. They do not directly operate with physical machines, they handle the VM requests as discussed in the previous sub-section. They can be classified into two subtypes:

Mid-level schedulers. They are placed in the middle of the hierarchy, they forward VM requests from their parents to their directly managed schedulers. They can either manage both high and low-level schedulers.

Ingress schedulers. They have an IaaSService as their parents, and they can act as either mid-level schedulers (if acting as an entry to a hierarchy) or as regular schedulers (if no hierarchy is needed).

Creating scheduler profiles. Before running the simulator with automated scheduling hierarchy management for realistic workloads, the hierarchy creation technique needs a profile for the user provided low-level schedulers (for the schedulers integrated in DISSECT-CF these profiles are already done). The profile creation starts with the specification of the total number of CPU cores – C_{max} – the simulated infrastructures should have during the profiling session. Next, several IaaSServices are created with varying number of CPUs in each physical machine – $c_{pm} \in C_{pm}$, where $C_{pm} = \{\forall x : (x \in \mathbb{N}) \wedge (1 < x < C_{max})\}$ – ensuring that the complete size of the infrastructure is matching the previously given total (e.g., with 8 CPU physical machines the number of physical machines should be $C_{max}/8$). Then, on each IaaSService the same randomly generated VM utilisation trace is executed (the trace's properties can be user defined). The execution time – $t_{ex} : C_{pm} \rightarrow \mathbb{R}$ – is recorded for all cases and they are written to the profile for the user provided low-level scheduler (examples of such profiles can be seen in Figures 4 and 5). The profiling executed on this way to ensure that the simulated infrastructure's size does not unnecessarily prolong the profiling period, and instead, only the PM count related scaling properties of the low-level scheduler do.

From the collected profile, the automated hierarchy creator determines the optimal amount of physical machines:

$$\begin{aligned} pm_{opt} &:= C_{max}/c_{opt} \quad (1) \\ \text{where} \quad t_{ex}(c_{opt}) &= \min_{\forall(c_{pm})}(t_{ex}(c_{pm})) \end{aligned}$$

The optimal amount of machines is calculated as the ratio between the maximum amount of CPU cores used during the profiling – C_{max} – and the specific number of CPU cores per machine – c_{opt} – which resulted in the smallest profiling execution time. Each low-level scheduler has a specific optimum value. Users can also specify an operationally acceptable pm count range for their scheduler relative to the pm_{opt} value. The range is specified with $\tau \in]0..1]$ which shows the difference allowed in the percentage of the execution time entries in the profile compared to the minimum execution time recorded for pm_{opt} (i.e., $t_{ex}(c_{opt})$). Thus:

$$C_{acc} := \{\forall c_x : c_x \in C_{pm} \wedge |1 - \frac{t_{ex}(c_x)}{t_{ex}(c_{opt})}| < \tau\} \quad (2)$$

$$PM_{acc} := \{C_{max}/c_x : c_x \in C_{acc}\} \quad (3)$$

Where the set of C_{acc} defines all the CPU core num-

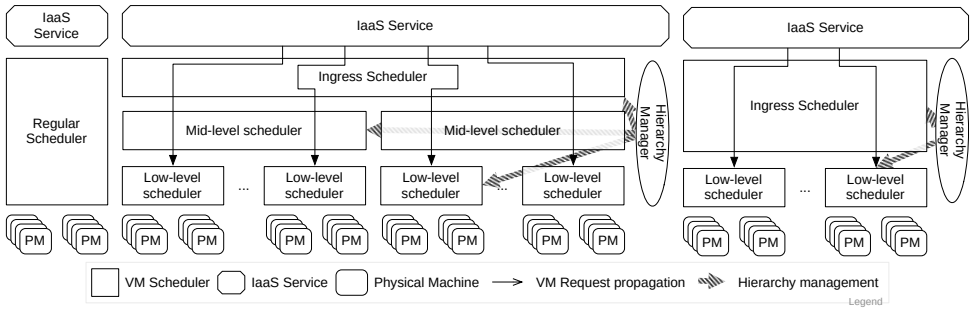


Figure 2: The kinds of scheduling hierarchies and schedulers possible in the new extended simulator

bers that resulted in profiling execution times within the user specified range. While PM_{acc} is the set of PM numbers for which the profiling suggests acceptable scheduling performance according to the user specified range. The hierarchy creator will use only the minimum $pm_l = \min PM_{acc}$ and maximum $pm_h = \max PM_{acc}$ values from the PM_{acc} set to determine under what conditions it does not need to alter the previously constructed hierarchy.

Managing scheduler hierarchies using profiling results. When a new PM is registered in the infrastructure under the control of a high-level scheduler, the hierarchy manager will register the PM in a round robin fashion to the underlying schedulers. Once the supervised PM count for a particular low-level scheduler is reached (i.e., it reaches over pm_h), the hierarchy manager realigns the PMs amongst the currently registered schedulers (ensuring a uniform PM count distribution amongst all its managed low-level schedulers). If it is not possible to realign the PM set without violating the maximum PM count limit set by the user, then the hierarchy manager will create a new low-level scheduler and start the realignment process again. This process is repeated until the number of physical machines registered under a particular low-level scheduler reaches pm_{opt}.

As PMs belong to a particular low-level scheduler, when they are deregistered, their past low-level scheduler will have a less balanced pm count. The hierarchy manager will not react to this unbalance until the PM count in one of the low-level schedulers reach pm_l. In such case, first the hierarchy manager tries to realign the PM set of all low-level schedulers so their managed PM set will be equally sized. If this approach still leaves some low-level schedulers with too low PM counts then those schedulers are eliminated from the system and their PMs are distributed amongst the still remaining low-level schedulers by the hierarchy manger. The elimination process is continued until the PM set of each managed low-level scheduler is sized around pm_{opt}.

So far we have discussed the situation for high-level schedulers that are directly in contact with low-level ones. In some cases the number of their directly managed schedulers reach the boundaries of the optimally operated scheduler set size. In those cases they get in touch with the hierarchy manager. The manager will either create secondary schedulers alongside the scheduler in question or eliminate one with similar techniques discussed for low level schedulers. The difference: the profile created for these high-level schedulers are defined in terms of the number of directly managed schedulers instead of the number of supervised PMs in a low-level scheduler. In both cases the hierarchy manager tries to equalise the number of PMs under a particular scheduler before creating or eliminating a high-level scheduler. The physical machine counts are also automatically determined and realigned if some low-level schedulers are preferred more than the others.

Finally, the hierarchy manager is operated alongside the ingest scheduler, the top scheduler in the hierarchy. This scheduler is the one that will be responsible for receiving the VM management operations from the `IaaSService`. Also, as future work, the automated

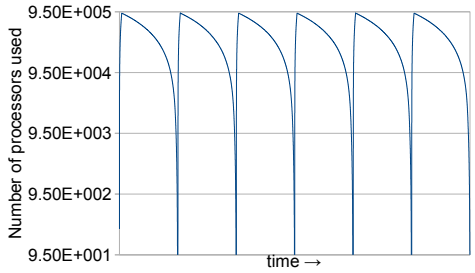


Figure 3: Example synthetic workload

hierarchies can follow the rules of software defined networking to have a more realistically constructed data centre layout.

IV. Evaluation

IV.1 Simulation Setup

Throughout the evaluation section, the following machine was used for executing the simulations: AMD Athlon FX 8120, 16 GB RAM, 128 GB SSD. The simulations were all set up and run in a controlled environment, the machine was never executing anything else just the simulators themselves. As workload traces for peta scale infrastructures are not yet public, in all the following cases the simulators were running a synthetic workload with similar characteristics than the one shown in Fig. 3. This synthetic workload has a peak utilisation with the same number of nodes as the simulated infrastructure has. This utilisation is the result of a randomly generated amount of virtual machine requests at a given time, and filling up the VMs with as many jobs as many needed to reach the expected utilisation profile at the given time instance (e.g., a particular VM in the load could host no jobs at all, but in some cases they can be filled up with several hundred if the VM's resources would not get exhausted by that many jobs). Also, the number of utilisation peaks can be configured and during the evaluation runs it was set up to be between 4-10 (the actual number of peaks was determined so it has had a stabilising effect on the simulation runtime – i.e., the number of peaks was set so the consecutive simulation runs have had small variance in their runtimes). Throughout this paper the following simulators were used: (i) DISSECT-CF 0.9.5 – as the old reference point that represents the latest stable release of the simulator without any peta-scale optimisations, (ii) DISSECT-CF 0.9.6 – as the last released version, (iii) DISSECT-CF 0.9.7* – the simulator incorporating the hierarchical extensions of this paper, and (iv) CloudSim 3.0.3 DVFS extensions – the independent reference point. Finally, it was ensured that independently from the simulator used, the program that set up and run the simulations was never consuming more CPU than 1% of the total CPU consumption of a simulation (this step assured that the below presented results are influenced mainly by the investigated simulators and not by the additional code used for the evaluation).

The simulated results of the extended DISSECT-CF simulator were *validated* by comparing its results to the past validated DISSECT-CF 0.9.5 results as well as to the results obtained from CloudSim 3.0.3 DVFS. After executing a workload in a simulation, the termination time of the last virtual machine was noted. These times were compared with both past simulators. The new simulations yielded final termination times within 0.05% of the older simulators, while significantly improving on simulation performance.

IV.2 Measurements

During the first measurements, basic scaling properties were investigated for all simulators. For this experiment, a 5000 core infrastructure was set-up in the simulators. The composition of the infrastructure was changed to range from a single node (with 5000 cores) to 5000 nodes (with a single core each). This experiment was designed to show the bottleneck situations in both the simulator's resource sharing mechanism and in its default virtual machine scheduler. Also, this experiment actually replicates the profiling technique introduced in Section III.3.2.

Figure 4 reveals the results of this first experiment. In the experiments utilising simulated infrastructures with less than 10 hosts, the resource sharing mechanism of the simulator is more dominant (i.e., the mechanism that determines how the resources of a single physical machine are shared amongst the VMs it is hosting). Similarly, experiments, with infrastructures consisting of over 1000 hosts, were dominated by the

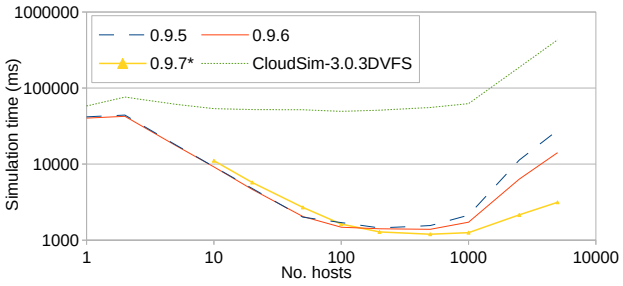


Figure 4: Small-scale measurement utilising an infrastructure with 5000 CPU cores (distributed amongst a varying number of hosts)

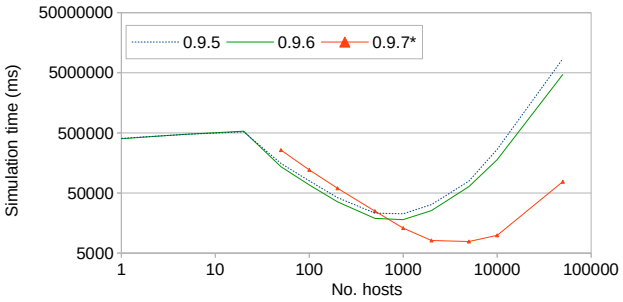


Figure 5: Medium-scale measurement using an infrastructure consisting of 50000 cores (distributed amongst a varying number of hosts)

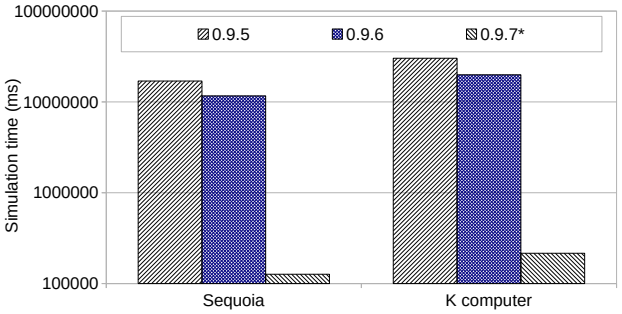


Figure 6: Peta-scale measurement

virtual machine scheduler's performance (i.e., in these cases it was very unlikely to have multiple virtual machines hosted in a single physical machine, thus the assignment mechanism of the virtual machines to physical ones become more dominant in execution time). As it can be seen, DISSECT-CF has a bigger overhead for resource sharing than CloudSim: DISSECT-CF previously had a 33% performance loss if the 5000 cores needed to be scheduled by the resource sharing mechanism of the simulator, while in case of CloudSim a virtual machine scheduler has a 86% performance loss. Next, the evaluation of DISSECT-CF 0.9.7 and its hierarchical extensions show that an over 8.8× improvement is achievable by just switching to the hierarchical scheduler. Remarks: the hierarchy built up by the new automated mechanism consisted of 10 low-level virtual machine schedulers and one high-level scheduler acting as ingress. Also the number of utilisation peaks was set to 10 as that gave the most stable measurements after evaluating the sample standard deviation of the measured runtimes of simulations.

Because of performance issues, only the first experiment was evaluated with CloudSim. The rest of the simulations are compared to past DISSECT-CF versions only as CloudSim based results were not obtainable in feasible time.

Next, the above discussed experiment was repeated with 50000 cores. This experiment was executed to show how the simulator scales in a well defined and controlled environment. The number of utilisation peaks was kept at 10 in order to allow a more direct comparison with the results from the previous experiment. The hierarchy built up by the new automated mechanism consisted of 50 low-level virtual machine schedulers. The results of this second experiment are shown in Fig. 5. As it can be seen, the hierarchical extension now improves with over 111× compared to the original DISSECT-CF 0.9.5 version. The figure also reveals, that it is not recommended to use hierarchical virtual machine schedulers in case there are too few physical machines on which these schedulers can place the requested VMs (e.g., performance degradation is observable for simulations with less than 600 hosts – the worst case degradation reaches 33%). In terms of scaling, the 5000 node infrastructure performs 1.72× better when there are more CPU cores supervised by a single node (the 50000 core experiment executed with 64VMs/ms – virtual machines processed per millisecond – while the 5000 core experiment achieved a performance of 37VMs/ms).

Finally, two peta-scale experiments were also exe-

cuted. For these experiments, two of the top supercomputers were selected and their infrastructures were constructed in the simulator. The selected supercomputers were chosen with two criteria: they should be listed amongst the top 10 from the top500 supercomputers list, and they should not have accelerators augmenting their computing power (this second requirement is needed because the current simulator cannot handle tasks running simultaneously on an accelerator and on a CPU core). The two supercomputers fulfilling these requirements:

- Sequoia – with 1,572,864 CPU cores in 98,304 nodes.
- K Computer – with 705,024 CPU cores in 88,128 nodes.

These two large-scale systems were simulated in the various versions of DISSECT-CF with 4 utilisation peaks for Sequoia and 6 utilisation peaks for the K Computer. Figure 6 reveals the measured results. As it can be seen, without the hierarchical extensions of DISSECT-CF, the simulations complete in over 3 hours (best case with the fastest 0.9.6 version without hierarchical scheduling). Thus it is not realistic to expect users of the simulators that they can execute hundreds or even thousands of scenarios for their research. However, with the hierarchical extensions even the peta-scale simulation reaches a manageable less than 4 minutes runtime. The automatically created hierarchy is still only two levels deep for these large-scale infrastructures (the third level of the hierarchy would appear for exa-scale simulations), but now consists of a little over 100 smaller infrastructure fragments. The resulting performance increase is between 91-139$\times$ compared to the original hierarchy-less simulations.

V. Conclusions and Future Works

This paper presented a new hierarchical VM scheduling technique for the DISSECT-CF simulator. The new scheduling technique is aimed at large-scale simulations (in the current paper it is focused on simulating peta-scale systems). The paper shows that the new technique can successfully reduce the time required for simulations on such scale. The observable performance improvements are enabling the application of the DISSECT-CF simulator during the evaluation of even peta-scale cloud systems (e.g., systems similarly constructed as the Sequoia or K computer present in recent top500 lists).

Regarding future works, the results presented in this paper is just the first step in many to enable a generic cloud and distributed systems simulator to cope with such large-scale systems like the commercially available Amazon EC2. In the following, the simulator is intended to be improved with GPGPU support to follow the trends of the accelerator based systems. Also, further research is needed to identify the peculiarities of the various interconnects in these large-scale systems. This new research will allow DISSECT-CF based simulations to handle infrastructures with mixed interconnects or federation of infrastructures with different interconnect technologies.

Acknowledgement

This work was partially supported by the European Union's Horizon 2020 programme under grant agreement No 644179 (ENTICE), by the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS) and by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

Software availability

This paper described the behaviour and features of DISSECT-CF version 0.9.7. Its source code is open and available (under the licensing terms of the GNU LGPL 3) at the following website:
`https://github.com/kecskemeti/dissect-cf`

References

[1] M. Armbrust, A. Fox, R. Griffith *et al.*, "Above the clouds: A berkeley view of cloud computing," University of California at Berkley, Tech. Rep. UCB/EECS-2009-28, February 2009.

[2] G. Sakellari and G. Loukas, "A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing," *Simul. Model. Pract. Th.*, vol. 39, pp. 92–103, 2013.

[3] G. Kecskemeti, "DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds," *Simulation Modelling Practice and Theory*, to appear, DOI: 10.1016/j.simpat.2015.05.009, pp. 1–28, 2015.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, January 2011.

[5] S. K. Garg and R. Buyya, "NetworkCloudSim: modelling parallel applications in cloud simulations," in *Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*. Victoria, NSW: IEEE, December 2011, pp. 105–113.

[6] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, Sept 2012.

[7] X. Li, X. Jiang, P. Huang, and K. Ye, "DartCSim: An enhanced user-friendly cloud simulation system based on CloudSim with better performance," in *2nd International Conference on Cloud Computing and Intelligent Systems (CCIS)*, vol. 1. Hangzhou: IEEE, Oct 2012, pp. 392–396.

[8] S. Sotiriadis, N. Bessis, N. Antonopoulos, and A. Anjum, "SimIC: Designing a new inter-cloud simulation platform for integrating large-scale resource management," in *27th International Conference on Advanced Information Networking and Applications (AINA)*, 2013, pp. 90–97.

[9] Y. Shi, X. Jiang, and K. Ye, "An energy-efficient scheme for cloud resource provisioning based on CloudSim," in *IEEE International Conference on Cluster Computing (CLUSTER)*. Austin, TX: IEEE, Sept 2011, pp. 595–599.

[10] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. Alsaleh, "Teachcloud: a cloud computing educational toolkit," *Int. J. of Cloud Computing*, vol. 2, no. 2/3, pp. 237–257, 2012.

[11] H. Casanova, "SimGrid: A toolkit for the simulation of application scheduling," in *First IEEE/ACM International Symposium on Cluster Computing and the Grid*. Brisbane, Qld.: IEEE, May 2001, pp. 430–437.

[12] T. Hirofuchi, A. Lèbre, L. Pouilloux *et al.*, "Adding a live migration model into SimGrid, one more step toward the simulation of infrastructure-as-a-service concerns," in *5th IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom)*, Bristol, UK, December 2013, pp. 96–103.

[13] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "Groudsim: an event-based simulation framework for computational grids and clouds," in *Euro-Par 2010 Parallel Processing Workshops*, ser. Lecture Notes in Computer Science, vol. 6586. Springer, 2011, pp. 305–313.

[14] S. Ostermann, K. Plankensteiner, D. Bodner, G. Kraler, and R. Prodan, "Integration of an event-based simulation framework into a scientific workflow execution environment for grids and clouds," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science. Poznan, Poland: Springer, 2011, vol. 6994, pp. 1–13.

[15] G. Kecskemeti, S. Ostermann, and R. Prodan, "Fostering energy-awareness in simulations behind scientific workflow management systems," in *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, Dec 2014, pp. 29–38.

[16] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente, "Design of a new cloud computing simulation platform," in *Computational Science and Its Applications-ICCSA 2011*, ser. Lecture Notes in Computer Science. Santander, Spain: Springer, 2011, vol. 6784, pp. 582–593.

[17] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware

cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[18] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[19] C. Engelmann, "Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale," *Future Gener. Comp. Sy.*, vol. 30, pp. 59–65, 2014.

[20] K. Wang, K. Brandstatter, and I. Raicu, "Simmatrix: Simulator for many-task computing execution fabric at exascale," in *Proceedings of the High Performance Computing Symposium*. Society for Computer Simulation International, 2013, p. 9.

NUMA impact on network storage protocols over high-speed raw Ethernet

Pilar González-Férez† and Angelos Bilas‡

†Universidad de Murcia, Spain, pilar@ditec.um.es
‡FORTH-ICS and University of Crete, Greece, bilas@ic.forth.gr

Abstract

Current storage trends dictate placing fast storage devices in all servers and using them as a single distributed storage system. In this converged model where storage and compute resources co-exist in the same server, the role of the network is becoming more important: network overhead is becoming a main limitation to improving storage performance. In our previous work we have designed Tyche, a network protocol for converged storage that bundles multiple 10GigE links transparently and reduces protocol overheads over raw Ethernet without hardware support. However, current technology trends and server consolidation dictates building servers with large amounts of resources (CPU, memory, network, storage). Such servers need to employ Non-Uniform Memory Architectures (NUMA) to scale memory performance. NUMA introduces significant problems with the placement of data and buffers at all software levels.

In this paper, we first use Tyche to examine the performance implications of NUMA servers on end-to-end network storage performance. Our results show that NUMA effects have significant negative impact and can reduce throughput by almost 2x on servers with as few as 8 cores (16 hyper-threads). Then, we propose extensions to network protocols that can mitigate this impact. We use information about the location of data, cores, and NICs to properly align data transfers and minimize the impact of NUMA servers. Our design almost entirely eliminates NUMA effects by encapsulating all protocol structures to a "channel" concept and then carefully mapping channels and their resources to NICs and NUMA nodes.

Keywords NUMA, memory affinity, network storage, Tyche, I/O throughput

I. Introduction

Technology trends for efficient use of infrastructures dictate that storage converges with computation by placing storage devices, such as NVM (Non-Volatile Memory) based cards and drives, in the servers themselves. With converged storage, compute servers are used as a single distributed storage system, in a departure from traditional SAN (Storage Area Network) and NAS (Network Attached Storage) approaches. In this model, where computation and storage are co-located, the role of the network becomes more important for achieving high storage I/O throughput.

For efficiency and scalability purposes modern servers tend to employ multiple resources of each kind, namely processors, memories, and storage/network links, in Non-Uniform Memory Access (NUMA) architectures (Figure 1).

Scaling networked storage throughput on such servers is becoming an important challenge. NUMA servers use multiple processor sockets with memory attached to each socket, resulting in non-uniform latencies from processor to different memories. In NUMA architectures each I/O device is attached to a specific NUMA node via an I/O hub (Figure 1). Processors, memories, and I/O hubs are connected through high-speed interconnects, e.g. QPI [1]. I/O requests as well DMA transfers to and from devices are routed through the memory-processor interconnect. Accessing remote memory (in a different NUMA node) incurs

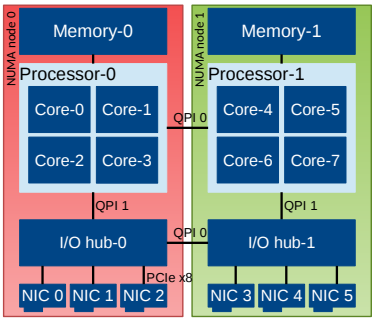
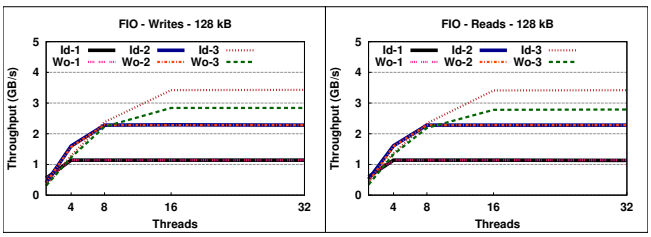


Figure 1: Internal data paths in NUMA servers.



(a) 128 kB writes (b) 128 kB reads

Figure 2: FIO [7] throughput with direct I/O, random writes and reads, 128kB and several threads, with (Id) and without (Wo) Tyche affinity over and 1, 2 and 3 NICs. Note that curves for Id-1, Wo-1 and Id-2, Wo-2 are overlapping.

significantly higher latency than accessing local memory [2, 3], up to a factor of 2x. In addition, it consumes throughput from the inter-processor link(s). Thus, for I/O performance and scalability purposes, it is important to explore how the network protocol can be designed to cater for affinity among memory, processing cores, and network interfaces (NIC) for data and protocol data structures.

In our previous work Tyche [4, 5] we examine the design of network storage protocols over raw Ethernet to achieve high throughput without hardware support. We argue that raw Ethernet is cost-effective and Tyche delivers high I/O throughput and low I/O latency using several techniques: bundling multiple NICs transparently, copy-reduction, storage-specific packet processing, RDMA (remote direct memory access)-type communication primitives, memory preallocation, and transparent NIC bundling.

In our previous work we observe that NUMA affinity is an important issue that spans the whole I/O path and has a significant performance impact.

Therefore, in this work, we use Tyche to analyze in detail the impact of NUMA affinity on networked storage access. In addition, we examine whether we can mitigate these performance effects on NUMA servers by properly re-designing the network protocol.

We evaluate this issue on two Linux servers with 8 cores (16 hyper-threads) and 6 x 10GigE Myricom 10G-PCIE-8A-C NICs attached to each server. We analyze the data traffic transferred through the QPI links with the Intel ® Performance Counter Monitor (PCM) [6].

Our results show that NUMA effects indeed have a very large negative impact on performance and they can reduce throughput by almost 2x. Our re-designed protocol that detects and uses NUMA affinity across buffers, threads, and NICs, improves I/O throughput by up to 85%.

This behavior is shown in Figure 2 for two configurations: Ideal (Id) case that allocates all the resources in NUMA node 0, and Worst (Wo) case that allocates them in node 1. We use 1, 2, and 3 NICs, all of them attached to the node 0. For 1 and 2 NICs, there is no difference in performance between both configurations. However, when using 3 NICs Ideal significantly outperforms Worst up to 23.7%. When using 6 NICs, this difference is larger. For instance, when the in-house micro-benchmark zmIO [8] is run with direct I/O and random 64kB read requests, Tyche obtains only 3.61 GB/s when no affinity is applied, whereas it achieves 6.67 GB/s when NUMA is taken into account. Additionally, our analysis shows that this difference in performance is reflected in QPI traffic. With NUMA affinity, data traffic mainly comes through the local QPI-1 link. Without NUMA affinity, a large amount of traffic comes through QPI-0 links or the remote QPI-1, and performance drops significantly. Sections IV and V present these results in more detail.

To mitigate these NUMA effects at high network throughput, we carefully design the send and receive paths. We encapsulate important structures and flow control in a "channel" concept that essentially corresponds to the end-to-end I/O path. We map channels to NICs and to NUMA nodes and allocate their resources in the same node where the NIC is attached.

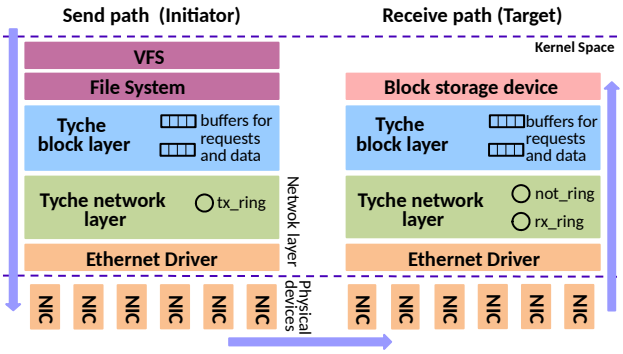


Figure 3: Overview of the send and receive path from the initiator (client) to the target (server).

