Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

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

Jorge Barbosa
Faculdade de Engenharia da Universidade do Porto (FEUP)
Departamento de Engenharia Informatica
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
E-mail: jbarbosa@fe.up.pt

Ricardo Morla
Faculdade de Engenharia da Universidade do Porto (FEUP)
ECE department
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
E-mail: rmorla@fe.up.pt

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

ISBN: 978-84-617-2251-8

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

This document also is supported by:

Printed in Madrid — November 2014

Preface

Network for Sustainable Ultrascale Computing (NESUS)

We are very excited to present the proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014), 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 Porto (Portugal) on August 27-28, 2014.

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.
The project have already held two important meetings: one for work groups in Madrid in July and another in Oporto (Portugal) at the end of August, attended by representatives of the research groups that participate as well as Project Officers from the EU's H2020 program. By reducing duplication of work and providing a more comprehensive vision of all the researchers, this COST Action hopes to increase the value of these groups at the European level, promoting European leadership in this area of knowledge, as well as enhancing its impact on science, the economy and society.
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

November 2014

NESUS

Network for Sustainable Ultrascale Computing

COST Action IC1305 **2014 | 2018**

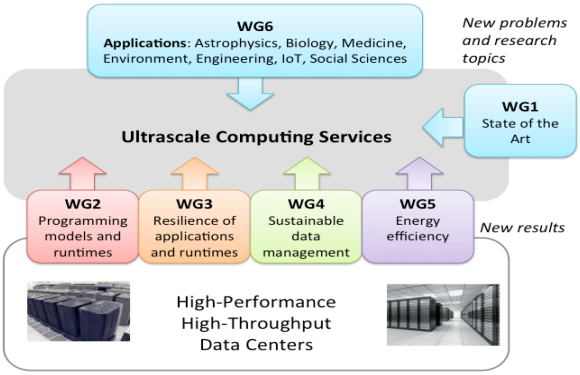
Aim

- 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)

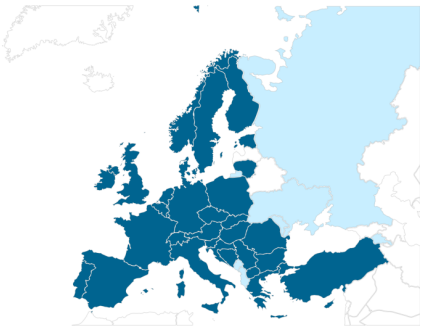
Participating countries: 45

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

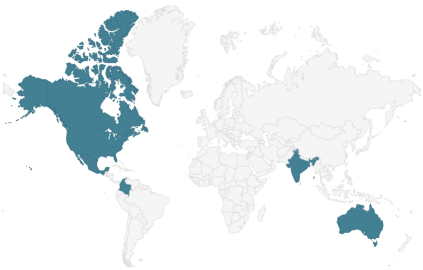
NNC countries: 6

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

WORKSHOP PROGRAM

First NESUS Workshop (NESUS 2014)

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop. • October 2014 • Vol. I, No. 1

Improving the Performance of the MPI_Allreduce Collective Operation through Rank Renaming

Juan-Antonio Rico-Gallego
University of Extremadura, Spain
jarico@unex.es

Juan-Carlos Díaz-Martín
University of Extremadura, Spain
juancarl@unex.es

Abstract

Collective operations, a key issue in the global efficiency of HPC applications, are optimized in current MPI libraries by choosing at runtime between a set of algorithms, based on platform-dependent beforehand established parameters, as the message size or the number of processes. However, with progressively more cores per node, the cost of a collective algorithm must be mainly imputed to process-to-processor mapping, because its decisive influence over the network traffic. Hierarchical design of collective algorithms pursuits to minimize the data movement through the slowest communication channels of the multi-core cluster. Nevertheless, the hierarchical implementation of some collectives becomes inefficient, and even impracticable, due to the operation definition itself. This paper proposes a new approach that departs from a frequently found regular mapping, either sequential or round-robin. While keeping the mapping, the rank assignation to the processes is temporarily changed prior to the execution of the collective algorithm. The new assignation makes the communication pattern to adapt to the communication channels hierarchy. We explore this technique for the Ring algorithm when used in the well-known MPI_Allreduce collective, and discuss the obtained performance results. Extensions to other algorithms and collective operations are proposed.

Keywords MPI Collectives, Parallel Algorithms, Message Passing Interface, Multi-core Clusters

I. Introduction

MPI [1] collective functions involve a group of processes communicating by message passing in an isolated context, known as *communicator*. Each process of a communicator is identified by its *rank*, an integer number ranging from 0 to $P-1$, where P is the size of the communicator. The optimisation of collectives is a key issue in HPC applications. A collective operation can be executed by different algorithms, each suitable for a given network technology, communicator size, message size, etc. For example, in the MPICH library [2], the implementation of *MPI_Allreduce* uses two algorithms for medium and large messages when the number of processes is a power of two, namely *Recursive Doubling* and *Ring*. The switch from the first to the second algorithm is done at execution time, with platform-dependent beforehand established message size and process number thresholds.

Current parallel systems are composed of multi-core nodes connected by a high performance network. The communication cost between two MPI ranks depends on their location, being lower if they share memory, and higher if they are in different nodes. Therefore the performance of an application depends on the assignation of the ranks to the processors of the cluster (mapping). In general, two types of mapping cover the necessities of most applications: *sequential* and *round-robin*. In the sequential mapping, ranks bind to processors so that a domain is completed (e.g. socket or node) before moving to the next domain. In round-robin, ranks are bound to domains by rotating on the existing domains.

Mapping affects to the performance of the underlying algorithms of collective operations. Interestingly, a given mapping may favour an algorithm and, at the same time, being harmful to another algorithm, not matter if both are used in the implementation of the same collective. For example, in the implementation of the *allreduce* operation in MPICH, referred above, the Recursive Doubling algorithm shows a better performance when the mapping is round-robin, while the Ring algorithm runs faster under the sequential mapping.

An approach to the issue of collectives performance is building algorithms that are aware of the different capacities of the available communication channels, as shared memory and network. These algorithms, known as *hierarchical*, stand on minimizing the communications through the slower channels, but the implementation for some collectives as *allgather* is not as effective as expected, even impracticable, and hence it is not provided in well-known MPI libraries as Open MPI [3].

This paper describes a new approach to the optimization of collectives in multi-core clusters. The goal is to obtain the best possible communication throughput. For instance, in the Ring algorithm, the communication takes place between consecutive ranks. If consecutive ranks are mapped to different nodes, all the communications progress through the network. Instead, a schedule of consecutive ranks to processes placed in the same multi-core node favours the much more efficient shared memory communication. Our method is based on a temporal reassignment of ranks. That neither modifies the algorithm nor the physical mapping. Instead, it is carried out by means of a transformation function prior to the execution of the algorithm. The function is simple and efficient, and converts a sequential mapping to round-robin and vice versa only during the execution of the algorithm.

This paper focuses on the *Ring* algorithm in the context of the *allreduce* operation. Besides, the methodology described is directly applicable to other algorithms used in the implementation of MPI

First NESUS Workshop. • October 2014 • Vol. I, No. 1

collectives. Platform considered is characterized by P, the number of processors (or processes involved in the operation), and M, the number of nodes in the cluster. $Q = P/M$ is the number of processors per node. Two channels are considered in the system, shared memory and network, with different performance. The study is conducted under two different mappings, sequential and round-robin, under the assumption of a homogeneously distributed number of processes over the nodes of the system. A hierarchical implementation of the algorithm is examined as well. The attained cost reduction depends on the number of nodes and the number of processes per node. In the used experimental platform, even with a small number of processes and nodes, the improvement reaches up to $2\times$ for long messages.

With respect to the structure of this article, following this introduction, section II reviews proposals of optimization of collective operations in a broad range of platforms. Section III studies the allreduce *Ring* algorithm in multi-core clusters based on the incoming mapping, and also a hierarchical allreduce implementation. The section exposes our proposal to improve the performance of the algorithm and the section IV outlines extensions to cases not covered in this paper. Section V shows the obtained performance figures, and section VI concludes the paper.

II. Related Work

MPI collectives performance is a key issue in high performance computing applications, and significant work has been invested in their design and optimization. Collectives in the MPI standard can be implemented from several of a set of algorithms available.

For instance, *MPI_Allreduce* can be implemented using the *Recursive Doubling* algorithm, that improves the latency when P is a power of two for small messages because is optimum with regard to the number of stages, however, the *Ring* algorithm performs better for larger messages. Both algorithms are also used in the implementation of *MPI_Allgather*, for which, in addition, other proposed algorithms improve the performance when requirements related to message size, process number or hardware and network technologies are met. *Bruck* algorithm [4] is more efficient for very short messages, even though it needs additional temporal memory, *Neighbour Exchange* algorithm in [5] requires half the stages than the Ring algorithm when the number of processes is even, and it exploits the piggy-backing feature of the TCP/IP protocols, as well as the *Dissemination* algorithm, proposed in [6], based on processes pairwise exchange of messages. Also related to the improvement of the performance by exploiting some networks capabilities, Mamidala et al. [7] evaluate the RDMA capacity for allowing concurrent direct memory access by the processes either in the same or different node of a multi-core cluster. Ma et al. [8] discuss the intra-node processes direct copy communication through shared memory by using the capacities of the operating system, and in [9] evaluate its impact in the collectives operations. Kielmann et al. [10] focus on the optimization of collective communications for clustered wide area systems.

The use of several algorithms in the same collective, based on system dependant beforehand established thresholds for message size and number of processes is shown by Thakur et al. in work [11] in a monoprocessor cluster of workstations. This approach has been adopted by the MPICH library, and it is available in the Open MPI library through its Modular Component Architecture [12]. Vadhiyar et al. [13] evaluate such improvement of performance through previous executed series of experiments conducted in an specific platform.

Multi-core clusters introduce a new actor in the scene. Performance becomes dependant on the effective use of the different communication channels. Hierarchical algorithms are specifically built to minimize the use of slower communication channels, and usually execute in several stages [14]. The process group splits in subgroups, with a local root per subgroup. Processes in a subgroup communicate through the faster communication channel, usually shared memory, hence, a subgroup is assigned to a node in the system. The application of these kind of algorithms to several implementations of the MPI standard and hardware platforms is extensively evaluated in [15], [16] and [17]. Based on analytical communication models, Karonis et al. [18] demonstrated the advantages of a multilevel topology-aware implementation of algorithms with respect to optimal *plain* algorithms. Sack and Gropp [19] show that a suboptimal algorithm in terms of inter-domain communications may produce lesser congestion that an optimal algorithm, and therefore to achieve a faster execution.

Former approximations adapt algorithms to the underlying communication capabilities. An inverse approach is to improve the performance through the calculation of the best layout of the processes over the processors of the cluster. Kravtsov et al. [20] define and propose an efficient solution to the *topology-aware co-allocation problem*, and Jeannot et al. proposes the *TreeMatch* algorithm in [21] applied to multi-core clusters. The challenge is optimally mapping the graph that defines the communication necessities of an application to the graph of the available resources. The solution can be applied to MPI collective operations, provided that they are built as a set of point-to-point transmissions [22]. Algorithms to automatically build the optimal *distance-aware collective communication topology*, based on the distance information between processes, are proposed in [23]. The results are applied to *Binomial Tree broadcast* and *Ring allgather* collectives.

III. MPI_Allreduce Ring Algorithm

In the *MPI_Allreduce* collective operation every process contributes with a buffer of size m bytes and gets in the output buffer the result of applying an specified operation to all the P processes buffers.

Ring algorithm implementation of the allreduce collective first copies data from the input buffer to the output buffer. Next, it operates on the output buffer in two phases: *computation* and *distribution*. The algorithm does not preserve order of operations. As a consequence, it can not be used with non commutative operations.

The *computation phase* is done in $P-1$ stages. The data buffer is divided up in segments of size m/P. In each stage k, from $k=0$, a process p sends its $p-k$ segment to process $p+1$, and next receives in a temporary buffer a segment from process $p-1$, that operates with local $p-k-1$ segment, with wraparounds. The operated segment in each process will be sent in the next stage. After $P-1$ stages, each process p has a full operated segment in the $p+1$ position of the output buffer.

Distribution phase performs an allgather to distribute these segments between processes also using a Ring algorithm. The algorithm operates in $P-1$ stages. All processes contribute with an m/P bytes segment at offset $p+1$ and receive P segments ordered by rank, for

First NESUS Workshop. • October 2014 • Vol. I, No. 1

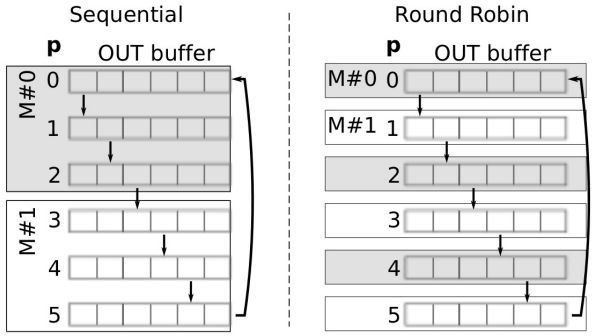


Figure 1: Representation of the communications in the stages of the Allreduce Ring (both computation and distribution phases) algorithm when processes are sequentially and round robin mapped, in a machine with $P = 6$ processes and $M = 2$ nodes.

a total of m bytes. In each stage, process p sends to process $p+1$ the m/P bytes received in the previous stage from process $p-1$, with wraparound.

Figure 1 represents the transmissions between processes in a machine with $P = 6$ processes and $M = 2$ nodes under both sequential (left) and round robin (right) mapping. In the Ring algorithm sequential mapping minimizes the point-to-point transmissions across the network. The first process of each node receives from the former node, and the last process sends to the next node. The rest of transmissions take place in shared memory and they progress in parallel with a total of M inter-node flows. Nevertheless, when the incoming mapping is round-robin $M \times Q$ inter-node transmissions take place at a time, giving rise to a much higher contention that degrades the communication. In networks as Ethernet, for instance, the contention may lead to a performance breakdown that grows with the number of simultaneous transfers. In section V, we evaluate the performance of Ring algorithm with sequential and round-robin mappings in a cluster based on Infiniband.

The fact is that the layout of the processes over the processors has a great impact in the effective cost. In the allreduce, both phases, computation and distribution, communicate each rank with the nearest next and previous rank numbers, hence, consecutive ranks must run in the same node in order to increase the data transmissions inside a node and minimize the network contention. The next section explores the design of algorithms which take into account the mapping of processes on the physical hierarchy of communications.

III.1 Hierarchical Allreduce Algorithm

An algorithm can be designed to minimize the data movement through the slowest communication channels in a multi-core cluster for the allreduce collective. For example, the implementation found in *Open MPI* library is composed of three phases, that progress sequentially.

The first phase performs a *reduce* operation local to each node in the system. One process per node, called local root, obtains the full operated Q segments in the output buffer. The second phase performs an inter-node allreduce between local roots. The number of processes running allreduce decreases with respect to a simple allreduce operation in section III, from P to M, but the size of messages contributed by each process increases from m/P to $m \cdot Q/P$. Thus, the amount of data transmitted through the network is the same as the allreduce Ring algorithm with sequential mapping. Nevertheless, this hierarchical design of the allgather minimizes the network contention regardless of the initial process mapping. Finally, in the third phase, the local root process broadcasts its resulting buffer to the rest of the processes in the same node.

Additional communicators must be created to perform collectives inside each node, and the inter-node allreduce between local roots.

III.2 Allreduce Mapping Transformation at Run-Time

Under awareness of a regular mapping, such as sequential or round-robin, the programmer would be in the position of exploiting that knowledge to increase the algorithm performance.

Necessity of minimize network communication advises a physical rearrangement of the processes that guarantees a sequential mapping before starting Ring algorithm. In practice, however, physically moving the processes conveys an excessive latency and cached data invalidation penalties. We propose instead a mere previous logical rearrangement of the ranks, a solution that is applied dynamically and efficiently. Logical renaming of processes ranks can be applied to both computation and distribution phases. The new algorithm is denoted as Ring*.

Let be a rank set $R = \{r_0, r_1, \ldots, r_{P-1}\}$ assigned to the P processes of a communicator following a Round Robin mapping. For simplicity and without loss of generality, we define $r_p = p$. The set R can be transformed into another set $S = \{s_0, s_1, \ldots, s_{P-1}\}$, which shows a sequential mapping, with $s_p = f_{SQ}(p)$. The *transformation function* f_{SQ} is defined as:

$$f_{SQ}(p) = ((p \times Q)\,\%P) + \lfloor p \,/\, M \rfloor \tag{1}$$

The number of nodes M must be known and the processes must be homogeneously distributed between nodes, i.e., the number of processes per node (Q) must be constant. See section IV for explanations about extensions to irregular mappings.

Similarly, a rank set $S = \{s_0, s_1, \ldots, s_{P-1}\}$ with $s_p = p$ and a sequential mapping can be transformed into a set $R = \{r_0, r_1, \ldots, r_{P-1}\}$, which shows a round robin mapping, with $r_p = f_{RR}(p)$. The transformation function f_{RR} (inverse of f_{SQ}) is defined as:

$$f_{RR}(p) = ((p \times M)\,\%P) + \lfloor p \,/\, Q \rfloor \tag{2}$$

In the allreduce computation phase, renaming of processes is applied prior to the execution of the Ring algorithm. A process with rank p behaves as a process with rank $f_{SQ}(p)$, applying the definition in (1). Then, in the stage k, a process sends the segment $f_{SQ}(p) - k$ to process behaving as $f_{SQ}(p)+1$, calculated as $f_{RR}\left(f_{SQ}(p)+1\right)$. Next, it receives a segment from process with rank $f_{RR}\left(f_{SQ}(p)-1\right)$ in a temporary buffer, that operates with local segment $f_{SQ}(p) - k - 1$.

For instance, process $p = 2$ in Figure 2 behaves as $f_{SQ}(2) = 1$. In the first stage $k = 0$, it sends the segment $f_{SQ}(2) - k = 1$ to process behaving as the rank $f_{SQ}(2)+1$, which is calculated as $f_{RR}\left(f_{SQ}(2)+1\right) = f_{RR}(2) = 4$, and receives from

First NESUS Workshop. • October 2014 • Vol. I, No. 1

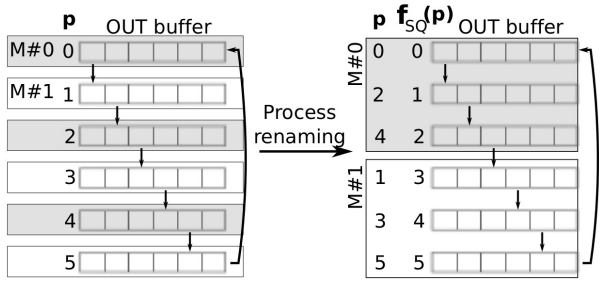


Figure 2: Allreduce Ring algorithm computation phase, stage $k = 0$, with round-robin mapping, $P = 6$ and $M = 2$. The buffer total size is divided up in P segments. Processes renaming through the transformation functions changes the mapping from round-robin to sequential before starting.

$f_{RR}\left(f_{SQ}(2) - 1\right) = f_{RR}(0) = 0$ a segment to be operated with the local segment number $f_{SQ}(2) - k - 1 = 0$.

Again, allreduce distribution phase requires renaming of processes from p into $f_{SQ}(p)$ prior to the execution of the regular Ring allgather algorithm. The operating principle is the same as the computation phase.

IV. Extensions of the Method

Our approach consists of departing from the a priori knowledge of a layout with regular mapping and keeps the original algorithm after having switched to another regular mapping, much more favourable in terms of performance. Such mapping information could be available through the processes manager module of the particular MPI implementation.

The transformation functions can be applied to algorithms with similar communication patterns to the Ring algorithm. For instance, *Neighbour Exchange* and *Binomial Tree* perform better when processes are sequentially mapped. Other algorithms have opposite requirements, such as *Recursive Doubling* and *Dissemination* algorithms, better suited to initial round robin mapping, because the distance between rank numbers communicating exponentially grows in each stage. The mapping needs to change from sequential to round robin, through the inverse application of the transformation functions.

The above-mentioned algorithms are used in a wide variety of collectives operations defined in the MPI standard, as *Broadcast*, *Scatter*, *Allgather*, etc. proving the method as highly generic.

Nevertheless, we can not always make assumptions about the deployment of the ranks over the cluster, all the more so as this layout may change with the creation of new communicators at run time, that could assign different ranks to the processes. On that case, with non-regular mappings, each rank involved in the collective operation will need to have information about the layout of all the ranks in the communicator. Resource requirements for that information are under study by the authors.

Performance measurements in clusters with other network technologies, such as Ethernet, confirms the expected results, with an increase in the difference of performance between mappings that is proportional to the difference in bandwidth capabilities between the channels.

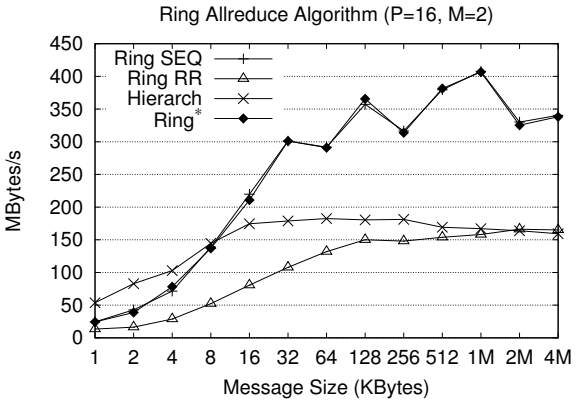


Figure 3: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 2$ nodes and $Q = 8$ processes by node, for a total of $P = 16$ processes.*

V. Performance Evaluation

The experimental platform used, named *Fermi*, is composed of eight nodes connected by a QDR Infiniband network. Each node has two 2.27 GHz Quad core Intel Xeon E5520 processors, with 8MB of shared L3 cache size, making a total of eight cores per node. The operating system is Linux 2.6.32. We used IMB (Intel MPI Benchmark), version 3.2, to obtain the latency data. Bandwidth is calculated as the message size divided by the latency, and showed in figures for the sake of clarity. A high number of iterations are executed for each collective algorithm and mean time is taken. IMB runs on Open MPI 1.8, the library that provides the allreduce algorithms through its *Tuned* and *Hierarch* collective components. Nonetheless it should be noted that MVAPICH2 yields similar results, as well as MPICH, which has been tested in Ethernet networks.

The allreduce Ring algorithm performance is plotted in Figures 3 to 5. Figures represent the bandwidth of sequential (SEQ) and round robin (RR) mappings, as well as the *Ring** algorithm and the hierarchical implementation of the collective operation, for increasing number of nodes (M), with $Q = 8$. The difference in bandwidth between sequential and *Ring** algorithm with respect to less favourable round robin mapping is nearly to $2\times$, for all the range of messages. *Ring** overload to the Ring algorithm is very low, because it is only attributable to the execution of transformation functions.

Hierarchical implementation of the allreduce algorithm leads to a higher performance than round robin, but it degrades with the size of the message because phases must progress sequentially. Performance depends as well on the algorithms used in each phase. In this paper we use the *binomial tree* algorithm for the *Reduce* and *Broadcast* algorithms in the phases 1 and 3, and allreduce *Ring* algorithm in the inter-node phase 2. This configuration outperforms even the allreduce Ring algorithm for short and medium messages.

In Figure 6 we plot the relative mean bandwidth (measured along the whole range of messages plotted in figures) between different cases for a constant number of nodes ($M = 8$), and a growing number of processes per node. Note that the difference between the sequential and the round-robin mapping grows with Q. As expected,

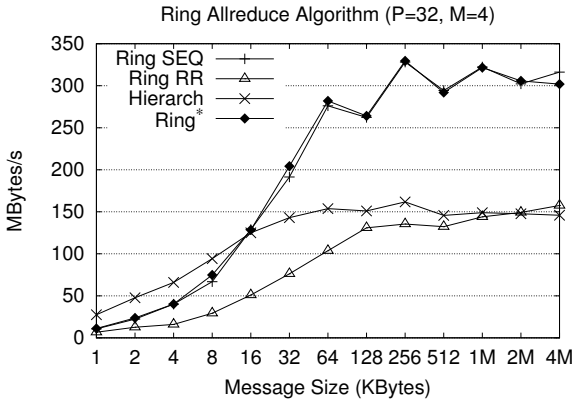


Figure 4: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 4$ nodes and $Q = 8$ processes by node, for a total of $P = 32$ processes.*

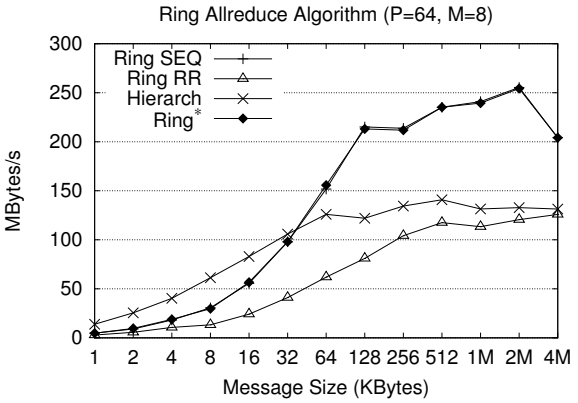


Figure 5: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 8$ nodes and $Q = 8$ processes by node, for a total of $P = 64$ processes.*

the difference between sequential and *Ring** remains constant, and denotes a minimal overload. Respect to the hierarchical case, the difference is constant for that small number of nodes.

It can be observed in Figure 3 that the change of mapping of the *Ring** algorithm shows an improvement with respect to the round robin mapping even in a minimal configuration with only $M = 2$ nodes.

VI. Conclusions

The performance of MPI collective algorithms in multi-core clusters highly depends on the deployment of the processes on the processors of the system. These algorithms usually establish a communication pattern between ranks that, if under specific regular mappings, use the communication resources effectively, other mappings significantly worsen their performance. The hierarchical design pursues the optimal use of the system available communication channels, regardless of the process mapping, but they are only efficient in a limited subset of collectives operations.

This paper proposes a more generic approach, whose goal is to adapt the mapping of processes to the communication pattern of the collective algorithm in run-time to reduce network traffic and contention. Such a switch does not require process migration, but a renaming of the processes ranks prior to the execution of the original algorithm.

Performance improvements of *MPI_Allreduce* collective is evaluated when built upon the *Ring* algorithm, which performs better when processes are mapped sequentially. The figures show that the processes renaming adds a low impact upon the cost of the original algorithm. Results are also compared to the hierarchical implementation of the collective.

Our approach can be applied to other algorithms commonly used in MPI collective operations, as the Recursive Doubling, Neighbour Exchange, Dissemination or Binomial Tree, with different incoming mapping necessities, covering a broad range of communication patterns. In addition, the paper discusses extensions to cover non-regular mapping of processes and other collective operations.

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)', and by the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

References

[1] MPI Forum. MPI: A Message-Passing Interface Standard, Version 3.0., September 2012.

[2] MPICH. MPICH High Performance Portable MPI Implementation, 2013.

[3] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[4] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 8(11):1143–1156, 1997.

[5] Jing Chen, Linbo Zhang, Yunquan Zhang, and Wei Yuan. Performance evaluation of allgather algorithms on terascale linux cluster with fast ethernet. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, HPCASIA '05, pages 437–, Washington, DC, USA, 2005. IEEE Computer Society.

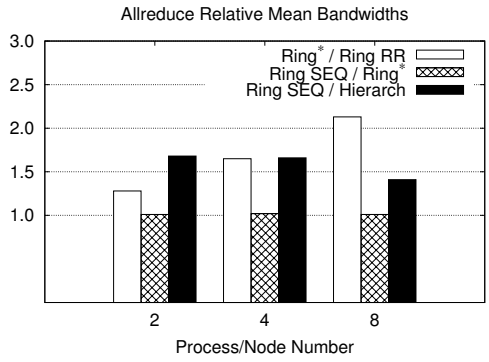


Figure 6: Allreduce relative mean bandwidths (calculated for the whole range of messages) of the Ring algorithm with respect to the Ring with round robin mapping, of Ring with sequential mapping with respect to Ring*, and of Ring* with respect to the hierarchical implementation. All the tests are executed with $M = 8$ nodes and $Q = 8$ processes by node, for a total of $P = 64$ processes.*

[6] Gregory D. Benson, Cho wai Chu, Qing Huang, and Sadik G. Caglar. A comparison of mpich allgather algorithms on switched networks. In *In Proceedings of the 10th EuroPVM/MPI 2003 Conference*, pages 335–343. Springer, 2003.

[7] Amith R. Mamidala, Abhinav Vishnu, and Dhabaleswar K. Panda. Efficient shared memory and rdma based design for mpi_allgather over infiniband. In *Proceedings of the 13th European PVM/MPI User's Group conference on Recent advances in parallel virtual machine and message passing interface*, EuroPVM/MPI'06, pages 66–75, Berlin, Heidelberg, 2006. Springer-Verlag.

[8] Teng Ma, G. Bosilca, A. Bouteiller, and J.J. Dongarra. Hierknem: An adaptive framework for kernel-assisted and topology-aware collective communications on many-core clusters. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 970–982, 2012.

[9] Teng Ma, G. Bosilca, A. Bouteiller, B. Goglin, J.M. Squyres, and J.J. Dongarra. Kernel assisted collective intra-node mpi communication among multi-core and many-core cpus. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 532–541, Sept 2011.

[10] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. Magpie: Mpi's collective communication operations for clustered wide area systems. *SIGPLAN Not.*, 34(8):131–140, May 1999.

[11] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[12] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[13] S.S. Vadhiyar, G.E. Fagg, and J. Dongarra. Automatically tuned collective communications. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 3–3, Nov 2000.

[14] Meng-Shiou Wu, R.A. Kendall, and K. Wright. Optimizing collective communications on smp clusters. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 399–407, 2005.

[15] Hao Zhu, David Goodell, William Gropp, and Rajeev Thakur. Hierarchical collectives in mpich2. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 325–326, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] A.R. Mamidala, R. Kumar, D. De, and D.K. Panda. Mpi collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 130–137, 2008.

[17] Richard L. Graham and Galen Shipman. Mpi support for multicore architectures: Optimized shared memory collectives. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 130–140, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] Nicholas T. Karonis, Bronis R. de Supinski, Ian T. Foster, William Gropp, and Ewing L. Lusk. A multilevel approach to topology-aware collective operations in computational grids. *CoRR*, cs.DC/0206038, 2002.

[19] Paul Sack and William Gropp. Faster topology-aware collective algorithms through non-minimal communication. *SIGPLAN Not.*, 47(8):45–54, February 2012.

[20] Valentin Kravtsov, Martin Swain, Uri Dubin, Werner Dubitzky, and Assaf Schuster. A fast and efficient algorithm for topology-aware coallocation. In *Proceedings of the 8th international conference on Computational Science, Part I*, ICCS '08, pages 274–283, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] E. Jeannot, G. Mercier, and F. Tessier. Process placement in multicore clusters:algorithmic issues and practical techniques. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4):993–1002, April 2014.

[22] Jin Zhang, Jidong Zhai, Wenguang Chen, and Weimin Zheng. Process mapping for mpi collective communications. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 81–92, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] Teng Ma, T. Herault, G. Bosilca, and J.J. Dongarra. Process distance-aware adaptive mpi collective communications. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 196–204, 2011.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop. October 2014

A Workflow-oriented Language for Scalable Data Analytics

Fabrizio Marozzo, Domenico Talia, Paolo Trunfio

DIMES - University of Calabria, Italy
fmarozzo@dimes.unical.it, talia@dimes.unical.it, trunfio@dimes.unical.it

Abstract

Data in digital repositories are everyday more and more massive and distributed. Therefore analyzing them requires efficient data analysis techniques and scalable storage and computing platforms. Cloud computing infrastructures offer an effective support for addressing both the computational and data storage needs of big data mining and parallel knowledge discovery applications. In fact, complex data mining tasks involve data- and compute-intensive algorithms that require large and efficient storage facilities together with high performance processors to get results in acceptable times. In this paper we describe a Data Mining Cloud Framework (DMCF) designed for developing and executing distributed data analytics applications as workflows of services. We describe also a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF. We finally present a data analysis application developed with JS4Cloud, and the scalability achieved executing it on DMCF.

Keywords Cloud computing, Data analytics, Workflows, JS4Cloud

I. Introduction

Cloud computing provides elastic services, high performance and scalable data storage to a large and everyday increasing number of users [1]. Clouds enlarged the offer of distributed computing systems by providing advanced Internet services that complement and complete functionalities of distributed computing provided by the Web, Grid computing and peer-to-peer networks. In fact, Cloud computing systems provide large-scale infrastructures for complex high-performance applications. Most of those applications use big data repositories and needs to access and analyze them to extract useful information.

Big data is a new and over-used term that refers to massive, heterogeneous, and often unstructured digital content that is difficult to process using traditional data management tools and techniques. The term includes the complexity and variety of data and data types, real-time data collection and processing needs, and the value that can be obtained by smart analytics. Advanced data mining techniques and associated tools can help extract information from large, complex datasets that are useful in making informed decisions in many business and scientific applications including advertising, market sales, social studies, bioinformatics, and high-energy physics. Combining big data analytics and knowledge discovery techniques with scalable computing systems will produce new insights in a shorter time [5].

Although a few cloud-based analytics platforms are available today, current research work foresees that they will become common within a few years. Some current solutions are open source systems such as Apache Hadoop and SciDB, while others are proprietary solutions provided by companies such as Google, IBM, EMC, BigML, Splunk Storm, Kognitio, and InsightsOne. As more such platforms emerge, researchers will port increasingly powerful data mining programming tools and strategies to the cloud to exploit complex and flexible software models such as the distributed workflow paradigm.

The growing use of service-oriented computing could accelerate this trend. Developers and researchers can adopt the software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) models to implement big data analytics solutions in the cloud. In such a way, data mining tasks and knowledge discovery applications can be offered as high-level services on Clouds. This approach creates a new way to delivery data analysis software that is called data analytics as a service (DAaaS).

Here we describe a Data Mining Cloud Framework (DMCF) that we developed according to this approach. In DMCF, data analysis workflows can be designed through visual programming, which is a very effective design approach for high-level users, e.g. domain-expert analysts having a limited understanding of programming. Recently, we extended the DMCF system to support also script-based data analysis workflows, as an additional and more flexible programming interface for skilled users. To this end, in [4] we introduced a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF.

II. Data Mining Cloud Framework

The DMCF has been designed to be implemented on different Cloud systems. However, a first implementation of the framework has been carried out on the Windows Azure cloud platform and has been evaluated through a set of data analysis applications executed on a Microsoft Cloud data center. The remainder of the section describes system architecture, application execution, user interface, and visual workflow programming.

II.1 System architecture

The architecture includes different kinds of 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 results evaluation.

- The *Data Table*, *Tool Table* and *Task Table* that contain metadata information associated with data, tools, and tasks.
- The *Task Queue* that manages the tasks to be executed.