I/O requests can use any channel. We dynamically detect the appropriate channel for each request, based on the location of the request data, and we direct each request accordingly. This approach aligns buffers and NICs and almost entirely eliminates NUMA effects.

Overall, results show that network storage protocols for modern servers with multiple resources need to be designed for NUMA affinity to achieve high network throughput. Otherwise, when affinity is not taken into account, performance is significantly downgraded.

The rest of this paper is organized as follows. Section II presents the necessary background on Tyche. Section III describes how Tyche deals with NUMA affinity. Sections IV, V and VI present our experimental results. Finally, we present related work in Section VII and we draw our conclusions in Section VIII.

II. Background

Tyche [4, 5] is an end-to-end network storage protocol on top of raw Ethernet that achieves high I/O throughput and low latency without hardware support (Figure 3). Tyche presents the remote storage device locally by creating at the client (initiator) a virtual local device that can be used as a regular device. Tyche is independent of the storage device and supports any file system. It provides reliable delivery, Ethernet-framing, and transparent bundling of multiple NICs.

To reduce message processing overhead, Tyche uses a copy reduction technique based on virtual memory page remapping, reduces context switches, and uses RDMA-type operations. The server (target) avoids all copies for writes by interchanging pages between the NIC receive ring and Tyche. The initiator requires a single copy for reads, due to OS-kernel semantics for buffer allocation. Tyche reduces overheads for small I/O requests by avoiding context switches for low degrees of I/O concurrency and by dynamically batching messages for high degrees of I/O concurrency. Tyche does not use RDMA over Ethernet, instead our protocol uses a similar, memory-oriented abstraction that allows us to reduce messaging overhead by avoiding packing and unpacking steps that are required over streaming-type abstractions, such as sockets. Additionally, there are several optimizations, such as avoiding dynamic memory allocations, that are typical in network protocol implementations.

Tyche uses small request messages for requests and completions, and data messages for data pages. A request message corresponds to a request packet that is sent using a small Ethernet frame. A data message corresponds to several data packets that are sent using Jumbo Ethernet frames of 4 or 8kB.

Tyche uses the concept of a communication "channel" to establish a connection between initiator and target. Each channel allows a host to send/receive data to/from a remote host. A channel is directly associated to the NIC that uses for sending/receiving data. Although a channel is mapped to a single NIC, several channels can be mapped to the same NIC. Tyche is able to simultaneously manage several channels, and it creates at least a single channel per NIC.

Each channel has two pre-allocated buffers, one for each kind of message, for sending and receiving messages. The initiator handles both buffers by specifying in the message header its position on them, and, on its reception, a message is directly placed on its buffer's position. At the target, the buffer for data messages contains lists of pre-allocated pages for sending and receiving data messages, and for issuing I/O requests to the local device. The initiator has no pre-allocated pages, it uses the pages of the I/O requests.

The initiator send path can operate in two modes. In the "inline" mode (Figure 3), the application context issues requests to the target with no context switch. In the "queue" mode, requests are inserted in a queue at the block level, and a thread dequeues them and

issues them. There is no other difference. Regarding performance, the inline mode outperforms the queue mode for small requests; the queue mode significantly outperforms the inline mode when there are many outstanding writes of large size. The target uses a work queue for sending completions back, because local I/O completions run in an interrupt context that cannot block.

At the receive path, a network thread per NIC processes packets and messages. When several channels are mapped to the same NIC, this thread will process packets for all the channels. At the block layer, several threads per channel process I/O requests.

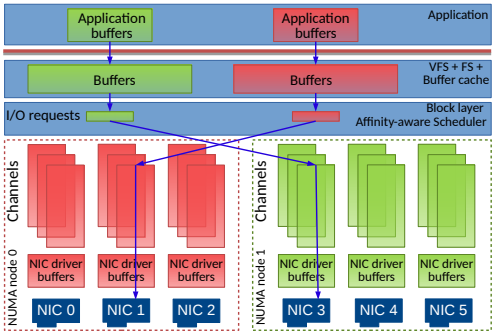


Figure 4: Affinity-aware scheduler selecting a channel for two I/O requests in a Tyche system with six NICs, three per NUMA node, and three channels per NIC.

III. Protocol design for NUMA affinity

To achieve high throughput in a NUMA architecture such as the one depicted in Figure 1 we need to consider affinity among different resources [2, 3]. In the I/O path, there are four resources related to NUMA affinity: application buffers, protocol data structures, kernel (I/O and NIC) data buffers, and NIC location in server sockets. The placement of threads plays a role as well, and it affects application threads, protocol threads, work queues, and interrupt handlers. Tyche orchestrates affinity of memory and threads by considering the system topology and the location of NICs. It creates a communication channel per NIC, and associates resources exclusively with a single channel.

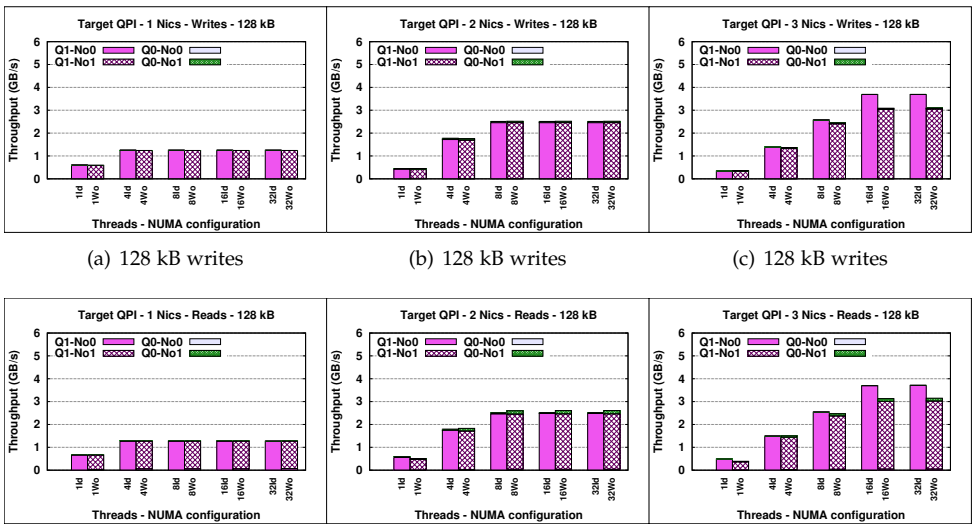
Each channel allocates memory for all purposes and pins its threads to the same NUMA node where its NIC is attached. For instance, in the architecture of Figure 1 a channel mapped to NIC-0 uses memory in Memory-0 and runs its threads in cores within Processor-0.

The NIC driver uses per NIC data structures: a transmission ring and two receive rings. We force the allocation of these rings in the same node where the NIC is attached as well, making them part of the NIC channel.

We implement a NUMA-aware work queue because in the Linux kernel we use it is not possible to apply affinity during assignment of tasks to work queues. Our work queue launches a thread per core that is pinned in its corresponding core. The target submits completion messages to the work queue by using its NUMA information. Conceptually, there is a work queue per channel.

There are a few remaining parts of the end-to-end path that are not affinity-aware: In the Linux kernel (a) it is not possible to control placement of buffer cache pages, (b) controlling application thread placement may have adverse effects on application performance, and (c) it is not possible to control placement of device I/O completions (on the target side). Next, we discuss how we deal with (a) and (b), whereas our results show that the impact of (c) is not significant.

To deal with affinity of I/O request buffers that are allocated before the request is passed to Tyche, we use an "assignment" approach. We allow requests to arrive with pre-allocated buffers, anywhere in the system. Then, we dynamically detect where buffers are allocated, we identify a NIC that is located in the same NUMA node as the request buffers, and we assign the request to a channel that uses this NIC. For this purpose, Tyche implements a scheduler to select a channel through which the next I/O request will be issued. If there are several channels on this node, it uses a fairness metric, by default equal kBs to each channel, to select one of them. Figure 4 depicts a Tyche system composed of six NICs, three per NUMA node, three channels per NIC, and the scheduling of two I/O requests. Our evaluation contrasts this affinity-based scheduling to a simple round-robin approach that does not consider buffer location and merely distributes I/O requests to channels in a round-robin manner.



(a) 128 kB writes (b) 128 kB writes (c) 128 kB writes

(d) 128 kB reads (e) 128 kB reads (f) 128 kB reads

Figure 5: QPI traffic, in GB/s, for Ideal (Id) and Worst (Wo), with FIO, 128 kB requests, and random writes and reads, and 1, 2 and 3 NICs.

IV. Experimental environment

Our experimental platform consists of two systems (initiator and target) connected back-to-back. Both nodes have two, quad core, Intel(R) Xeon(R) E5520 CPUs running at 2.7 GHz. The operating system is the 64-bit version of CentOS 6.3 testing with Linux kernel version 2.6.32. The target node is equipped with 48 GB DDR-III DRAM, and the initiator with 12 GB. The target uses 12 GB as main memory, and 36 GB as ramdisk. Each node has 6 Myricom 10G-PCIE-8A-C cards that are capable of about 10 Gbits/s throughput in each direction.

We use the open-source Intel ® Performance Counter Monitor (PCM) [6], that provides core-level CPU information, and supports different kind of metrics. We use the estimation of data traffic transferred through the Intel ® QuickPath interconnect links. For each node, the tool provides data traffic for "QPI-1" (inside the same node), and for "QPI-0" (traffic coming from a remote node). We analyze **Q1-No0**, **Q1-No1**, **Q0-No0** and **Q0-No1**, that correspond to the traffic for QPI-1 and QPI-0 in NUMA nodes 0 and 1, respectively. The traffic coming through "QPI-0" link is the traffic between processors (Figure 1), Q0-No0 corresponds to the data traffic to Processor-0 from Processor-1, and Q0-No1 to the traffic in the other direction. The tool does not provide the traffic between I/O hubs.

We use two micro-benchmarks: FIO (Flexible I/O) is a workload generator with many parameters, including number of threads, synchronous and asynchronous operations, request size, access pattern, etc. [7]. zmIO is an in-house benchmark that uses the asynchronous I/O API of the Linux kernel to issue concurrent I/Os with minimal CPU utilization [8]. In this work we start from the Tyche version implemented in Linux kernel 2.6.32 [4]. Tyche uses the queue mode (see Section II) to avoid lock contentions [4].

V. Degradation of I/O throughput due to NUMA

We first perform an analysis for the impact of NUMA effects on the throughput of networked storage I/O and the associated QPI traffic.

We use the baseline version of Tyche that has no support for NUMA effects. We examine the extent of

NUMA impact on throughput as follows. We attach 1, 2, or 3 NICs to NUMA node 0 and create one channel per NIC with round-robin scheduler. Given that all the NICs are on the NUMA node 0, the role of the scheduler is minimal. Then, we create two extreme configurations: Ideal and Worst. In Ideal, we manually allocate memory and threads for Tyche and the benchmark in node 0 where the NICs are also attached. In Worst, we allocate all memory and threads for Tyche and the benchmark in NUMA node 1.

We use FIO with direct I/O, random reads and writes of 128kB, and 60s of runtime. We run the test with 1, 4, 8, 16, and 32 application threads. The storage device is accessed in a raw manner (no file system). Figure 2 provides throughput, in GB/s, achieved by Tyche as a function of the number of application threads.

To analyze the QPI traffic due only to our networked storage protocol and exclude traffic due to the storage device, the target does not use the ramdisk during this test, and it completes the requests without performing the actual I/O. Figure 5 depicts, in GB/s, the traffic through each QPI-node link at the target during the test execution as a function of the number of application threads.

Regarding performance, with 1 and 2 NICs, both configurations obtain the same throughput, and there is no difference between them. With 3 NICs, Ideal significantly outperforms Worst by up to 23.7%. As we explain below, the reason is a bottleneck that appears on the QPI path followed by the data.

Regarding the QPI analysis, with Ideal, since both, NICs and resources, are in NUMA node 0, the QPI traffic is only through Q1-No0 (the Q1 link at NUMA node 0), and the throughput provided by this link is quite similar to the throughput provided by Tyche. There is no data traffic through Q0-No0, Q0-No1, and Q1-No1.

With Worst, the behavior is rather different, and the QPI traffic is through Q1-No1 (the Q1 link at NUMA node 1). The data goes through QPI-1 that connects I/O hub-1 with Processor-1 (Q1-No1) and through the QPI link that connects the I/O hubs (this traffic is not reported by the tool).

With 1 or 2 NICs, the data traffic generated does not saturate this path and the system achieves maximum performance. With 3 NICs, the amount of data traffic generated saturates this path and QPI-0 becomes the bottleneck. QPI and Tyche throughput drop by up to 26.2% and 23.9%, respectively.

Table 1: Configuration of the tests run for the NUMA study. RR stands for round robin scheduling

Test	NUMA affinity		Channel scheduler
	Tyche	Application	
Ideal	Yes	Yes	Affinity-aware
TyNuma	Yes	No	RR
Worst	No	No	RR

VI. Improvement due to protocol NUMA extensions

We now analyze the impact of NUMA effects depending on memory placement applied by Tyche and the application. To perform this analysis, we evaluate three configurations: Ideal, TyNuma and Worst. Table 1 summarizes these configurations. With Ideal, we manually configure the NUMA placement of the application: half of the application threads and their corresponding resources are allocated in each NUMA node. With TyNuma and Worst, we run the application without any affinity hint.

To perform this set of experiments, we use six NICs, three on each NUMA node, and we open one channel per NIC. Now, the target uses the ramdisk, and it performs the actual I/O. Consequently, at the target, there is data traffic due to the network traffic and due to the copy of the ramdisk, and we are not able to distinguish between them. Therefore, we only analyze the QPI traffic at the initiator.

To achieve maximum performance, NUMA affinity should be applied not only by Tyche but also by the application. Therefore, the performance achieved depends also on the placement performed by the application. In addition, it is interesting to see if our protocol extensions can hurt performance for hand-tuned applications. For this reason, we examine two cases: (a) Regular applications that are not tuned for NUMA. For this purpose we use zmIO that allocates resources (threads and buffers) without any particular attention to NUMA. (b) Hand-tuned applications. For

this purpose we use FIO that allocates resources in a balanced manner.

Do protocol extensions for NUMA help performance? We perform the analysis with zmIO, because when zmIO is run without affinity hint, it allocates 99% of writes and around 75% of reads to a single NUMA node (node-0). Therefore, almost all writes issued to channels allocated in node 1 have their resources allocated in node 0, and for reads, this rate is only 50%. Consequently, with zmIO, performance also depends on the request type.

We run zmIO with random reads and writes, direct I/O, request sizes of 64 kB, 128 kB, and 512 kB, and a runtime of 60 s. The remote storage device is accessed as a raw device (no file system). We run 1, 8, 16, and 32 application threads. Since this test is base on time, each time a different amount of data is read or written. Figure 6 provides throughput, in GB/s, achieved by Tyche as a function of the number of application threads. Figure 7 depicts the percentage of the total traffic through each QPI-node link as a function of the application threads and configuration.

For writes, Figure 6 shows that only by applying the right placement, Ideal configuration, Tyche achieves its maximum throughput, being 6.77 GB/s with 32 threads and 512 kB request size. Figure 7 shows that with Ideal, almost all the data traffic comes through the QPI-1 link, having a similar amount of traffic both nodes.

With Worst (the opposite case), Tyche only obtains up to 4.67 GB/s again with 32 threads and 512 kB request size. Indeed, by applying affinity, Ideal outperforms Worst by up to 85%. Figure 7 shows that for writes, Worst only has data traffic through the QPI-1 link on node 0, since almost all the user requests are allocated in this node. There is no data traffic through QPI-1 on node 1, and there is a significant amount of traffic through QPI-0 as well.

With TyNuma, Tyche only achieves up to 5.00 GB/s again with 32 threads and 512 kB request size. Ideal improves throughput up to 37.60%. Figure 7 shows that TyNuma behaves like Worst, and the data traffic through the QPI-1 link is mainly on node 0.

For writes, due to the QPI data traffic (see Figure 7), Worst and TyNuma are not able to provide better performance. TyNuma outperforms Worst because, at the Target, TyNuma is applying NUMA affinity, whereas Worst is not.

For reads, Ideal and TyNuma achieve up to 6.86 GB/s and 6.78 GB/s, respectively, whereas, Worst only up to 4.79 GB/s. Ideal improves throughput by up to 84.8% comparing with Worst.

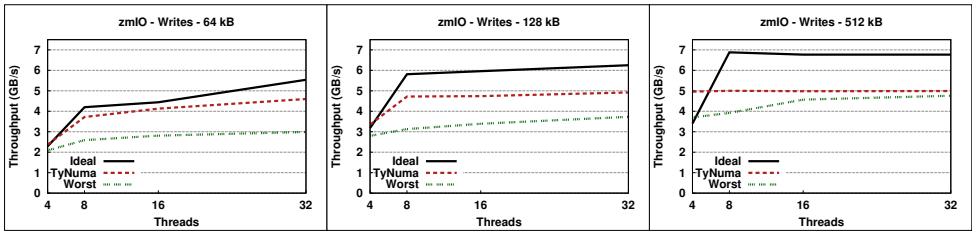
As we can see in Figure 7, this difference in performance between Ideal and Worst is due to the QPI traffic. With reads, Ideal has all the data traffic through the QPI-1 links, having both nodes the same amount of traffic. However, Worst has up to 33% of the total traffic coming through QPI-0.

When comparing Ideal and TyNuma, there only is a small difference in throughput and they exhibit a quite similar behavior. However, regarding QPI traffic, TyNuma behaves more similar to Worst. With TyNuma, at the initiator the QPI traffic is quite similar to with Worst, but at the target, the QPI traffic is the same as with Ideal. At the target, Ideal and TyNuma are applying memory and threads placement, and having a quite similar behavior. With Worst, the target does not apply affinity, so a significant amount of traffic goes through QPI-0 links and consequently the throughput drops significantly. At the initiator, since application buffers are allocated there, QPI traffic with TyNuma behaves like with Worst, since, with both, the application is not applying NUMA affinity, and the application buffers are only allocated at the initiator.

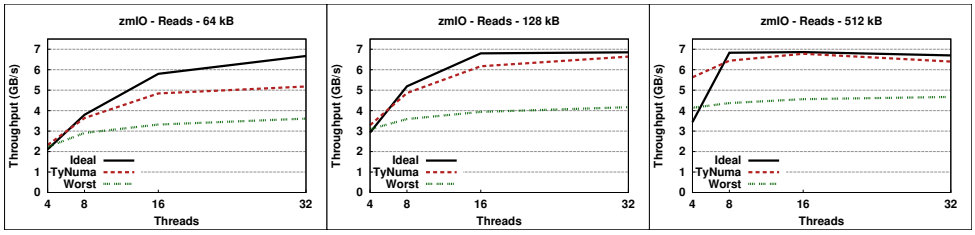
Note that with only 4 threads, all of them, and their resources, are allocated in NUMA node 0, for this reason, with Ideal, there is only data traffic through the QPI-1 link of node 0.

Do protocol extensions for NUMA hurt performance for hand-tunned applications? We analyze the QPI traffic with FIO. When we run FIO with no affinity hint, FIO, by itself, makes a quite balanced distribution of resources. Consequently, even selecting the channel in a round-robin order, around 50% of the requests are issued through a channel allocated in the same node where the request's buffers are allocated.

We use FIO with random reads and writes, direct I/O, a 256 MB file size, and 4 kB, 128 kB, and 512 kB request sizes. We run 1, 4, 8, 16, and 32 application threads. Each thread has its own file and makes 30

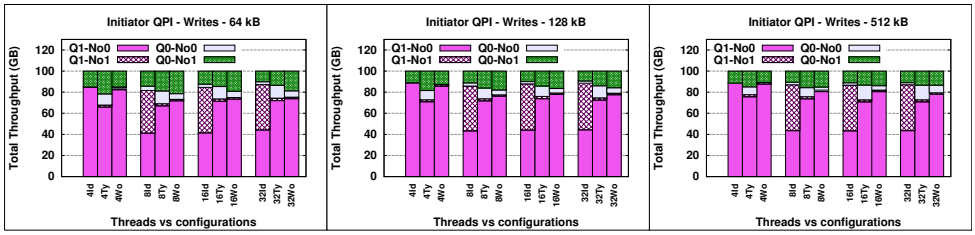


(a) 64 kB writes (b) 128 kB writes (c) 512 kB writes

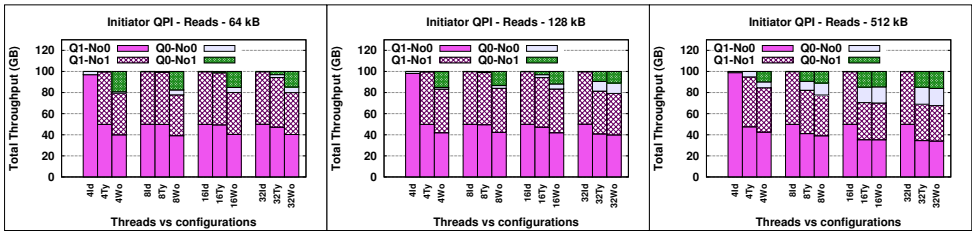


(d) 64 kB reads (e) 128 kB reads (f) 512 kB reads

Figure 6: Throughput, in GB/s, achieved by Tyche depending on the affinity, with zmIO for 64 kB, 128 kB and 512 kB requests, and random writes and reads.



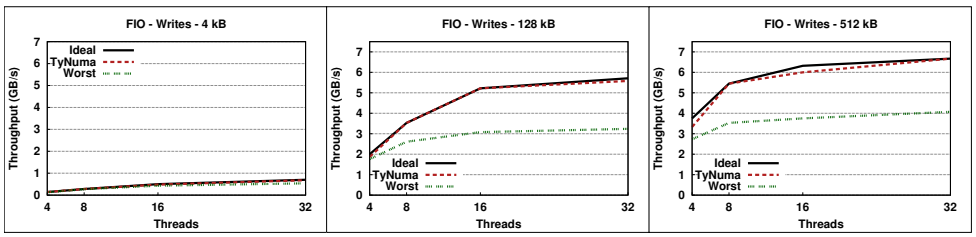
(a) 64 kB writes (b) 128 kB writes (c) 512 kB writes



(d) 64 kB reads (e) 128 kB reads (f) 512 kB reads

Figure 7: Percentage of QPI for Tyche depending on the affinity, with zmIO for 64 kB, 128 kB and 512 kB request size, and random writes and reads.

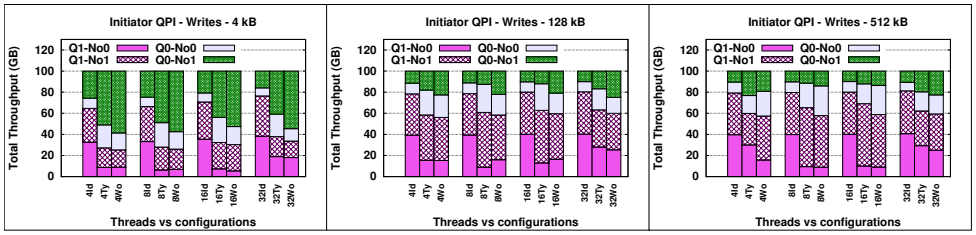
iterations over it, thus, with all the request sizes, the same amount of data is always read or written. We use XFS as the file system. Figure 8 provides throughput, in GB/s, achieved by Tyche as a function of the number of application threads. Figure 9 depicts the percentage of the total traffic through each QPI-node link as a



(a) 4 kB writes (b) 128 kB writes (c) 512 kB writes

(d) 4 kB reads (e) 128 kB reads (f) 512 kB reads

Figure 8: Throughput, in GB/s, achieved by Tyche depending on the affinity, with FIO for 4 kB, 128 kB and 512 kB request size, and random writes and reads.



(a) 4 kB writes (b) 128 kB writes (c) 512 kB writes

(d) 4 kB reads (e) 128 kB reads (f) 512 kB reads

Figure 9: Percentage of QPI for Tyche depending on the affinity, with FIO for 4 kB, 128 kB and 512 kB request size, and random writes and reads.

function of the application threads and configuration.

Figure 8 shows that with Ideal, Tyche obtains its maximum throughput, being 6.67 GB/s for writes and reads, with 32 threads and 512 kB request size. Whereas, with Worst, Tyche obtains only up to 4.21 GB/s and 4.07 GB/s for writes and reads, re-

spectively, also with 32 threads and 512 kB request size. Indeed, by applying the right placement (Ideal configuration), Tyche improves throughput up to 76% (32 threads and 128 kB writes) when comparing with the Worst configuration.

As we can see in Figure 9, this difference in performance is due to the QPI traffic, as explained for reads with zmIO. With Ideal, almost all the traffic comes through the QPI-1 link, and both nodes have a similar amount of traffic. But with Worst, a significant amount of traffic, up to 48% of the total, comes through the QPI-0 link.

Note that although Ideal exhibits perfect placement, for writes, there is traffic at QPI-0 due to cacheline invalidations to the remote NUMA node. With reads, this type of traffic is not present.

TyNuma behaves similar to Ideal due to the QPI traffic at the target, as explained for zmIO. The target applies affinity for both Ideal and TyNuma, so they behave quite similar. In Worst, the target does not apply affinity, therefore a significant amount of traffic goes through QPI-0 and the throughput drops significantly.

This behavior is presented also for small requests. For 4 kB requests, memory placement has also a significant impact, and Ideal improves throughput up to 66.2% and 44.6% for writes and reads, respectively, compared to Worst. Again, there is a small difference in performance between Ideal and TyNuma.

VII. Related work

A lot of work has been done for NUMA-aware process scheduling and memory management in the context of many-core processors and systems. Regarding I/O performance, for instance, Mavridis *et al.* propose Jericho [9], an I/O stack that consists of a NUMA-aware file system and a DRAM cache organized in slices mapped to NUMA nodes. Their results show that Jericho improves performance up to 2× by doing more than 95% of memory accesses local. Zheng *et al.* [10] propose a scalable user-space cache for NUMA machines. By partitioning the cache by processors, they break the page buffer pool into a large number of small page sets and manages each set independently. Note that, in this work, we show that NUMA placement is a key aspect to achieve maximum throughput with *network storage* protocols as well.

Regarding NUMA and networking, Moreaud *et al.* [11] study NUMA effects on high-speed networking in multi-core systems and show that placing a task on a node far from the network interface leads to a performance drop, and especially bandwidth. Their results show that NUMA effects on throughput are asymmetric since only the target destination buffer appears to need placement on a NUMA node close to the interface. In our case, NUMA affects both sides, target and initiator.

Ren *et al.* [12] propose a system that integrates an RDMA-capable protocol (iSER), multi-core NUMA tuning, and an optimized back-end storage area network. They apply NUMA affinity by using the `numactl` utility for binding a dedicated target process to each logical NUMA node, and achieve an improvement of up to 19% in throughput for write operations. In contrast, Tyche applies NUMA affinity, at both initiator and target, by defining channels that are mapped them to NICs and to NUMA nodes, and their resources are allocated in the same node where the NIC is attached. This approach almost entirely eliminates NUMA effects, and achieves an improvement of up to 85%.

Dumitru *et al.* [13] also analyze, among other aspects, the impact of NUMA affinity on NICs capable of throughput at the range of 40 GBits/s, without, however, to propose a solution. Pesterev *et al.* [14] analyze NUMA effects on TCP connections by proposing Affinity-Accept that ensures that all processing for a given TCP connection to occur on the same core. They reduce time spent in the TCP stack by 30% and improves overall throughput by 24%. They use the NICs to spread incoming packets among many RX DMA rings to ensure packets from a single flow always map to the same core. However, our study shows that even the NIC resources should be allocated in the same NUMA node where the NIC is attached to obtain maximum performance.