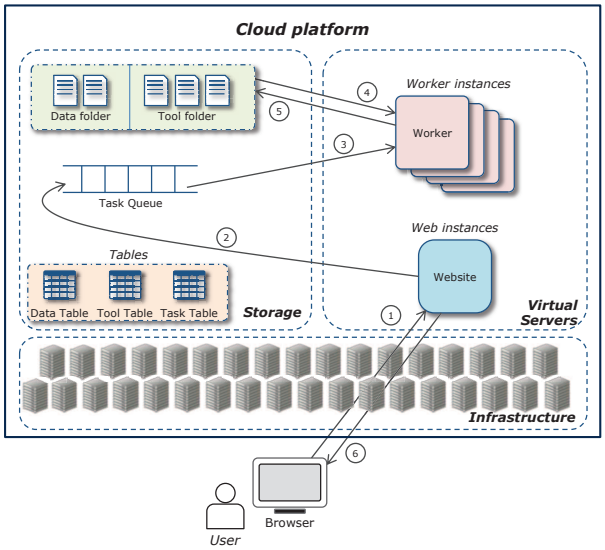


Figure 1: System architecture and application execution steps.

The virtual machines components are:

- A pool of *Worker instances*, which is in charge of executing the data mining tasks submitted by users.
- A pool of *Web instances* host the Website, by allowing users to submit, monitor the execution, and access the results of their data mining tasks.

The Website is the user interface to three functionalities: i) *App submission*, which allows users to submit single-task, parameter sweeping, or workflow-based applications; ii) *App monitoring*, which is used to monitor the status and access results of the submitted applications; iii) *Data/Tool management*, which allows users to manage input/output data and tools.

II.2 Applications execution

Figure 1 shows the main steps carried out for designing and executing a knowledge discovery application:

1. A user accesses the Website and designs the application (either single-task, parameter sweeping, or workflow-based) through a Web-based interface.
2. After application submission, the system creates a set of tasks and inserts them into the Task Queue on the basis of the application.
3. Each idle Worker picks a task from the Task Queue, and concurrently executes it.
4. Each Worker gets the input dataset from the location specified by the application. To this end, a file transfer is performed from the Data Folder where the dataset is located, to the local storage of the Worker.
5. After task completion, each Worker puts the result 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 the type of application submitted by a user. In the case of a single-task application, just one data mining task is inserted into the Task Queue. If the user submits a parameter sweeping application, the set of tasks corresponding to the combinations of the input parameters values are executed in parallel. If a complex workflow-based application must be executed, the set of tasks created depends on how many data mining tools are invoked within the workflow. Initially, only the workflow tasks without dependencies are inserted into the Task Queue.

II.3 User interface

The App submission section of the Website is composed of two main parts: one pane for composing and running both single-task and parameter-sweeping applications and another pane for programming and executing workflow-based knowledge discovery applications. As an example, Figure 2 shows a screenshot of the App submission section, taken during the execution of a parameter-sweeping application.

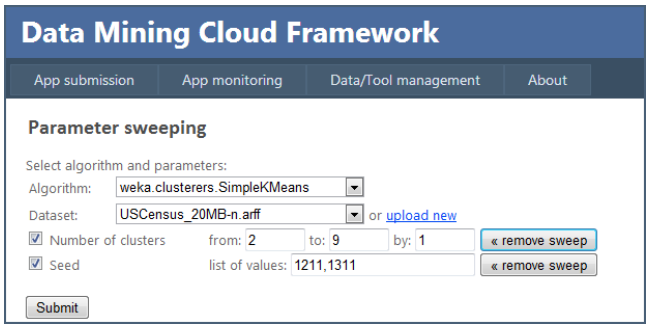


Figure 2: Screenshot of the App submission section.

Users can monitor the status of each single task through the App monitoring section, as shown in Figure 3. For each task, the current status (submitted, running, done or failed) and status update time are shown. Moreover, for each task that has completed its execution, two links are enabled: the first one (Stat) gives access to a file containing some statistics about the amount of resources consumed by the task; the second one (Result) visualizes the task result.

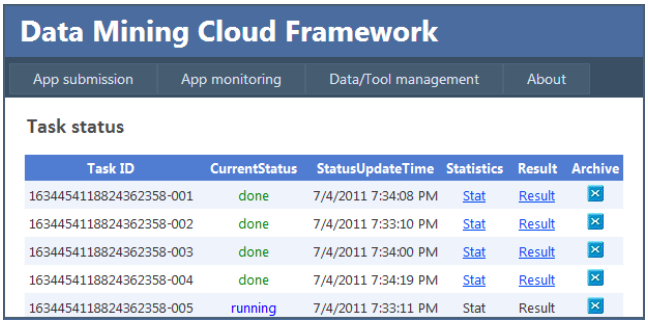


Figure 3: Screenshot of the App monitoring section.

II.4 Visual workflow programming

The DMCF includes a visual programming interface and its services to support the composition and execution of workflow-based knowledge discovery applications. Workflows provide a paradigm that may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. In data-driven discovery processes, knowledge discovery workflows can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories.

Visual workflows in DMCF are directed acyclic graphs whose nodes represent resources and whose edges represent the dependencies among the resources. Workflows include two types of nodes:

- *Data* node, which represents an input or output data element. Two subtypes exist: Dataset, which represents a data collection, and Model, which represents a model generated by a data analysis tool (e.g., a decision tree).
- *Tool* node, which represents a tool performing any kind of operation that can be applied to a data node (filtering, splitting, data mining, etc.).

The nodes can be connected with each other through direct edges, establishing specific dependency relationships among them. When an edge is being created between two nodes, a label is automatically attached to it representing the kind of relationship between the two nodes. Data and Tool nodes can be added to the workflow singularly or in array form. A data array is an ordered collection of input/output data elements, while a tool array represents multiple instances of the same tool.

Figure 4 shows a data mining workflow composed of several sequential and parallel steps as an example for presenting the main features of the visual programming interface of the DMCF [3]. The example workflow analyzes a dataset by using several instances of a classification algorithm that run in parallel on several cloud servers.

III. Script-based workflow programming

JS4Cloud (JavaScript for Cloud) is a JavaScript-based language for programming data analysis workflows [4]. The Web interface of DMCF allows to design and execute workflows programmed by the JS4Cloud language, by providing an environment similar to that used to develop visual workflows in the same framework.

The main benefits of JS4Cloud are: *i*) it is based on a well known scripting language, so that users do not have to learn a new programming language from scratch; *ii*) it implements a data-driven task parallelism that automatically spawns ready-to-run tasks to the available Cloud resources; *iii*) it exploits implicit parallelism so application workflows can be programmed in a totally sequential way.

Two key programming abstractions in JS4Cloud are *Data* and *Tool* elements:

- *Data* elements denote input files or storage elements, or output files or stored elements.
- *Tool* elements denote algorithms or software tools.

For each Data and Tool element included in a JS4Cloud workflow, an associated descriptor, expressed in JSON format, will be included in the environment of the user who is developing the workflow.

A Tool descriptor includes a reference to its executable, the required libraries, and the list of input and output parameters. Each parameter is characterized by name, description, type, and can be mandatory or optional. The JSON descriptor of a new tool is created automatically through a guided procedure provided by DMCF, which allows users to specify all the needed information for invoking the tool (executable, input and output parameters, etc.).

Similarly, a Data descriptor contains information to access an input or output file, including its identifier, location, and format. Differently from Tool descriptors, Data descriptors can also be created dynamically as a result of a task operation during the execution of a JS4Cloud script. For example, if a workflow W reads a dataset D_i and creates (writes) a new dataset D_j, only D_i's descriptor will be present in the environment before W's execution, whereas D_j's descriptor will be created at runtime.

Another key element in JS4Cloud is the *task* concept, which represents the unit of parallelism in our model. A task is a Tool, invoked from the script code, which is intended to run in parallel with other tasks on a set of Cloud resources.

According to this approach, JS4Cloud implements *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. A task T_j does not depend on a task T_i belonging to the same workflow (with $i \neq j$), if T_j during its execution does not read any data element created by T_i.

III.1 JS4Cloud functions

JS4Cloud extends JavaScript with three additional functionalities, implemented by the set of functions listed in Table 1:

- *Data Access*, for accessing a data element stored in the Cloud;
- *Data Definition*: to define a new data element that will be created at runtime as a result of a tool execution;
- *Tool Execution*: to invoke the execution of a tool available in the Cloud.

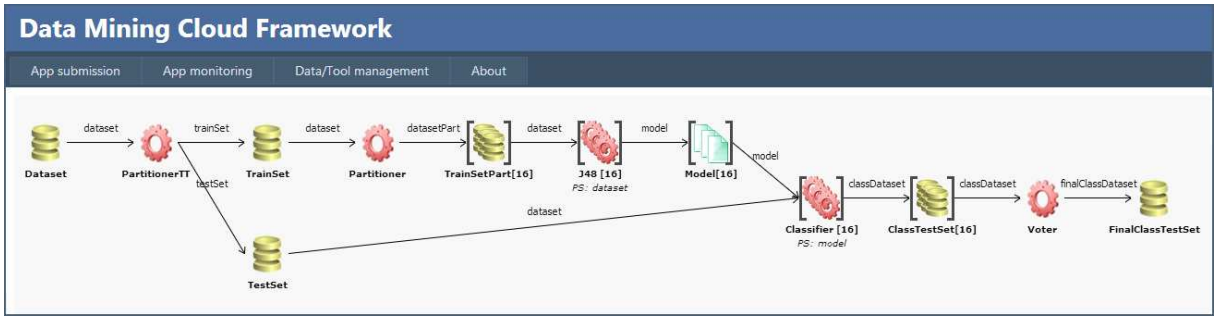


Figure 4: A visual workflow for parallel classification.

Table 1: JS4Cloud functions.

Functionality	*Function*	*Description*
Data Access	Data.get(<*dataName*>);	Returns a reference to the data element with the provided name.
	Data.get(new RegExp(<*regular expression*>));	Returns an array of references to the data elements whose name match the regular expression.
Data Definition	Data.define(<*dataName*>);	Defines a new data element that will be created at runtime.
	Data.define(<*arrayName*>,<*dim*>);	Define an array of data elements.
	Data.define(<*arrayName*>,[<dim_1>,...,<dim_n>]);	Define a multi-dimensional array of data elements.
Tool Execution	<*toolName*>(<par_1>:<val_1>,...,<par_n>:<val_n>);	Invokes an existing tool with associated parameter values.

Data Access is implemented by the `Data.get` function, which is available in two versions: the first one receives the name of a data element, and returns a reference to it; the second one returns an array of references to the data elements whose name match the provided regular expression. For example, the following statement:

```
var ref = Data.get("Census");
```

assigns to variable `ref` a reference to the dataset named `Census`, while the following statement:

```
var ref = Data.get(new RegExp("^CensusPart"));
```

assigns to `ref` an array of references (`ref[0]...ref[n-1]`) to all the datasets whose name begins with `CensusPart`.

Data Definition is done through the `Data.define` function, available in three versions: the first one defines a single data element; the second one defines a one-dimensional array of data elements; the third one defines a multi-dimensional array of data elements. For instance, the following piece of code:

```
var ref = Data.define("CensusModel");
```

defines a new data element named `CensusModel` and assigns its reference to variable `ref`, while the following statement:

```
var ref = Data.define("CensusModel", 16);
```

defines an array of data elements of size 16 (`ref[0]... ref[15]`). In both cases, the data elements will be created at runtime as result of a tool execution.

Differently from Data Access and Data Definition, there is not a named function for Tool Execution. In fact, the invocation of a tool T is made by calling a function with the same name of T. For example, the following statement:

```
DTree({dataset:DRef, confidence:0.05, model:MRef});
```

invokes a tool named DTree, where `DRef` is a reference to the dataset to be analyzed, previously introduced using the `Data.get` function, `MRef` is a reference to the model to be generated, previously introduced using `Data.define`.

III.2 Basic patterns

Several workflow patterns can be implemented with JS4Cloud [4]. Figure 5 shows four examples of patterns that can be defined in JS4Cloud workflows, namely data partitioning, data aggregation, parameter sweeping and input sweeping. For each pattern, the figure shows an example as a visual DMCF workflow, and how the same example can be coded using JS4Cloud.

The *data partitioning* pattern produces two or more output data from an input data element, as in Figure 5-a1, where a Partitioner tool divides a dataset into a number of splits. With JS4Cloud, this can be written as shown in Figure 5-a2.

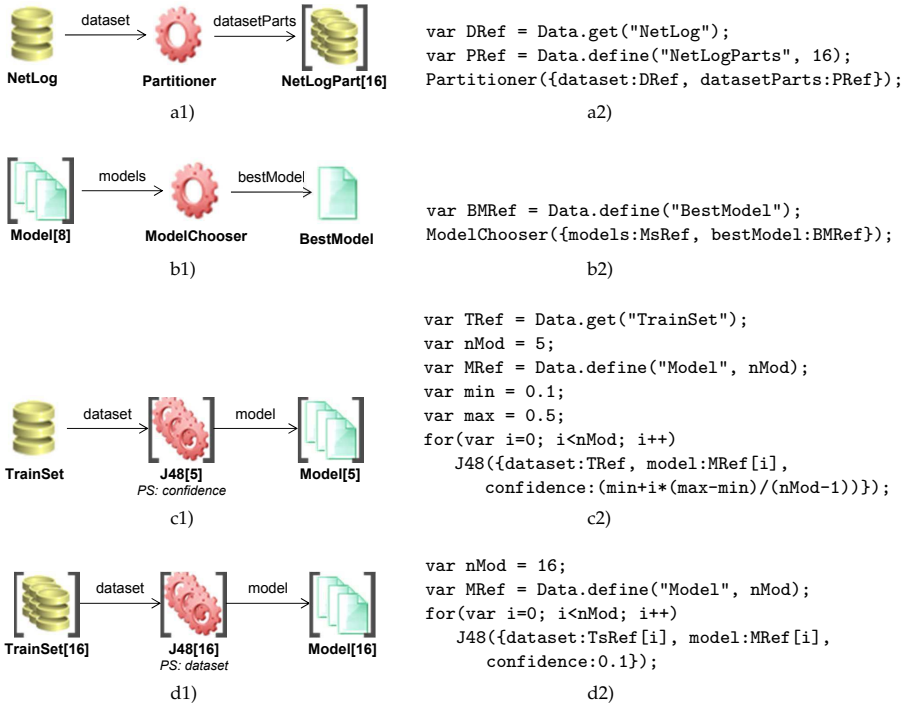


Figure 5: Visual (left) and JS4Cloud (right) workflow patterns: a) data partitioning; b) data aggregation; c) parameter sweeping; d) input sweeping.

The *data aggregation* pattern generates one output data from multiple input data, as in Figure 5-b1, where a ModelChooser tool takes as input eight data mining models and chooses the best one based on some evaluation criteria. The same task can be coded using JS4Cloud as shown in Figure 5-b2.

Parameter sweeping is a data analysis pattern in which a dataset is analyzed by multiple instances of the same tool with different parameters, as in the example shown in Figure 5-c1. In this example, a training set is processed in parallel by 5 instances of the J48 data classification tool to produce the same number of data mining models. The J48 instances differ each other by the value of a single parameter, the *confidence* factor, which has been configured (through the visual interface) to range from 0.1 to 0.5 with a step of 0.1. The equivalent JS4Cloud script is shown in Figure 5-c2.

Finally, *input sweeping* is a pattern in which a set of input data is analyzed independently to produce the same number of output data. It is similar to the parameter sweeping pattern, with the difference that in this case the sweeping is done on the input data rather than on a tool parameter. An example of input sweeping pattern is represented in Figure 5-d1. In this example, 16 training sets are processed in parallel by 16 instances of J48, to produce the same number of data mining models. The corresponding JS4Cloud script is shown in Figure 5-d2.

III.3 Example of JS4Cloud workflow

We describe a JS4Cloud workflow that analyzes a dataset using n instances of the J48 classification algorithm that work on n partitions of the training set and generate n knowledge models. By using the n generated models and the test set, n classifiers produce in parallel n classified datasets (n classifications). In the final step of the workflow, a voter generates the final classification (in the file `FinalClassTestSet`) by assigning a class to each data item. This is done by choosing the class predicted by the majority of the models [6].

The input dataset, containing about 46 million tuples and with a size of 5 GB, was generated from the *KDD Cup 1999*'s dataset, which contains a wide variety of simulated intrusion records in a military network environment.

Figure 6 shows the JS4Cloud code of the workflow. At the beginning, the input dataset is split into training set and test set by a partitioning tool (*line 3*). Then, the training set is partitioned into 64 parts using another partitioning tool (*line 5*). As third step, the training sets are analyzed in parallel by 64 instances of the J48 classification algorithm, to produce the same number of classification models (*lines 7-8*). The fourth step classifies the test set using the 64 models generated on the previous step (*lines 10-11*). The classification is performed by 64 classifiers that run in parallel to produce 64 classified test sets. As the last operation, the 64 classified test sets are passed to a voter that produces the final classified test set.

Beside each code line number, a colored circle indicates the status of execution. The green circles at lines 3 and 5 indicate that the two partitioners have completed their execution; the blue circle at line 8 indicates that J48 tasks are still running; the orange circles indicates that the corresponding tasks are waiting to be executed.