VIII. Conclusions

Here, we analyze and evaluate the impact of NUMA affinity on the network layer supporting the converged storage paradigm over high-speed Ethernet. We analyze the impact of memory placement by studying

the amount of data traffic through the Intel ® QPI links. This analysis shows that NUMA effects can have a large negative impact on performance, reducing network throughput up to 2x.

To mitigate NUMA effects, we encapsulate all protocol data structures and flow control in "channels" that essentially correspond to the structures required to serve a request through the full end-to-end I/O path. Then, we carefully map channels to NICs and NUMA nodes to ensure proper affinity. Our approach improves throughput by up to 85%, to a large extent eliminating inter-processor QPI traffic and NUMA effects.

Acknowledgment

We thankfully acknowledge the support of the European Commission under the 7th Framework Programs through the NanoStreams (FP7-ICT-610509) project, the HiPEAC3 (FP7-ICT-287759) Network of Excellence, and the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.

References

[1] An Introduction to the Intel® QuickPath Interconnect. http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html, 2009.

[2] Matthew Dobson, Patricia Gaughen, Michael Hohnbaum, and Erich Focht. Linux Support for NUMA Hardware. In *Ottawa Linux Symposium*, 2003.

[3] Christoph Lameter. Local and Remote Memory: Memory in a Linux/NUMA System. In *Ottawa Linux Symposium*, 2006.

[4] Pilar González-Férez and Angelos Bilas. Tyche: An efficient Ethernet-based protocol for converged networked storage. In *Proceedings of the IEEE Conference on MSST*, 2014.

[5] Pilar González-Férez and Angelos Bilas. Reducing CPU and network overhead for small I/O requests in network storage protocols over raw Ethernet. In *Proceedings of the IEEE Conference on MSST*, 2015.

[6] Thomas Willhalm (Intel). Intel ® Performance Counter Monitor - A better way to measure CPU utilization. https://software.intel.com/en-us/articles/intel-performance-counter-monitor, 2012.

[7] FIO Benchmark. http://freecode.com/projects/fio.

[8] zmIO Benchmark. http://www.ics.forth.gr/carv/downloads.html.

[9] Stelios Mavridis, Yannis Sfakianakis, Anastasios Papagiannis, Manolis Marazakis, and Angelos Bilas. Jericho: Achieving Scalability through Optimal Data Placement on Multicore systems. In *Proceedings of the IEEE Conference on MSST*, 2014.

[10] Da Zheng, Randal Burns, and Alexander S. Szalay. Toward millions of file system iops on low-cost, commodity hardware. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013.

[11] Stéphanie Moreaud and Brice Goglin. Impact of NUMA effects on high-speed networking with multi-opteron machines. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, 2007.

[12] Yufei Ren, Tan Li, Dantong Yu, Shudong Jin, and Thomas Robertazzi. Design and Performance Evaluation of NUMA-aware RDMA-based End-to-end Data Transfer Systems. In *Proceedings of international conference for High Performance Computing, Networking, Storage and Analysis*, 2013.

[13] Cees de Laat Cosmin Dumitru, Ralph Koning. 40 Gigabit Ethernet: Prototyping Transparent End-to-End Connectivity. In *Proceedings of the Terena Networking Conference*, 2011.

[14] Aleksey Pesterev, Jacob Strauss, Nickolai Zeldovich, and Robert T. Morris. Improving network connection locality on multicore systems. In *Proceedings of the 7th ACM European Conference on Computer Systems*, 2012.

Evaluating data caching techniques in DMCF workflows using Hercules

Francisco Rodrigo Duro†, Fabrizio Marozzo∗, Javier Garcia Blas†

frodrigo@arcos.inf.uc3m.es, fmarozzo@dimes.unical.it, fjblas@arcos.inf.uc3m.es

Jesus Carretero†, Domenico Talia∗, Paolo Trunfio∗

jesus.carretero@uc3m.es, talia@dimes.unical.it, trunfio@dimes.unical.it

† ARCOS, University Carlos III, Spain
∗ DIMES, University of Calabria, Italy

Abstract

The Data Mining Cloud Framework (DMCF) is an environment for designing and executing data analysis workflows in cloud platforms. Currently, DMCF relies on the default storage of the public cloud provider for any I/O related operation. This implies that the I/O performance of DMCF is limited by the performance of the default storage. In this work we propose the usage of the Hercules system within DMCF as an ad-hoc storage system for temporary data produced inside workflow-based applications. Hercules is a distributed in-memory storage system highly scalable and easy to deploy. The proposed solution takes advantage of the scalability capabilities of Hercules to avoid the bandwidth limits of the default storage. Early experimental results are presented in this paper, they show promising performance, particularly for write operations, compared to the performance obtained using the default storage services.

Keywords DMCF, Hercules, data analysis, workflows, in-memory storage, Microsoft Azure

I. Introduction

In the last decade, most of the scientific computing problems are increasing their needs to process large quantities of data. Large simulations, data visualization, and big data problems are some of the application areas leading the trends in scientific computing. This evolution is moving needs from a computing-centric power point of view to a data-centric approach. Current trends in High Performance Computing (HPC) also include the use of cloud infrastructures as a flexible approach to virtually limitless computing resources. Given this current scenario, a solution that combines HPC, data analysis, and cloud computing is becoming more and more necessary.

According to their elastic feature, cloud computing infrastructures can serve as effective platforms for addressing the computational and data storage needs of most big data applications that are being developed nowadays. However, coping with and gaining value from cloud-based big data requires novel software tools and advanced analysis techniques. Indeed, advanced data mining techniques and innovative tools can help users to understand and extract what is useful in large and complex datasets for making informed decisions in many business and scientific applications.

The Data Mining Cloud Framework (DMCF), developed at University of Calabria, is an environment for designing and executing data analysis workflows in

cloud platforms. Currently, DMCF uses the storage provided by the cloud provider for any I/O related job. This implies that the I/O performance of DMCF is limited by the performance of the default storage. Moreover, it is influenced by the contention that occurs when other I/O tasks are concurrently executed in the same region. Finally, the cost of using persistent storage service to store temporary data should be also taken into account.

The solution proposed here consists in using Hercules as the default storage system for temporary data produced in workflows. Hercules is a distributed in-memory storage system, easy to deploy and highly scalable. This system has been developed in the ARCOS research group, at University Carlos III Madrid, and it has also been proved in traditional HPC cluster with promising results.

This novel approach has three main objectives. The first one is taking advantage of the scalability of Hercules to avoid the bandwidth limits of the default storage. When the number of Hercules I/O nodes increases, the total available aggregated bandwidth usable by worker nodes is enhanced. The second objective is to allow the deployment, thanks to the easy deployment of Hercules, of an ad-hoc and independent in-memory storage system to avoid the contention produced during peak-loads in the cloud storage service. The last objective is the independence from the cloud platform used. While each cloud infrastructure have different APIs to access their storage services, Hercules has interfaces for commonly used APIs (like POSIX-like, put/get, MPI-IO) in order to imply minor modifications to existing code.

The main focus of this work is to deploy Hercules on a cloud infrastructure together with DMCF and to evaluate their performance with respect to the cloud storage service in different scenarios. This preliminary evaluation is aimed at demonstrating the capabilities of Hercules to be used as temporary storage of data analysis applications developed using DMCF.

The remainder of the paper is structured as follows. Section II describes the main features of DMCF. Section III introduces Hercules architecture and capabilities. Section IV emphasizes the advantages of integrating DMCF and Hercules and outlines how this integration will work. Section V presents preliminary results of the performance achieved by Hercules in the Azure cloud infrastructure and compares the results with Azure Storage. Section VI briefly presents other research work in the same field. Finally, section VII concludes the work and give some future research related to the presented work.

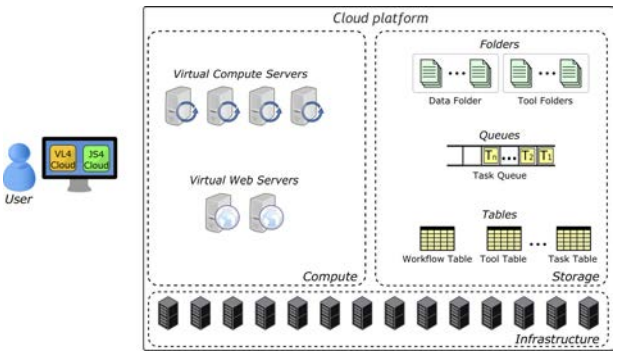


Figure 1: Architecture of Data Mining Cloud Framework.

II. Data Mining Cloud Framework

The Data Mining Cloud Framework (DMCF) [6] is a software system designed for designing and executing data analysis workflows on Clouds. A Web-based user interface allows users to compose their applications and to submit them for execution to the Cloud platform, following a Software-as-a-Service (SaaS) approach.

The architecture of DMCF includes different components that can be grouped into storage and compute components (see Figure 1). The storage components include:

- A *Data Folder* that contains data sources and the results of knowledge discovery processes. Similarly, a *Tool Folder* contains libraries and executable files for data selection, pre-processing, transformation, data mining, and evaluation of results.
- *Data Table*, *Tool Table* and *Task Table* contain metadata information associated with data, tools, and tasks.
- The *Task Queue* contains the tasks that are ready for execution.

The compute components are:

- A pool of *Virtual Compute Servers*, which are in charge of executing the data analysis tasks.
- A pool of *Virtual Web Servers* that host the Web-based user interface.

The DMCF architecture has been designed to be implemented on top of different Cloud systems. The implementation used in this work is based on Microsoft Azure[1].

A user interacts with the system to perform the following steps for designing and executing a knowledge discovery application:

1. The user accesses the Website and designs the workflow through a Web-based interface.
2. After submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the workflow.
3. Each idle Virtual Compute Server picks a task from the Task Queue, and concurrently executes it.
4. Each Virtual Compute Server gets the input dataset from the location specified by the workflow. To this end, a file transfer is performed from the Data Folder where the dataset is located to the local storage of the Virtual Compute Server.
5. After task completion, each Virtual Compute Server puts the results on the Data Folder.
6. The Website notifies the user as soon as her/his task(s) have completed, and allows her/him to access the results.

The set of tasks created on the second step depends on how many data analysis tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue. All the potential parallelism of the workflow is exploited by using all the needed Virtual Compute Servers.

DMCF allows to program data analysis workflows using two languages:

- VL4Cloud (Visual Language for Cloud), a visual programming language that lets users develop applications by programming the workflow components graphically.
- JS4Cloud (JavaScript for Cloud), a scripting language for programming data analysis workflows based on JavaScript [6].

Both languages use two key programming abstractions:

- Data elements, denoting input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
- Tool elements, denoting algorithms, software tools or complex applications performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

Another common element is the Task concept, which represents the unit of parallelism in our model. A task is a Tool invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a data-driven task parallelism. This means that, as soon as a task does not depend on any other task in the same workflow, the runtime asynchronously spawns it to the first available virtual machine (VM). A task Tj does not depend on a task Ti belonging to the same workflow (with $i \neq j$), if Tj during its execution does not read any data element created by Ti.

III. Hercules

Hercules [3] is a distributed in-memory storage system based on the key/value Memcached database [4]. The distributed memory space can be used by the applications as a virtual storage device for I/O operations and has been specially adapted in this work for being used as in-memory shared storage for cloud infrastructures. Our solution relies on an improved version of Memcached servers, which provides an alternative storage solution to the default storage service.

As can be seen in the Figure 2, Hercules architecture has two levels: worker library and servers. On top

[1] http://azure.microsoft.com

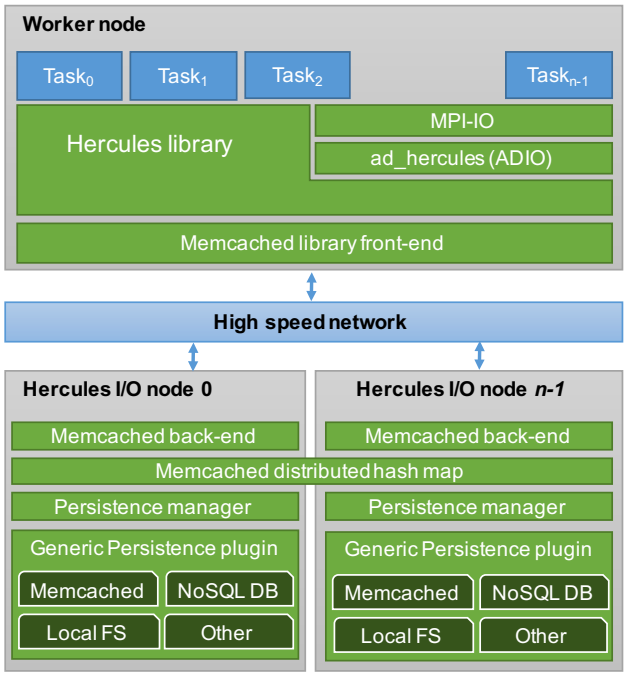


Figure 2: Hercules architecture. On the top the worker side, a user-level library. On the bottom the server side with the Hercules I/O nodes divided in modules.

is the worker user-level library with a layered design. Back-ends are based on the Memcached server, extending its functionality with persistence and tweaks. Main advantages offered by Hercules are: scalability, easy deployment, flexibility, and performance.

Scalability is achieved by fully distributing data and metadata information among all the nodes, avoiding the bottlenecks produced by centralized metadata servers. Data and metadata placement is completely calculated in the worker-side by a hash algorithm. The servers, on the other hand, are completely stateless.

Easy deployment and flexibility at worker-side are tackled using a POSIX-like user-level interface (open, read, write, close, etc.) in addition to classic put/get approach existing in current NoSQL databases. The existing software requires minimum changes to run with Hercules. The layered design allows for performing any future change with the minimum required effort. Servers can be deployed in any kind of Linux systems at user level. Persistence can be easily configured us-

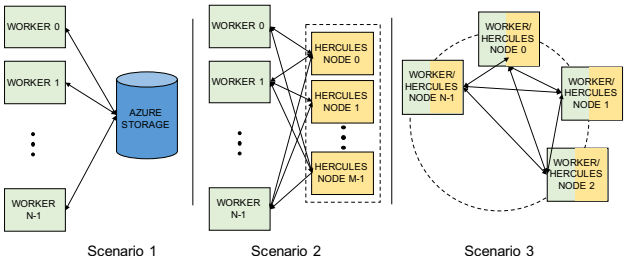


Figure 3: Deployment scenarios for the combination of Hercules and DMCF infrastructures.

ing the existing plugins or developing new ones. An MPI-IO interface is also available for legacy software relying on MPI as communication system.

Finally, performance and flexibility at server-side are targeted by exploiting the parallel I/O capabilities of Memcached servers. Flexibility is achieved by Hercules due to its easiness to be deployed dynamically on as many nodes as necessary. Each node can be accessed independently, multiplying the total throughput peak performance. Furthermore, each node can serve requests in a concurrent way thanks to a multi-threading approach. The combination of these two factors results in full scalability: both when the number of nodes increases and when the number of workers running on each node increases.

IV. Integration between DMCF and Hercules

The final objective of this joint research work is the integration of DMCF and Hercules. As can be seen in Figure 3, Hercules and DMCF can be configured in more than one deployment scenarios to achieve different levels of integration.

The first scenario shows the current approach of DMCF, where every I/O operation is done against the cloud storage service offered by the cloud provider, which is Azure Storage in this work. While this storage service is suitable for persistent data, it could be inefficient for temporary data. The main benefits of a cloud storage service are the convenience of using every tool offered by the same provider and the persistence options offered, even in different geographical regions.

Nevertheless, there are, at least, four disadvantages about this approach. First, proprietary interfaces and tools to access the storage service offered by different providers. Second, the performance offered by this services could have limitations that can not be avoided and performance could not be stable when there are peaks of use by other users. Third, the storage services are offered in a closed configuration, and can not be customized to the necessities of users at any time. Fourth, the cloud philosophy is tightly related with the pay-per-use concept. However, it does not make sense to pay for temporary data as if it was persistent data.

The second scenario, and the first contribution of this paper, is to use Hercules as the default storage for temporary generated data. Temporary data is becoming more and more popular in data analysis and many-task based applications. Most of these applications are developed as a sequence of tasks that communicate by using temporary files. Hercules I/O nodes can be deployed on as many VM instances as needed by the user depending on the required performance and the characteristics of data. Even the instance type can be configured according to the necessities of each different application. As stated in Section III, Hercules offers different user-level interfaces such as POSIX-like, put/get, and MPI-IO, allowing a more flexible deployment of legacy applications than the default cloud storage service. Cost-wise it is needed to better study the competition between using a persistence-focused service against launching Hercules I/O node instances as temporary storage.

The third scenario shows an even tighter integration of DMCF and Hercules infrastructures. In this scenario Hercules I/O nodes share virtual instances with the DMCF workers. If the data needed by the DMCF worker is stored inside the Hercules I/O node running in the same instance, it will not be necessary to use the network for accessing data, and every I/O operation will be completely local. This functionality, paired with the improved data placement algorithm that stores all the data related with one file in the same Hercules I/O node, and with a DMCF scheduler that co-locates the tasks in the nodes where the data is stored, can lead to even better performance, exposing and exploiting data locality.

Before implementing the system integration, we need to analyze the potential performance improvement that Hercules can offer on a public cloud infrastructure, specially against Azure Storage, which is the storage service chosen in the current DMCF implementation. This preliminary evaluation is presented in Section V.

V. Evaluation

As mentioned before, to demonstrate the capabilities of Hercules in accelerating the I/O operations of DMCF workers, we evaluated the performance of the Azure Storage service against our proposed solution. For this purpose, we have designed and implemented a simple benchmark, referred from now on as *Filecopy Benchmark*. In this benchmark, a configurable number of workers perform two simple tasks per worker: the first one is writing files to the configured storage (Azure Storage or Hercules) and, after the write task is complete, a read task starts over the data written previously. The benchmark is fully configurable in terms of:

- *Number of worker nodes*: each worker node is a VM deployed in Azure.
- *Number of workers per node*: worker processes running in the same node in parallel. This parameter is important to evaluate how the storage solutions will behave in multi-core architectures and how they perform when different worker processes share the same network interface.
- *File size*: the total size in MegaBytes (MB) of the file can be configured to simulate different problem sizes.
- *Chunk size*: in Azure storage, a BLOB object is divided into blocks (maximum block size of Azure Storage is 4 MB, not enough for large files). The Java library used for accessing to Azure Storage, automatically divides a block object in the required number of block objects. In addition to this behavior, our implementation divides a file into different BLOB objects. Chunk size parameter is the size of each of the block objects that are part of a complete file. In Hercules, it corresponds to

Table 1: Azure instance type characteristics.

Type	Cores	RAM (GB)	Bandwidth (Mbps)[1]	Price
A0	1	0.75	<100	0.0
A1	1	1.75	~240	0.0
A2	2	3.50	~480	0.1
A3	4	7.00	~960	0.2
A4	8	14.00	~1700	0.4
D1	1	3.50	~480	0.0
D2	2	7.00	~900	0.1
D3	4	14.00	~1600	0.2
D4	8	28.00	~2000	0.776

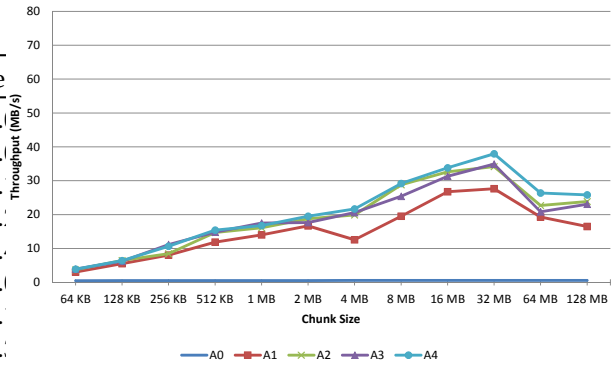
the buffer size of the POSIX write operation. Internally, Hercules divides the files in blocks adapted to the key-value hashmap of Memcached.

The computing resources used during the evaluation are completely based on Microsoft Azure. Table 1 shows the characteristics of the different instance types used during our evaluation. All the resources used were located on the "Western Europe" region and the OS installed on the VMs was Ubuntu 14.04 LTS. It is also worth to be noted that, as the objective of the research work is to use Hercules as temporary storage, persistence features are disabled.

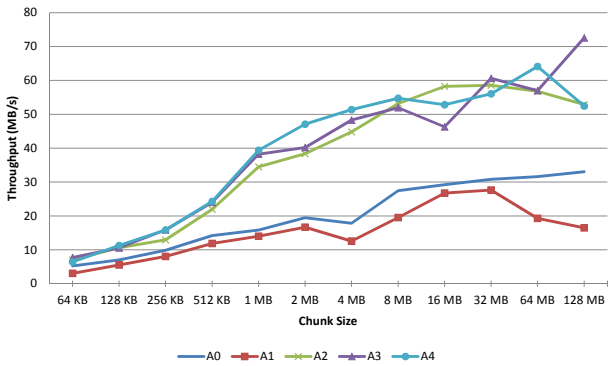
V.1 Chunk size evaluation

For the first evaluation case, we have fixed the file size to 128 MB, to have a file size that is big enough to show the performance with different chunk sizes. The chunk size will vary during the evaluation and we have used the five standard (A0-A4) instance types. Figure 4(a) shows the performance achieved during the write operations and Figure 4(b) the read operations performance. As it can be seen in these figures, Azure Storage performs much better for read (up to 72 MB/s) than for write operations (up to 38 MB/s). Also, the performance increases with the chunk size, achieving the best performance around the 32 MB mark. Finally, it is interesting to note how the performance varies

[1]Bandwidth measured experimentally using *iperf* tool between two VMs of the same instance type in the same region.



(a) Throughput of Azure Storage by using the Filecopy Benchmark (128 MBytes) for evaluating the block size for writes.



(b) Throughput of Azure Storage by using the Filecopy Benchmark (128 MBytes) for evaluating the block size for reads.

Figure 4: File copy benchmark configured for evaluating the Azure Storage performance depending on the block object size.

with the instance type used: as expected, the most expensive instances have the better performance.

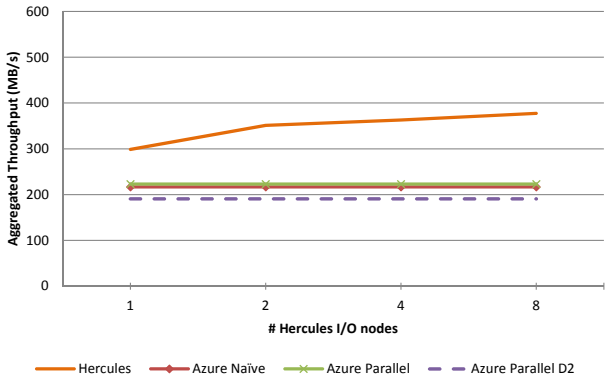
V.2 Hercules I/O nodes scalability

The next phase in the evaluation process is the measurement of the performance difference between Azure Storage and Hercules using different configurations. Also, we evaluate how Hercules scales its performance as the number of deployed I/O nodes increases. Based on the preliminary nature of this evaluation, our budget was limited to VMs running with a maximum

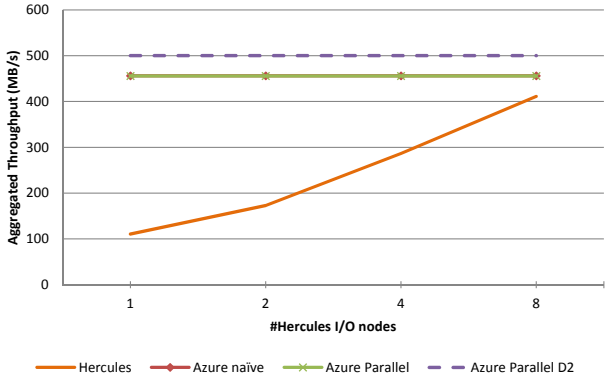
number of 25 cores in total. After some quick bandwidth evaluation cases (results showed in Table 1), we selected D1 and D2 instances as the best performers in network bandwidth per core ratio. D1 instances achieve a peak performance of 60 MB/s using one core while D2 tops at around 115 MB/s with two cores, managing to reach almost the best possible performance of the available Gigabit virtual network interface. This is 2x the bandwidth available per core compared with Standard 'AX' instance types. In the future, it would be interesting to evaluate the performance achieved by Hercules running in the A8 and A9 network optimized instances with Infiniband network, and 56 and 112 Gigabytes of RAM respectively. This network optimized instances should be the optimal option for running Hercules I/O nodes.

The final selection for this test is 8 VMs (D1 instances) as worker nodes and up to 8 VMs (D2 instances) as Hercules I/O nodes. Figure 5 plots the filecopy benchmark results, configuring the experiment with a file size of 512 MB, with 32 MB of chunk size and executing one read/write operation per worker node (one worker process per node) which implies a 4096 MB problem size (512 MB x 8 worker nodes). We have compared four different cases. The first one is the performance obtained by Hercules using between 1 and 8 I/O nodes. The second case is Azure Storage baseline approach, using the default access pattern offered by the Java API, without any optimizations. Third case is Azure Storage applying some optimizations to the code, specially important is setting up the *BlobRequestOptions* object property *setConcurrentRequestCount* with 8 threads per process, using 8 concurrent threads to parallel access to Azure Storage. The last case can not be directly compared with the performance achieved by Hercules, because it uses the reserved D2 instances as worker nodes, instead of using them as I/O nodes, to show the peak performance achievable by Azure Storage with fully working Gigabit interface, hence the dotted line. In the Hercules case, the peak performance is limited by the aggregated bandwidth available worker-side (8x60 MB/s ~480 MB/s) not by the server-side 8x115 MB/s (~920 MB/s).

Figure 5(a) shows the performance evolution as the number of Hercules I/O nodes increase compared



(a) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for writes. We set up the experiment with 8 worker nodes, writing 512 MBytes each one (4 GBytes in total).



(b) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for reads. We set up the experiment with 8 worker nodes, reading 512 MBytes each one (4 GBytes in total).

Figure 5: File copy benchmark configured for evaluating the Hercules I/O nodes scalability. 8 worker processes running on 8 worker nodes access 4 Gigabyte of data. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in the best read cases.

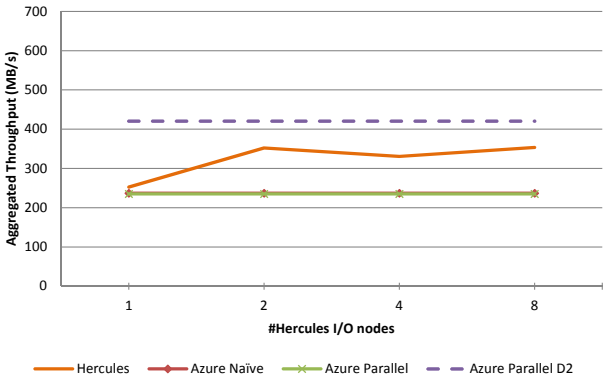
to the different Azure Storage approaches. The figure clearly demonstrates how Hercules performance tops near the 400 MB/s mark, which is near the maximum theoretical peak performance of 8x60 MB/s (~480

MB/s). This peak performance achieved using 8 I/O nodes for parallel access is nearly 2x the performance achieved by Azure Storage in any of the configurations. Some interesting sights in the Azure Storage side are how both the baseline and the parallel approach performance is nearly identical caused by only being one core available in D1 instances. Also, it is interesting how the D2 instances performance using parallel accesses is even lower, exposing the deficiencies of Azure Storage performance in write operations.