Figure 7 shows the turnaround times of the workflow, obtained

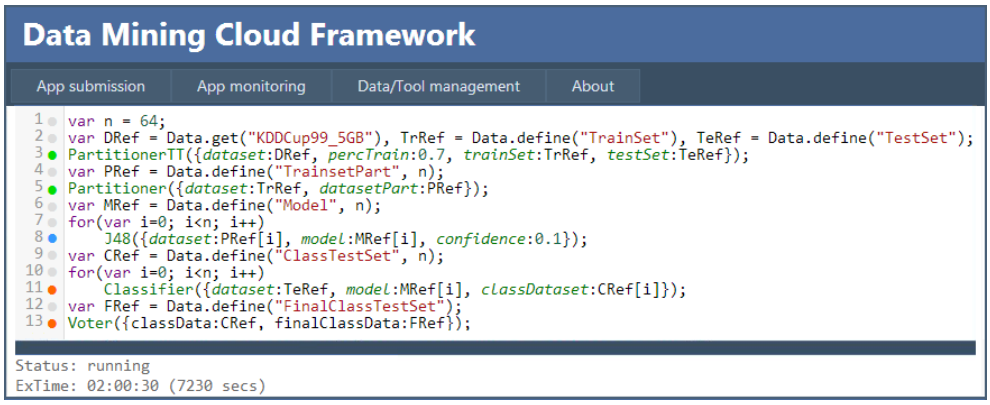


Figure 6: JS4Cloud workflow running in the DMCF's user interface.

varying the number of virtual servers used to run it on the Cloud from 1 (sequential execution) to 64 (maximum parallelism). As shown in the figure, the turnaround time decreases from more than 107 hours (4.5 days) by using a single server, to about 2 hours on 64 servers. This is an evident and significant reduction of time, with a speedup ranging from 7.64 using 8 servers to 50.78 using 64 servers. This is a very positive result, taking into account that some sequential parts of the implemented application (namely, partitioning and voting) do not run in parallel.

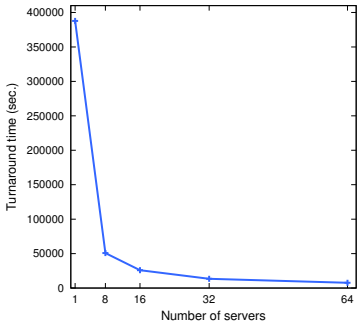


Figure 7: Turnaround time vs number of available servers.

IV. Concluding remarks

Cloud computing [2] provides scalable resources for Big data mining and parallel knowledge discovery applications. In fact, Clouds offer large and efficient storage facilities with high performance processors to get results in reduced times. In this paper we presented a Data Mining Cloud Framework (DMCF) designed for developing and running distributed data analytics applications as collections of services. In this framework, data sets, data mining algorithms and knowledge models are implemented as services that can be combined through a visual interface to produce distributed workflows executed on Clouds.

Recently, we extended the DMCF system to support also script-based data analysis workflows, as an additional and more flexible programming interface for skilled users. To this end, we introduced a workflow-oriented language, called JS4Cloud, to support the design and execution of script-based data analysis workflows on DMCF. Experimental performance results, obtained designing and executing JS4Cloud workflows in DMCF, have proven the effectiveness of the proposed language for programming data analysis workflows, as well as the scalability that can be achieved by executing such workflows on a public Cloud infrastructure.

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] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[2] Cloud Computing Expert Group. The future of cloud computing. Report from European Commission, January 2010.

[3] F. Marozzo, D. Talia, and P. Trunfio. A cloud framework for big data analytics workflows on azure. In *Proc. of the 2012 High Performance Computing Workshop, HPC 2012*. 2012.

[4] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Scalable script-based data analysis workflows on clouds. In *Proc. of the 8th Workshop on Workflows in Support of Large-Scale Science (WORKS 2013)*, pages 124–133, Denver, CO, USA, November 2013. ACM Press.

[5] Domenico Talia. Clouds for scalable big data analytics. *IEEE Computer*, 46(5):98–101, 2013.

[6] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowl. Inf. Syst.*, 24(3):415–439, 2010.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop • October 2014

Scheduling Real-Time Jobs in Distributed Systems - Simulation and Performance Analysis

Georgios L. Stavrinides and Helen D. Karatza

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

Abstract

One of the major challenges in ultrascale systems is the effective scheduling of complex jobs within strict timing constraints. The distributed and heterogeneous system resources constitute another critical issue that must be addressed by the employed scheduling strategy. In this paper, we investigate by simulation the performance of various policies for the scheduling of real-time directed acyclic graphs in a heterogeneous distributed environment. We apply bin packing techniques during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling methods. The simulation results show that the proposed policies outperform all of the other examined algorithms.

Keywords Scheduling, Distributed systems, Real-time jobs, Simulation, Performance evaluation

I. Introduction

The rapid developments in computing and communication technologies have led to the emergence of *ultrascale computing*, which provides a large-scale, heterogeneous distributed platform for the processing of complex jobs [1, 2, 3, 4]. The sustainability of a computing environment of such scale and complexity is one of the most crucial aspects of ultrascale computing.

I.1 Motivation

One of the major challenges in ultrascale systems is the *effective scheduling* and processing of a large number of *interdependent* tasks within strict *timing constraints*. Such tasks often have precedence constraints among them and thus form a *real-time directed acyclic graph (DAG)*, with an end-to-end deadline. In case a real-time job cannot meet its deadline, then depending on its criticality, its result will be useless or even worse, this may have catastrophic consequences on the environment under control [5]. The *distributed* and *heterogeneous* resources of the target system constitute another critical issue that must be addressed during the scheduling of real-time complex jobs [6].

I.2 Contribution

We investigate by simulation the performance of various policies for the scheduling of real-time DAGs in a heterogeneous distributed environment. Our goal is to apply *effective techniques* during the scheduling process, in order to guarantee that every real-time job will meet its deadline.

I.3 Related Work

A large number of job scheduling techniques have been developed and studied in the literature [7, 8, 9, 10, 11, 12, 13]. The most commonly used real-time scheduling algorithm is the *Earliest Deadline First (EDF)* [14]. According to this policy, the job with the earliest deadline has the highest priority for execution. An efficient and practical method for scheduling directed acyclic graphs, is the *list scheduling approach*, according to which the tasks are arranged in a prioritized list. Subsequently, each task is allocated to the processor that minimizes a cost function, such as the task estimated start time [15]. A simple list scheduling algorithm is the *Highest Level First (HLF)* [16], which prioritizes each component task according to the longest path from the particular task to an exit task in the DAG.

Based on the observation that idle time slots may form in the schedule of a processor due to the data dependencies of the tasks in a DAG, Kruatrachue and Lewis in [17] propose the *Insertion Scheduling Heuristic (ISH)*. According to this method which is based on HLF, during the processor selection phase, a task may be inserted into an idle time slot in a processor's schedule, as long as it does not delay the execution of the succeeding task in the schedule and provided that it cannot start earlier on any other processor. Topcuoglu et al. in [18] present the *Heterogeneous Earliest Finish Time (HEFT)* list scheduling strategy, which is essentially an alternative version of ISH, adapted for heterogeneous systems.

An improved version of HEFT is presented in [15] by Arabnejad and Barbosa. It introduces a look ahead feature based on an optimistic cost table. Jiang et al. in [19] present a novel clustering algorithm, the *Path Clustering Heuristic with Distributed Gap Search (PCH-DGS)*, for the scheduling of multiple DAGs in a heterogeneous cloud. Their proposed method tries to insert each group of tasks into the first available idle time slot in a processor's schedule (a DAG's tasks are partitioned into groups in an attempt to minimize the communication cost between them). In case the time gap cannot accommodate all of the tasks of the group, the rest of the group's tasks are inserted into the next available schedule gap of the same or other processor.

All of the above algorithms are static and do not take into account any timing constraints. Moreover, they essentially utilize schedule gaps according to the First Fit bin packing technique [20]. Cheng et al. propose in [21] a scheduling heuristic, *Least Space-Time First (LSTF)*, that takes into account both the precedence and the timing constraints among the tasks. However, their algorithm does not utilize any schedule idle time slots. In this paper, we apply various bin packing techniques (First Fit, Best Fit and Worst Fit) during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling methods. Moreover, our policies are suitable for the dynamic scheduling of multiple real-time DAGs.

II. System and Workload Models

The real-time complex jobs arrive in a Poisson stream with rate λ at a heterogeneous cluster that consists of a set of q fully connected processors. Each processor p_i serves its own local queue of tasks (it has its own local memory) and has an execution rate μ_i. The transfer rate between two processors p_i and p_j is denoted by v_{ij}. The processor execution rates and the communication links data transfer rates may vary. The heterogeneous cluster is dedicated to real-time jobs and it may be part of a computational grid or cloud. The jobs arrive at a central scheduler [22], where their unscheduled tasks wait in a global waiting queue until they get ready to be scheduled. A task becomes ready to be scheduled when it has no predecessors or when all of its parent tasks have finished execution.

The *heterogeneity factor HF* of the system denotes the difference in the speed of the processors, as well as in the transfer rate of the communication links. The execution rate of each processor in the system is uniformly distributed in the range $[\overline{\mu} \cdot (1 - HF/2), \overline{\mu} \cdot (1 + HF/2)]$, where $\overline{\mu}$ is the mean execution rate of the processors. The data transfer rate of each communication link is uniformly distributed in the range $[\overline{v} \cdot (1 - HF/2), \overline{v} \cdot (1 + HF/2)]$, where $\overline{v}$ is the mean data transfer rate of the communication links.

Each job that arrives at the cluster is a directed acyclic graph $G = (V, E)$, where V is the set of the nodes of the graph and E is the set of the directed edges between the nodes. Each node represents a component task n_i, whereas a directed edge e_{ij} between two tasks n_i and n_j represents the data that must be transmitted from task n_i to task n_j. Each node n_i in a DAG has a weight w_i, which denotes its *computational volume* (i.e. the amount of computational operations needed to be executed). The *computational cost* of the task n_i on a processor p_j is given by:

$$Comp(n_i, p_j) = w_i / \mu_j \tag{1}$$

where μ_j is the execution rate of processor p_j. The *level* L_i of a task n_i is the length of the longest path from the particular task to an exit task. The length of a path in the graph is the sum of the computational and communication costs of all of the tasks and edges, respectively, on the path.

Each edge e_{ij} between two nodes n_i and n_j has a weight c_{ij} which represents its *communication volume* (i.e. the amount of data needed to be transmitted between the two tasks). The *communication cost* of the edge e_{ij} is incurred when data are transmitted from task n_i (scheduled on processor p_m) to task n_j (scheduled on processor p_n) and is defined as:

$$Comm\left((n_i, p_m), (n_j, p_n)\right) = c_{ij} / v_{mn} \tag{2}$$

where v_{mn} is the data transfer rate of the communication link between the processors p_m and p_n.

The *communication to computation ratio CCR* of a job is the ratio of its average communication cost to its average computational cost on a target system and is given by:

$$CCR = \frac{\sum_{e_{ij} \in E} \overline{Comm(e_{ij})}}{\sum_{n_i \in V} \overline{Comp(n_i)}} \tag{3}$$

where V and E are the sets of the nodes and the edges of the job respectively. $\overline{Comm(e_{ij})}$ is the average communication cost of the edge e_{ij} over all of the communication links in the system, whereas $\overline{Comp(n_i)}$ is the average computational cost of the task n_i over all of the processors in the system. An example task graph is illustrated in figure 1.

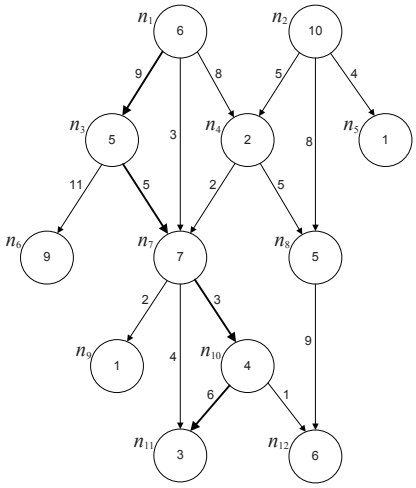


Figure 1: An example DAG with two entry tasks and five exit tasks. The number in each node denotes the average computational cost of the represented task. The number on each edge denotes the average communication cost between the two tasks that it connects. The critical path (i.e. the longest path) of the graph is depicted with thick arrows.

III. Scheduling Strategies

In order to schedule the ready tasks in the global waiting queue, a list scheduling heuristic is employed. This method consists of two phases: (a) a *task selection phase* and (b) a *processor selection phase*.

III.1 Task Selection Phase

Each task is assigned a priority according to one of the following policies:

- ***Earliest Deadline First (EDF)*:** the priority value of each task is equal to the absolute end-to-end deadline of its job. The task with the smallest priority value has the highest priority for scheduling.
- ***Highest Level First (HLF)*:** the priority value of each task is equal to its level. The task with the largest priority value has the highest priority for scheduling.
- ***Least Space-Time First (LSTF)*:** the priority value of a task n_i is equal to its *space-time* ST_i parameter, which is defined as $ST_i(t) = D - t - L_i$, where D is the absolute end-to-end deadline of the task's job, t is the current time instant and L_i is the task's level.

The tasks are arranged in a list, according to their priority. The task with the highest priority for scheduling is placed first in the list. In case two or more tasks have the same priority value, they are arranged in descending order of average computational costs.

III.2 Processor Selection Phase

Once a task is selected by the scheduler, it is allocated to the processor that can provide it with the earliest *estimated start time* EST (ties are broken randomly). The EST of a ready task n_i on a processor p_n is given by:

$$EST(n_i, p_n) = \max\{T_{data}(n_i, p_n), T_{idle}(n_i, p_n)\} \quad (4)$$

where $T_{data}(n_i, p_n)$ is the time at which all input data of task n_i will be available on processor p_n, whereas $T_{idle}(n_i, p_n)$ is the time at which p_n will be able to execute task n_i.

In order to calculate the term $T_{idle}(n_i, p_n)$, the *potential position* of task n_i on processor p_n is determined. This is the position at which the task n_i would be placed according to its priority in the local waiting queue of processor p_n, if it was actually assigned to that particular processor. An alternative, more effective method to determine the potential position of a task in a processor's queue is described below.

III.3 Alternative Method of Potential Position Calculation

According to our proposed method, during the processor selection phase of the scheduling process, the potential position of a ready task in a processor's queue is determined by taking into account not only the task's priority, but also the idle time slots in the processor's schedule that can be utilized. Specifically:

- **Step 1:** We first find the *initial potential position* at which the ready task n_i would be placed in the processor's queue, according to its priority and so that it does not precede the task that is placed after the last exploited idle time slot in the schedule of the processor. The scheduled tasks placed in the area between the head of the queue and the initial potential position of task n_i, form the exploitable area of the queue.
- **Step 2:** The tasks in the exploitable area of the queue are examined whether they can give idle time slots, starting from the task at the head of the queue. An idle time slot is candidate for exploitation by the ready task n_i only when it can accommodate its computational cost. Moreover, task n_i must not delay any succeeding tasks in the processor's queue.

  The task is inserted into an idle time slot according to one of the following bin packing policies:

  - ***First Fit (FF)*:** the task is placed into the first idle time slot where its computational cost fits.
  - ***Best Fit (BF)*:** the task is placed into the idle time slot where its computational cost fits and where it leaves the minimum unused time possible.
  - ***Worst Fit (WF)*:** the task is placed into the idle time slot where its computational cost fits and where it leaves the maximum unused time possible.

  The above procedure has as a result the calculation of the *final potential position* of the ready task n_i.

The pseudocode for the method described above is given in algorithm 1. The scheduling method used in this paper is an enhanced version of the one described in our previous work in [23]. Specifically, in this paper, in case a job misses its deadline, not only are its scheduled tasks that are waiting in processor local queues aborted, but also, all of the other tasks that are waiting in the particular queues are rescheduled (on the same processors), according to their priority. This is necessary, due to the fact that a task removal from a queue may lead to the cancellation of utilized idle time slots or to the creation of new ones that could be exploited by other tasks that are waiting in the queue. Other differences with our previous work in [23] include: (a) the CCR parameter is defined differently in this paper and (b) different values for the simulation input parameters are used.

IV. Performance Evaluation

IV.1 Performance Metric

The performance of the investigated scheduling policies was evaluated by simulation. In order to have full control on all of the required system and workload parameters, we implemented our own discrete-event simulation program in C++. As a performance metric, the *job guarantee ratio* JGR was employed, which is defined as:

$$JGR = \frac{TNJG}{TNJA} \quad (5)$$

where $TNJG$ is the total number of jobs guaranteed, i.e. the total number of jobs that met their deadline. $TNJA$ is the total number of job arrivals at the system, during the time period the system was observed.

IV.2 Simulation Input Parameters

In our simulation experiments we used synthetic workload, in order to obtain unbiased results. The task graphs were generated randomly, using our own custom DAG generator, as described in [24].

Algorithm 1 Alternative method of potential position calculation.

Input: A ready task n_i and a processor p_n.
Output: Final potential position of task n_i in p_n's queue.

```
find initialPotentialPosition of task n_i in p_n's queue
determine exploitable area of p_n's queue
finalPotentialPosition ← initialPotentialPosition
spareTime ← −1
get first task n_j in exploitable area of p_n's queue
repeat
    if task n_j forms a schedule hole then
        if Comp(n_i, p_n) ≤ T_data(n_j, p_n) − EST(n_i, p_n) then
            if Bin Packing Policy = First Fit then
                finalPotentialPosition ← currentQueuePosition
                return finalPotentialPosition
            else if Bin Packing Policy = Best Fit then
                if spareTime = −1 or spareTime > spareTimeOfThisScheduleHole then
                    spareTime ← spareTimeOfThisScheduleHole
                    finalPotentialPosition ← currentQueuePosition
                end if
            else if Bin Packing Policy = Worst Fit then
                if spareTime < spareTimeOfThisScheduleHole then
                    spareTime ← spareTimeOfThisScheduleHole
                    finalPotentialPosition ← currentQueuePosition
                end if
            end if
        end if
    end if
    get next task n_j in exploitable area of p_n's queue
until all tasks in exploitable area of p_n's queue are examined
return finalPotentialPosition
```

The simulation input parameters are summarized in table 1. The computational volume of a task in a graph is exponential with mean $\overline{w}$. The communication volume of an edge is exponential with mean $\overline{c}$. The relative deadline of each job is uniformly distributed in the range $[CPL, 2CPL]$, where CPL is the length of the critical (i.e. the longest) path in the graph. The heterogeneity factor of the system is considered to be equal to $HF = 0.5$. That is, the target system is considered to feature a moderate degree of heterogeneity.

Parameter description	Value
Number of processors in the system	$q = 64$
Mean execution rate of processors	$\overline{\mu} = 1$
Mean data trans. rate of comm. links	$\overline{v} = 1$
Heterogeneity factor	$HF = 0.5$
Number of tasks in each job	$a \sim U[1, 64]$
Arrival rate of the jobs	$\lambda = \{0.2, 0.25, 0.3, 0.35\}$
Relative deadline of each job	$RD \sim U[CPL, 2CPL]$
CCR of the jobs	$CCR = \{0.1, 1, 10\}$
Mean comp. volume of the tasks	$\overline{w} = 10$ ($CCR = 0.1$) and $\overline{w} = 1$ ($CCR = \{1, 10\}$)

Table 1: Simulation input parameters.

IV.3 Simulation Results

We investigated the performance of the scheduling strategies included in table 2, with respect to the arrival rate of the jobs, for DAGs with various communication to computation ratios:

- computationally intensive DAGs ($CCR = 0.1$);
- moderate DAGs ($CCR = 1$);
- communication intensive DAGs ($CCR = 10$).

Scheduling Strategy	Task Selection Phase (task prioritization)	Processor Selection Phase (utilization of idle time slots)
EDF	Earliest Deadline First	No
EDF_FF	Earliest Deadline First	First Fit
EDF_BF	Earliest Deadline First	Best Fit
EDF_WF	Earliest Deadline First	Worst Fit
HLF	Highest Level First	No
HLF_FF	Highest Level First	First Fit
HLF_BF	Highest Level First	Best Fit
HLF_WF	Highest Level First	Worst Fit
LSTF	Least Space-Time First	No
LSTF_FF	Least Space-Time First	First Fit
LSTF_BF	Least Space-Time First	Best Fit
LSTF_WF	Least Space-Time First	Worst Fit

Table 2: Examined scheduling strategies.

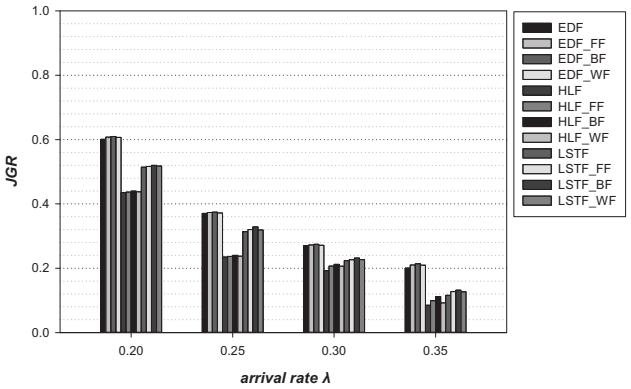


Figure 2: JGR vs. λ for $CCR = 0.1$.

Figures 2, 3 and 4 show the simulation results in each of the above cases, respectively.

The simulation results suggest that the scheduling strategies that employ the EDF policy in the task selection phase, exhibit better performance than the strategies that employ the HLF and the LSTF policies. This is more obvious in the case of computationally intensive DAGs. Furthermore, the proposed alternative versions of the scheduling algorithms that utilize idle time slots in the processor selection phase, outperform their respective counterparts that do not utilize idle time gaps.

Figure 5 shows the average improvement in the system performance for the proposed scheduling policies, compared to their counterpart methods that do not utilize schedule gaps. The improvement is more apparent in the case of computationally intensive workload. Specifically, the average improvement in this case is shown in table 3 for each scheduling strategy. Even though the scheduling strategies that employ the HLF and the LSTF policies in the task selection phase benefit more by the utilization of idle time slots in the processor selection phase than the respective strategies that use EDF, the latter outperform their corresponding counterparts.

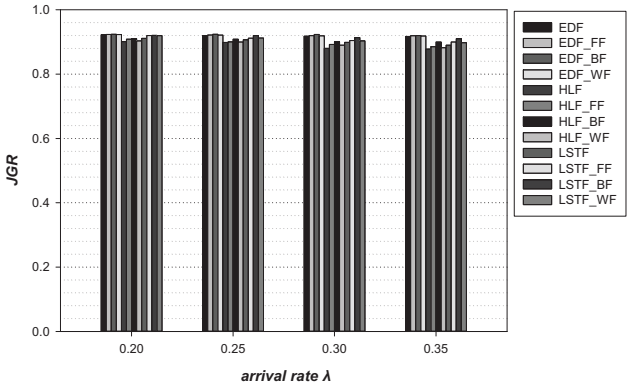


Figure 3: JGR vs. λ for $CCR = 1$.

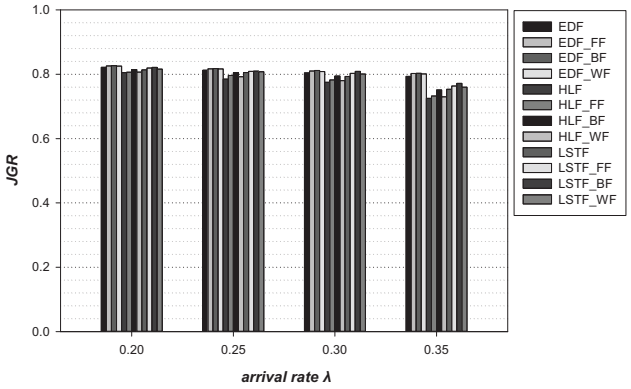


Figure 4: JGR vs. λ for $CCR = 10$.

V. Conclusions and Future Directions

In this paper, we investigated by simulation the performance of various policies for the scheduling of real-time DAGs in a heterogeneous distributed environment. We applied bin packing techniques during the processor selection phase of the scheduling process, in order to utilize schedule gaps and thus enhance existing list scheduling algorithms.

The simulation results suggest that in the case where the utilization of idle time slots is based on the Best Fit bin packing technique, the system exhibits better performance than in the case where the First Fit and the Worst Fit policies are used. Overall, the proposed EDF_BF scheduling strategy outperforms all of the other examined algorithms.

Ultrascale systems may utilize multicore architectures. Moreover, energy efficiency and fault tolerance are vital aspects of their sustainability [25, 26]. Therefore, our future research plans include the adaptation of our proposed scheduling strategies in order to meet those needs.

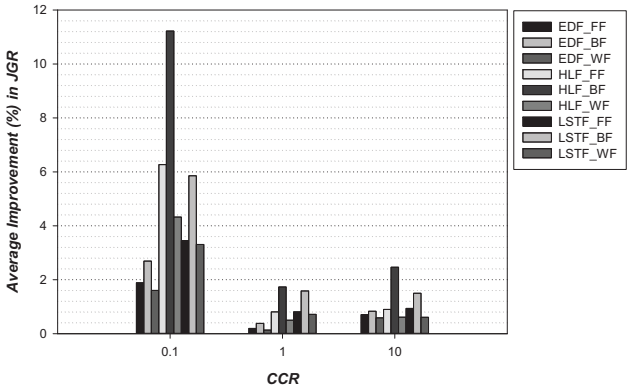


Figure 5: The average improvement (%) in the system performance for the proposed scheduling strategies, compared to their counterpart policies that do not utilize idle time slots.

Scheduling Strategy	Average Improvement in JGR
EDF_FF	1.89%
EDF_BF	2.69%
EDF_WF	1.60%
HLF_FF	6.27%
HLF_BF	11.22%
HLF_WF	4.32%
LSTF_FF	3.45%
LSTF_BF	5.86%
LSTF_WF	3.31%

Table 3: The average improvement in the system performance for the proposed scheduling strategies, in the case of computationally intensive workload.

Acknowledgment

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

References

[1] R. Boutaba, L. Cheng, and Q. Zhang, "On cloud computational models and the heterogeneity challenge", *Journal of Internet Services and Applications*, Springer, vol. 3, no. 1, pp. 77-86, 2012.

[2] J. Carretero and J. D. Garcia, "The Internet of Things: connecting the world", *Personal and Ubiquitous Computing*, Springer-Verlag, vol. 18, no. 2, pp. 445-447, 2014.

[3] J. Carretero and J. G. Blas, "Introduction to cloud computing: platforms and solutions", *Cluster Computing*, Springer, to appear.

[4] G. L. Stavrinides and H. D. Karatza, "The impact of resource heterogeneity on the timeliness of hard real-time complex jobs", in *Proceedings of the 7th International Conference on Pervasive Technologies Related to Assistive Environments (PETRA'14), Workshop on Distributed Sensor Systems for Assistive Environments (Di-Sensa)*, Island of Rhodes, Greece, May 2014, to appear.

[5] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed., Springer, New York, U.S.A., 2011.

[6] G. L. Stavrinides and H. D. Karatza, "The impact of input error on the scheduling of task graphs with imprecise computations in heterogeneous distributed real-time systems", in *Proceedings of the 18th International Conference on Analytical and Stochastic Modelling Techniques and Applications (ASMTA'11)*, Venice, Italy, June 2011, pp. 273-287.

[7] H. D. Karatza, "Performance of gang scheduling policies in the presence of critical sporadic jobs in distributed systems", in *Proceedings of the 2007 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'07)*, San Diego, U.S.A., July 2007, pp. 547-554.

[8] G. L. Stavrinides and H. D. Karatza, "Performance evaluation of gang scheduling in distributed real-time systems with possible software faults", in *Proceedings of the 2008 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'08)*, Edinburgh, U.K., June 2008, pp. 1-7.

[9] H. D. Karatza, "Performance of gang scheduling strategies in a parallel system", *Simulation Modelling Practice and Theory*, Elsevier, vol. 17, no. 2, pp. 430-441, 2009.

[10] G. L. Stavrinides and H. D. Karatza, "Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes", *Future Generation Computer Systems*, Elsevier, vol. 28, no. 7, pp. 977-988, 2012.

[11] K. Chatterjee, A. Kossler, and U. Schmid, "Automated analysis of real-time scheduling using graph games", in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, Philadelphia, U.S.A., April 2013, pp. 163-172.

[12] H. Yu, Y. Ha, and B. Veeravalli, "Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems", *IEEE Transactions on Computers*, IEEE, vol. 62, no. 10, pp. 2026-2040, 2013.

[13] H. D. Karatza, "Scheduling jobs with different characteristics in distributed systems", in *Proceedings of the 2014 International Conference on Computer, Information and Telecommunication Systems (CITS 2014)*, Jeju Island, South Korea, July 2014, pp. 1-5.

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *Journal of the ACM*, ACM, vol. 20, no. 1, pp. 46-61, 1973.

[15] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table", *IEEE Transactions on Parallel and Distributed Systems*, IEEE, vol. 25, no. 3, pp. 682-694, 2014.

[16] T. L. Adam, K. M. Chandy, and J. R. Dickson, "A comparison of list schedules for parallel processing systems", *Communications of the ACM*, ACM, vol. 17, no. 12, pp. 685-690, 1974.

[17] B. Kruatrachue and T. G. Lewis, "Duplication scheduling heuristic, a new precedence task scheduler for parallel systems", Technical Report 87-60-3, Oregon State University, Corvallis, Oregon, U.S.A., 1987.

[18] H. Topcuoglu, S. Hariri, and M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Transactions on Parallel and Distributed Systems*, IEEE, vol. 13, no. 3, pp. 260-274, 2002.

[19] H. J. Jiang, K. C. Huang, H. Y. Chang, D. S. Gu, and P. J. Shih, "Scheduling concurrent workflows in HPC cloud through exploiting schedule gaps", in *Proceedings of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, Melbourne, Australia, October 2011, pp. 282-293.

[20] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo, "Bin packing approximation algorithms: survey and classification", *Handbook of Combinatorial Optimization*, Springer, pp. 455-531, 2013.

[21] B. C. Cheng, A. D. Stoyenko, T. J. Marlowe, and S. K. Baruah, "LSTF: a new scheduling policy for complex real-time tasks in multiple processor systems", *Automatica*, Elsevier, vol. 33, no. 5, pp. 921-926, 1997.

[22] X. Zhu, X. Qin, and M. Qiu, "QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters", *IEEE Transactions on Computers*, IEEE, vol. 60, no. 6, pp. 800-812, 2011.

[23] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques", *Simulation Modelling Practice and Theory*, Elsevier, vol. 19, no. 1, pp. 540-552, 2011.

[24] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations", *Journal of Systems and Software*, Elsevier, vol. 83, no. 6, pp. 1004-1014, 2010.

[25] G. L. Stavrinides and H. D. Karatza, "Fault-tolerant gang scheduling in distributed real-time systems utilizing imprecise computations", *Simulation: Transactions of the Society for Modeling and Simulation International*, Sage Publications, Special Issue on Performance Evaluation of Computer and Telecommunication Systems, vol. 85, no. 8, pp. 525-536, 2009.

[26] G. Terzopoulos and H. D. Karatza, "Performance evaluation and energy consumption of a real-time heterogeneous grid system using DVS and DPM", *Simulation Modelling Practice and Theory*, Elsevier, vol. 36, pp. 33-43, 2013.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
Book paper template • September 2014 • Vol. I, No. 1

Bi-objective Workflow Scheduling in Production Clouds: Early Simulation Results and Outlook

Juan J. Durillo, Radu Prodan

Institute of Computer Science, University of Innsbruck, Austria
juan,radu@dps.uibk.ac.at

Sorina Camarasu-Pop, Tristan Glattard

CREATIS – CNRS, Lyon-Vileurbanne, France
tristan.glatard,sorina.pop@creatis.insa-lyon.fr

Frédéric Suter

IN2P3 Computing Center, Lyon-Villeurbanne, France
frederic.suter@cc.in2p3.fr

Abstract

We present MOHEFT, a multi-objective list scheduling heuristic that provides the user with a set of Pareto tradeoff optimal solutions from which the one that better suits the user requirements can be manually selected. We demonstrate the potential of our method for multi-objective workflow scheduling on the commercial Amazon EC2 Cloud by comparing the quality of the MOHEFT tradeoff solutions with a state-of-the-art multi-objective approach called SPEA2 for three types of synthetic workflows with different parallelism and load balancing characteristics. We conclude with an outlook into future research towards closing the gap between the scientific simulation and real-world experimentation.*

Keywords Scheduling, scientific workflows, Cloud computing, multi-objective optimisation, list-based heuristics

I. Introduction

In previous e-science research [21], many scientists have found in scientific workflows an attractive model of building large scale applications for heterogeneous wide-area parallel and distributed computing systems such as Grids. Typically, a scientific workflow application [20] consists of several (legacy) programs (referred from now on as tasks or activities) in the form of a dependency graph, where the input of some of these programs may depend on the output of the others. Once the application is composed as a workflow, its performance depends on how the individual tasks are mapped (scheduled) on to the available parallel and distributed resources. Traditionally, finding an optimal schedule of the tasks minimising the *makespan* or completion time of the whole workflow has been the main objective and a major NP-complete challenge [23]. As a consequence, many heuristics and meta-heuristics [4] for approximating a solution to this problem have been proposed [18, 22].

In the context of Cloud computing, the computed mapping must additionally optimise the *economic cost* incurred by renting resources. Today, most commercial Clouds offer heterogeneous types of resources at different prices and with different performance. For example, in Amazon EC2 (`http://aws.amazon.com/ec2/pricing/`) a user can choose among different types of instances, where the fastest resource is about eight times more expensive than the slowest one[1]. In these circumstances, the workflow scheduling problem has to be formulated as a *multi-objective optimisation problem (MOP)* which aims at optimising at least two conflicting criteria: makespan and economic cost of workflow's execution. The main characteristic of MOPs is that no single solution exists that is optimal with respect to all objectives, but a set of tradeoff solutions known as *Pareto front* [9]. Solutions within this set cannot be further improved in any of the considered objectives without causing the degradation of at least another objective. Most related work [13, 17, 2] simplifies workflow scheduling optimising several competing objectives to a single-objective problem by aggregating all the objectives in one analytical function. The main drawback of these approaches is that the aggregation of the different objectives is made a priori, with any knowledge about the workflow, infrastructure, and in general about the problem being solved. Therefore, the computed solution may not properly capture the user preferences. On the other hand, few approaches computing the tradeoff solutions have been proposed. Their main advantage over the aggregative ones is that the user is provided with a set of optimal solutions from which the one that better suits the requirement or preferences can be manually selected.

To address this gap, we introduce in this paper a new multi-objective workflow scheduling method called *Multi-Objective Heterogeneous Earliest Finish Time (MOHEFT)* as an extension to the

[1] These prices only refer to the Standard On-Demand Instances (September 2013)

Book paper template • September 2014 • Vol. I, No. 1

well-known HEFT [22] mono-objective workflow scheduling algorithm. Our proposal is a new heuristic-based method that computes a set of tradeoff solutions with a small additional overhead compared to the traditional single objective methods. In doing this, MOHEFT builds several intermediate workflow schedules in parallel in each step instead of a single one. To ensure the quality of the tradeoff solutions, MOHEFT uses dominance relationships and a metric called crowding distance to guarantee their diversity. MOHEFT is generic in the number and type of objectives, applied in this paper for optimising the makespan and economic cost of running workflow applications in an Amazon-based commercial Cloud.