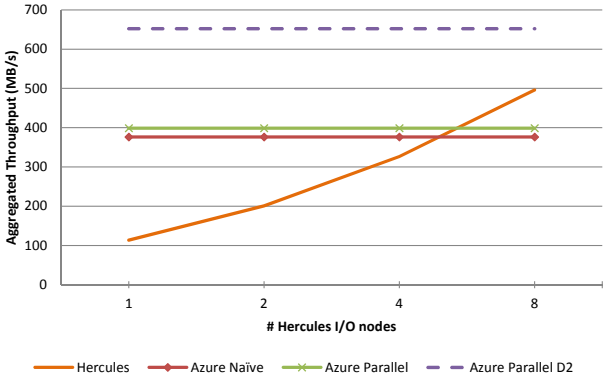
In Figure 5(b), which depicts the read operations performance, can be clearly seen how the Hercules performance evolves as the number of I/O nodes available increases. With only one I/O node available, the performance is ~100 MB/s, the maximum offered by the network interface of the I/O node (D2 instance). As the number of I/O nodes increases, the performance evolves, reaching a peak performance of ~400 MB/s, again near the theoretical up mark of 480 MB/s and near the performance of Azure Storage that slightly outperforms Hercules in this case. Azure Storage performs at the peak performance of the available network, with same performance in naive and parallel approaches using D1 instances while performing marginally better when D2 instances are used as worker nodes.

Third evaluation case is an evolution of the previous test for a scenario with higher congestion using the same infrastructure (8 D1 instances as worker nodes and 8 D2 instances as Hercules I/O nodes). In this case, instead of having 1 worker running on each node, we launched 4 workers running in parallel on each of the worker nodes, keeping the problem size in 4096 MB. For this purpose, each worker process writes, and then reads, a 128 MB file, with the same chunk size of 32 MB.

Figure 6(a), showing the performance in write operations, reports a very similar behavior of Hercules compared to the previous test case, but achieving a lower peak performance. At the same time, Azure Storage performance with D1 instances increases and the difference between Hercules and Azure Storage is narrowed to a 50% difference in favor of Hercules. Furthermore, using more than one process per node in the dual-core D2 instances, doubles the performance obtained by Azure Storage than Hercules in this special case.



(a) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for writes. We set up the experiment with 8 clients and 4 process per node, writing 128 MBytes each process (4 GBytes in total).



(b) Throughput of Hercules by using the Filecopy Benchmark for evaluating the scalability of I/O nodes for reads. We set up the experiment with 8 clients and 4 process per node, reading 128 MBytes each process (4 GBytes in total).

Figure 6: File copy benchmark configured for evaluating the Hercules I/O nodes scalability. 32 worker processes running on eight worker nodes (4 processes per node) access 4 Gigabyte of data. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in the best read cases.

On the other hand, on Figure 6(b), related with read operations, the peak performance of Hercules is even higher than the previous case, fully utilizing the ~480

MB/s of the available aggregated bandwidth at client-side and surpassing the peak throughput performance of Azure Storage accessed from D1 instances. When Azure Storage is accessed by D2 instances with more than one process reading in parallel from different files, the performance is almost doubled, in a similar way seen in the write operations.

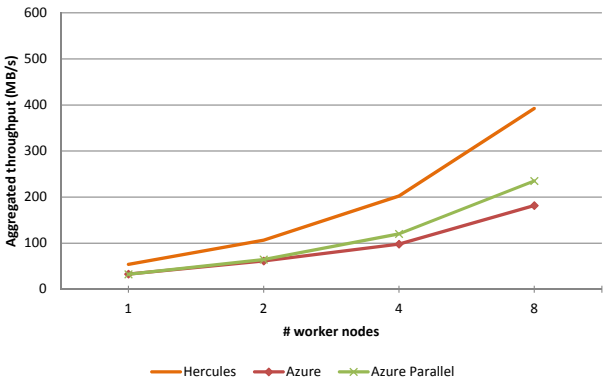
As conclusions of the last two cases, we can emphasize how the aggregated throughput of the workers accessing to the Hercules storage system approaches the theoretical maximum bandwidth available in every studied case, showing the scalability capabilities of our proposed solution. The performance in write operations is between 1.5x and 2x the performance achieved by Azure Storage with a similar architecture, while the performance in read operations in first case is marginally in favor of Azure and in the second case is comparable.

V.3 Worker nodes strong scalability

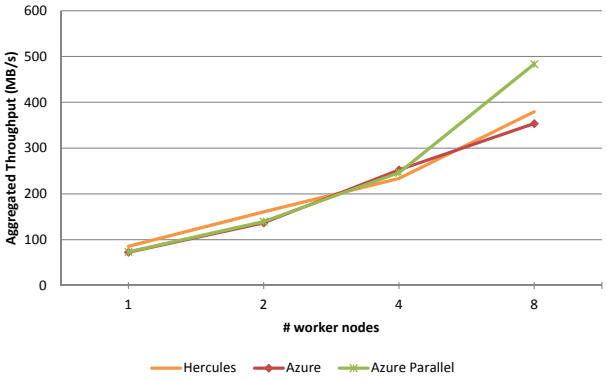
The last test cases focus on evaluating the behavior of our solution with an increasing number of worker nodes accessing the Hercules storage system. The objective is to evaluate the impact of the congestion against Azure Storage. The test cases are equivalent to the previous test cases, with Hercules using always 8 I/O nodes, while Azure Storage is evaluated using the native approach and the optimized parallel implementation. The aim of this test is to study a strong scalability scenario, where an increasing number of worker nodes perform the same total work: writing 8x512 MB files, a total problem size of 4096 MB, and then reading them. As expected, as the number of worker nodes increases, the total available bandwidth increases at the same pace, leading to better peak throughput performance, but the bottleneck continues at client-side.

Figure 7 shows the same trends already explained in the previous test cases. In Figure 7(a), which represent the aggregated throughput in write operations, can be seen how Hercules is always reaching the theoretical peak performance of each configuration, and how its performance is better than Azure Storage in every case, even doubling the performance in the most favorable one.

In the read operations performance case, Figure 7(b), again Hercules takes advantage of the available bandwidth in every case and competes really well with Azure Storage but the case of 8 clients where the Azure Parallel performance is better.



(a) Throughput varing the worker nodes from 1 to 8, writing 8 files (512 MBytes) per node. We set up the experiment with 8 I/O nodes in case of Hercules.



(b) Throughput varing the worker nodes from 1 to 8, reading 8 files (512 MBytes) per node. We set up the experiment with 8 I/O nodes in case of Hercules.

Figure 7: File copy benchmark configured for comparing Azure Storage and Hercules performance with an increasing number of worker nodes accessing to the storage concurrently. Hercules is configured with 8 I/O nodes and from 1 to 8 worker nodes access to the storage systems concurrently. Hercules performance is up to 2x better than Azure Storage in write operations while performing nearly as good as Azure Storage in most cases.

From the results of the evaluation, we can conclude that Hercules is capable of fully utilize the available bandwidth of every infrastructure where it has deployed. Furthermore, the scalability is assured in any case, on one hand when the number of I/O nodes deployed increases and, on the other hand, when the number of concurrent worker nodes scales and the congestion is higher. Compared to Azure Storage, our proposed solution is up to 2x better in performance during write operations and competes on equal conditions on read operations. Furthermore, should be noted that every test case evaluated in this work uses the best possible configuration for Azure Storage, as explained at the beginning of this section, and it could be predicted the same performance for Hercules in other scenarios while Azure Storage is expected to be penalized.

The potential of our proposed solution is clearly exposed in this preliminary benchmark evaluation. However, we are still working on test cases with a greater number of workers and I/O nodes to better show the scalability capabilities of the Hercules storage system deployed on a cloud infrastructure. Our final objective is to find the limitations in performance of Azure Storage and to evaluate how many number of Hercules I/O nodes are needed to achieve a comparable performance.

VI. Related work

The continued growth in popularity of *many-task computing* has caused many researchers to focus on research to improve the performance of storage systems, one of the major bottlenecks in this type of paradigms.

Previous solutions for providing in-memory storage are Parrot, Chirp, and AHPIOS. Parrot [7] is a tool to adapt existing systems using a remote I/O through the POSIX interface and Chirp [8]. Chirp is a user-level filesystem for collaboration across distributed platforms such as clusters, clouds, and grid computing systems.

AHPIOS (Ad-Hoc Parallel I/O system for MPI applications) [5] is a fully scalable system for I/O parallel MPI applications. AHPIOS relies on dynamic partitions and elastic demand partitions for distributed deployment applications. AHPIOS provides different memory caches levels. Hercules shares many of its features: (1) the user-level deployment without special privileges, transparency using a widely and easy deployment by using simple commands, (2) Hercules is designed to achieve high scalability and performance by leveraging many compute nodes as possible for I/O nodes, (3) Hercules uses main memory for temporal storage in order to improve performance in access. Costa et al. [1, 2] propose using the file attributes of MosaStore to provide communication between the workflow engine and file system by using hints. The workflow engine can provide these hints directly to the file system or file system can infer patterns by analyzing the data. The MosaStore approach is radically different from Hercules, because it uses a centralized metadata server rather than a focus on easy deployment and fully distributed as is our proposal. This server could became a bottleneck in large-scale systems.

The AMFS framework [9] offers programmers a simple scripting language for scripting execution of parallel applications in memory. Hercules shares with AMFS and treatment approach distributed metadata. A difference in AMFS must explicitly specify which data is to memory and what will be persistent while the goal is to be able to offer Hercules persistence transparently to the programmer.

HyCache+ [10] is a distributed storage middleware that allows effectively use the network bandwidth of the high-end massively parallel systems. HyCache + acts as main storage of recently accessed data (metadata, intermediate results for the analysis of large-scale data, etc.), and only exchange data asynchronously with the remote file system. One of the similarities between HyCache+ and Hercules is fully distributed metadata approach, the usega of computer network rather than the network shared storage, and high scalability. HyCache+ is totally based on POSIX while Hercules offers the possibility of using a POSIX interface and *get/set* operators. HyCache+ focuses on improving parallel file systems, while Hercules is designed to accelerate workflow execution engines, facilitating the exploitation of data locality in current cloud-based applications.

There are also studies that focus on the study of performance storage platforms in the cloud. Zhao et

al. [11] compares the I/O performance of S3FS, HDFS, and FusionFS [12]. As demonstrated in the experimental evaluation conducted in this paper, the performance obtained by Hercules equals or exceeds S3FS.

VII. Conclusions and future work

In this work we have presented the integration of the Hercules system and the Data Mining Cloud Framework in order to design and evaluate an ad-hoc storage system for temporary data produced inside data analysis workflow applications.

The evaluation results discussed in this paper clearly demonstrate the potential performance of Hercules, which is able to use more than 80% of the available bandwidth in every case and showing its scalability capabilities in every evaluated scenario. The performance achieved by Hercules is up to 2x the performance of Azure Storage in write operations while our proposed solution has been proved competitive in any scenario with read operations against the cloud storage service evaluated here.

Given the good results of this preliminary evaluation, our objective in the near future is to evaluate Hercules in more complex scenarios, with an increasing number of workers and I/O nodes, to better know the potential capabilities to work together with DMCF, and in addition to investigate the limitations of Azure Storage. Furthermore, it will be interesting to evaluate Hercules against Azure Storage in scenarios where Azure Storage is expected to have worse performance: changing the chunk size, changing the file size, changing the access patterns, etc.

After this first analysis of the capabilities of Hercules in complex cases, we will continue working in the integration of Hercules and DMCF, and in the evaluation of the price/performance ratio reached by Hercules in contrast with different cloud storage services. The final objective of our joint research is a fully working DMCF solution using Hercules as temporary storage for real data analysis applications.

Acknowledgement

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS). This work is partially supported by the grant TIN2013-41350-P, *Scalable Data Management Techniques for High-End Computing Systems* from the Spanish Ministry of Economy and Competitiveness.

References

[1] Samer Al-Kiswany, Abdullah Gharaibeh, and Matei Ripeanu. The case for a versatile storage system. *Operating Systems Review*, 44(1):10–14, 2010.

[2] L.B. Costa, H. Yang, E. Vairavanathan, A. Barros, K. Maheshwari, G. Fedak, D. Katz, M. Wilde, M. Ripeanu, and S. Al-Kiswany. The case for workflow-aware storage:an opportunity study. *Journal of Grid Computing*, pages 1–19, 2014.

[3] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. A hierarchical parallel storage system based on distributed memory for large scale systems. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 139–140, New York, NY, USA, 2013. ACM.

[4] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[5] Florin Isaila, Francisco Javier Garcia Blas, Jesús Carretero, Wei-Keng Liao, and Alok Choudhary. A Scalable Message Passing Interface Implementation of an Ad-Hoc Parallel I/O System. *Int. J. High Perform. Comput. Appl.*, 24(2):164–184, May 2010.

[6] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Js4cloud: script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency and Computation: Practice and Experience*, pages n/a–n/a, 2015.

[7] Douglas Thain and Miron Livny. Parrot: Transparent user-level middleware for data-intensive computing. *Scalable Computing: Practice and Experience*, 6(3), 2005.

[8] Douglas Thain, Christopher Moretti, and Jeffrey Hemmes. Chirp: a practical global filesystem

for cluster and grid computing. *Journal of Grid Computing*, 7(1):51–72, 2009.

[9] Zhao Zhang, Daniel S. Katz, Timothy G. Armstrong, Justin M. Wozniak, and Ian Foster. Parallelizing the execution of sequential scripts. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 31:1–31:12, New York, NY, USA, 2013. ACM.

[10] Dongfang Zhao, Kan Qiao, and Ioan Raicu. Hycache+: Towards scalable high-performance caching middleware for parallel file systems. In *IEEE/ACM CCGrid*, 2014.

[11] Dongfang Zhao, Xu Yang, Iman Sadooghi, Gabriele Garzoglio, Steven Timm, and Ioan Raicu. High-Performance Storage Support for Scientific Applications on the Cloud. *ScienceCloud'15*, June 2015.

[12] Dongfang Zhao, Zhao Zhang, Xiaobing Zhou, Tonglin Li, Ke Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu. FusionFS: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 61–70, Oct 2014.

Analyzing Power Consumption of I/O Operations in HPC Applications

Pablo Llopis, Manuel F. Dolz,
Javier García-Blas, Florin Isaila,
Jesús Carretero

University Carlos III, Spain
{pllopis,mdolz,fjblas,
florin,jcarrete}@arcos.inf.uc3m.es

Mohammad Reza Heidari,
Michael Kuhn

University of Hamburg, Germany
{heidari,kuhn}@informatik.uni-hamburg.de

Abstract

Data movement is becoming a key issue in terms of performance and energy consumption in high performance computing (HPC) systems, in general, and Exascale systems, in particular. A preliminary step to perform I/O optimization and face the Exascale challenges is to deepen our understanding of energy consumption across the I/O stacks. In this paper, we analyze the power draw of different I/O operations using a new fine-grained internal wattmeter while simultaneously collecting system metrics. Based on correlations between the recorded metrics and the instantaneous internal power consumption, our methodology identifies the significant metrics with respect to power consumption and decides which ones should contribute directly or in a derivative manner. This approach has the advantage of building I/O power models based on a previous set of identified utilization metrics. This technique will be validated using write operations on an Intel Xeon Nehalem server system, as writes exhibit interesting patterns and distinct power regimes.

Keywords HPC, I/O operations, power analysis, system metrics, statistical analysis.

I. Introduction

Modern scientific discoveries have been driven by an insatiable demand for high computing performance. However, as we progress on the road to Exascale systems, energy consumption becomes a primary obstacle in the design and maintenance of HPC facilities. A simple extrapolation shows that an Exascale platform based on the current most energy efficient hardware available in the Green500 [1] would consume 120 MW. The power wall being set to 20 MW [2], this system would still exceed this limit by a factor of five, thus turning it economically unfeasible due to its projected TCO. Indeed, systems will need to reach an energy efficiency of 50 GFLOPS/Watt to face the Exascale challenge. Actually, hardware vendors are already trying to provide more energy-efficient parts and software developers are gradually increasing power-awareness in the current software stack, from applications to operating systems. For example, recent advances in processor technologies have enabled operating systems to leverage new energy efficient mechanisms such as DVFS (Dynamic Voltage and Frequency Scaling) or DCT (Dynamic Concurrency Throttling) to limit power consumption of computing systems.

Data movement has been identified as an extremely important challenge among many others on the way towards the Exascale computing [2]. The low performance of the I/O operations especially in I/O-intensive scientific domains and simulations continues to present a formidable obstacle to reaching Exascale computing in the future large-scale systems. CPU speed and HDD capacity are boosted approximately by factors of 400 and 100 every 10 years, respectively. HDD speed, however, develops at a slower pace and can only be increased by a factor of 20 every 10 years. This issue

triggers a special interest in optimizing storage systems in data centers, and motivates the need for more research to improve the energy efficiency of storage technologies. Therefore, a first step to develop I/O optimizations is to further understand how energy is consumed in the whole I/O stack.

Due to the key role of power constraints, future Exascale systems are expected to work with a limited power budget, and be able to allocate power to different subsystems dynamically. In this scenario, the capability of predicting power consumption based on data movement and I/O operations is a useful resource. In this paper, we take advantage of a new internal wattmeter to deeply analyze the power drawn at every single wire leaving the PSU and feeding the hardware components of a server platform. While some existing works have focused on studying power consumption of system components such as storage devices, CPUs and memory [3], we offer a detailed view of the whole I/O stack power usage across all system components, from operating system mechanisms down to storage devices.

Given the foregoing, this paper makes the following contributions:

- Leveraging our power measurement and system metrics frameworks, we can benefit from data exploration and analysis to provide insights into the relation between power and data movement of I/O operations.
- We present a methodology that identifies key system metrics which are greatly correlated with power usage during data movement and I/O operations.
- Using our methodology, we conclude the most useful metrics in practical terms and narrow them down to a subset that can reflect the power consumption resulting from the data movement across the I/O stack.

The rest of this paper is structured as follows: In the second section, we present some related works about power analysis and modeling. In Section III, we detail our data acquisition framework, which consists of a power measurement and a system data collection framework as well as a detailed description of our new wattmeter. Section IV describes the proposed methodology for analyzing the data acquired from our software and hardware frameworks. Section V presents the results of applying our methodology and provides key insights into how the system consumes power when performing write operations. Finally, the conclusion section summarizes of our contributions and suggests some future works.

II. Related Work

Current approaches for analyzing power usage and estimating energy consumption fall into different categories: power modeling at the hardware level [4, 5], power modeling at the performance counters level [6], and power modeling at the simulation level [7, 8, 9].

Simulation techniques are commonly used for evaluating both performance and energy consumption. Prada et al. [8] describe a novel methodology that aims to build fast simulation models for storage devices. The method uses a workload as a starting point and produces a random variate generator that can be easily integrated into large-scale simulation models. A disk energy simulator, namely Dempsey [10], reads I/O traces and interprets both performance and power consumption of each I/O operation using the DiskSim simulator. Dempsey was only validated on mobile disk drives. This solution predicts energy consumption using the simulated disks characteristics instead of system metrics. Manousakis et al. [5] present FDIO, a feedback-driven controller that improves DVFS for I/O-intensive applications. This solution relies on the node being instrumented for obtaining fine-grained power measurement readings. Their feedback controller detects I/O phases, quickly switches the CPU frequency to all possible states and then selects the optimal setting regarding its power/performance ratio. El-Sayed et al. [11] demonstrated that energy-optimized adaptive policies result in higher quality energy/runtime tradeoffs than the static (constant) policies. While our work also describes the physical instrumentation to obtain fine-grained power readings, we use this instrument to analyze data movement patterns and detect power regimes. Our proposed model does not require having this kind of invasive instrumentation in order to predict energy consumption. Lewis et al. [6] uses

the node temperature to predict energy consumption. The authors discuss the interaction of the different components for their modeling. The authors propose using read and writes per second metric (obtained by iostat) for modeling I/O workloads. Allalouf et al. [4] develop a scalable power modeling method that estimates the power consumption of storage workloads (STAMP). The modeling concept is based on identifying the major workload contributors to the power consumed by the disk arrays. Deng et al. [12] model the flash memory based storage systems constructed as a RAID by leveraging the semantic I/O. In a contribution similar to ours, the authors calculate the cost and energy consumption of storage devices for running a variety of workloads, categorized by their dominant requirements [13]. In contrast, our solution predicts the energy consumed by the complete I/O stack (including CPU and memory consumption) for single/multi core access patterns, in addition to the storage devices. For estimating energy consumption, some works also focus on system performance counters [14, 15, 16]. These works propose linear models that are able to provide run-time power estimations, and are validated with instrumented hardware. Other works concentrate on reducing energy consumption of individual storage devices. Zhu et al. [17] optimize disk energy consumption by tuning cache-replacement strategies that increase idle time and reduce disk spin-ups.

Our work focuses on exploring, analyzing, and modeling the power consumption of the operating system's I/O stack across all components. Like Li et al. [18], we aim to build models that help better understand and reduce the energy consumption of the storage stack. Unlike most works mentioned in this section, we do not provide a generic power model for computation, or limit our analysis to a single system component, but focus on energy consumption caused by data movement patterns across the memory hierarchy and I/O stack.

III. Data Acquisition Framework

In this section, we describe the power performance measurement framework that we use to instrument our platform to perform the I/O analysis in detail. Specifically, we leverage our `pyprocstat` tool for tracing `procfs` and `sysfs` system metrics and the PMLib framework to support both external and internal wattmeter devices. However, we will focus on the data obtained from the internal wattmeter, as this device provides more fine-grained measurements and is able to detect rapid power variations. We use the external wattmeter only for validation purposes. Figure 1 depicts the ontology of our framework, divided in both system and power measurement categories with their corresponding tools.

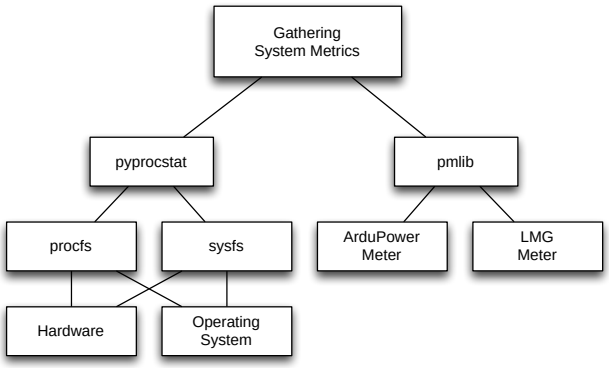


Figure 1: Ontology of system metrics used in our measurement framework.

III.1 System metrics collection framework

We instrument our platform to gather live system metrics in order to analyze workload behaviours and detect the correlation of the system activities with power consumption. These data traces can be easily correlated with power consumption traces for further exploration and analysis, as demonstrated in Section V.

While common UNIX tools such as `top` and `iostat` are able to gather live system information, they are not well-suited for our purposes. Our goal is to obtain traces that are aggregated into a time series consisting of different system metrics. Scripting existing tools in order to generate the resulting data traces is not very practical, since this method ends up launching processes several times and causing unnecessary overheads. Similarly, data stemming from performance counters gathered with the Linux `perf` framework are limited because their main objective is timing function

calls and counting function calls, with limited support for inspecting runtime values. Hence, they are better suited to other tasks such as profiling and performance debugging.

Instead, we develop `pyprocstat` [19], an easy to use, low-overhead, modular and flexible tool specifically built for our purposes. This tool consists of different modules, each of which is in charge of gathering information from different parts of the system. These modules usually collect information directly from kernel-provided interfaces such as procfs and sysfs, providing system data with a low overhead. In this paper, we collect system information from I/O devices, virtual memory, interrupts and per-CPU utilization, adding up to a total of 120 different system metrics.

III.2 Power measurement framework

To measure power consumption, we leverage the PMLib framework, a well-established package for investigating power usage in HPC applications [20]. Its implementation provides a general interface to utilize a wide range of wattmeters, including *i*) external devices, such as commercial PDUs, WattsUp? Pro .Net, ZES Zimmer Lmg450, etc., *ii*) internal wattmeters, directly attached to the power lines leaving from PSU, such ArduPower, *iii*) commercial DAS from National Instruments (NI) and *iv*) integrated power measurement interfaces such as Intel RAPL, NVIDIA NVML, IPMI, etc. The PMLib client side provides a C library with a set of routines in order to measure the application code. The traces obtained can be easily integrated into existing profiling and tracing frameworks such as Extrae+Paraver [21] or VampirTrace+Vampir [22].

III.2.1 ArduPower as a low-cost internal wattmeter

In this section, we describe our new internal ArduPower wattmeter in detail. ArduPower has been conceived as a low-cost DAS to measure the instantaneous DC power consumption of the internal components in computing systems wherever the PSU output power lines are accessible [23]. In general, it offers a spatially fine-grained power measurement by providing 16 channels to monitor the power consumption of several parts, e.g., mainboard, HDD and GPU cards etc., simultaneously, with a sampling rate varying from

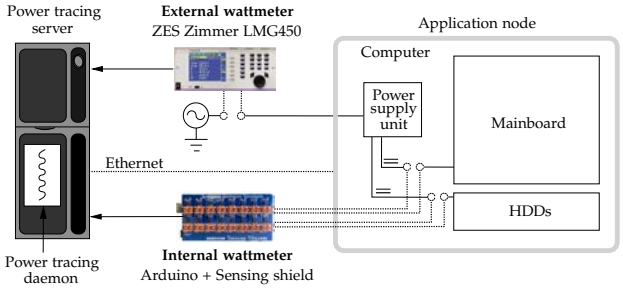


Figure 2: Power measurement setup combining both internal ArduPower *and external* Lmg450 *wattmeters.*

480 to 5,880 Sa/s, depending on the number of selected channels. The total production cost of the wattmeter is approximately 100 €, so it can be easily accommodated in moderate-/large-scale HPC platforms in order to investigate power consumption of scientific applications. As shown in Figure 3, ArduPower wattmeter consists of two basic hardware components: a shield of 16 current sensors and an Arduino Mega 2560 processing board detailed below.

Sensing shield ArduPower comprises a self-designed shield, responsible for sensing the DC currents passing through the wattmeter and providing the outputs to the processor board of the Arduino. It consists of 16 Allegro ACS713 current sensors [24], each of them converting the DC current passing through it to a proportional voltage. The Hall-effect elements provide a highly accurate, low noise output voltage signal proportional to the applied DC current, sensing up to 20 A with a total output error of $\pm 1.5\%$ and a low internal resistance of 1.2 mΩ.

Microcontroller ArduPower also comprises an Arduino Mega 2560 processing board that is connected directly to the power sensing shield. It benefits from one Atmel ATmega2560 [25] as a high-performance low-power 8-bit AVR RISC-based microcontroller running at 16 MHz and combining 256 KiB ISP flash memory, 8 KiB SRAM and 4 KiB EEPROM. The complete board of Arduino Mega 2560 has 16 analog inputs supported by a 10-bit ADC, 4 UARTs (hardware serial ports) and a USB link.

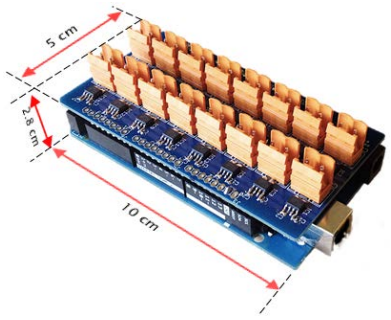


Figure 3: ArduPower *wattmeter, power sensing shield and Arduino Mega 2560 processing board.*

The sensing shield is placed on the top of Arduino Mega 2560 to let an analog to digital converter (ADC) read the outputs of the 16 current sensors and transform them into digital values. The Arduino board communicates with the target tracing server through a serial USB link and sends the measured DC current values to a PMLib server in order to calculate the instantaneous power consumption.