While in theory a Cloud user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in Amazon this maximum number is limited to $N = 20$ and could be enlarged through offline communication. Within this maximum number N, the user flexibly can choose between the different types of instances offered by the Cloud provider (e.g. `m1.small`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge` for Amazon EC2) with different performance and prices. The question which instances should compose the set of maximum size N for running the workflow becomes critical and has no single answer, since different combinations produce different tradeoff schedules. Moreover, the set of N instances does not need to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance is mostly beneficial towards the end. These additional constraints imposed by commercial Cloud systems require modifications to the proposed algorithms originally designed for heterogeneous distributed computing systems. Additionally, we also aim at highlighting the potential of the Pareto front as a tool for decision support, analysing how the user can exploit this information for improving the workflow schedule.

The paper is organised as follows. Section II defines the abstract workflow, resource, and problem definition underneath our approach. In Section III, we present our multi-objective workflow scheduling algorithm, adapted to the case of commercial Clouds. We present in Section IV the experimental setup for evaluating our technique on several synthetic and real-world workflows on Amazon EC2 (Section V). Finally, we summarise the conclusions and the future work in Section VI.

II. Model

This section formally describes the workflow, resource, and problem definition underneath our approach.

II.1 Workflow Model

We model a *workflow application* as a directed acyclic graph: $W = (A, D)$ consisting of n tasks (also referred in the remaining of this paper as activities) $A = \bigcup_{i=1}^{n} \{A_i\}$, interconnected through control flow and data flow dependencies; $D = \{(A_i, A_j, Data_{ij}) \mid (A_i, A_j) \in A \times A\}$, where $Data_{ij}$ represents the size of the data which needs to be transferred from activity A_i to activity A_j. We use $pred(A_i) = \{A_k \mid (A_k, A_i, Data_{ki}) \in D\}$ to denote the *predecessor* set of activity A_i, (i.e. activities to be completed before starting A_i). Finally, we assume that the computational workload of every activity A_i is known and is given by the number of machine instructions required to be executed.

II.2 Resource Model

We assume that our hardware platform consists of a set of m heterogeneous resources $R = \cup_{j=1}^{m} R_j$, which can be of any type as provided by Amazon EC2 (e.g. `m1.small`, `m1.medium`, `m1.large`, `m1.xlarge`, `c1.medium`, `c1.xlarge`). For a given resource R_j of a certain type, we know its average performance measured in GFLOPs. In our workflow model we assume that an activity that is executed in any of these resource can benefit from a parallel execution using all the virtual cores exposed by the instance, achieving the performance indicated in the last column of Table 1. The use of any of these resources is charged per every hour of computation following the Amazon prices indicated in the third column of that table. The final price is based not only on the resources' usage, but also in the data stored and transferred among different instances which depends on four components: (1) price per hours of resource's usage PE_{R_i}; (2) price per MB of data storage PS_{R_i}; (3) price per MB of data received PI_{R_i}; (4) price per MB of data sent PO_{R_i}.

The prices of these components depend on the Cloud provider. Currently, Amazon EC2 does not charge for internal data transfers among EC2 instances which do not require a public IP address, i.e. which do not require to be publicly reachable from Internet. Amazon EC2 also does not charge for incoming data from Internet to EC2 instances. In the case of outgoing data, the first GB transferred each month is free, and up to 10TB of information can be transferred at a relative low price of 0.120$ per GB. For the data storage, the price charged by Amazon EC2 is 0.10$ per stored GB.

Finally, commercial Clouds such as Amazon EC2 introduce constraints that must considered. While in theory a user can access an infinite pool of resources, in practice most providers restrict this number to a maximum of N instances that can be simultaneously acquired. For example, in case of Amazon this maximum number is limited to 20 and can be enlarged through offline communication. Within this maximum number N, the user can flexibly choose between the different types of instances with different performance and prices. The question which instances to compose the set of maximum size N for running the workflow becomes critical and has no single answer since different systems of maximum size N will produce different tradeoff schedules. Moreover, the set of N instances does not have to be invariant during the whole workflow execution. For example, it may occur that one type of instance is particularly good at the beginning of the workflow execution, and a different type of instance the most beneficial at the end.

II.3 Problem Definition

Our problem consists in scheduling the execution of the workflow tasks on Cloud resources such that the makespan and the economic costs are minimised. In the rest of this paper, we will use $sched(A_i)$ to denote the resource on which the task A_i is scheduled to be executed. We describe in the following how the two objectives of interest are computed.

II.3.1 Makespan

For computing the workflow makespan, it is first necessary to define the *execution time* $t_{(A_i,R_j)}$ of an activity A_i on a resource $R_j = sched(A_i)$ as the sum of the time required for transferring the biggest input data from any $A_p \in pred(A_p)$ and the time required to complete A_i in R_j:

$$t_{(A_i,R_j)} = \max_{A_p \in pred(A_i)} \left\{ \frac{Data_{pi}}{b_{pj}} \right\} + \frac{workload(A_i)}{s_j}, \quad (1)$$

where $Data_{pi}$ is the size of the data to be transferred between A_p and A_i, b_{pj} is the bandwidth of one TCP stream between the resource where task A_p was executed and the resource R_j, $workload(A_i)$ the length of the task A_i in machine instructions, and s_j the speed of the resource R_j in number of machine instructions per second. Next, we can compute the *completion time* T_{A_i} of activity A_i considering the execution time of itself and its predecessors:

$$T_{A_i} = \begin{cases} t_{(A_i,sched(A_i))}, & pred(A_i) = \emptyset; \\ \max\limits_{A_p \in pred(A_i)} \left\{ T_{A_p} + t_{(A_i,sched(A_i))} \right\}, & pred(A_i) \neq \emptyset. \end{cases} \quad (2)$$

The workflow makespan is finally defined as the maximum completion time of all the activities in the workflow:

$$T_W = \max_{i \in [1,n]} \left\{ T_{(A_i,sched(A_i))} \right\}. \quad (3)$$

II.3.2 Economic Cost

The economic cost depends on two terms: the computation cost $C^{(comp)}$ and the cost of data transfer and storage $C^{(data)}$. We define $C^{(data)}_{(A_i,R_j)}$ as the cost of the data transfers $In(A_i)$ and $Out(A_i)$ and storage $Data(A_i)$ from executing activity A_i on resource R_j:

$$C^{(data)}_{(A_i,R_j)} = Data(A_i) \cdot t_{(A_i,R_j)} \cdot PS_{R_i} + In(A_i) \cdot PI_{R_i} + Out(A_i) \cdot PO_{R_i}, \quad (4)$$

For defining the cost $C^{(comp)}_{R_j}$ of using a resource R_j, we assume that for each task A_i executed on R_j we record two timestamps: $t^{(start)}_{A_i}$ when the activity starts and $t^{(end)}_{A_i}$ when the activity finishes its execution. The value $t^{(end)}_{A_i}$ can be computed as $t^{(start)}_{A_i} + t_{(A_i,R_j)} + \max_{A_i \in pred(A_p)} \left\{ \frac{Data_{ip}}{b_{jp}} \right\}$. We consider that the times for transferring the input $In(A_i)$ and the output data $Out(A_i)$ are included in the interval between $t^{(start)}_{A_i}$ and $t^{(end)}_{A_i}$. In other words, these time stamps indicate the period of time on which the resource R_j needs to be active due to the execution of the activity A_i.

Let us consider the set of p activities scheduled on resource R_j denoted as $\{J_1, \ldots, J_p\}$, where $p < n$ and $sched(J_i) = R_j, i \in [1,p]$, sorted based on their start timestamp: $t^{(start)}_{J_1} < \ldots < t^{(start)}_{J_p}$. Based on this ordering, we cluster them in $q \leq p$ different *groups* $G^{(j)}_k$, $1 \leq k \leq q$, so that all activities in one group are executed consecutively without releasing the resource. After the activity with the largest start timestamp in the group completes, the resource is released.

We construct the first group $G^{(j)}_1 = \{J_1, \ldots, J_r\}, r \leq p$, based on the following three rules:

1. The first activity J_1 belongs to the first group: $J_1 \in G^{(j)}_1$;

2. Every activity $J_i \in G^{(j)}_1, 2 \leq i \leq r$ completes before the resource is released. This means that J_i starts when the resource is still leased because of the execution of J_{i-1}:

$$t^{(start)}_{J_i} < t^{(start)}_{J_1} + \left\lceil \frac{t^{(end)}_{J_{i-1}} - t^{(start)}_{J_1}}{3600} \right\rceil \cdot 3600. \quad (5)$$

   We divide the total time in seconds of using a resource by 3600 in order to convert it to hours, and use the ceiling operator to round this value to complete hours of computation. Obviously, the resource will be rented for as many hours as required for finishing all the activities within this group.

3. The next activity (not part of the previous group) $J_{r+1} \notin G^{(j)}_1, r+1 \leq p$ starts in an instant of time $t^{start}_{J_{r+1}}$ when the resource has been already released, i.e., task J_r has finished its execution, the last rented period of one hour for executing J_r has expired, and the resource R_j was not needed in the period of time elapsed between $t^{end}_{J_r}$ and $t^{start}_{J_{r+1}}$. Mathematically, it can be expressed as:

$$t^{(start)}_{J_1} + \left\lceil \frac{t^{(end)}_{J_r} - t^{(start)}_{J_1}}{3600} \right\rceil \cdot 3600 < t^{(start)}_{J_{r+1}}. \quad (6)$$

Successive groups are built until the last activity J_p has been assigned. The second group $G^{(j)}_2$ is constructed in the same way starting from the task J_{r+1} instead of J_1. The same strategy is used for the rest of the groups. Once all the groups have been created, we define the cost $C^{(comp)}_{R_j}$ of using the resource R_j as the number hours required for executing all groups multiplied by the cost per hour:

$$C^{(comp)}_{R_j} = PE_{R_j} \cdot \sum_{k=1}^{q} \left\lceil \frac{\sum_{A_i \in G^{(k)}_{R_j}} t_{(A_i,R_j)}}{3600} \right\rceil. \quad (7)$$

We compute the economic cost of executing the entire workflow $W = (A, D)$ as the computation cost on all m resources plus the cost for transferring and storing the data:

$$C_W = \sum_{j=1}^{m} C^{(comp)}_{R_j} + \sum_{(A_i,A_j,Data_{ij}) \in D} C^{(data)}_{(A_i,R_j)}. \quad (8)$$

III. Cloud-aware Multi-Objective Heterogeneous Earliest Finish Time Algorithm

The original HEFT algorithm builds a solution by iteratively mapping the workflow tasks onto the available resources. That mapping is aimed at minimising the completion time of every task, so in every iteration only the resource which minimises this goal is considered. When multiple objectives are considered, the goal is to compute a set of tradeoff solutions by allowing the creation of several solutions at the same time instead of building a single one. Therefore, instead of mapping every task onto the resource where it is finishes earlier,

Algorithm 1 Cloud-aware MOHEFT algorithm.

```
Require: W = (A, D), A = ⋃_{i=1}^{n} A_i                          ▷ Workflow application
Require: N                                                        ▷ Maximum simultaneous instances
Require: I                                                        ▷ Number of different instance types
Require: R = ⋃_{j=1}^{m} R_j                                      ▷ Set of resources, where m = N · I
Require: K                                                        ▷ Number of tradeoff solutions
Ensure: S = ⋃_{i=1}^{K} sched_W, sched_W = {(A_i, sched(A_i)) | ∀A_i ∈ A}   ▷ Set of K tradeoff schedules
1: function MOHEFT(W,R,K)
2:     Rank ← B-RANK(A)                                           ▷ Order the tasks according to B-rank
3:     for k ← 1, K do                                            ▷ Create K empty workflow schedules
4:         S_k ← ∅
5:     end for
6:     for i ← 1, n do                                            ▷ Iterate over the ranked tasks
7:         S' ← ∅
8:         for j ← 1, m do                                        ▷ Iterate over all resources
9:             for k ← 1, K do                                    ▷ Iterate over all tradeoff schedules
10:                s ← S_k ∪ (Rank_i, R_j)                        ▷ Extend all intermediate schedules
11:                if COUNTRESOURCES(sched_W, m) > N then         ▷ More than N instances used
12:                    T_s ← ∞                                    ▷ Mark schedule as non-valid
13:                    C_s ← ∞
14:                end if
15:                S' ← S' ∪ {s}                                  ▷ Add new mapping to all intermediate schedules
16:            end for
17:        end for
18:        S' ← SORTCROWDDIST(S', K)                              ▷ Sort according to crowding distance
19:        S ← FIRST(S', K)                                       ▷ Choose K schedules with highest crowding distance
20:    end for
21: return S
22: end function
```

we should allow mapping it to resources that provide a tradeoff between the considered objectives.

MOHEFT described in pseudocode in Algorithm 1 extends the original HEFT algorithm by approximating a set of tradeoff solutions K instead of a single one. Similar to HEFT, it ranks first the tasks using the B-rank metric (line 2). However, instead of creating an empty solution as in HEFT, it creates a set S of K empty solutions (lines 3–5). Afterwards, the mapping phase of MOHEFT begins (lines 6–20). MOHEFT iterates first over the list of tasks (line 6) sorted by their computed rank. The idea is to extend every solution in S by mapping the next task to be executed onto all m possible resources and store them in a temporal set S' which is initially empty (line 7). For creating these new solutions, we iterate over the set of resources (line 8) and the solution set S (line 9), and add the new extended intermediate schedules to the new set S' (line 15). This strategy results in an exhaustive search if we do not include any restrictions. Therefore, we save only the best K tradeoffs solutions from the temporary set S' into the set S (lines 18–19). We consider that a solution belongs to the best tradeoff if it is not dominated by any other solution and if it contributes to the diversity of the set. For this last criterion, we employ the crowding distance [10], which gives a measure of the area surrounding a solution where no other tradeoff solution is placed. Our criterion is to prefer solutions with a higher crowding distance, since this means that the set represents a wider area of different tradeoff solutions. The constraint on the number of resources is checked in line 11. If the constraint is not violated, the makespan and cost are computed as before, otherwise they are set to infinite. This will cause the algorithm to discard that solutions later on line 18, producing only tradeoff solutions which use at most N instances. After assigning all the tasks (line 21), the algorithm returns the set of K best tradeoff solutions.

Given a set of n activities and m resources, the computational complexity of HEFT is $O(n \cdot m)$. MOHEFT only introduces two main differences with respect to HEFT: the creation of several solutions in each iteration of the algorithm, and the possibility of considering resources providing a tradeoff solution. These two modifications only require an additional loop in MOHEFT (see Algorithm 1, lines 9 – 16). Considering that the set of tradeoff solutions is K, the extra loop in MOHEFT performs only K iterations, rendering a complexity of $O(n \cdot m \cdot K)$. Usually, the number of tradeoff solutions is a constant much lower than n and m. For example, a workflow can be composed of thousands of tasks and the set of tradeoff solutions can be accurately represented with tens of solutions. Thus, the complexity can be approximated as almost $O(n \cdot m)$, as in HEFT.

IV. Experimental Setup

We describe in this section the experiments carried out for validating the Cloud-aware MOHEFT algorithm.

IV.1 Evaluation Metrics

We consider three criteria for comparing the quality of solutions. First, we consider the shortest makespan of the schedules computed by the three analysed techniques. Second, we focus on the economic aspect of the schedules, analysing the cheapest solution reported by each technique. The idea of these two indicators is to assess the behaviour of the different approaches optimising each individual criterion. Finally, we consider the hypervolume indicator for assessing the quality of computed tradeoff solutions. Second, we analyse the tradeoff solutions for different workflow types. Although we compute the tradeoff between cost and makespan, for the sake of highlighting the potential of the obtained results, we will plot the cost savings versus the makespan deterioration, as percentages relative to the most makespan-efficient solution, computed by HEFT. Third, we study the number and the type of instances selected by the different scheduling solutions computed by the three approaches.

We compare the MOHEFT algorithm with SPEA2* [25], a version of the SPEA2 genetic algorithm proposed in [26] which was shown to outperform NSGA-II and PAES for multi-objective workflow scheduling in [25]. We implemented SPEA2* using the jMetal framework [11], slightly modified to deal with the limitation imposed by commercial Clouds on the maximum number of simultaneous resources. This algorithm requires the same input parameters as MOHEFT and works with a population (set) of candidate solutions which are iteratively recombined with the aim of evolving them towards the optima. In our experiments, we used $K = 10$, apply the recombination operator with a probability of 0.9 and the mutation with 0.5. This configuration is the same one used in the original paper where SPEA2* is described. In order to avoid our conclusions be biased by any hazard effect of this stochastic behaviour, we run SPEA2* for five times and always consider the run producing the front with the largest hypervolume.

IV.2 Workflow Applications

We generated three types of synthetic workflows using the random workflow generator described in [24]. Our interest is to analyse how the number of independent activities influences the scheduling results. Therefore, the defined types are intended to cover a wide spectrum of workflow structures from this point of view:

- *Type-1* where the number of tasks that can be executed in parallel ranges between one and two;

Table 1: Performance and price of various Amazon EC2 instances.

Instance	Mean performance [GFLOPS]	Price [$/h]	GFLOPS/$
`m1.small`	2.0	0.1	19.6
`m1.large`	7.1	0.4	17.9
`m1.xlarge`	11.4	0.8	14.2
`c1.medium`	3.9	0.2	19.6
`c1.xlarge`	50.0	0.8	62.5

- *Type-2* where the number of tasks that can be executed in parallel is high, and the workflow is balanced (same number of tasks in every level);
- *Type-3* where the number of tasks that can be executed in parallel is high, but the workflow is unbalanced (different number of tasks in every level).