IV. Data Exploration and Methodology

Our goal is to explore the data to understand how the electrical power is consumed in a computing system to perform I/O operations. The analysis should reveal the system metrics which are correlated with the specific I/O operations and are useful for modeling the power usage related to data movement and I/O operations. In order to measure data movement, we perform simple sequential write I/O operations while collecting data as detailed in section III. To carry out this micro benchmark, we leverage `fio` [26], which is a commonly used micro benchmarking tool developed by the lead developer and maintainer of the Linux block IO subsystem.

In spite of the apparent simplicity of these operations, write I/O operations exhibit interesting irregular patterns due to the way the operating system manages data while it moves across the I/O stack, as depicted in Figure 4. However, read I/O operations do not exhibit these patterns and, therefore, this is why this work entirely focuses on write I/O operations.

We identify different power and performance

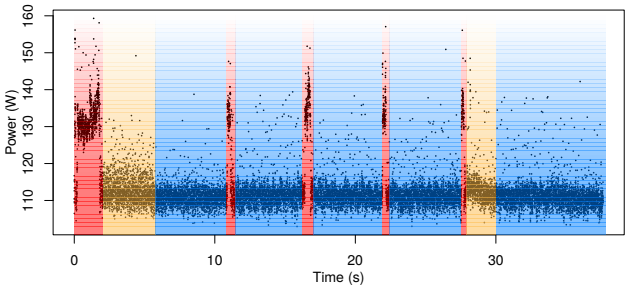


Figure 4: Power regimes during a sequential write of a 4 GiB file.

regimes that correspond to temporal regions in these I/O operations. Figure 4 shows that for a simple write I/O operation, the power consumption can vary significantly. This is due to the fact that the system transitions between different power and performance regimes while data moves from main memory and is written to disk. This shows that a simple straw-man approach to modeling writes using average power and write duration as input would not be sufficient, especially for short write operations. This also motivates the need to have a way for estimating the power consumption of I/O operations.

Using the data collected as a time series, we design a methodology in order to detect highly correlated metrics with regard to power consumption, following a similar approach as in [27]. Identifying these metrics is important for developing new power usage models that are more sophisticated than the aforementioned straw-man approach. This methodology leverages the Pearson's correlation and consists of the following steps:

1. For each collected system metric, we calculate its correlation with power consumption.
2. For each collected system metric, we compute the derivative and calculate its correlation with power consumption.
3. Every correlation whose absolute value is below than an empirically determined threshold t is discarded.
4. The union from both correlations results in a table of system metrics that are relevant for power usage

during data movement of I/O operations.

Note that in the design of the methodology, we have taken several aspects into account. First, some metrics are cumulative values, and therefore monotonically increasing with time, e.g, number of interruptions occurred since boot. Others are defined as a rate or instantaneous value and might vary with time accordingly, e.g., power consumption varies depending on the load of the machine. Therefore, a methodology should be aware of this fact in order to avoid correlations of metrics of different nature, and convert cumulative time series to instantaneous values so all metrics can be compared. The transformation from accumulated to instantaneous values is normally performed by subtracting the previous observation r_{t-1} from the current one r_t at the time t.

Second, since data movement has a direct impact on the power consumption, metrics that are measured in quantities of data should be transformed into the rates of movement by computing their corresponding derivatives. For example, as shown in Figure 5, the dirty memory metric measures the number of bytes in memory that must be written back to the disk at a given time[1]. However, the page dirtying rate (or speed) needs to be derived from the dirty memory metric in order to be compared with the power consumption properly. By calculating the dirtying rate, it is possible to measure the amount of data that any user-space application is writing to main memory. Information closely related to data movement can be obtained from the available system metrics. We believe this is a fundamental step when developing a methodology that identifies most correlated metrics with regard to the power usage.

V. Analysis of System I/O Operations

In this section, we perform an analysis of I/O operations using our measurement framework and the proposed methodology described in Section IV for sequential I/O writes. First, we describe the hardware setup and the software configuration that have been used for this work in detail.

[1]Note that the absolute value of the derivative is computed in order to superimpose positive and negative rates on a single normalized plot.

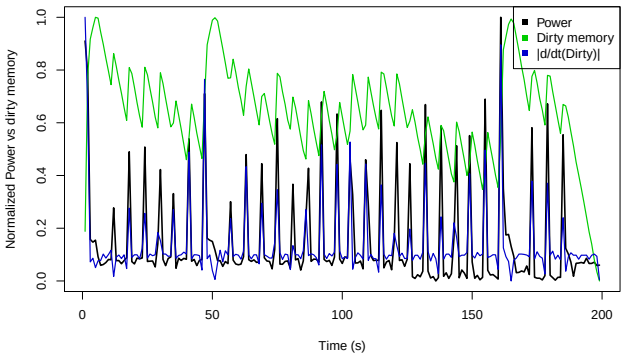


Figure 5: Dirty memory, $|d/dt(\text{dirty memory})|$ and power consumption of a sequential write of a 20 GiB file.

V.1 Configuration setup

Target platform. The analysis and evaluation have been carried out on a server platform, denoted as Nehalem, equipped with 2× Intel Xeon X5560 (total of 8 physical cores) running at 2.80 GHz, 12 GB of RAM memory, and a Seagate Barracuda 500 GB S-ATA II HDD, equipped with a Supermicro PSU 720W 80+ Gold (≈82% energy efficiency).

Software instrumentation framework. We use `pyprocstat` [19] to gather system metrics and obtain time series that describes live system information. We configure this tool to use the following built-in modules: `meminfo` (collects data from `/proc/meminfo`), `stat` (collects CPU utilization data, interrupts, context switches, etc from `/proc/stat`), `vmstat` (collects virtual memory data from `/proc/vmstat`), and `io` (collects I/O data from `sysfs`).

Power measurement framework. As mentioned before, we use the PMLib software package to investigate power usage of HPC applications. Power measurement can be controlled by the applications using a collection of routines that allow the user to query information on the power measurement units, create counters associated with a device where power data is stored, start, continue and terminate power sampling, etc. All this

information is managed by the PMLib tracing server, which is in charge of acquiring data from the devices and sending back the appropriate answers to the invoking client application via the proper PMLib routines (see Figure 2).

Wattmeters. We use the ArduPower wattmeter connected to all the power lines leaving from the PSU and feeding the different components of the server machine. We leverage 16 channels of ArduPower to measure the 3.3 V, 5 V and 12 V lines from a 24 ATX motherboard connector, 2× 4-pin connectors, and a 4-pin Molex connector. We avoid ground and negative voltage lines. On the other hand, we also employ an external ZES Zimmer Lmg450 wattmeter measuring Nehalem in order to verify that the internal measurements are well correlated with the external ones. This wattmeter can measure up to 20 Sa/s.

V.2 Analysis of write operations

Applying our proposed methodology, we obtain one power usage time series and 120 system metric time series for every benchmark run. Figure 6 depicts the correlations of all the 120 system metrics with power consumption during a sequential write of a 4 GiB file on Nehalem. The top plot shows the direct correlations, while the bottom plot takes the derivative of the data before computing the correlation with power. Indeed, the bottom plot clearly shows that only one system metric is highly correlated with power (B4) and the rest have a very low correlation. Table 1 lists the most significant system metrics. As it is shown in the table, those metrics which have values below our empirically obtained threshold of 0.75 have been discarded.

Where `cpu_system` is the system CPU utilization, `Dirty` is the number of dirty memory pages, `softirq` is the number of Linux software IRQs, `procs_running` is the number of running processes, and `cpu_user` is the user mode CPU utilization. Not surprisingly, CPU utilization is highly correlated with power usage since the CPU is the most power intensive component during these operations. Interrupts are also highly correlated with the I/O power usage. However, we argue that the most relevant system metric for write operations

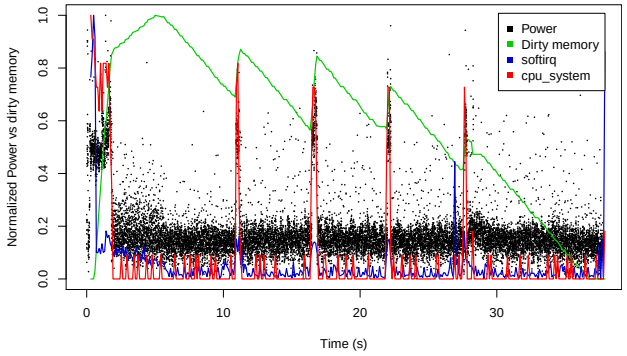


Figure 7: Power consumption vs. dirty memory for the sequential write of a 4 GiB file.

is the derivative of the number of dirty pages. While the system CPU utilization shows a slightly higher correlation, we can hardly use this metric to reflect the level of power consumption used by I/O operations since the system CPU utilization correlates with the power consumption of other workloads running on the machine and, in consequence, this metric is not useful for decoupling I/O from other workloads. Similarly, the number of running processes cannot be considered a good metric due to its nature. Therefore, we conclude that the page dirtying rate, d/dt(`Dirty`), is the best system metric to reflect the I/O-related power usage. Figure 7 clearly shows how well the `Dirty` system metric describes data movement with regard to power consumption.

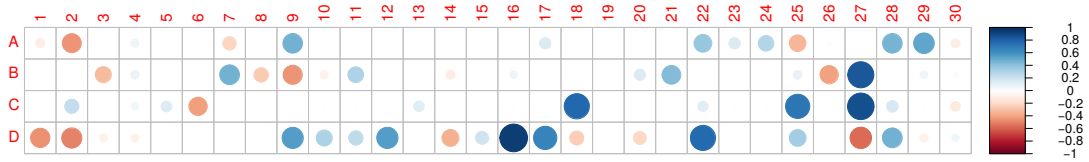
VI. Conclusions

In this paper, we leverage a power and system tracing framework in order to deeply analyze power usage due to data movement across the I/O stack. Among power consumption measurements collected in this framework, we also gather system metrics obtained from the `procfs` and `sysfs` file systems. Next, we present a new methodology to determine which system metrics are highly correlated to power consumption. We validate this technique by performing write operations on an Intel Xeon Nehalem server system instrumented with ArduPower and Lmg450, a fine-grain internal DC wattmeter and one external AC wattmeter, respectively.

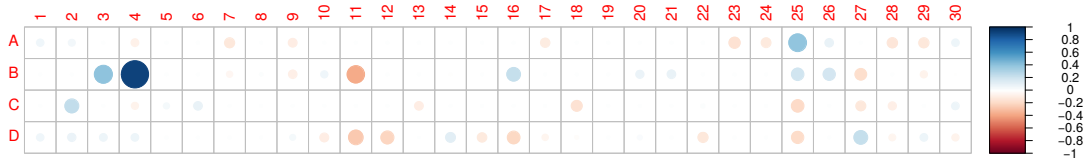
Several aspects are taken into account in designing

Table 1: Correlation of system metrics to power for a sequential write.

Metric	Corr(data)	Corr(d/dt(data))	Corrplot Tile
`cpu_system`	0.94	-0.20	D16
`Dirty`	0.08	0.92	B4
`softirq`	0.87	-0.12	C27
`procs_running`	0.84	-0.17	B27
`cpu_user`	0.77	-0.12	D22



(a) Raw system metrics.



(b) Derivative system metrics w.r.t. time.

Figure 6: Correlation plot between power usage and system metrics for a 4 GiB write.

the methodology. First, some of the metrics measured are reported as accumulated values, e.g., number of interruptions occurred since last read of a counter or the number of dirty pages at a given time. Due to their nature, some metrics do not appear to have direct impact on the power consumption; however, if their derivatives are computed, their instantaneous rates of change can correlate better with power consumption. In other words, the presented methodology can determine if a metric is highly correlated to power in the direct or derivative mode.

The analysis results demonstrate that only a small portion of the metrics, such as the CPU utilization, the rate of change in the number of dirty pages, software interruptions and number of processes running, have direct impacts on the power consumption and, due to their strong correlation, they can eventually be incorporated into I/O power models.

For future works, we plan to refine our methodology in order to analyze the correlations at the PSU wire levels, as obtained from the internal ArduPower wattmeter, and extend our benchmark suite to comprise more I/O operations with different configurations. We also aim to automatically build I/O power models using the methodology presented in this paper.

Acknowledgements

The work presented in this paper has been partially supported by the EU Project FP7 318793 "EXA2GREEN" and partially supported by the EU under the COST Programme Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)" and by the grant TIN2013-41350-P, *Scalable Data Management Techniques for High-End Computing Systems* from the Spanish Ministry of Economy and Competitiveness.

References

[1] The Green500 Editors. Green500. `http://www.green500.org/`, 6 2015. Last accessed: 2015-8.

[2] US Department of Energy. Top Ten Exascale Research Challenges. Technical report, Department of Computer Science, Michigan State University, February 2014. `http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf`.

[3] Anne-Cecile Orgerie, Marcos Dias de Assuncao, and Laurent Lefevre. A survey on techniques for improving the energy efficiency of large-scale distributed systems. *ACM Comput. Surv.*, 46(4):47:1–47:31, March 2014.

[4] Miriam Allalouf, Yuriy Arbitman, Michael Factor, Ronen I. Kat, Kalman Meth, and Dalit Naor. Storage modeling for power estimation. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 3:1–3:10, New York, NY, USA, 2009. ACM.

[5] Ioannis Manousakis, Manolis Marazakis, and Angelos Bilas. Fdio: A feedback driven controller for minimizing energy in i/o-intensive applications. In *Proceedings of the 5th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'13, pages 16–16, Berkeley, CA, USA, 2013. USENIX Association.

[6] Adam Lewis, Soumik Ghosh, and N.-F. Tzeng. Run-time energy consumption estimation based on workload in server systems. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower'08, pages 4–4, Berkeley, CA, USA, 2008. USENIX Association.

[7] Guangyu Sun, Yongsoo Joo, Yibo Chen, Dimin Niu, Yuan Xie, Yiran Chen, and Hai Li. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, Jan 2010.

[8] Laura Prada, Javier Garcia, Alejandro Calderon, J. Daniel Garcia, and Jesus Carretero. A novel black-box simulation model methodology for predicting performance and energy consumption in commodity storage devices. *Simulation Modelling Practice and Theory*, 34(0):48 – 63, 2013.

[9] Timo Minartz, JulianM. Kunkel, and Thomas Ludwig. Simulation of power consumption of energy efficient cluster hardware. volume 25, pages 165–175. Springer-Verlag, 2010.

[10] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies*, FAST '03, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[11] Nosayba El-Sayed and Bianca Schroeder. To checkpoint or not to checkpoint: Understanding energy-performance-i/o tradeoffs in hpc checkpointing. In *IEEE International Conference on Cluster Computing (CLUSTER)*, pages 93–102. IEEE, 2014.

[12] Yuhui Deng, Lijuan Lu, Qiang Zou, Shuqiang Huang, and Jipeng Zhou. Modeling the aging process of flash storage by leveraging semantic i/o. *Future Generation Computer Systems*, 32(0):338 – 344, 2014.

[13] Yan Li and D.D.E. Long. Which storage device is the greenest? modeling the energy cost of i/o workloads. In *IEEE 22nd International Symposium on Modelling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 100–105, Sept 2014.

[14] Gilberto Contreras and Margaret Martonosi. Power prediction for intel XScale® processors using performance monitoring unit events. In *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, pages 221–226. IEEE, 2005.

[15] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system

power analysis and modeling for server environments. International Symposium on Computer Architecture-IEEE, 2006.

[16] Tao Li and Lizy Kurian John. Run-time modeling and estimation of operating system power consumption. In *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '03, pages 160–171, New York, NY, USA, 2003. ACM.

[17] Qingbo Zhu, F.M. David, C.F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management. In *In Proceedings of IEE Software*, pages 118–118, Feb 2004.

[18] Jing Li, Anirudh Badam, Ranveer Chandra, Steven Swanson, Bruce Worthington, and Qi Zhang. On the energy overhead of mobile storage systems. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST'14, pages 105–118, Berkeley, CA, USA, 2014. USENIX Association.

[19] Pablo Llopis. A powerful and modular tool for gathering live system information as time series. `https://github.com/pllopis/pyprocstat`.

[20] S. Barrachina, M. Barreda, S. Catalán, M.F. Dolz, G. Fabregat, R. Mayo, and E.S. Quintana-Ortí. An integrated framework for power-performance analysis of parallel scientific workloads. In *ENERGY 2013, The 3rd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 114–119, 2013.

[21] Paraver: the flexible analysis tool. `http://www.cepba.upc.es/paraver`. [Last access: June 2015].

[22] The vampir performance analysis tool-set. `https://www.vampir.eu/`. [Last access: June 2015].

[23] Manuel F. Dolz, Mohammad Reza Heidari, Michael Kuhn, and Germán Fabregat. ArduPower: a low-cost wattmeter to improve energy efficiency of HPC applications. In *5th International Green & Sustainable Computing Conference*, Las Vegas, NV, USA, December 2015. To appear.

[24] LLC Allegro MicroSystems. ACS713: Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Voltage Isolation and a Low-Resistance Current Conductor. `http://www.allegromicro.com`, 2015. [Last access: June 2015].

[25] Atmel Corporation. ATmega2560: 8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash. `http://www.atmel.com/devices/atmega2560.aspx`, 2015. [Last access: June 2015].

[26] Jens Axboe. Flexible i/o tester. `http://freecode.com/projects/fio`.

[27] M. F. Dolz, J. Kunkel, K. Chasapis, and S. Catalán. An analytical methodology to derive power models based on hardware and software metrics. *Computer Science - Research and Development*, 2015.

FriendComputing: Organic application centric distributed computing

Beat Wolf, Loïc Monney, Pierre Kuonen

University of Applied Sciences Western Switzerland, HES-SO//Fribourg
beat.wolf@hefr.ch

Abstract

Building Ultrascale computer systems is a hard problem, not yet solved and fully explored. Combining the computing resources of multiple organizations, often in different administrative domains with heterogeneous hardware and diverse demands on the system, requires new tools and frameworks to be put in place. During previous work we developed POP-Java, a Java programming language extension that allows to easily develop distributed applications in a heterogeneous environment. We now present an extension to the POP-Java language, that allows to create application centered networks in which any member can benefit from the computing power and storage capacity of its members. An accounting system is integrated, allowing the different members of the network to bill the usage of their resources to the other members, if so desired. The system is expanded through a similar process as seen in social networks, making it possible to use the resources of friend and friends of friends. Parts of the proposed system has been implemented as a prototype inside the POP-Java programming language.

Keywords Java, distributed computing

I. Introduction

Scientific and commercial applications face an increase of computational resource requirements. This can be observed in various domains, such as medical research applications, material simulations, weather forecasts or multimedia processing. In all those domains the data to be analysed is produced at increasing speeds, with data processing applications not being able to keep up with the analysis. In the past, mainly large organizations have addressed these types of analysis as they had access to large computing infrastructures. The constant improvements of computers as well as the reduction of their prices, has attracted the interest of small organizations to tackle this type of calculations. Nevertheless, the constantly increasing amounts of data produced as well as the complexity of the types of analyses to perform still restricts many small organizations to really address this domain.

Several technologies emerged over the past years to assist these organizations to cope with the demands of modern applications. Especially, technologies like cloud computing enabled these organizations to expand their computing infrastructure for cases where it is not sufficient. While for many organizations this is an acceptable solution, it is not a solution for all use cases. This is especially true for organizations working with sensitive data, for example medical data, that may not be allowed to be sent to any remote location. This problem is even more emphasized when the cloud provider is located in a different jurisdiction, a likely scenario in regards to currently popular cloud providers like Amazon.

Various commercial and academic computing grids have been created in the last decades. They regroup the infrastructures of several organizations into a single grid, available for use to all members. Grids such as the Open Science Grid (OSG) [1] or the Worldwide LHC Computing Grid (WLCG) [2] come in the form of traditional grids which can be used for a multitude of applications by various users. Other grids, like Folding@Home [3] and the BOINC [4] based grids are

application specific, generally used by a single user providing the analysis tasks.

The main drawback of the approach of the grid is due to the fact that the code must be adapted to different operating systems and hardware present in the grids. This leads to costly IT developments that are often beyond the capacity of these small organizations. This problem has been largely reduced in the clouds thanks to the virtualization.

In addition, the setup and maintenance of a grid environment is a complicated task and many organizations do not have the necessary technical knowledge to do so. A use case which is quite common is that multiple partners need a significant amount of computing power to perform the same type of analysis. In such a situation the partners often use the same software to perform their analysis and could benefit from using each other's infrastructure to do so. It is this particular use case that we address in this poster.

II. Friend computing

The concept of friend computing is to create an application centered network of so called "friends", which share the same goal. Multiple users of a certain software which performs computationally complex analyses can group together and benefit from each other's infrastructure. The group is expanded through a process similar to how social networks work. Any new member gets invited by an existing member and once part of the network can access the computing power of every other installation. This approach is similar to the friend-to-friend computing as presented in [5], with the main difference that in [5], authors focus on data sharing only. In [6] the authors have shown that friend-to-friend computing can also been applied for sharing computing resources, an idea on which we expand upon.

We based our first prototype implementation of friend computing on POP-Java [9], an extension of the Java programming language which implements the POP (Parallel Object Programming) model [7]. The POP programming model was initially implemented as an extension of the C++ language, called POP-C++ [8]. POP-Java was chosen as it offers an excellent base for the concept of friend computing.

The POP-Java language has as one of its main features the possibility to create objects on remote computers, making it possible for the programmers to combine local and remote objects. By default POP-Java will either use any available computer in a locally configured POP-Java network, or a specific computer defined by the programmer. The introduction of Friend computing allows the programmer to search for a computing resource in the friend network automatically, with the ability to specify certain criteria such as processing power or storage space. This can also be used to bring calculations to the place where the data is stored, which can reduce privacy concerns in cases where the data itself is sensitive.

The concept of friend computing was also approached from a commercial viewpoint, giving the users of a network an incentive to make their resources available. Because of this an accounting system was integrated, in which every member of the network logs the usage of their resources by other members. This makes it possible to bill the different users of the network based on their usage, increasing the incentive to participate in the network as well as giving the possibility to monetize their infrastructure.

III. Prototype

We created a prototype of the concept of friend computing using POP-Java. It consists of an extended version of the POP-Java language, as well as an example application using those features. This prototype application was used to verify the correct implementation of the POP-Java implementation which includes the friend computing extension.

The prototype allows the creation of new friend networks, allowing to define an ID and purpose of the friend computing network. This newly created network can then be extended by inviting new members to the network. To get invited, the joining party needs to provide an IP address to a member of the network and have the prototype application running. The prototype application will show a notification of the network invitation, and upon accepting sets up the required friend computing network configurations. Once a member of the network, any member can launch a simple calculation, in this case the factorization of a

number. When the calculation is launched, available resources inside the friend computing network are automatically discovered and the calculations distributed over multiple computers.

Any usage of the resources is journaled and can later be used to bill the individual members of the network (if so wished).

IV. Conclusion

We showed a concept on how to approach distributed software development in an Ultrascale world. The ability to dynamically grow a distributed computing network based around a specific application with the possibility to bill other members based on their usage is an interesting approach to use ultrascale systems. Our current prototype has only been verified on a small scale network and further tests will show how it scales to larger numbers. Further work will be required to scale the current concept to Ultrascale systems, but the current concept already allows for the design of very large distributed applications. This is especially true if like in every object orientated application, the scope of every object is limited as much as possible. This reduces not only the complexity of the application, but also the complexity of the network traffic between the different distributed objects.

Other open works include the improvement of the resource discovery protocol. While the currently used resource discovery protocol, which is the one used by POP-Java, works well when searching for resources with a certain amount of RAM or CPU power, in the context of friend computing it would be helpful to be able to search for computing nodes which have certain data stored locally. This would allow bringing the computations to the data instead of the other way around, greatly reducing data confidentially issues that can arise in distributed systems.

The prototype and POP-Java in general also does not yet handle firewalls and systems behind a NAT. Including support for those would greatly help adoption on a larger scale including commercial applications like GensearchNGS [10], an genetic diagnostics application which served as the initial inspiration for this project.

References

[1] R. Pordes, *et al.. The open science grid.* Journal of Physics: Conference Series. Vol. 78. No. 1. IOP Publishing, 2007

[2] D. Bonacorsi and T. Ferrari, *WLCG Service Challenges and Tiered architecture in the LHC era.*, IFAE 2006. Springer Milan, 2007. 365-368.

[3] S. M.Larson, *et al.*, *Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology.* 2002

[4] D.P. Anderson, *Boinc: A system for public-resource computing and storage*, Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing. IEEE, 2004

[5] B. Popescu, *et al.*, *Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System*, Security Protocols, 2006

[6] U.Norbisrath, *et al.*, *Friend-to-friend computing instant messaging based spontaneous desktop grid*, Proceedings - 3rd International Conference on Internet and Web Applications and Services, ICIW 2008

[7] T. A.Nguyen and P. Kuonen, *A model of dynamic parallel objects for metacomputing*, The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications, 2002.

[8] C. D.Jiogo *et al.*, *Parallel Object Programming in POP-C++: A Case Study for Sparse Matrix-vector Multiplication*, Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP06), France, 2006

[9] B. Wolf *et al.*, *POP-Java : Parallélisme et distribution orienté objet*, ComPAS 2014 : confÃl'rence en parallélisme, architecture et systèmes, 2014

[10] B. Wolf *et al.*, *DNAseq Workflow in a Diagnostic Context and an Example of a User Friendly Implementation*, BioMed Research International, vol. 2015

Multilevel parallelism in sequence alignment using a streaming approach

Beat Wolf[1,2], Pierre Kuonen[1], Thomas Dandekar[2]

[1] University of Applied Sciences Western Switzerland, HES-SO//Fribourg
[2]University of Würzburg//Germany
beat.wolf@hefr.ch

Abstract

Ultrascale computing and bioinformatics are two rapidly growing fields with a big impact right now and even more so in the future. The introduction of next generation sequencing pushes current bioinformatics tools and workflows to their limits in terms of performance. This forces the tools to become increasingly performant to keep up with the growing speed at which sequencing data is created. Ultrascale computing can greatly benefit bioinformatics in the challenges it faces today, especially in terms of scalability, data management and reliability. But before this is possible, the algorithms and software used in the field of bioinformatics need to be prepared to be used in a heterogeneous distributed environment. For this paper we choose to look at sequence alignment, which has been an active topic of research to speed up next generation sequence analysis, as it is ideally suited for parallel processing. We present a multilevel stream based parallel architecture to transparently distribute sequence alignment over multiple cores of the same machine, multiple machines and cloud resources. The same concepts are used to achieve multithreaded and distributed parallelism, making the architecture simple to extend and adapt to new situations. A prototype of the architecture has been implemented using an existing commercial sequence aligner. We demonstrate the flexibility of the implementation by running it on different configurations, combining local and cloud computing resources.

Keywords Ultrascale systems, NESUS, Template

I. Introduction

In the field of bioinformatics the advance of next generation sequencing (NGS) technologies increased the amount of data produced at a very high speed. They produce the sequencing data at a higher speed, with longer sequences that have increasingly better quality. The amount and speed at which the data is produced increased much faster than the capacities of computers evolved during the same time. This makes it challenging for sequence laboratories to analyse the produced data in a reasonable amount of time. While in the beginning of DNA sequencing the produced data could still be analysed by hand, the amounts of data produced today for one sample can range from 10 giga base pairs in whole exome sequencing to hundreds in the case of whole genome sequencing. Powerful computing infrastructures are needed to analyse this type of data.

Ultrascale computing presents itself as a possible solution to many of the issues faced in bioinformatics. But to benefit from the possibilities of ultrascale computing, many algorithms and tools used need to be adapted to work in such a distributed environment. While much effort is spent to distribute and parallelize various tools, they are often tied to a specific environment, be it a grid or a cloud. In this paper we look at the issue of sequence alignment and create a generic way to distribute the workload over a heterogeneous distributed system. This work can be used as the basis of future work to bring bioinformatics data processing to ultrascale systems.

Sequence alignment is an active topic of research, with various approaches taken to solve the issue. The basic idea of sequence alignment is to find the best position of a sequenced read on an already known reference genome. The complexity of the task comes from the size of the reference genome, the amount of sequences to be aligned and the fact that the sequenced genome does not perfectly match the reference genome. As an example, the he human reference genome is 3 Giga base pairs long. Millions of sequences with a typical length between 100 and 1000 base pairs are then aligned against this reference. The differences between those sequences and the reference can range from simple single nucleotide mismatches to complicated insertions or deletions.

Various approaches exist to the problem of sequence alignment, an overview of the different techniques can be found at [1]. While the different alignment tools approach the issue from different angles, they do all have in common that they find the optimal placement of a single sequence on the reference genome. They do so by looking only at one sequence at a time, without any expectations of the order at which the sequences are aligned. This makes sequence alignment an ideal candidate for parallel processing. Every sequenced read can be processed independently and in any order. The order in which they are input for the alignment and their output order also does not matter. In fact most alignment algorithms make use of multi-core systems through multithreading to speed up the alignment. Some, like the recently released tool nvBowtie, even use GPUs to make use of the thousands of cores of a GPU to accelerate sequence alignment. The use of parallel processing does not stop here, but also extends to distributed systems. Examples of such tools are Crossbow [3], Cloudburst [4], ScalaBLAST [5] or custom frameworks like the one presented in [6].

In this paper we explore a possible architecture which uses parallelism on multiple levels. It is used to distribute the sequence alignment over multiple cores, multiple computers in the same network and cloud resources. While several of the previously mentioned tools support some of those approaches, none combine all of them at once. A stream based approach is used to distribute the workload over all the different systems. The way a remote computer and a local thread are integrated into the alignment process is very similar in this architecture. A prototype of the system has been implemented in a commercial NGS data analysis software, GensearchNGS [9]. The proposed architecture is verified by measuring its scalability over multiple heterogeneous computing nodes and the flexibility of the design is discussed. While the sequence alignment algorithm distributed in this example is a custom one used by GensearchNGS, the architecture and design of the distribution is not restricted to that algorithm.

II. Methods

The implementation of the distributed sequence aligner uses Java 6+ as the programming language. The communication with remote installations is done through DIRMI, a replacement for the by default in Java integrated remote method invocation library RMI. DIRMI was chosen over RMI as it allows for bidirectional connections to remote objects, making it possible to work behind a NAT while using remote computing resources.

The developed sequence aligner can be run locally using the multiple available cores of a machine and distributed over multiple computers with the software installed and running. For this prototype the installed software is GensearchNGS. Additionally, the user can also dynamically launch remote cloud instances of the aligner. The integration of cloud resources, for the moment using the Amazon AWS EC2 cloud service, is done through the generic Java clouds library jclouds. It allows for easy integration of cloud instances on various clouds, making it possible to launch and destroy cloud computing instances from inside Java. The aligner which is distributed is based on the aligner used in GensearchNGS [7]. The aligner uses a hash based index for the initial seed detection. The seeds are evaluated used a custom heuristic algorithm with the final alignment being performed by the Gotoh [2] algorithm, ideally suited for gapped alignment. While for this implementation of the distributed alignment we use this specific algorithm, the architecture could also be applied to other existing alignment algorithms.

The dataset used to test the prototype comes from the genome comparison & analytic testing project (GCAT) [8], which provides standardized

datasets to test different NGS data analysis software. The illumina-100bp-pe-exome-150x dataset has been used, which is available on the GCAT website (http://www.bioplanet.com/gcat/). The dataset contains 45'038'905 read pairs with an average read length of 100 bp. For the benchmarks in this paper, only the first 5 million read pairs have been used.

III. Architecture

The architectural choices are based on a typical scenario. Small genetic research laboratories often do not have the required infrastructure to perform big sequence alignments. But what they do have is a modest computing infrastructure with multiple multi-core desktop systems used for data analysis. A common evolution for those laboratories is to start NGS data analysis using targeted sequencing for a specific set of genes. Later they expand to whole exome analysis and in the end doing whole genome sequencing. While targeted sequencing can be done using a modest desktop computer, moving to whole exome and especially whole genome sequencing data quickly brings laboratories to their limits. Thus the goal of the proposed architecture is to optimally use the existing resources but also provide the option to expand the infrastructure when needed to outside resources. Possible options to expand to outside resource are, depending on the laboratory policies, cloud computing services like Amazon.

The core concept of the architecture is based on the idea of using stream processing for sequence alignment. As shown by [6], using stream processing for DNA sequence alignment can greatly increase the total processing time. This comes mainly from that fact, that uploading the data to a remote computing resource and analysing it can be done at the same time, drastically reducing the overall analysis time.

The concept of stream processing was not only used to integrate the remote computing resources, but to connect all parts of the system together, including local alignment threads. The core system, as shown in figure 1, is composed of 3 main elements: The `Data reader`, the `Sequence aligner` and the alignment `Writer`. For this implementation we choose to use `BlockingQueues` to communicate between the different elements. It

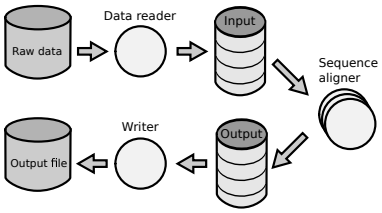


Figure 1: Overall stream based aligner architecture, connecting the different system elements through queues

is this choice of using Queues for the communication between the different parts that made the architecture very flexible. The `Data reader` reads the input data provided by the sequencer, usually in the FASTQ file format, and puts the individual reads on the Queue to be aligned. The different local threads implementing the `Sequence aligner` take those sequences from the queue and align them against the reference sequence. After having aligned a sequence, the `Sequence aligner` puts the created alignment in a different queue, the one containing the finished alignments. The alignment `Writer` takes the aligned sequences from the output queue and writes them into an alignment file, typically a SAM or BAM file. All the different elements of the system are run in their own thread.

Using queues to communicate between the different parts of the aligner makes it extremely flexible and scalable. This is shown in figure 2 which expands the basic local parallelization to work in a distributed environment. The addition of `Sequence aligner` and `Distributed client` allow to easily distribute the work to a remote machine. In fact, `Sequence aligner` and `Distributed client` can run simultaneously on the same machine, performing some alignment work locally while distributing other. It is even possible to have a `Distributed server` which in turn distributes the alignment workload further with its own `Distributed client`. This flexibility allows for interesting configurations which adapt to various real life situations discussed later.

To be able to connect to a remote machine, all that is required is that the remote machine is running the `Distributed server` object, accessible by the client machine. In the context of this prototype every user

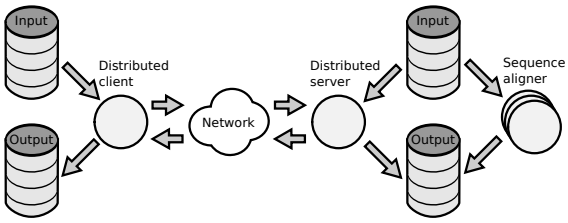


Figure 2: Integration of remote computing resources

of the GensearchNGS application has the option to let other users in the same network connect to his installation and use it to perform sequence alignment. The different installations discover each other using a custom automatic service discovery protocol. When starting the sequence alignment, the different local sequence alignment threads are created, and the remote `Distributed server` objects are automatically connected.

This concept has been expanded to include cloud computing. This has been done through the integration of the JClouds library, which allows to initialize cloud resources on demand. When starting the sequence alignment, the user is provided with the choice of how many cloud instances of the sequence aligner he wants to execute. Depending on his choice, a certain number of cloud resources are started, running a preconfigured image with the software preinstalled and executed upon booting. Once connected to the `Distributed server` object, there is no cloud specific code to perform the alignment. The system only sees an object that takes raw sequences from the input queue and puts aligned sequences back on the output queue.

The previously described possibility to create `Distributed server` objects which in turn distribute the workload further using `Distributed client` allows to handle particular limitations certain cloud providers have. In certain cloud environments it is desirable to only start a single cloud instance with a public IP address. Any additional cloud resource is then instantiated in a private network inside the cloud, only accessible from the outside through the public instance. Instead or additionally to a local sequence aligner in the `Distributed server` object, the `Distributed server` can run multiple `Distributed client` objects. This allows the public cloud instance to route the incoming raw sequencing stream from the outside to the multiple instances in the private network.

Another interesting side effect of this architecture is that it allows to perform sequence alignment on machines which are normally not powerful enough to handle the alignment. Aligning against the human reference sequence using the GensearchNGS aligner requires approximatively 5 GB of RAM. Especially older desktop systems do not always have that amount of RAM. The discussed architecture makes it possible to launch the sequence alignment locally on those machines, but without creating local `Sequence aligner` objects.

IV. Results & Discussion

To demonstrate the flexibility and performance of our architecture, we tested the prototype using multiple configurations. To do this, four typical configurations have been tested. The first one tested the performance using a single laptop. The second configuration added a second desktop computer located in the same network to speed up the calculations The third configuration expanded upon the second one by adding one instance of an Amazon AWS EC2 virtual machine. The fourth configuration uses no local alignment on the laptop which starts the alignment process, but offloads all of the alignment to two instances in the Amazon AWS EC2 cloud. In this configuration, the laptop only does the work of reading the raw data and saving the aligned sequences in the output file. The configurations of the tests have been chosen to represent a typical scenario in a small genetics laboratory. Figure 3 shows how the different computers used for the four configurations are connected.

The work laptop is equipped with an Intel Core i7-3520M dual core CPU, clocked at 3.6 Ghz with 8 GB of RAM. The desktop computer uses an Intel Core I7 870 with 4 cores clocked at 3.6 Ghz, also with 8GB of RAM. The Amazon AWS EC2 cloud instances of the aligner are launched on a virtual server of type c3.xlarge, which has a CPU of type Intel Xeon E5-2680 with 4 cores and 7.5 GB of RAM. The laptop and the desktop computer are connected with a 1 Gb/s switch, and both of them are connected to the internet with a

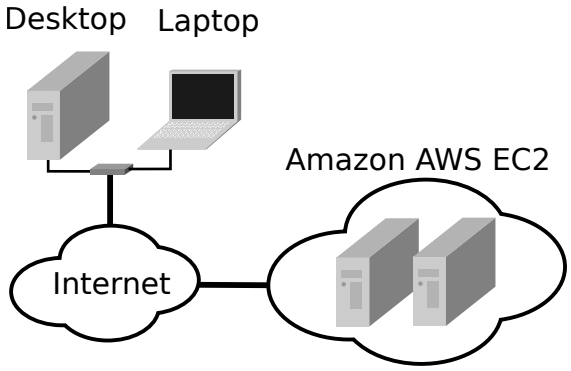


Figure 3: Topology of all elements used in the benchmark configurations.

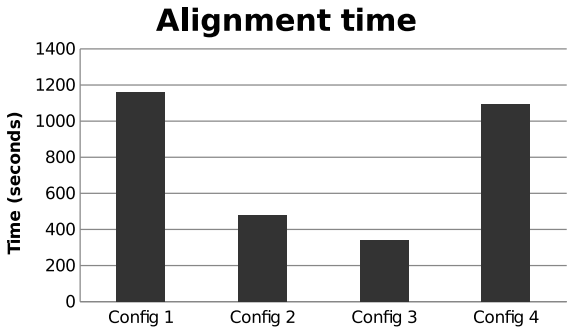


Figure 4: Alignment times on all 4 example configurations

100Mbs (up and down) connection.

Figure 4 shows the benchmark times for the distributed alignment on the different configurations. The measured times contain only the time required for the sequence alignment, not including initialization times which is about 3 minutes. Those initialization times contain the time required to load the reference sequence into memory as well as starting the cloud instances.

The raw times for the different configurations settings were: 1'163 seconds for configuration one, using only the laptop. 478 seconds for configuration 2, which used the laptop and the desktop computer. 339 seconds for the configuration 3, using the laptop, the desktop computer as well as one cloud instance. Configuration 4 finished in 1'096 seconds, using the laptop as a base station but doing all alignment work on two cloud instances.

Looking at the benchmark results we can conclude that the proposed architecture adapts well to all tested configurations. In the various configurations the different systems were well saturated, without slower systems badly affecting the overall performance. Load balancing is indeed a side-effect of the chosen design where every computing resource, be it a local thread or a remote computer, takes sequences to align out of a common queue. As long as I/O speeds permit, every computing resource is provided with the amount of work needed to saturate it.

The way remote computing resources are handled, be it in the local network or in the cloud, also permits for any amount of computers to fail. As long as at the computer launching the alignment and at least one computing resource which performs the alignment keep running, the alignment will successfully finish. This is achieved by keeping a local copy of every sequence sent to a remote computer. This local copy is only deleted once a successful alignment has been received for the sequence. If the remote computer disconnects, all the local copies of the sequences which have been sent to him but for which no alignment has been found yet, are put back in the input queue. They are then recovered and aligned by another computing resource which is still running.

V. Future work

While the developed prototype nicely shows the ability of the architecture to adapt to various situations and distribute the workload over all systems in the example configurations, there are still issues to be addressed. The first issue being the bandwidth required to distribute the workload over multiple computers, especially if they are located in a remote cloud. While the amount of data sent to the remote resource has already been minimized as much as possible, including efficient encoding of DNA sequences, not every possible optimization has yet been done to reduce the bandwidth requirements. A home internet connection will quickly saturate, putting an upper limit to the cloud instances that can be used simultaneously. While the internet connection of a genetic laboratory is usually higher than a standard home internet connection, the exact bandwidth requirements and limitation still need

to be evaluated and optimized. Once this step is done, the architecture will be compared to other distributed aligners like the previously mentioned ones [3, 4, 5, 6].

The second issue is the one of data security and confidentiality in distributed systems. In many data laboratories the data privacy rules restrict or forbid the usage of remote resources for any patient data. This is currently an active topic of research and no particular security measures were taken to improve the data security in this prototype. The two main features related to datasecurity and are planned to be implemented are: encrypting all data sent to and from the cloud and pooling multiple samples to be aligned simultaneously.

VI. Conclusion

We presented a generic architecture for stream based multilevel parallel alignment. The architecture uses the same concepts to distribute the alignment over multiple cores on one system and over multiple computers. The implemented prototype is able to adapt to various real life situations, using locally available computing resources or extend them using cloud resources. The effortless combination of the different distribution methods is a unique feature of this prototype, showing the potential for bioinformatics software to optimally use existing infrastructure and extend it if needed. While the implemented prototype has been integrated into a commercial NGS data analysis software, GensearchNGS, the proposed architecture is applicable to other alignment algorithms. The current implementation and its source code is not publicly accessible, but we have plans to release it at a later date under the GNATY (GensearchNGS Analysis Tools librarY) project. GNATY is in the process of being published and is free of access. Future work will include the optimization of the architecture as well as addressing the issue of datasecurity in the cloud related to DNA sequencing data.

References

[1] Li, H., & Homer, N. (2010). *A survey of sequence alignment algorithms for next-generation sequencing*, Briefings in Bioinformatics, 11(5), 473-83.

[2] O. Gotoh, *An improved algorithm for matching biological sequences*, Journal of Molecular Biology 162, 705-708, 1982.

[3] B. Langmead, M. C. Schatz, J. Lin, M. Pop and S. L. Salzberg, *Searching for SNPs with cloud computing*, Genome biology, 10:R134, 2009.

[4] M. C. Schatz, *CloudBurst: highly sensitive read mapping with MapReduce*, Bioinformatics, 25, 11, 1363-1369, 2009.

[5] C. S. Oehmen and D. J. Baxter, *ScalaBLAST 2.0: rapid and robust BLAST calculations on multiprocessor systems*, Bioinformatics, 29, 6, 797-798, 2013.

[6] S. A. Issa *et al.*, *Streaming Support for Data Intensive Cloud-Based Sequence Analysis*, BioMed research international, vol. 2013, Art.no. 791051, 2013.

[7] B. Wolf, P. Kuonen and D. Atlan, *Distributed DNA alignment, a stream based approach*, Doctoral Workshop on Distributed Systems, Bern, Switzerland, Proc., 39-41, 2012.

[8] G. Highnam *et al.*, *An analytical framework for optimizing variant discovery from personal genomes*, Nature Communications, 6, Art.no. 6275, 2015.

[9] B. Wolf *et al.*, *DNAseq Workflow in a Diagnostic Context and an Example of a User Friendly Implementation*, BioMed Research International, vol. 2015

Exploiting Heterogeneous Compute Resources for Optimizing Lightweight Structures

ROBERT DIETZE, MICHAEL HOFMANN, GUDULA RÜNGER

Department of Computer Science, Chemnitz University of Technology, Chemnitz, Germany

{dirob,mhofma,ruenger}@cs.tu-chemnitz.de

Abstract

Optimizing lightweight structures with numerical simulations leads to the development of complex simulation codes with high computational demands. The optimization approach for lightweight structures consisting of fiber-reinforced plastics is considered. During the simulated optimization, independent simulation tasks have to be executed efficiently on the heterogeneous computing resources. In this article, several scheduling methods for distributing parallel simulation tasks among compute nodes are presented. Performance results are shown for the scheduling and execution of synthetic benchmark tasks, matrix multiplication tasks, as well as FEM simulation tasks on a heterogeneous compute cluster.

Keywords Numerical simulations, scheduling, heterogeneous clusters

I. INTRODUCTION

The development of complex simulations in science and engineering leads to various challenges for application programmers, especially when targeting at future ultrascale computing systems [4]. The sustainability and portability of application codes represent important non-functional requirements, which can be provided, for example, with an appropriate methodology for the development process as well as technical support in the form of dedicated programming libraries [7]. By encapsulating the data exchange operations of coupled simulations, it is possible to achieve a flexibly distributed execution of the simulation components on distributed systems. However, for compute-intensive simulations also the efficient utilization of various computing resources, such as HPC servers or clusters, is important.

As an application example for complex simulations, we consider the optimization of lightweight structures based on numerical simulations which are studied in the research project MERGE[1]. The simulations cover the manufacturing process of short fiber-reinforced plastics and the characterization of their mechanical properties for specific operating load cases [6]. For solving the optimization problem, the simulations are performed several times with different parameter sets in order to develop an optimal set of parameters. The efficient execution of the simulations on HPC platforms leads to a task scheduling problem with the following properties:

- The tasks are independent from each other and the number of tasks is usually in the order of tens or hundreds.
- Since each task represents an execution of the same parallel simulation application, all tasks behave almost the same. This means that the expected parallel runtime is the same for all tasks and can be determined previously, for example, with separate benchmark measurements.
- The compute resources to be utilized are hierarchically organized and can comprise several compute clusters. Each cluster consists of several compute nodes and each node contains several compute cores.
- The compute resources are heterogeneous in the sense that each compute node has an individual performance.
- A parallel task can be either a shared-memory application that can be executed only on a single node including several cores or a distributed-memory application that can be executed on a cluster including several nodes.

[1]MERGE Technologies for Multifunctional Lightweight Structures, http://www.tu-chemnitz.de/merge

In this article, we investigate the use of scheduling algorithms for assigning simulation tasks to compute resources with the goal to reduce the total parallel runtime of the entire set of simulations. We employ task and data parallel scheduling methods and propose a new scheduling algorithm called WATER-LEVEL method. The WATER-LEVEL method is designed as a trade-off between task and data parallel executions and is based on a best-case estimation of the total parallel runtime of all tasks. All presented methods were implemented to solve to scheduling problem described above. We show performance results with different simulation tasks on a heterogeneous compute cluster.

The rest of this article is organized as follows: Section II presents the application example from mechanical engineering. Section III describes the approaches for scheduling the execution of the simulations on heterogeneous compute resources. Section IV shows corresponding performance results. Section V discusses related work and Sect. VI concludes the article.

II. SIMULATION AND OPTIMIZATION OF LIGHTWEIGHT STRUCTURES

The numerical optimization of lightweight structures consisting of fiber-reinforced plastics can be performed by a simulation approach which is described in the following.

II.1 Simulation of fiber-reinforced plastics

The lightweight structures can be manufactured by injection molding, which represents one of the most economically important processes for the mass production of plastic parts. The parts are produced by injecting molten plastic into a mold, followed by a cooling process. Fillers, such as glass or carbon fibers are mixed into the plastic to improve the mechanical properties, such as the stiffness or the durability of the parts. Besides the properties of the materials used, the orientation of the fibers and the residual stresses within the parts have a strong influence on the resulting mechanical properties. Thus, determining the mechanical properties of such short fiber-reinforced plastics requires to consider both the manufacturing process and specific operating load cases for the potential use of the plastic parts.

The manufacturing process can be simulated with computational fluid dynamics (CFD) that simulates the injection of the material until the mold is filled. The input data of the CFD simulation include the geometry of the part, the material properties, such as the viscosity or the percentage of mixed in fibers, and the manufacturing parameters, such as the injection position or pressure. The simulation results describe the fiber orientation and the temperature distribution within the part. These result data are used for simulating the subsequent cooling process with an approach based on the finite element method (FEM) that computes the residual stresses within the freezed part.

The simulation of the manufacturing process is followed by an evaluation of the resulting part. Mechanical properties are determined by simulating the behavior of the manufactured part for specific operating load cases of its future use. These simulations are also performed by FEM simulations using boundary conditions that correspond to the given load cases. The FEM application code employs advanced material laws for short fiber-reinforced plastics and uses the previously determined fiber orientation and residual stresses within the part as input data. The final simulation results describe the behavior of the part, for example, its deformation under an applied surface load.

II.2 Optimizing manufacturing parameters

The goal of the simulation process is not only to simulate one specific manufacturing process of a plastic part but to optimize the properties of the lightweight structures. This is done by an optimization process that varies selected material and manufacturing parameters, such as the fiber percentage or the injection position. The optimization is executed by repeatedly selecting specific values for the variable parameters and then simulating the manufacturing process and the load cases for the selected parameter configurations as described in the previous subsection. Thus, there are a number of simulation tasks to be executed (i. e., one for each parameter configuration to be simulated) that are independent from each other. The specific number of independent simulation tasks strongly depends on the number of variable parameters or on the optimization method employed and is usually expected to be in the order of tens or hundreds.

Figure 1 (left) shows an example of a plastic part, which is a plate made of short fiber-reinforced plastics with a hole on one side. The plate is clamped on two sides and a circular surface load is applied leading to the shown deflection in force direction. The shown optimal injection point leads to a fiber orientation within the plate that minimizes the deflec-

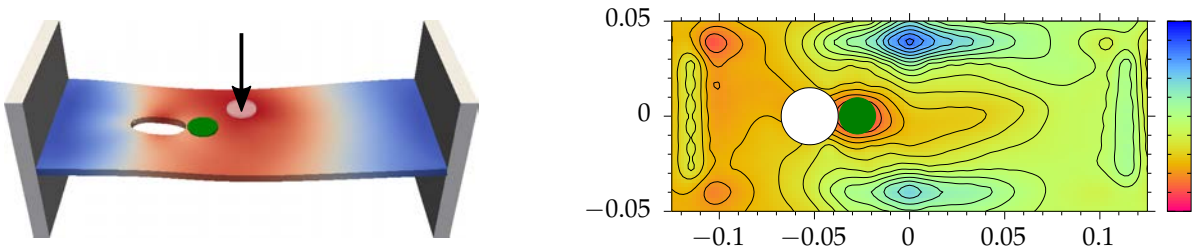


Figure 1: Left: Clamped plate with hole, applied surface load (arrow), and optimal injection point (green). Right: Contour plot of the objective function including the obtained minimum (green).

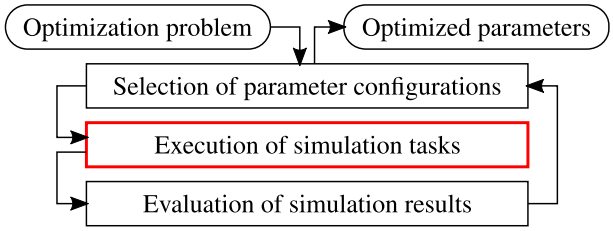


Figure 2: Overview of the optimization process.

tion. Figure 1 (right) shows a contour plot of the objective function for the corresponding optimization problem. The function values were determined during a Kriging-based optimization method [12]. This method creates an arbitrary number of candidate points for the optimal solution which are then recursively improved. The candidate points can be computed independently from each other, thus leading to a number of simulation tasks that can be executed at the same time. Figure 2 gives an overview of the optimization process. The repeated execution of the simulation tasks is the most time consuming part of the optimization process. Thus, performing these computations efficiently on HPC platforms is required and can be supported by an appropriate scheduling method for the parallel execution of simulation tasks on various compute nodes.

III. Distributing Simulations on Heterogeneous HPC Platforms

The optimization process described in Sect. II leads to independent numerical simulations that need be executed efficiently on HPC platforms. In the following, we describe a corresponding scheduling problem and present several scheduling algorithms for utilizing heterogeneous HPC clusters.

III.1 Scheduling problem

Task model: The independent numerical simulations are given as n_T parallel tasks $T_1, \ldots, T_{n_T}$. We assume that the parallel runtime of the simulations was previously determined with benchmark measurements on a specific reference compute node. Thus, for each task T_i, $i \in \{1, \ldots, n_T\}$, the given function $t_i(p)$ specifies the parallel runtime of the simulation when using p processor cores. Furthermore, it is known whether the tasks are capable of being executed either on a single node only (e. g., for OpenMP-based codes) or on a cluster of nodes (e. g., for MPI-based codes).

Machine model: The compute resources of the HPC platform to be used consist of n_N compute nodes $N_1, \ldots, N_{n_N}$. For each node N_j, $j \in \{1, \ldots, n_N\}$, its number of processor cores p_j and a performance factor f_j (with respect to the reference compute node) is given. The nodes are grouped into n_C clusters $C_1, \ldots, C_{n_C}$ such that each cluster is a subset of nodes and each node is part of exactly one cluster. Each cluster has to be able to execute an appropriate parallel task (e. g., MPI-based) on all its nodes.

Schedule: The goal is to determine an assignment of the given tasks to the compute resources of the HPC platform such that the total runtime for executing all tasks (i. e., the makespan) is minimized. For each task, the resulting schedule contains the compute resources to be used (i. e., nodes and utilized numbers of cores) and the estimated start time. Furthermore, for each task, the list of tasks that utilize the same compute resources immediately before is given. With this information, it will be possible to wait for their completion, especially if the runtimes in practice differ from the estimated runtimes.

III.2 Task and data parallel executions

Scheduling parallel tasks requires to determine the number of parallel cores to be used by each task. The following task and data parallel schemes will be used as reference methods:

Pure task parallel: This scheduling strategy uses only one core for each task and, thus, allows the execution of as many tasks as possible at the same time. The scheduling is performed by creating a list of all cores, using the core from the front of the list for the task to be scheduled next and then moving this core to the back of the list.

Pure data parallel: This scheduling strategy uses as many cores as possible for each task. Depending on the properties of the tasks (see Sect. III.1), either all cores of a node or all cores of a cluster are used as compute resources. The scheduling is performed by creating a list of all compute resources (i. e., either nodes or clusters), using the compute resource from the front of the list for the task to be scheduled next and then moving this compute resource to the back of the list.

Both methods schedule the tasks in their given order and use the compute resources in a round-robin scheme independently from their performance or utilization. We study the following adaptations to create further variants of the task and data parallel scheduling methods: The tasks are sorted in descending order based on their sequential runtimes to favor an early execution of long running tasks. Furthermore, scheduling a task is now performed by selecting the compute resource that provides the earliest finish (EF). This strategy replaces the round-robing scheme and is especially important for heterogeneous compute resources. Overall, we consider four task and data parallel scheduling variants, i. e., the original methods (TASKP and DATAP) and the variants with the earliest finish (TASKP-EF and DATAP-EF).