We generated the length of every task and the data produced using a Gaussian distribution. For every type, we considered 100 different instances having between 100 and 1000 different tasks.

IV.3 Resource Infrastructure

Amazon EC2 offers fourteen different types of on-demand instances with different performance and price. In [1], Iosup *et al.* evaluated them for scientific computing and reported the average performance in millions of floating point operations per second (GFLOPs) of five different instance types thorough extensive benchmark experimentation. Table 1 summarises the average performance, the price per hour of computation, and the ratio GFLOPs per invested dollar of these resources. In this work, we will evaluate our multi-criteria workflow scheduling method on these five instance types.

We consider that the user has access to the default maximum number of $N = 20$ Amazon instances which can be of any of the five types summarised in Table 1 (i.e. $I = 5$ and $m = N \cdot I = 20 \cdot 5 = 100$). We assume that no public IP addresses are required for running the experiments on the Amazon EC2 infrastructure. Additionally, the output data transfers from Amazon to the outside Internet are constant, take place only at the end of the workflow execution and thus, do not influence the scheduling results. In this situation, we assume in our experiments that the prices for data sent and received are zero: $PI_{R_i} = 0$ and $PO_{R_i} = 0$.

V. Evaluation

We present in this section the evaluation results for the synthetic workflow first, then for the real-world ones. Finally, we analyse how the solutions computed by the algorithms change when the constraint of simultaneously using 20 resources is relaxed.

V.1 *Type-1* Workflows

First, Fig. 1a shows that MOHEFT outperformed SPEA2 in terms of hypervolume for all evaluated *Type-1* workflow instances. We did not include HEFT in this comparison because it only delivers a single solution with the optimal makespan. It is remarkable that, for this workflow type, MOHEFT always computed solutions with the same hypervolume value, meaning that the shape of the optimal set of tradeoff solutions in this case does not vary with the workflow size.

In terms of makespan (see Fig. 1b), all the three methods computed the same solution, which confirms that the performance of MOHEFT does not degrade compared to HEFT. In case of SPEA2*, the results are not surprising since the algorithm is initialised with the solution computed by HEFT. Both MOHEFT and SPEA2 computed the same cheapest schedule illustrated in Fig. 1c, which considers the cheapest instance (`m1.small`) for the entire workflow. This fact is a consequence of the low degree of parallelism of this workflow. The solution computed by HEFT is always the most expensive one.

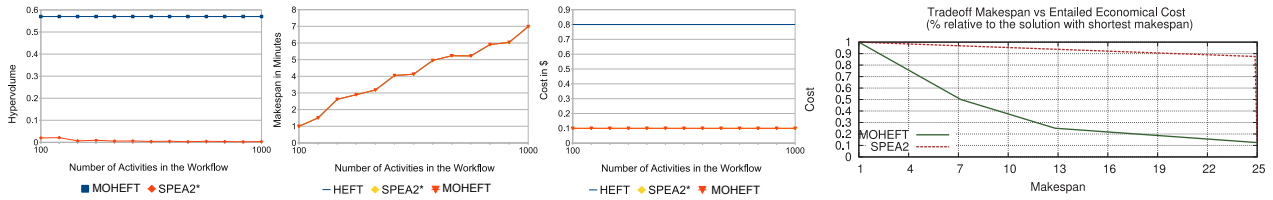
Fig. 1d shows an example of tradeoff solutions computed by MOHEFT and SPEA2*. The higher quality of the solutions computed by MOHEFT can be easily visualised in this chart. In particular, we observe that our method computed a schedule which halves the price of the solution with the optimal makespan by only introducing a 7% of time overhead. In case of SPEA2*, a solution with the same cost would have required an increase of 25% in makespan. These results highlight the importance of the Pareto front as a decision support tool, since computing a single schedule at a time would have hidden this information.

V.2 *Type-2* Workflows

In terms of the quality of the set of tradeoff solutions, MOHEFT has again outperformed SPEA2* for all *Type-2* workflow sizes, as indicated by the hypervolume indicator in Fig. 2a. In this case, different workflow sizes result in Pareto fronts with different hypervolumes, meaning that the shape of the Pareto front for this problem depends on the number of tasks that can be executed in parallel. If we focus on the makespan (see Fig. 2b), it is worth mentioning that MOHEFT and SPEA2* were able in some cases to compute solutions with better makespans than HEFT. The explanation for this behaviour is that, due to its greedy nature, HEFT easily converges towards a local optimum, situation which is overcome by MOHEFT and SPEA2* due to a larger exploration of the search space. In terms of economic cost (see Fig. 2c), MOHEFT and SPEA2* computed the same solution which is a lot cheaper than the solution with the best makespan.

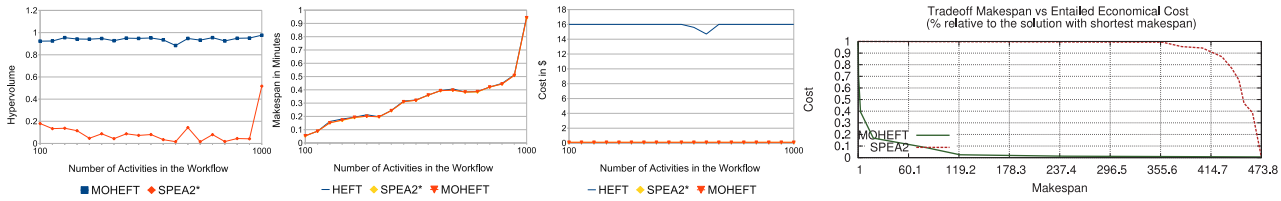
Fig. 2d shows a comparison of the tradeoff solutions computed by MOHEFT and SPEA2*. The differences between both algorithms are even more noticeable than for workflows of *Type-1*. In this case, MOHEFT computed a schedule which reduced the cost by 30% incurring only a 1.4% increase in makespan. Computing a solution of similar price for SPEA2* would have required increasing the makespan by more than 450%. This huge difference between our approach and SPEA2* clearly points out the potential of MOHEFT for multi-objective workflow scheduling in terms of the quality of the computed solutions.

For this workflow type, many solutions computed by SPEA2* required more than 20 resources, thus invalidating its adoption for workflow scheduling in the context of commercial Clouds with limitations on the maximum number of instances that can be simultaneously rented. In particular, 66% of the computed schedules required more than the 20 resource limit imposed by Amazon EC2 (see Fig. 2d). This behaviour does not appear in the solutions computed by MOHEFT or HEFT, which always provided schedules with at most 20 resources.



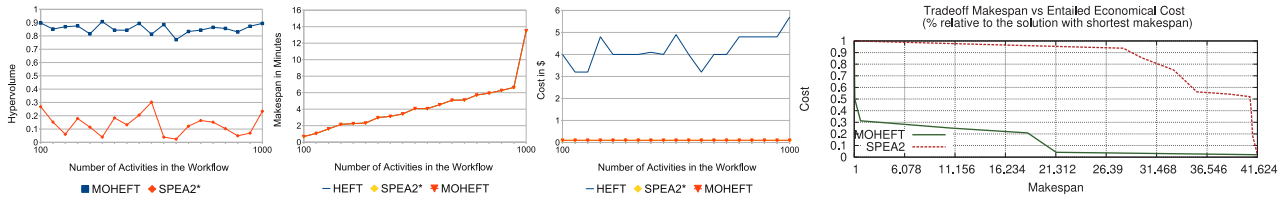
(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 1: Evaluation results for synthetic workflows of Type-1.



(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 2: Evaluation results for synthetic workflows of Type-2.



(a) Hypervolume. (b) Makespan. (c) Economic cost. (d) Cost – makespan tradeoff.

Figure 3: Evaluation results for synthetic workflows of Type-3.

V.3 *Type-3* Workflows

The results for this workflow type, summarised in Fig. 3, confirm the findings of the previous two types.

The hypervolume of the tradeoff sets for the *Type-3* workflows (see Fig. 3a) shows that MOHEFT outperforms SPEA2* also in this case. As in the previous case, the hypervolume reflects a different shape of the Pareto front for workflows with different number of activities. This fact validates the hypothesis that the shape of the tradeoff solutions depends on the number of activities of the workflow that can be executed in parallel. All three techniques computed the schedule which minimises the makespan, confirming again the suitability of this method for workflow scheduling if the user is only interested in optimising this goal. Similar to the previous case, MOHEFT and SPEA2* computed the best solutions in terms of economic cost. The difference between HEFT and the other two methods tends to increase with the number of activities composing the workflow.

An example of the tradeoff solutions computed by MOHEFT and SPEA2* is shown in Fig. 3d. For this workflow type, MOHEFT was able to compute solutions that halve the maximum price with only 1% increase in makespan, while SPEA2* required at least a 40% of extra time for a solution of the same cost. These results indicate once more the better suitability of MOHEFT for multi-objective workflow scheduling on the Amazon EC2 Cloud. In this case, the three techniques always computed schedules meeting the restriction of using at most 20 on-demand instances.

VI. Discussion and Outlook

Designing, optimising, scheduling, and executing scientific applications for heterogeneous computing infrastructures, including production Clouds, involve multiple cycles of code development, small testing followed by real executions, performance monitoring, data collection, optimisation and tuning, which is a cumbersome, tedious, and time-consuming multi-experimental process if not supported by appropriate tools. Working with heterogeneous and dynamic production platforms, such as the European Grid Infrastructure (EGI) or the Amazon EC2 Cloud, brings additional complexity related to performance variability (due to external factors), non-deterministic parallel executions, virtualization overheads, or reliability issues requiring repeated experimentation to produce statistically relevant

results. Repeated experimentation, however, has the drawback of being time and resource consuming (in both manpower and hardware), of disrupting regular production on the platforms, and of limiting freedom in exploration of new cases (e.g. for the sake of curiosity). This issue becomes critical in public Cloud infrastructures, that through their new pay-as-you-go cost resource provisioning model, make the experimentation costs transparent. Finally, energy consumption has recently become another argument against "wasting natural resources" for curiosity science that may yield validated results only in few cases. Faced with this situation, scheduling and executing scientific applications in production distributed computing infrastructures (DCI), including Clouds, has become an increasingly complex multi-objective optimisation problem involving several conflicting metrics for which no appropriate tool support exists due to the increased difficulty in deploying and testing new heuristic methods in real production environments. As a first step in this direction, we proposed a truly multi-objective workflow scheduler called MOHEFT, which extends a well-known list scheduling heuristic in a multi-dimensional objective space of tradeoff solutions. We applied the algorithm in the context of makespan and economic cost optimisation, extended to deal with the realistic constraints imposed by commercial Clouds that restrict the total number of resources that can be simultaneously acquired, but keep their type flexible depending on the temporal needs.

To research and validate such new scheduling heuristics, computer scientists rely nowadays on mathematical models [19, 14], simulators [3, 6, 8], or experimental platforms [5] to reproduce real systems in controlled conditions, which nonetheless remains a challenge [16, 12]. Among these, simulation tools have emerged as important exploration means to facilitate the conduction and management of thousands of experiments, freeing scientists and developers from the complexity and variability of the underlying infrastructure. Simulation tools not only allow easier prototyping and testing of new methods, but also enable their thorough evaluation in situations not easily encountered in real-world scenarios through deterministic and reproducible experiments. Inline with these considerations, we validated and compared MOHEFT with the original HEFT algorithm and with SPEA2*, an extension of the state-of-the-art multi-objective optimisation algorithm SPEA2 by simulating three types of synthetic workflows with different parallelization and work balancing characteristics on Amazon EC2 resources. We showed that the visualisation of the Pareto front can represent a powerful decision making tool for selecting the most appropriate tradeoff solutions. For example, it revealed that certain workflows can be executed twice as cheap by conceding a marginal 5% increase in makespan. In all experiments, MOHEFT computed schedules with the same makespan as the HEFT but with better economic cost, and outperformed SPEA2* in terms of hypervolume used as an indicator of the quality of the set of tradeoff solutions. A visual analysis of the tradeoff solutions revealed that SPEA2* computed in many cases solutions with a higher economic for the same makespan. Finally, our experiments revealed that MOHEFT was able to meet the resource constraints imposed by current commercial Clouds, while SPEA2* failed on this issue.

Our validation, however, is limited to simulation of synthetic workflows and lacks a real-world validation. A reason for this is a major drawback of the existing simulation tools is that they are not properly tuned for production platforms. As argued in [7], most of the existing works validate their research (including scheduling) based on simulation without giving sufficient proofs that the simulator accurately reproduces the behaviour of a real infrastructure, which makes the accuracy and relevance of the results doubtful. Furthermore, most works do not validate the simulated results by comparing them with real executions in a real infrastructure. A reason that brought to this unfortunate situation is the lack of integrated tools that close the cycle between experimentation of new basic research methods (such as investigation of new scheduling optimisation algorithms), their extensive and accurate validation through realistic infrastructure modelling and simulation tools, and finally their integration and deployment on the real platform for production runs delivering the expected improved performance.

We plan in the future to validate our proposed method for real-world scientific workflows on production Cloud infrastructures, which is a real challenge due to the lack of appropriate validation tools. Faced with this research problem, University of Innsbruck together two INRIA divisions (CREATIS – CNRS and IN2P3 Computing Center) aim to research a framework enabling lightweight exploration and evaluation of new scheduling heuristic methods for scientific applications on real Cloud infrastructures through extensive realistic simulations, before deploying them for production runs as illustrated in Figure 4. This work therefore aims to reduce the experimentation costs and contribute to shortening the application lifecycle from its design to production operation, maintenance and tuning. We intend to research methods to build *realistic simulations of real DCIs* from observations and existing simulation toolboxes, with particular focus on the EGI Cloud platform. We will specifically focus on the simulation of the *Virtual Imaging Platform (VIP)* [15], one of the most used scientific computing platforms on the EGI that facilitates sharing of medical image simulators and digital models of the human body. We intend to extend our objective space with other metrics of interest alongside makespan and economic cost, such as energy consumption, reliability, utilisation, fairness, security, and any other Quality of Service or functional application-specific parameter.

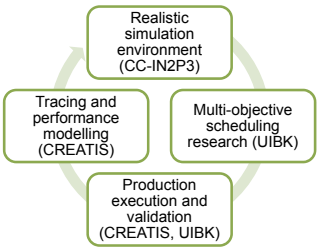


Figure 4: Closing the modelling and simulation – research – production execution cycle.

Acknowledgements

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] I. Alexandru, S. Ostermann, M.N. Yigitbasi, R. Prodan, T. Fahringer, and D.H.J Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Transactions onf Parallel and Distributed Systems*, pages 1–16, 2010.

[2] I. Assayad, A. Girault, and H. Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, Firenze, Italy, June 2003. IEEE.

[3] William H. Bell, David G. Cameron, Luigi Capozza, A. Paul Millar, Kurt Stockinger, and Floriano Zini. Optorsim – a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, 17(4):403–416, 2003.

[4] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and c onceptual comparison. *ACM Computing Surveys*, 35(3):268 – 308, 2003.

[5] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[6] Rajkumar Buyya and Manzur M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[7] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and systems. *Journal of Parallel and Distributed Computing*. in press.

[8] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a generic framework for large-scale distributed experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.

[9] Carlos A Coello Coello, Gary B Lamont, and David A Van Veldhuisen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[11] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[12] Dick H.J. Epema. Twenty years of grid scheduling research and beyond (keynote talk). In *CCGrid'2011*, pages xxxi – xxxiii, Ottawa, CA, may 2012.

[13] Saurabh Kumar Garg, Rajkumar Buyya, and H. J. Siegel. Scheduling parallel applications on utility grids: time and cost trade-off management. In *Proceedings of the Thirty-Second Australasian Conference on Computer Science - Volume 91*, ACSC '09, pages 151–160, Darlinghurst, Australia, Australia, 2009. Australian Computer Society, Inc.

[14] T. Glatard and S. Camarasu-Pop. Modelling pilot-job applications on production grids. In *7th international workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms (Heteropar 09)*, Delft, The Netherlands, 2009.

[15] T. Glatard, C. Lartizien, Bernard Gibaud, R. Ferreira da Silva, G. Forestier, F. Cervenansky, M. Alessandrini, H. Benoit-Cattin, O. Bernard, S. Camarasu-Pop, N. Cerezo, P. Clarysse, Alban Gaignard, Patrick Hugonnard, H. Liebgott, S. Marache, A. Marion, J. Montagnat, J. Tabary, and D. Friboulet. A virtual imaging platform for multi-modality medical image simulation. *IEEE Transactions on Medical Imaging*, 32(1):110–118, 2013.

[16] Jens Gustedt, Emmanuel Jeannot, and Martin Quinson. Experimental validation in large-scale systems: a survey of methodologies. *Parallel Processing Letters*, 19(3):399–418, 2009.

[17] Mourad Hakem and Franck Butelle. Reliability and scheduling on systems subject to failures. In *Proceedings of the 2007 International Conference on Parallel Processing*, ICPP '07, pages 38–, Washington, DC, USA, 2007. IEEE Computer Society.

[18] E. Ilavarsan and P. Thambidurai. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2):94–103, 2007.

[19] J.T. Moscicki, M. Lamanna, M. Bubak, and P.M.A. Sloot. Processing moldable tasks on the grid: Late job binding with lightweight user-level overlay. *Future Generation Computer Systems*, 27(6):725–736, 2011.

[20] Munindar P. Singh and Mladen A. Vouk. Scientific Workflows: Scientific Computing Meets Transactional Workflows, 1996.

[21] Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[22] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260 –274, mar 2002.

[23] J.D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[24] J. Yu, R. Buyya, and K. Ramamohanarao. Workflow scheduling algorithms for grid computing. In F. Xhafa and A. Abraham, editors, *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 109–153. Springer Berlin, 2008.