III.3 WATER-LEVEL method

In addition to the task and data parallel execution schemes described in the previous subsection, we present a further strategy for assigning tasks to compute resources which we call WATER-LEVEL method (WATERL). The method uses the given runtime functions of the tasks and the performance factors of the compute nodes to determine the compute resources for a each task. For realistic tasks, we assume the following

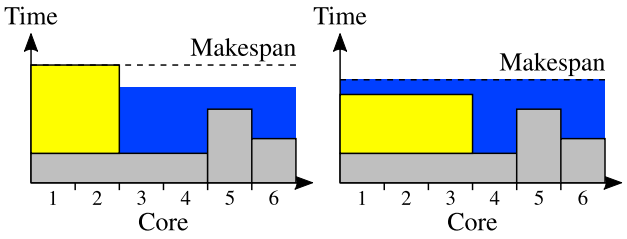


Figure 3: Scheduling of one task (yellow) either on two (left) or three (right) cores, previously scheduled tasks (gray), and optimally executed remaining tasks (blue).

behavior: The parallel runtime of a task decreases for increasing numbers of parallel cores until an optimal number of cores is reached and, thus, a higher number of parallel cores (up until the optimal number) should be preferred. However, the decreasing of the parallel runtime is usually restricted by parallelization overheads (e. g., due to communication or synchronization) and, thus, a lower number of parallel cores should be preferred. Therefore, the WATER-LEVEL method increases the number of cores for a task only up until an estimation of the resulting makespan reaches a minimum. This estimation is determined by assigning a task temporarily to a specific number of cores and assuming all remaining tasks can be executed optimally in parallel on the compute resources. Support for heterogeneous compute resources is achieved by taking the performance factors $f_j, j = 1, \ldots, n_N$, of the compute nodes into account for the estimation.

Figure 3 shows an illustration of the WATER-LEVEL strategy in which the current task to be scheduled (yellow) will use either two (left) or three (right) cores. All remaining tasks (blue) are assumed to be executed optimally in parallel on all cores (i. e., distributed like "water" over the "task landscape"). In this case, the current task would be assigned to three cores since the estimation of the resulting makespan (i. e., the "water level") reaches a minimum.

The pseudocode of the WATER-LEVEL method for tasks that can be executed only on single compute nodes is shown in Figure 4. The method starts by determining the total work W required for executing all tasks sequentially (line 4). Scheduling the tasks proceeds similar to the task and data parallel methods of the previous subsection: The tasks are sorted in descending order based on their sequential runtimes to favor an early execution of long running tasks (line 5) and a loop iterates over all tasks in the sorted order (line 7).

```
1  input  : tasks T_i, i = 1,...,n_T, with runtimes t_i(p)
2  input  : nodes N_j, j = 1,...,n_N, with p_j cores
3  output : compute resource and start time for each task
4  seq. work W = sum_{i=1}^{n_T} t_i(1)
5  sort T_i, i = 1,...,n_T in descending order of t_i(1)
6  // assume T_1,...,T_{n_T} are sorted
7  for i = 1,...,n_T do
8      W = W - t_i(1)
9      minimal makespan m* = infinity
10     for j = 1,...,n_N do
11         p = 0
12         repeat
13             p = p + 1
14             select p cores of N_j as resource R with start time s
15             estimate makespan m with optimally parallelized
               seq. work W and task T_i assigned to resource R
16             if m < m* then
17                 m* = m ; R* = R ; s* = s
18         until p >= p_j or t_i(p) is minimal
19     use resource R* and start time s* for task T_i
```

Figure 4: Pseudocode of the WATER-LEVEL *method for tasks that can be executed only on single compute nodes.*

In each iteration for the current task $T_i, i \in \{1,\ldots,n_T\}$, the sequential work W of all remaining tasks is calculated (line 8). Then, two loops iterate over the compute nodes (line 10) and their number of cores (line 12) to determine the compute resources for the task T_i that lead to a minimal makespan. The inner loop stops earlier if the parallel runtime $t_i(p)$ reaches a minimum for the current number of cores p. It depends on the given runtime function whether and how this minimum can be determined.

For tasks that can be executed only on a single compute node, a compute resource R consists of a specific node and the number of cores to be used on that node. The final schedule also requires the corresponding start time s on the compute resource (line 14). The selected compute resource R is temporarily used for task T_i and the resulting makespan m with optimally parallelized work W of the remaining tasks is estimated (line 15). If this estimated makespan m is smaller than the current minimal makespan m^*, then the corresponding compute resource R and start time s are stored (line 17) such that they can be later used for the task T_i (line 19).

Nodes	Processors	Cores	GHz
`cs1,cs2`	Intel Xeon E5345	$2 \times 2 \times 4$	2.33
`sb1`	Intel Xeon E5-2650	$1 \times 2 \times 8$	2.00
`ws1,...,ws5`	Intel Xeon X5650	$5 \times 2 \times 6$	2.66

Table 1: List of the compute resources used.

For tasks that can be executed on a cluster of nodes, the pseudocode shown in Fig. 4 has to be modified. The clusters $C_1,\ldots,C_{n_C}$ have to be provided as input and loops over the clusters and their cores replace the lines 10 and 12. Additionally, a compute resource R for the current task T_i, $i \in \{1,\ldots,n_T\}$ will then contain a subset of the nodes of the current cluster and the numbers of cores to be used on each of these nodes. In general, the distinction between the two kinds of tasks could also be performed on a per task basis in an implementation of the WATER-LEVEL method.

IV. PERFORMANCE RESULTS

The task and data parallel methods as well as the WATER-LEVEL method have been used for the scheduling of different simulation tasks. In the following, we present performance results on a heterogeneous compute cluster.

IV.1 Experimental setup

The heterogeneous compute cluster used consists of 8 compute nodes, each with two multi-core processors. Table 1 lists the nodes and their specific processors. The parallel runtime of the tasks required for the scheduling is determined with separate benchmark measurements on the reference compute node `cs1`. The performance factors of the other compute nodes are derived from the sequential runtimes of a task on those compute nodes. In the following subsections, we show total parallel runtimes for executing a number of tasks according to the determined schedules. Starting the tasks is performed by a Python script running on a separate front-end node of the cluster using SSH connections to the compute nodes to be utilized. Each schedule is executed 5 times and the average result is shown.

IV.2 Benchmark tasks

As synthetic benchmark, we employ "sleep" tasks that perform no computations, but only wait for a specific time

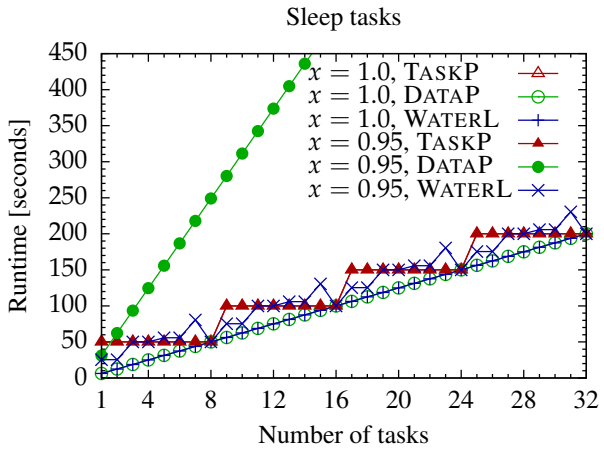


Figure 5: Parallel runtime of different scheduling methods using sleep tasks without ($x = 1.0$) and with parallelization overhead ($x = 0.95$) executed on node `cs1`.

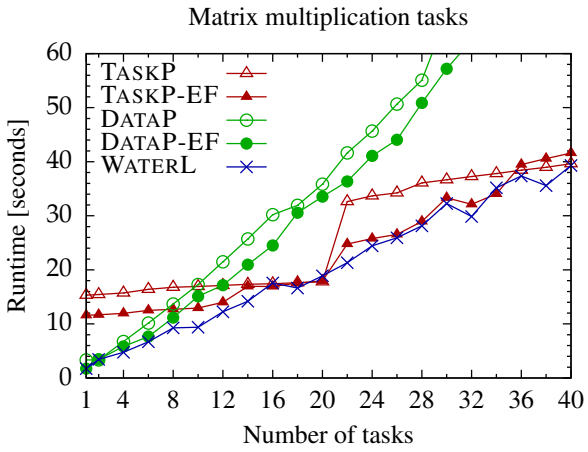


Figure 6: Parallel runtime of different scheduling methods using matrix multiplication tasks executed on nodes `cs1` and `ws1` with a total of 20 cores.

$t(p) = 50s \cdot \left[x \cdot \frac{1}{p} + (1-x) \cdot (\log p + p)\right]$. The runtime comprises of a fraction x which decreases linearly with the number of cores p (e. g., computation) and a remaining part, which increases logarithmically and linearly (e. g., parallelization overhead).

Figure 5 shows parallel runtimes of sleep tasks with $x = 1.0$ (i. e., without parallelization overhead) and $x = 0.95$ (i. e., with parallelization overhead) executed on node `cs1` depending on the number of parallel tasks using different scheduling algorithms. The TaskP method achieves the same results for both kinds of tasks, because the tasks are always executed sequentially. The runtime shows a step-wise increase after every 8 additional tasks, since with these task numbers all 8 cores of the compute node are equally utilized. The DataP method uses always the maximum number of 8 cores for each task and shows strong differences between the two kinds of tasks. With $x = 1.0$, the parallel runtime of a task decreases linearly and the minimum runtime is achieved using the maximum number of cores. However, with $x = 0.95$, using the maximum number of cores leads to a strong increase of the runtime due to the increasing parallelization overhead. The Water-Level method achieves a trade-off between the TaskP and DataP method. With $x = 1.0$, the optimal result of the DataP method is achieved. With $x = 0.95$, the runtime results are close to the runtime results of the TaskP method, but show a different shape with a more continuous increase instead of the step-wise increase.

The matrix multiplication operation (DGEMM) from the OpenBLAS library is used as parallel benchmark tasks that can be executed on one compute node only. The number of cores utilized is controlled with the environment variable `OPENBLAS_NUM_THREADS`. The matrix size is set to 4000×4000. Figure 6 shows parallel runtimes depending on the number of parallel tasks using different scheduling algorithms and the compute nodes `cs1` and `ws1`. The TaskP methods shows a step-wise increase similar to Fig. 5. Additionally, there is a slight increase between each step since the matrix multiplication tasks on the same compute node influence each other. Selecting the compute nodes according to the earliest finish (TaskP-EF) causes an earlier utilization of the faster node `ws1` and thus, decreases the runtime for specific task numbers. The DataP method shows a strong increase of the runtime due to the parallelization overhead caused by always using the maximum number of cores. The DataP-EF method (i. e., with earliest finish) selects the faster compute node `ws1` more often, thus leading to a smaller runtime. The results of the Water-Level method demonstrate the trade-off between the task and data parallel schemes. With small numbers of tasks, the runtime of the Water-Level method is equal or below the best data parallel scheme and for higher numbers of tasks, the runtime of the Water-Level method is similar to the best task parallel scheme.

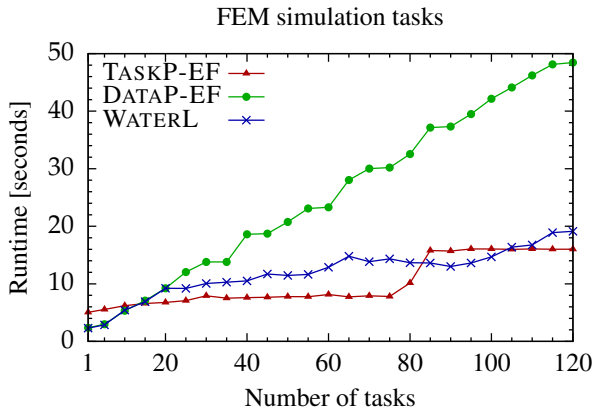


Figure 7: Parallel runtime of different scheduling methods using FEM simulation tasks executed on all nodes listed in Table 1 with a total of 92 cores.

IV.3 FEM simulation tasks

An OpenMP parallel FEM code [3] is used as simulation tasks in the optimization process for lightweight structures as described in Sect. II. Figure 7 shows parallel runtimes depending on the number of parallel tasks using different scheduling algorithms and all compute nodes listed in Table 1. The data parallel scheme with the DATAP-EF method leads to strongly increasing runtimes, which is caused by the low speedup achieved with the parallel FEM code. Thus, using a data parallel execution with high numbers of cores is only advantageous if there are few FEM simulation tasks (i. e., about twice the number of compute nodes). The task parallel scheme with the TASKP-EF method often leads to the smallest runtimes, but shows a steep increase if the number of tasks approaches the total number of cores. The WATER-LEVEL methods leads to the same results as the DATAP-EF method for small numbers of tasks. However, for higher numbers of tasks, the WATER-LEVEL methods is up to a factor of two slower than the TASKP-EF methods. This behavior is caused by the less efficient parallel execution of the FEM code that favors an execution with small numbers of cores. In contrast to that, the WATER-LEVEL method assumes an optimal parallel execution of the unscheduled tasks for the estimation of the makespan. Since this estimation differs strongly from the actual parallel runtime of the tasks, the schedule of the WATER-LEVEL method differs significantly from the faster task parallel schedule.

V. RELATED WORK

Scheduling is a popular problem in computer science involving different application areas and approaches [9]. One of those areas is the scheduling of sequential or parallel tasks to be executed on a given set of hardware resources (e. g., processors) while additional dependencies between the tasks may restrict their execution order. Determining an optimal schedule (e. g., with minimal Makespan) for tasks with dependencies is an NP-hard problem that is usually solved with heuristics or approximation algorithms [8]. The layer-based scheduling algorithm from [5] decomposes a set of tasks with dependencies into layers of independent tasks and schedules each layer separately with a so-called list scheduling algorithm. Since the simulation tasks in our optimization process are independent, we can omit a decomposition into layers.

List scheduling algorithms add priorities to the single tasks and assign the tasks in descending order of their priority to the processors. Algorithms, such as *Largest Processing Time* (LPT) [2] and *Longest Task First* (LTF) [14], use the given runtime of the tasks as priorities, thus scheduling compute intensive tasks first. Algorithms for heterogeneous architectures, such as *Heterogeneous Earliest Finish Time* (HEFT) [13] and *Predict Earliest Finish Time* (PEFT) [1], also take the runtime of the tasks on individual processors into account for the priorities. The proposed WATER-LEVEL method is also a list scheduling algorithm that prioritizes the tasks according to their runtime. However, the WATER-LEVEL method uses only the sequential runtime as priority and uses the individual processor speeds of a heterogeneous architecture for the allocation of cores by parallel tasks and for the selection of compute nodes.

Scheduling parallel tasks with dependencies can also be performed with a two-step approach consisting of an allocation step and a scheduling step. The scheduling step assigns the parallel tasks to specific processors and is usually based on a list scheduling algorithm. The allocation step determines the number of processors for each parallel task. This step is usually performed iteratively starting with an initial allocation (e. g., one processor per tasks) and then repeatedly assigning additional processors to tasks (e. g., to shorten the critical path). Examples for such algorithms are *Critical Path Reduction* (CPR) [10] and *Critical Path and Allocation* (CPA) [11]. The WATER-LEVEL method performs the allocation of cores only once for each task during the list scheduling and, thus, omits repeated allocation and scheduling steps.

VI. Conclusion

In this article, we have proposed the WATER-LEVEL scheduling method for parallel tasks without dependencies on heterogeneous HPC platforms. The method performs an iterative assignment of tasks to compute resources and uses a best-case estimation of the makespan to determine the number of cores to be used for each task. Performance results for benchmark tasks demonstrate a good trade-off between task and data parallel execution schemes. However, for simulation tasks with low parallel efficiency, the best-case estimation may differ strongly from the actual runtimes achieved. This disadvantage might be solved by using the given parallel runtimes of the tasks for a better estimation of the makespan.

Acknowledgment

This work was performed within the Federal Cluster of Excellence EXC 1075 "MERGE Technologies for Multifunctional Lightweight Structures" and supported by the German Research Foundation (DFG). Financial support is gratefully acknowledged.

References

[1] H. Arabnejad and J.G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *Transactions on Parallel and Distributed Systems*, 25(3):682–694, 2014.

[2] K.P. Belkhale and P. Banerjee. An approximate algorithm for the partitionable independent task scheduling problem. In *Proc. of the 1990 Int. Conf. on Parallel Processing, (ICPP'90)*, pages 72–75, 1990.

[3] S. Beuchler, A. Meyer, and M. Pester. SPC-PM3AdH v1.0 - Programmer's manual. *Preprint SFB/393 01-08, TU-Chemnitz*, 2001.

[4] L. A. Bongo, R. Ciegis, N. Frasheri, J. Gong, D. Kimovski, P. Kropf, S. Margenov, M. Mihajlovic, M. Neytcheva, T. Rauber, G. Rünger, R. Trobec, R. Wuyts, and R. Wyrzykowski. Applications for ultrascale computing. *Supercomputing Frontiers and Innovations*, 2(1):19–48, 2015.

[5] J. Dümmler, T. Rauber, and G. Rünger. Programming support and scheduling for communicating parallel tasks. *J. of Parallel and Distributed Computing*, 73(2):220–234, 2013.

[6] M. Hofmann, F. Ospald, H. Schmidt, and R. Springer. Programming support for the flexible coupling of distributed software components for scientific simulations. In *Proc. of the 9th Int. Conf. on Software Engineering and Applications (ICSOFT-EA 2014)*, pages 506–511. SciTePress, 2014.

[7] M. Hofmann and G. Rünger. Sustainability through flexibility: Building complex simulation programs for distributed computing systems. *Simulation Modelling Practice and Theory, Special Issue on Techniques And Applications For Sustainable Ultrascale Computing Systems*, 2015. (to appear).

[8] J.T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.

[9] M.L. Pinedo. *Scheduling: Theory, algorithms, and systems*. Springer.

[10] A. Radulescu, C. Nicolescu, A.J.C. van Gemund, and P.P. Jonker. CPR: Mixed task and data parallel scheduling for distributed systems. In *Proc. of the 15th Int. Parallel and Distributed Processing Symposium (IPDPS'01)*, pages 1–8. IEEE, 2001.

[11] A. Radulescu and A.J.C. van Gemund. A low-cost approach towards mixed task and data parallel scheduling. In *Proc. of the Int. Conf. on Parallel Processing (ICPP'01)*, pages 69–76. IEEE, 2001.

[12] M. Strano. A technique for FEM optimization under reliability constraint of process variables in sheet metal forming. *Int. J. of Material Forming*, 1(1):13–20, 2008.

[13] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Task scheduling algorithms for heterogeneous processors. In *Proc. of the 8th Heterogeneous Computing Workshop (HCW'99)*, pages 3–14. IEEE, 1999.

[14] J. Turek, J.L. Wolf, and P.S. Yu. Approximate algorithms scheduling parallelizable tasks. In *Proc. of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 323–332. ACM, 1992.

Chameleon© C2HDL Design Tool In Self-Configurable Ultrascale Computer Systems Based On Partially Reconfigurable FPGAs

Anatoliy Melnyk, Viktor Melnyk, Lyubomyr Tsyhylyk

Intron Ltd, Ukraine

aomelnyk@intron-innovations.com vmelnyk@ intron-innovations.com l.tsyhylyk@gmail.com

Abstract

The FPGA-based accelerators and reconfigurable computer systems based on them require designing the application-specific processors soft-cores and are effective for certain classes of problems only, for which these soft-cores were previously developed. In Self-Configurable FPGA-based Computer Systems the challenge of designing the application-specific processors soft-cores is solved with use of the C2HDL tools, allowing them to be generated automatically. In this paper, we study the questions of the self-configurable computer systems efficiency increasing with use of the partially reconfigurable FPGAs and Chameleon© C2HDL design tool, corresponding to the goals of the project entitled "Improvement of heterogeneous systems efficiency using self-configurable FPGA-based computing" which is a part of the NESUS action. One of the features of the Chameleon© C2HDL design tool is its ability to generate a number of application-specific processors soft-cores executing the same algorithm that differ by the amount of FPGA resources required for their implementation. If the self-configurable computer systems are based on partially reconfigurable FPGAs, this feature allows them to acquire in every moment of its operation such a configuration that will provide an optimal use of its reconfigurable logic at a given level of hardware multitasking.

Keywords self-configurable computer systems, field-programmable gate arrays, high-performance computing, reconfigurable computing, C2HDL design tools.

I. Introduction

Today one of the most promising areas of activity in the field of high performance computing is creation of the reconfigurable computer systems (RCCS). RCCSs compete with other types of high-performance computer systems due to high speed characteristics of the modern field-programmable gate arrays (FPGAs) - the hardware base of a reconfigurable computing environment (RCE) of RCCS, and due to advances in the design technology of application-specific processors to be synthesized in RCE of RCCS.

Co-functioning of a computer system based on general-purpose processors with application-specific processors synthesized in RCE, whose structure considers executed algorithms features, allows increasing its overall performance by 2-3 orders of magnitude. The reconfigurability and ability to synthesize an application-specific processor (ASP) with a new structure and functionality in RCE allows changing the functional commitment of a created thereby RCCS with preserving its high performance in the new class of problems.

Along with high performance ensured by the RCCS, there are also challenges associated with their application. These particularly are significant timing expenses for task distribution between the processing units, often lack of IP cores required for implementation in the RCE which forces to develop them from scratch, and high additional requirements to the RCCS users qualification, as they, besides modeling and programming, must perform a system analysis, design the ASPs architecture, perform their synthesis and implementation in RCE.

Deprived of the above mentioned problems are self-configurable computer systems (SCCS), where these labor-intensive and time-consuming tasks are fully automated and replaced from the operator to the computer system. Taking into account the necessity of further improvement of the computer means performance

and extension of the reconfigurable devices application in computer systems, further development in the field of self-configurable computer systems is a topical task of scientific research and engineering.

II. Related Work

The concept of the self-configurable FPGA-based computer systems design, the method of information processing in them, and their structure are proposed in paper [1]. The issue of labor-intensive and time-consuming tasks, which is a characteristic for RCCS, is removed from SCCS owing to the new method of information processing applied in it.

The software tools that should be used in a SCCS as its components are available today. In this regard, we may consider an approach to the development of computational load balancing systems between a general-purpose computer and the hardware accelerator proposed in [2]. The IP cores' generators [3], [4], IP cores' libraries [5], and system level design tools and solutions available on the market could be used for the ASP design on the basis of its algorithm description by a high-level programming language [6] - [8].

III. Problem Statement

Implementation of the SCCS basing on partially reconfigurable FPGAs enables organization of multiple-task execution in the reconfigurable environment. This opportunity is provided as the subprograms of different tasks are executed independently in FPGA's different reconfigurable regions, and each of them is loaded into the FPGA as a partial configuration after initialization of the respective program. Such SCCS operation has a number of advantages, among which - the actual multitasking, an effective use of the reconfigurable logic and rationalization of energy consumption. At the same time, this mode of the SCCS operation imposes additional requirements for the generating system to create the application-specific processors HDL-models. Depending on the workload of the computer system, the amount of available for one separate task reconfigurable logic resources at a time can range from a maximum value that corresponds to all FPGA dynamic part resource, to the minimum value that corresponds to one or a number of reconfigurable regions, and vice versa. The question arises to organizing dynamic reallocation of the reconfigurable logic resources and replacing some running application-specific processors with others performing the same tasks but differing by the equipment volume. This should be done to provide an effective use of resources and the required level of multitasking.

To address this challenge it is necessary, during the program compilation, for each subprogram executed in the reconfigurable environment, to generate a number of application-specific processors HDL-models $ASPM\{ASPM_{opt}, ..., ASPM_{min}\}$, where $ASPM_{opt}$ is an optimum HDL-model that uses all the space-time properties of an algorithm given by the subprogram and to be implemented requires the largest amount of the reconfigurable logic resources among the $ASPM$ models; $ASPM_{min}$ is an HDL-model that to be implemented requires the minimum amount of the resources. In this regard, we propose the Chameleon© C2HDL design tool, which for each algorithm, given by the ANSI C program, can generate a set of application-specific processors VHDL soft-cores that differ by the amount of equipment to be implemented.

The paper structure is the following:

Section IV shows the principles of information processing in SCCS and its structure organization.

Section V highlights the partially reconfigurable FPGAs operation features.

Section VI introduces the characteristics and features of the Chameleon© C2HDL design tool.

Section VII shows an example of application of the Chameleon© C2HDL design tool in the SCCS for creation of a set of FFT processors VHDL models.

In our experiment the reconfigurable environment of the SCCS is built on the Altera FPGA, therefore created processors models are targeted at being implemented in this FPGA and differ mainly by the number of the embedded DSP blocks they use. The duration of these FFT processors VHDL models generation and their technical characteristics are shown.

Section VIII concludes the paper.

IV. Self-Configurable Computer Systems And The Method Of Information Processing In It

The self-configurable computer system is the computer system with reconfigurable logic where the program compilation includes automatically performed actions of creation of configuration. and which acquires that configuration automatically in the time of program loading for execution [1]. The SCCS automatically executes: 1) computational load balancing between the general-purpose processor and reconfigurable environment (RCE); 2) creation of an application-specific processor (ASP) HDL-model. Loading of the configuration files obtained after logical synthesis into the RCE is carried out by the operating system in parallel with loading of the computer subprogram executable file into its main memory after program initialization [1].

The method of information processing in the SCCS consists of three stages: compiling the program, its loading, and execution. The user creates a program P_{in} written in a high-level programming language and submits it into the SCCS. During compiling the SCCS automatically performs the following actions: divides this program into the general-purpose processor's subprogram P_{GPP} and RCE's subprogram P_{RCE}, performs P_{GPP} compilation, generates P_{GPP} executable file obj, creates ASP's HDL-model $ASPM$ to perform P_{RCE} subprogram, performs ASP's logic synthesis, and stores the obtained executable file obj and configuration files of RCE $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$, where K_{FPGA} is the number of FPGAs forming RCE, into the secondary memory.

These actions are performed in the SCCS with the following means:

1. The computational load balancing system for load balancing between a general-purpose processor and RCE. This system automatically selects fragments from the program P_{in} whose execution in the RCE reduces its execution time, and divides the program P_{in} into the P_{GPP} subprogram replacing the selected fragments in there by instructions for interaction with the RCE, and the P_{RCE} subprogram, formed from the selected fragments. An example of such a system is described in [2]. This system creates the RCE subprogram in x86 assembly language, thus it must be supported by the means for the assembly language code translation into a high-level language to be used in the SCCS. The tools of this type are available on the market, for example Relogix Assembler-to-C Translator [9] from MicroAPL.
2. A compiler for compiling P_{GPP} subprograms from the language that they are represented in into the object codes obj that can be directly executed by the general-purpose processor.
3. A generating system for the ASPs HDL-models creation. The system automatically generates models $ASPM$ from the RCE subprograms P_{RCE}, like Chameleon© [7], [10], that is discussed in this article, or Agility Compiler [11] and DK4 Design Suite [12] from Celoxica, or CoDeveloper from Impulse [13].
4. Logic synthesis tools and FPGA configuring tools for the ASPs HDL-models logic synthesis during the program compilation stage and FPGA configuring during the program loading stage. These tools are available from the FPGA vendors, for example, Vivado Design Suite, ISE, Alliance, Foundation from Xilinx; Quartus II, Max + II from Altera.

From the **conf** and obj files a combined executable file is formed and stored into the secondary memory. At the stage of the program loading after its initialization, the SCCS loads the executable file obj of the general-purpose processor's subprogram into the main memory using a conventional loader and, at the same time, loads the configuration files $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$ into the RCE and thus creates an ASP in there using the FPGA configuring tools. Then, the stage of the program execution is performed.

The major part of the SCCS basic software means represents a compiler which combines the computational load balancing system, a compiler for GPPs subprograms compilation, a generating system for ASPs HDL-models creation, and ASPs logical synthesis tools.

The reconfigurable environment of the SCCS can be realized on the base of partially reconfigurable FPGAs,

which gives an opportunity to organize the hardware multitasking there and brings a number of other benefits. The partially reconfigurable FPGAs operation features are briefly highlighted below.

V. Partially Reconfigurable FPGAs Operation Features

The ability to reconfigure a part of an FPGA circuitry after its initial configuration while the other parts remain unaffected is referred to as partial reconfiguration. The direct benefits of using this ability is a significant reduction of the duration of reconfiguring and reduction of the memory size required for the configuration storage (the size of the bit-stream is directly proportional to the number of resources being configured). Also, this ability opens new possibilities for the reconfigurable logic application in computers, particularly, it allows organizing hardware multitasking in FPGA and embodying the concept of Virtual Hardware, that is combined extremely well with the concept of SCCS design.

Partial reconfiguration is carried out in FPGA by downloading partial configurations files after its initial configuration, and thus - during the operation. These files specify only the configuration of the FPGA parts called Reconfigurable Partitions or Reconfigurable Regions, each of them contains separate device's modules. Reconfigurable partitions contain a certain amount of equipment and have a clearly defined location and boundaries in the FPGA circuitry. In this regard, the device needs a modular structure. The modules loaded into the reconfigurable partitions are called Reconfigurable Modules.

Partial reconfiguration can be static, when the device is not active during the reconfiguration process (while the partial configuration data is sent into the FPGA, the rest of it is stopped and brought up after the configuration is completed), and dynamic, also known as active partial reconfiguration, which enables changing the part of the FPGA while the rest of it is still operating.

Besides one or more reconfigurable regions, a partially reconfigurable FPGA also contains a static region which remains unchanged during partial reconfiguration. For example, partial reconfiguration controller, memory and interface logic can operate in this region. The Partial Reconfiguration Controller automates the mentioned process. The user can develop a controller by himself or can use ready available on the market solution. The controller can also be external to the FPGA device.

Two modes of the partial reconfiguration are used:

- Module-based - implies creation of a reconfigurable module and, with the help of relevant software, generation of its partial configuration code. This code completely replaces the previously synthesized reconfigurable module in the selected reconfigurable region. Note that this approach requires interfaces interoperability of all reconfigurable modules that operate in one reconfigurable region.

- Difference-based - implies introducing small changes to the scheme of the previously synthesized reconfigurable module. Partial configuration code contains information about the differences between the structures of the existing and new modules operating in the reconfigurable region, and is formed by "fusion" of the binary codes of the previously loaded to the FPGA configuration with the new one, for example, using XOR operation. This approach makes it possible to significantly reduce the size of configuration code. It is used, for example, to replace the contents of table operating device, memory contents, etc. This approach is especially interesting for implementation of evolutionary algorithms.

Partial reconfiguration design flow and mechanisms are being continuously improved. For example, in Virtex, Virtex-II, Virtex-II Pro and Virtex-E FPGAs from Xilinx, the configuration can be changed only by full columns of the reconfigurable matrix, and their numbers have to be multiples of 4 (4, 8, 12, ...). In Virtex-4 FPGAs this restriction is eliminated, while it is possible to change the configuration of an arbitrary rectangular area of the matrix, with some restrictions on its height. In modern Xilinx FPGAs (today it is 7th generation: Artix-7, Kintex-7, Virtex-7 and Zynq-7000 SoC) the minimum regions whose configuration can be changed independently are called Reconfigurable Frames. The width of the reconfigurable frames is one

element (there are different types of elements, including CLB, BRAM, DSP), while the height - the one clock region or input/output block. Some examples are as follows: in the Xilinx FPGAs 7th generation devices [14] - CLB: 50 x 1; DSP48: 10 x 1; RAM: 10 x 1; in the UltraScale devices [15] - CLB: 60 x 1; DSP48: 24 x 1; RAM: 12 x 1.

A partial configuration file consists of a certain number of configuration frames (not to be confused with the reconfigurable frames). The configuration frame is the minimum unit of information of this file and sets a configuration for one reconfigurable frame.

In the Altera FPGAs, the partial reconfiguration is implemented similarly.

VI. Chameleon© C2HDL Design Tool

The Chameleon© C2HDL design tool is initially targeted for use in the heterogeneous ultrascale computer systems. It is intended for the ASP's HDL-model automatic generation from the algorithm described in the ANSI C language [7], [10]. The developer, specifying an algorithm of the data processing on ANSI C, in return gets a fully debugged and synthesizable VHDL RTL model of the device that implements the described algorithm. The architecture of the device is fully optimized for the executed algorithm and maximally uses its ability for paralleling. The obtained VHDL design may be further implemented in the FPGA by any FPGA design solution, e.g. the Xilinx Vivado Design Suit or Altera Quartus II.

Besides the algorithm of the data processing, the input information for the Chameleon© C2HDL design tool are also the ASP's interface specification and technical characteristics, for example, desired performance or algorithm execution time boundaries. The platform for the ASP synthesis is configurable processor architecture configured according to the following input parameters: the number of Functional Units, an instruction set for each Functional Unit, the size and content of the instruction and data memory, the communication network structure.

The basic scheme of the Chameleon© C2HDL design tool operation is shown in Figure 1.

The Chameleon© C2HDL design tool features:

1. Short generation time. For example, generation of the FFT processor VHDL model with 50 Functional Units takes several minutes on a conventional PC.
2. Desired pre-set level of the algorithm parallelization.
3. Quick search of the appropriate level of parallelization to achieve the desired ASP's performance or power consumption.
4. The architecture of the ASP is tested and verified, which eliminates the probability of synthesis and operation errors.

Thus, this tool can be effectively used in the SCCS, and the example of its usage is shown in the next section.

VII. Experimental Results

We have used the Chameleon© C2HDL design tool as one of the basic software means of the SCCS compiler. The SCCS hardware platform is realized on the base of the conventional personal computer running on the Windows OS and the reconfigurable environment built on the Cyclone V FPGA from Altera. The RCE subprogram chosen for the experiment represents the algorithm of the 64-point Fast Fourier Transformation in the ANSI C language, its code is given in Figure 2. This program has been submitted to the input of the Chameleon© C2HDL design tool, and a set of the RTL VHDL-models of the 64-points FFT processors, whose structures contain a different number of Functional Units, has been automatically generated. As a most productive the one containing 15 Functional Units was determined by the Chameleon© C2HDL design tool; in all the models a number of these modules is determined automatically. The Functional Units are implemented in the Cyclone V FPGA as an embedded DSP blocks in relation 1x1.

Depending on the workload, the SCCS operating system can choose the FPGA partial configuration that contains an appropriate by the equipment amount or a performance FFT processor, and replace the operating in the RCE instance of the processor to another on

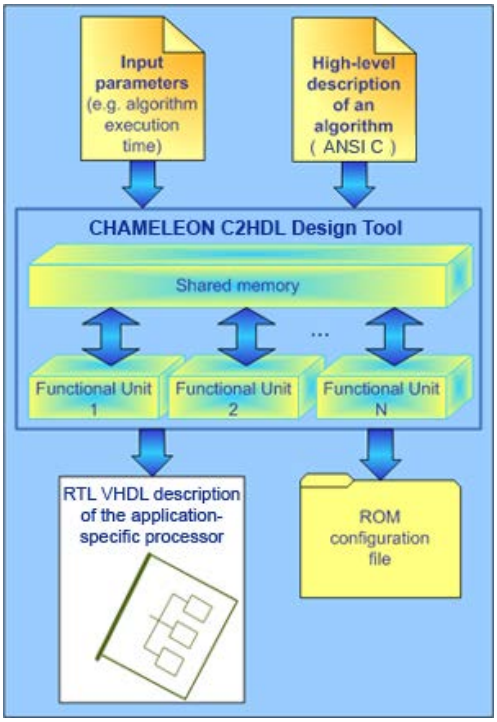


Figure 1: Basic Scheme of Chameleon© C2HDL Design Tool Operation.

the run. Table 1 shows, the technical characteristics of the FFT processors VHDL-models generated with Chameleon© C2HDL design tool, and synthesised in the Cyclone V 5CSEMA5F31C6 device by the Quartus II 13.1.0 Web Edition.

Number of the Functional Units	LUT utilization	Maximum Frequency (MHz)	Command count	FFT time (us)
1	1.809	204.08	1900	9.31
2	2.380	200,32	1015	5,07
4	3.054	216,8	575	2,65
7	4.806	174,52	388	2,22
8	4.858	190,73	352	1,85
10	6.715	171,85	311	1,81
13	9.200	149,7	190	1,27
15	10.198	138,48	180	1,30

Table 1: Technical Characteristics of FFT Processors.

Basing on this data, the SCCS operating system can choose which FFT processor configuration to acquire at a certain moment of its operation, depending on the actual workload. For example, the configuration consuming 1809 LUTs executes FFT in 9.31 us, and configuration consuming 9200 LUTs - in 1.27 us.

The time required by the SCCS for the FFT processors VHDL-models generation generally increases linearly with increasing the number of parallel Functional Units (see Figure 3). The main part of the generation time is spent on the algorithm parallelization and schematic optimization.

VIII. Conclusions

Implementation of the SCCS basing on partially reconfigurable FPGAs enables organization of the simultaneous multiple-task execution in the reconfigurable environment of the SCCS as the subprograms of different tasks are executed independently in different reconfigurable regions of the FPGA. Such SCCS operation has a number of advantages, among which, besides the actual multitasking - effective use of the reconfigurable logic and rationalization of energy consumption. At

```
#include "inout.h"
#define nx 64

void main()
{
        int x[nx*2]; int w[nx]; int ia, ib, i, j, k; int rtemp, itemp; int c, s; int n2 = nx >> 1;
        (a set of constants w[0]… w[63])
        (an array of the input data, real and imagine parts x[  0], x[ 127], sorted in bit-reverse order)
        for (k = 1; k < nx; k <<= 1) {
                ia = 0;
                for (j = 0; j < k; j++) {
                        c = w[2 * n2 * j];
                        s = w[2 * n2 * j + 1];
                        for (i = 0; i < n2; i++) {
                                ib = ia + k;
                                rtemp = (int)(((__int64)c * x[2 * ib])) - (int)(((__int64)s * x[2 * ib + 1]));
                                itemp = (int)(((__int64)c * x[2 * ib + 1])) + (int)(((__int64)s * x[2 * ib]));
                                x[2 * ib] = (x[2 * ia] - rtemp);
                                x[2 * ib + 1] = (x[2 * ia + 1] - itemp);
                                x[2 * ia] = (x[2 * ia] + rtemp);
                                x[2 * ia + 1] = (x[2 * ia + 1] + itemp);
                                ia += k << 1; }
                        ia = j + 1; }
                n2 >>= 1; }
        for (i = 0; i < (nx * 2); i++) {
                out_port(x[ i ], 1); } }
```

Figure 2: Program of 64-Point FFT Algorithm in ANSI C.

the same time, this mode of the SCCS operation imposes additional requirements for the generating system to create the application-specific processors HDL-models. The question arises to organizing dynamic re-allocation of the reconfigurable logic resources and replacing some running application-specific processors with others performing the same task but differing by the equipment volume. This should be done to provide an effective use of resources and the required level of multitasking. To address this challenge, it is necessary, during the program compilation, for each subprogram executed in the reconfigurable environment, to generate a number of application-specific processors HDL-models. We propose in this regard to use the Chameleon© C2HDL design tool.

In the article we consider the SCCS structure and the method of information processing in it, the partially reconfigurable FPGAs operation features, and the Chameleon© C2HDL design tool operation and features among which short generation time, desired preset level of the algorithm parallelization, automatic generation of tested and verified ASP HDL models. One of the features of the Chameleon© C2HDL design tool is its ability to generate a number of application-specific processor soft-cores executing the same algorithm differing by the amount of FPGA resources required for their implementation. For the self-configurable computer systems based on partially reconfigurable FPGAs this feature allows acquiring in every moment of its operation configuration that will provide an optimal use of its reconfigurable logic at a given level of hardware multitasking.

To estimate the benefit, we have experimented with the Chameleon© C2HDL design tool as one of the basic software means of the SCCS compiler. The SCCS hardware platform is realized on the base of the conventional personal computer running on the Windows OS and the reconfigurable environment built on the Cyclone V FPGA from Altera. Chosen for the experiment RCE subprogram represents the algorithm of the 64-point Fast Fourier Transformation in the ANSI C language. This program has been given to the input of the Chameleon© C2HDL design tool, and a set of the RTL VHDL-models of the 64-point FFT processors has been automatically generated. The experimental results have shown that the Chameleon© C2HDL design tool generates a set of FFT processors with high technical characteristics in very short time, and satisfies the basic requirements for a generating system of the SCCS to provide its effective operation.

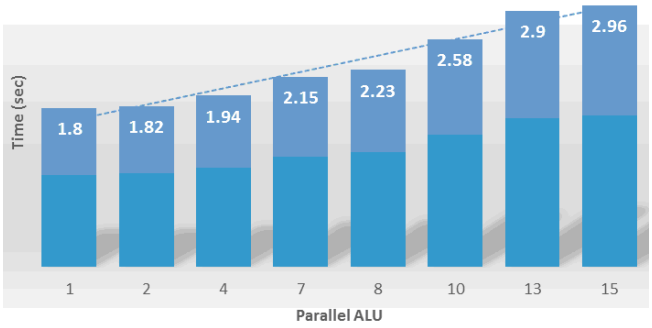


Figure 3: Time Required for 64-Point FFT Processors VHDL-Models Generation.

References

[1] A. Melnyk, V. Melnyk, "Self-Configurable FPGA-Based Computer Systems," *Advances in Electrical and Computer Engineering*, vol. 13, no. 2, pp. 33-38, 2013.

[2] V. Melnyk, V. Stepanov, Z. Sarajrech, "System of load balancing between host computer and reconfigurable accelerator," *Computer systems and components, Scientific Journal of Yuriy Fedkovych Chernivtsi National University*, vol. 3, no. 1, pp. 6-16, 2012.

[3] "A Proven EDA Solutions Provider makes all the difference". [Online]. *Available: http://www.aldec.com/en.*

[4] Xilinx Core Generator. Xilinx Inc. [Online]. *Available: http://www.xilinx.com/ise/products/coregen_overview.pdf - 2005.*

[5] Melnyk, A, Melnyk, V. "Organization of libraries of standardized and custom IP Cores for high-performance hardware accelerators", Proceedings of IV-th all-Ukrainian conference "Computer Technologies: Science and Education", Ukraine, Lutsk, 9-11 October 2009. -P. 113-117.

[6] Genest, G. "Programming an FPGA-based Super Computer Using a C-to-VHDL Compiler: DIME-C", Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference, 5-8 Aug. 2007. - P. 280 - 286.

[7] "Chameleon - the System-Level Design Solution," [Online]. *Available: http://intron-innovations.com/?p=sld_chame*

[8] ANSI-C to VHDL Compiler. [Online]. *Available: http://www.nallatech.com/FPGA-Development-Tools/dimetalk.html.*

[9] "Relogix Assembler-to-C translator," [Online]. *Available: http://www.microapl.co.uk/asm2c/*

[10] Melnyk, A., Salo, A., Klymenko, V., Tsyhylyk, L. "Chameleon - system for specialized processors high-level synthesis", Scientific-technical magazine of National Aerospace University "KhAI", Kharkiv, 2009. N5, pp. 189-195.

[11] Agility Compiler for SystemC. Electronic System Level Behavioral Design & Synthesis Datasheet. 2005. [Online]. *Available: http://www.europractice.rl.ac.uk/vendors/agility_compiler.pdf*

[12] Handel-C Language Reference Manual For DK Version 4. Celoxica Limited, 2005. - 348p.

[13] Impulse CoDeveloper C-to-FPGA Tools. [Online]. *Available: http://www.impulseaccelerated.com/products_universal.htm*

[14] Vivado Design Suite User Guide. Partial Reconfiguration. UG909 (v2015.2) June 24, 2015.

[15] UltraScale Architecture. Online. *Available: http://www.xilinx.com/products/technology/ultrascale.html*

HPC in Computational Micromechanics of Composite Materials

(Poster Summary)

Radim Blaheta, Alexej Kolcun, Ondřej Jakl, Kamil Souček, Jiří Starý, Ivan Georgiev

Institute of Geonics, Czech Academy of Sciences, Ostrava, Czech Republic
Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, Bulgaria
[blaheta, kolcun, jakl, soucek, stary]@ugn.cas.cz, ivan.georgiev@parallel.bas.bg

I. Introduction

By *micromechanics* we understand analysis of the macroscale response of materials through investigation of processes in their microstructure. Here by the macroscale, we mean the scale of applications, where we solve engineering problems involving materials. Examples could be analysis of aircraft constructions with different composite materials and analysis of rock behaviour and concrete properties in geo- and civil engineering applications. Analysis of bio-materials with many medicine applications is also rapidly developing. Different applications are distinguished by different characteristic size. At macroscale the materials mostly look as homogeneous or they are idealized as homogeneous or piecewise homogeneous. A substantial heterogeneity is hidden and appears only after more detailed zooming view into the material. This hidden heterogeneity can be called microstructure. In metals it is created by crystals and grains, in composite materials by matrix and inclusions, in concrete by gravel and mortar or iron reinforcement, in rock by mineral composition and possible grouting, etc. When the ratio between the characteristic dimensions on macro and microstructure subjects is sufficiently large, then we say that the scales are well separated. In this case, it is not possible to perform the macroscale analysis going into the microstructure details, but it is possible to analyse the macroscopic problems with the use of effective (homogenized) material properties, which are obtained by testing smaller samples of materials. In computational micromechanics, the testing of such samples means solution of boundary value problems on test domains involving the microstructure with loading provided by suitable boundary conditions.

In this work, we focus on X-ray CT image based micromechanics of geomaterials and concrete with the use of continuum mechanics and *finite element* (FE) computations of the microscale strains and stresses, see [2]. This means that basic information about the microstructure is provided by the analysis (segmentation) of 3D images of real samples. This information should be complemented by information on local material properties, i.e. material properties of the individual material constituents.

There is a strong need for high performance computing (HPC) at several levels of computational micromechanics, namely at:

- analysis of CT scans,
- high resolution finite element solution of boundary value problems,
- solution of inverse problems for determination or calibration of local material properties.

This contribution deals with the second point, i.e. solution of high resolution FE systems with tens or hundreds million *degrees of freedom* (DOF). We report about the performance of in-house solvers exploiting the *Schwarz domain decomposition method* with aggregation, c.f. [3], and outline possible future development in the area of *ultrascale computing*, which is necessary for building efficient solution methods for *inverse material identification problems*, see [4] and work in progress.

II. High resolution FE systems

In the analysis of heterogeneous materials with microstructure *(composites)* (see [2]), our test sample domain Ω is a cube with a relatively complicated microstructure. Its FE mesh is constructed from data obtained from industrial CT scanning, performed at the CT-lab of the Institute of Geonics. Two types of composites are considered in this work: A *coal-resin* geocomposites and *reinforced concrete* composites. See Tab. 1 for the main characteristics of the resulting FE problems *(benchmarks)*.

Benchmark	Discretization	Size in DOF	Data size
GEOC-2l	$257 \times 257 \times 1025$	203 100 675	33.5 GB
FIBER-3	$401 \times 401 \times 401$	193 443 603	32.2 GB

Table 1: Benchmark problems: Name, discretization, size of the resulting linear system and storage requirements.

The elastic response of a representative volume Ω is characterized by homogenized elasticity C or compliance S tensors ($S = C^{-1}$). The elasticity and compliance tensors are basically determined from the relations

$$C\langle\varepsilon\rangle = C\varepsilon_0 = \langle\sigma\rangle \quad \text{and} \quad S\langle\sigma\rangle = S\sigma_0 = \langle\varepsilon\rangle, \qquad (1)$$

respectively. Here $\langle\sigma\rangle$ and $\langle\varepsilon\rangle$ are volume averaged stresses and strains computed from the solution of elasticity problem

$$-\operatorname{div}(\sigma) = 0, \quad \sigma = C_m\varepsilon, \quad \varepsilon = (\nabla u + (\nabla u)^T)/2 \ \text{ in } \Omega, \qquad (2)$$

with boundary conditions

$$u(x) = \varepsilon_0 \cdot x \ \text{ on } \partial\Omega \quad \text{and} \quad \sigma \cdot n = \sigma_0 \cdot n \ \text{ on } \partial\Omega, \qquad (3)$$

respectively. Above, σ and ε denote stress and strain in the microstructure, C_m is the variable local elasticity tensor, u and n denote the displacement and the unit normal, respectively. The use of pure Dirichlet and pure Neumann boundary conditions allows us to get a upper and lower bounds for the upscaled elasticity tensor, see e.g. [4].

III. GEM solvers

GEM is an in-house FE software described in detail in [1], which makes use of linear tetrahedral finite elements for discretization. Arising systems of linear equations are processed by solvers based on the *preconditioned conjugate gradient* (PCG) method, with *stabilization* in the singular case [3]. PCG uses *overlapping domain decomposition preconditioners*. To solve the benchmarks, we employ two types of GEM solvers:

GEM-DD solver uses *one-level additive Schwarz* domain decomposition preconditioner with subproblems replaced by displacement decomposition incomplete factorization, see ref. in [3]. The resulting preconditioner is symmetric positive definite even for the singular case.

GEM-DD+CG solver implements *two-level Schwarz* domain decomposition preconditioning, arising from the GEM-DD above by additive involvement of a *coarse problem correction*. The coarse problem is created by a regular aggressive aggregation with 3 DoF' per aggregation. In the singular case, the coarse problem is also singular with a smaller null space containing only the rigid shifts. The coarse problem is solved only approximately by inner (not stabilized) CG method with a lower solution accuracy – relative residual accuracy $\varepsilon_0 \leq 0.01$.

IV. Computing resources

The computations were performed on two parallel platforms:

Enna 64-core NUMA *multiprocessor*, Institute of Geonics AS CR: eight octa-core Intel Xeon E7-8837 / 2.66 GHz processors; 512 GB of DDR2 RAM; CentOS 6.3, Intel Cluster Studio XE 2013.

Anselm *multicomputer* (cluster, 209 compute nodes), IT4Innovations National Supercomputing Center: two octa-core Intel E5-2665 / 2.4 GHz processors per node; 64 GB RAM per node; Infiniband QDR interconnection, fully non-blocking, fat-tree; Bullx Linux Server 6.3 (Red Hat clone), Intel Parallel Studio 13.1.

V. Computational experiments

Table 2 shows the timings of the GEM-DD+CG solver (with coarse grid problem applied) obtained for the **coal-resin geocomposite** benchmark GEOC-2l both on Enna and Anselm, and demonstrates the impact of the coarse grid size on the time of the solution. The *stopping criterion* $\|r\|/\|b\| \leq \varepsilon$, based on the relative residual accuracy, was 10^{-5}. On Enna, the best results (2483.6 s) were observed with aggregation $9 \times 9 \times 9$.

# Sd	Enna						Anselm	
	DD + CG		DD + CG		DD + CG		DD + CG	
	$9 \times 9 \times 9$		$9 \times 9 \times 18$		$9 \times 9 \times 27$		$9 \times 9 \times 27$	
	# It	T_{iter}	# It	T_{iter}	# It	T_{iter}	# It	T_{iter}
4	751	13719.0	858	15757.6	997	18518.4	997	12671.4
8	690	6237.7	800	6960.8	917	8062.9	917	5803.9
16	585	2717.4	674	4010.6	777	4815.6	777	2576.6
32	585	**2483.6**	622	2923.8	708	3452.5	708	1157.5
64					627	3637.0	627	558.8
128							652	358.5
256							631	**299.6**
512							649	333.5

Table 2: Timings of the GEOC-2l benchmark with pure Neumann boundary conditions and achieved by the GEM-DD+CG solver on the multiprocessor Enna and multicomputer Anselm: Iteration counts (# It) and wall-clock time (in seconds) for the solution time (T_{iter}) are provided now for different sizes of CG problem involved in computations and for various numbers of subdomains (# Sd).

The experiments confirm the advantage of multicomputers (systems with distributed memory) for greater number of subdomains, when the multiprocessors in general suffer from the memory-processor bandwidth contention. Thus, while on Enna the scalability fades out at about 32 cores, the turning point on Anselm is around 256 processing elements, when the small size of subdomains deteriorates the ratio between computation and communication. In absolute figures, we were able to solve the benchmark 8 times faster on Anselm than on Enna. A part of Anselm's advantage is to be credited to its newer Intel Sandy Bridge CPU architecture, which outperforms Enna's Westmere CPU in our applications by 20 - 40 % (separate test).

A bit surprising decrease of the number of iterations with increasing number of subdomains (processors) as reported in the above Tables, can be explained by the fact that smaller subdomain problems are solved more accurately in our implementation.

Anselm/GEOC-2l best time in Table 2 (299.6 s with 256 processing elements and aggregation $9 \times 9 \times 27$) was surpassed by another experiment (not shown in the table): aggregation $15 \times 15 \times 31$, 910 iterations, 512 subdomains (32 compute nodes employed), **249.8 s**.

The experiments carried out on the **fiber-reinforced concrete** FIBER-3 benchmark delivered similar results. So far, just Enna has been used for computations, confirming limited scalability on its multi-processor architecture. We observed great importance of the coarse grid and its proper dimensioning for efficient solution (computing times of GEM-DD are multiples of the computing times of GEM-DD+CG).

VI. Conclusions and future work

Micromechanics leads to large-scale problems, as illustrated by the presented benchmarks. The computational requirements can be further substantially increased in the case of an *inverse analysis for identification local material properties* (see e.g. [4]).

At the IT4Innovations National Supercomputing Center, there is a new massively parallel computer available, called Salomon. This multi-computer (cluster) has 1008 compute nodes and we plan to employ it in future experiments.

The approach described so far employs classical domain decomposition philosophy. Both facts, computational demands and availability of massively parallel computers, motivate further research in algorithms, which are efficient from the point of view of arithmetic operations and (even more important) from the point of view of communication.

In the future, we plan to test the effect of *communication avoiding* (CA) algorithms. For example, using the same ingredients as in our domain decomposition solvers, we can employ CA conjugate gradients and a deflation type implementation of the aggregation based coarse space.

Acknowledgement: This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project

(CZ.1.05/1.1.00/02.0070) and the COST NESUS project with an additional CZ MEYS LD15105 support.

References

[1] R. Blaheta, O. Jakl, R. Kohut, J. Starý: *GEM - A Platform for Advanced Mathematical Geosimulations.* In: R. Wyrzykowski et al. (eds.): PPAM 2009, Part I, LNCS 6067, Springer-Verlag, 2010, pp. 266-275.

[2] R. Blaheta et al: *Material parameter identification with parallel processing and geo-applications.* In: R. Wyrzykowski et al. (eds.): PPAM 2011, Part I, LNCS 7203, Springer-Verlag, 2012, pp. 366-375.

[3] R. Blaheta, O. Jakl, J. Starý, E. Turan: *Parallel solvers for numerical upscaling.* In: P. Manninen, P. Oster (eds.): PARA 2012, LNCS 7782, Springer-Verlag, 2013, pp. 375-386.

[4] R. Blaheta, R. Kohut, A. Kolcun, K. Souček, L. Staš, L. Vavro: *Digital image based numerical micromechanics of geocomposites with application to chemical grouting.* International Journal of Rock Mechanics and Mining Sciences 77, 2015, pp. 77-88.

List of Authors