[25] Jia Yu, Michael Kirley, and Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*, GRID '07, pages 10–17, Washington, DC, USA, 2007. IEEE Computer Society.

[26] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop • October 2014 • Vol. I, No. 1

An Approach Towards High Productivity Computing

Zorislav Shoyat

Centre for Information and Computer Science
Rudjer Boshkovich Institute, Zagreb, Croatia
sojat@irb.hr

Abstract

The notion of what exactly we mean by productivity is largely depending on the active paradigms of a particular field and, on a global level, on the present prevailing social, cultural, scientific and spiritual paradigms and environment. It follows that in a long term any specific definition of productivity will have to be changed. Unfortunately, due to the historical processes, present day human-computer communication is on an extremely low level of language complexity. Consequently our present day productivity in using computers from the idea till the implementation is very low. This is primarily due to the circulus vitiosus of interdependecy of (hardware) computer architectures and popular computer programming languages based on the designs of the first Electronic Brains of the mid-last century. The natural, human Language is the prime Human tool for building a common model of the Universe, a huge fractal dynamic system, i.e. machine, whose sub-machines are smaller fractal machines consisting of a series which goes through dialects, sociolects down to idiolects. On the other hand, regarding strictly formal non-adaptable "programming" languages we see that almost all our computer linguistic efforts are oriented towards fixed expressions which are simple enough to be easily and efficiently translated into the scalar serial presently prevailing computer architecture(s). Therefore a new, fresh approach is proposed, based on the idea that the lowest possible level of a computer system shall understand a natural-like communication language, which is contextful and deals with Information, not with Data without Meta-Data. By significantly leveling up the human-computer interaction towards the ideals of a semi-natural language completely new approaches for High Productivity Computing, both on Hardware and on Software level can be thought out, and the NESUS WG1 Focus Group High Productivity Computing has been established, to historically, futuristically and realistically define and, based on that, develop, through partner collaboration projects, such a (possible) High Productivity System based on specific hardware and software.

Keywords High Productivity Computing, Computer History, Human Computers, Natural Language, Programming Languages, Productivity, Data, Information, Data Processing, Information Processing, Focus Group High Productivity Computing, NESUS

I. What is Productivity?

The word *productivity* comes from the Latin words *pro* ("for") and *duco, 3.* ("to lead"). In this sense pro-duco means to lead towards some thing, and has generally the meaning of something which was made by a process aimed towards that result. Productivity is therefore actually lively occasioning of making something, and encompasses, in its general sense, the whole path to be followed from an idea to its realisation. Therefore we can speak of productivity in sciences, arts, technologies, and even sports.

Naturally, through constant change in the Human society, and the constant change of particular techniques being employed within each, old or newly developed, field of human enterprise, particular aspects of the term productivity are being differently emphasized. The standpoint from which we regard what actually productivity is as applied to a specific (sub-)field in a specific moment in time is directly depending on the present and projected future needs inside that particular field, as well as the present and projected needs of the society in which this field is embedded.

Therefore it is obvious that the notion of what exactly we mean by *productivity* is largely depending on the present state of affairs of the field we are applying the notion to, and, on a more global level, on the present prevailing social, cultural, scientific and spiritual paradigm and environment. It follows that in a long term any specific definition of productivity will have to be changed. However the general definition, productivity being the lively flowing over a process path from an idea to its realisation, holds in all cases. The length of the process path, be it in time, material, effort..., is the generic measure of productivity, a long path (highly time-consuming realisation, enormous amount of effort...) shows low productivity, a short path high productivity.

From this generic definition we can consequently easily adapt the specific notions of productivity in specific fields of human endeavour, according to the mentioned field and civilisation paradigms, standpoints and preferences. It is important to note that the notion of productivity in all of its wide semantic field is directly connected with the notion of *technique*. The use of a certain technique applied to certain processes will consequently directly induce the productivity of that application. A *technique* is actually the way in which a process (from inception to realisation of whatever is produced) is performed. A good technique is the 'tool' to achieve a short path, i.e. high productivity. Technology is therefore a field aiming towards high productivity by rationally organising applications of different techniques to processes.

II. Productivity and Computers

Once upon a time we had simple calculators, abaci and similar, and writing equipment (pen and paper, stylus and papirus, chisel and

First NESUS Workshop • October 2014 • Vol. I, No. 1

stone...). Therefore to make any more complicated calculations a computer, a person who knows how to calculate and compute, was needed. A Computator was known already in ancient Rome, and the first mention of people being Computers in English dates back to 1613. There were quite a lot of areas computers were employed in - ranging from purely scientific applications through navigation, commerce, balistics, finances, building, designing... to, and in the previous, 20^{th} century, even mostly, war efforts.

And, as sad as it is, exactly those war efforts of the last century were necessitating the employment of a huge number of computers, mostly women, to raise the productivity primarily of enemy code breaking and balistic calculations. But we have already seen in the history of technology that certain repetitive operations, exactly programmed, can be more productive when using mechanical means (as for example the programmed looms) or very entertaining (as e.g. the roll-piano). So, accumbent on the existing developments, the programmed loom/roll-piano, the Hollerith sorting machines, mechanical calculators, Babbage's machine constructions, the work of mrs. Ada Lovelace (and not to forget the Zuse mechanical computers), and naturally abaci, during the Second World War we started developing electromechanical, electrical and electronic equipment which could take over the most tedious parts of the jobs computers had to do, and therefore significantly raise the overall productivity of whatever they had to compute.

II.1 Initial Aim of Computers

It is hard exactly to know if the initial aim of non-human computers (or, better to say, calculating and computing equipment) was primarily to take off the burden of tedious repetitive operations in which humans make a lot of errors, or to shorten the time individual calculations in a process of computing something take, or both of those (which actually seems most probable). Anyway, the development of those first non-human computers radically changed the world in that instant (as strange as it sounds, but here we will not go into the philosophycal justification of this statement).

Regarding the productivity, the "mechanical" calculating and computing done by the electronic brains (!) of that time was much much faster than ever humans could imagine. The "Giant Brain" (as the press called it) ENIAC from 1946 could do unbelievable 35 divisions or square roots per second, and unimaginable 357 multiplications per second! This enormous speed of elementary, but for a human quite time-consuming mathematical operations was the highest peek of computing productivity anybody could conceive, as thousands and thousands computers would have to be employed to do the same amount of calculations in the same amount of time as the giant electronic brain(s).

It could be noted that this thrilling speed of something humans see as very complicated operations (multiplication, division, square root) raised the eyes of everybody towards the unfathomable heights of "intelligence" that were in front of us. Because, if that electronic brain can do so complicated operations so extremely fast, he must be able to be thought (instructed, programmed) to be highly intelligent! We just need to find how to programme intelligence into the giant brain... and soon: it will help us solve all of our problems / it will take over the world (choose by preference).

Therefore it is quite logical that with such an enormous difference of the calculation speed between the electronic computer and the human computer the productivity was fully focused on the electronic "brain". Any (well almost any) amount of human labour to prepare the calculation sequence, the way the computer will perform the process, the algorithm, and the data necessary - was a very small general effort in comparison with the amount of human computer effort saved by the use of the computing equipment.

However, it is interesting to note that the balance of productivity is always important. In 1948. a small stored programme memory was added to the ENIAC computer. This modification made the computer to work much slower, and disabled the possibility of use of parallelism in the processor. Although this modification reduced the speed of the computer by a huge factor of six and in the same time eliminated the ability of parallel computation, it did reduced the reprogramming time from days to hours. The speed of the computer was still high enough to be finally I/O bound, and the productivity was enhanced: the change in speed of programming was considered well worth the loss of performance in execution.

An important fact which has to be mentioned from the historical perspective is that the memories of the early (electronic) computers - both the data and the programme storages, independent of the architecture - were quite small, and their size never surpassed anything a human could not analyse on paper in a reasonable amount of time. Therefore a convenient method of getting rid of algorithmic, data or human errors in the calculations on a computer was the so called "core dump", meaning a full listing of the data in all computer memory locations. This would then be used very effectively by the humans to understand what went wrong. And just to mention an important historical fact: the first electronic computer programmers were women - the human computers employed during the war.

II.2 The "Middle Ages" of Computers

Very soon it was realised that instructing the computers on the basic level of wires, bits and switches is not the most productive way of translating algorithms into actual computing processes. More freedom of expression was needed for the programmers, more autonomy was needed from the computers.

The focus of productivity partly shifted from the previous era, as more and more computers, larger and larger computer memories and faster and faster processors began to be available. Now the challenge started being how to organise all of our ideas (and we just mentioned how high our hopes were flying) and somehow transfer them into something the computer would understand, as the performance of the computers was drastically raising even inside the same decade. Suddenly the computers started being "hungry" for programmes and new data, as most of the simple algorithms and small data sets would be "devoured" by the computing speed.

So, to cope with this change of the focus of productivity, as now the computers were already so stable (hardware-wise) and so fast that we, humans, started lagging behind with our job of programming, we had to invent a new way to organise our process of instructing the computers. That led to the invention of autocoders, assemblers and higher level programming languages.

However the way the computers worked, i.e. the way those very basic operations are organised in streams of operators and operands, was very influential on the development of the emerging "programming languages". Actually, the computers themselves were initially constructed to do sequential mathematical algorithms, the

same way a human computer would do them. They were never in construction then intended to process and render movable pictures, to translate natural languages, or to drive a car. Although between the first few computer programming languages ever invented we see several major families of completely different approaches, as in the families (in the broadest sense) started by Assemblers, Fortran, Lisp, APL and Cobol, hardware was primarily (and still is, unfortunatelly) developed based on the same philosophy which, out of necessity, produced the first electronic computers.

So consequently, most of the programming languages were/are based on the elementary serial calculating engine principles, as to be effectively executed on hardware. This was a combined consequence of the common computer architecture and the fact that vast majority of programmers, being thought about such computer internal working and architecture, very easily accepted just those languages, and not the other computer-linguistical streams like APL or Lisp.

II.3 Where Are We Now?

In the meantime our "necessities" have drastically changed. We do not have any more relatively simple needs that could be explained to the computer in several hundreds, thousands or even tens of thousands of elementary steps. We now, more and more, want computers to perform well in a very vast field of extremely diverse data and information processing domains. Suddenly, historically speaking, we have the wish, which we perceive as a need, to process huge amounts of data, to process extremely complicated algorithms, and we wish we would have computers which could process zillions of over-simple operations, common to our present equipment as they are inherited from the early history of computing equipment. Not many different hardware approaches did we ever try, and most of the alternatives we do not use. And, furthermore, the same type of equipment we also wish to drive cars, move robots, keep our doctor informed about our blood pressure, get the lights properly running in the theatre, and allow us to chat through some book of faces.

And naturally, consequently our focus of the notion of productivity changes again. This time we have an enormous amount of diverse computers, all based on processing very simple operations on any kind of operands presented to them as data, and most of them organised as scalar serial processors. And we have an extended set of programming languages almost all of the same kind, the basic historical computer-execution oriented kind. And even more, most of us use just the linguistically worst languages, i.e. those at the lowest machine level, as e.g. Fortran, C, C++, Java... etc. Though some of us love some of them, others some others, still others keep their faith in Lisp or APL families, we are all gravely and thoroughly aware of the fact that programming anything for the present day data communicating and processing environment is extremely tedious and with results which can not be predicted to be as we wished, wanted or needed.

The productivity is getting lower and lower. The path between the idea and the final realisation, the final answers or possibility of application of the computer is very very long. We cannot cope any more with the complexity which emerged.

So the focus of productivity in present day and, as much as we may predict the future, in the future shall be on the notion that computers, as being more and more inseparable part of our whole society and civilization, have to be user friendly to any educational level of potential users. Therefore the notion of High Productivity in this sense encompasses the productivity of any human necessitating help from information processing and computing equipment, whatever their needs be.

III. Humans and Computers

So we stand here, in front of unknown powers and possibilities, in front of the Giant Brain encompassing presently billions of computers of all kinds, intertwined into a twisty turbulent network of slow, medium, fast, very-fast, ultra-fast connections, and we would like to actually use it. As a personal assistant, as a scientific collaborator, as a companion in times of leisure, as a librarian and as a library, as a preserver and a collector, as a future telling machine, as a trusty banker and as an "intelligence enhancer". And not to forget, we would like this Great Brain also to drive our cars, to play waiters in bars and lounges, and restaurants, naturally, and also to produce cars, to take care of our health, and, most importantly, to entertain us.

There is a very interesting aspect of the interrelationship between Humans and Computers. Though this is not the main thematic of this article, it is a very important aspect of our present day overall civilisation development:

Imagine in one moment a huge electromagnetic storm comes onto our little planet Earth. And suddenly the internet is down. Completely. Could we succeed to live as we knew just twenty years ago? Could we survive with the non-connected computing technology?

Imagine in one moment a huge electromagnetic storm comes onto our little planet Earth. And suddenly no one computer works. All the buzzing computing equipment is dead still. Could we succeed to live as we knew just seventy years ago? There are still quite many between us who lived already seventy years ago. Could our present day civilisation survive the shock of being thrown back just half a century? (We would still have the knowledge!)

Would that not be good stories for a science-fiction horror film? Humans Without Computers!

Well, to continue now, do not worry and do not imagine that any of the above storms happened. The major problem we, as Humans, actually have with Computers from the very begin of their development is that the communication possibilities between a human with an idea and a computer into which it could be implement are so primitive that not many of us either can grasp all the necessary prerequisites, or have enough time to do it, or are interested at all to tackle such a complex field of communication. Even if we have strong intentions and enormous sitting stamina the job of explaining any idea to modern day computers, specially if we need more than one processor, or more than one computer, is a risky job, as a lot of unpredictable problems related to any level of modern computer realisation (hadware and appropriate accompanying software) can suddenly emerge and throw us into a next frenzy, undefined time frame of "development". Most of what we euphemistically call *development* of an application (of ideas onto computers) is actually headbanging debugging and search for solutions of practical implementation problems in a dimly lit space of extremely high complexity, meandering through the vast and confusing ocean of individual low level statements.

First NESUS Workshop • October 2014 • Vol. I, No. 1

III.1 Human Civilisation and the Language

The Language is the prime human tool for building a common model of the Universe. Cybernetically speaking, the Language is a huge fractal dynamic system, i.e. machine, whose sub-machines are smaller fractal machines consisting of a series which goes through dialects, sociolects down to idiolects. On each fractal level the language is defining the model of the world, from the individual, with his knowledges and prejudices, through a specific group, society, field of interest, with its specific terms and phrases and implicities, up to the national Language, with its approach towards life, its epistemology, paradigms, hopes, fears and spirituality. Not forgetting the highest level of human common, some and all languages encompassing, model of the environment in which each and every Human Being lives, as for example the notions of birth and death, the notions of Sun and Moon, leaves, trees, water and vapour, love, heroism, serenity, knowledge etc. etc. From that level down the facet machines, sub-languages which work inside a Language may, as we get towards the individual, be getting more awkward, slangish, highly distorted, very narow-minded, and logically must be entailing smaller amount of active and passive elements. The "meaning" of something is always seen and communicated through the informational structural position of notions transferred into speech by organisation into words and phrases and enveloped by idiomatic linguistical rules into a specific text, intended to have the possibility to "rearrange" the informational structural position of notions in the other person, group, society... One human knows something, many humans know much, all humans know all (what the Human race as a collective knows).

In other words, a natural Language is a very flexible web of interrelations between notions up to the level of text expression. That way it becomes a common model of all that a human and his human civilisation perceives in the world around. Through that it is obvious that all inter-human communication, to be able to express any reference to things not here and now, i.e. to use the frame of reference of a common model of the World and influence the change of particular aspects of the model in communication co-actors, has to be linguistically based. Even more, due to that baseness of Language in our perception of the environment, most of our internal thoughts are very often in the form of words and sentences of a Language which is internal enough for us (mother tongue or tongues or well known, interiorised, other languages). The Language, constantly changing and adapting itself, as a whole defines this active frame of reference through which we inspect our environment and communicate about it.

Are you gay? In a melancholic way? I hope you are not just sad! You know, presently most computers are women!

These few sentences probably exemplify the above text quite well. They seem strange in present day English, and they, to be properly understood (as they were intended to be by the speaker/writer) have to be read/heard within an appropriate *context*. How much information did you get by reading those sentences literaly? For this example the understood meaning during the Second World War America would be quite cheering up for a mathematically versed unemployed woman, staring through a window in a gay, but melancholic way.

III.2 Programming Languages

As already mentioned, the development of programming languages was a natural extension of what the computers could execute towards the perceived needs of what we would like to "compute". It was a huge step forward in the sense of productivity as opposed to the defining and rewiring of specific wires connecting computer subunits, or entering individual bits of instructions into the machine. The higher level syntax, rigid and formal, as to be able to be easily and unambiguously translated into elementary machine instructions, and the use of words similar to some words of natural languages (but absolutely strictly unambiguously defined) made instructing electronic computers, programming them to do something according to some explicitly in the programming language described algorithm, much easier and more convenient. However, the main benefit of using programming languages with a reasonable syntactic and sematic expressibility was (and is), naturally, the possibility to use a higher level of abstraction, and therefore to think easier and more about the algorithmic translation of ideas then the actual low level hardware execution prerequisites.

Therefore the introduction of formal computer programming languages meant a higher level of abstraction, enabled portability and generic algorithmic formalisation, and all that with much less steep learning curves.

III.3 Which Programming Language?

"When tackling a complex new problem first develop a formal language specifically oriented towards that problem, the problem is then much easier to solve" (paraphrasing William M. Waite), or use an existing formal language which is already developed for the necessary type of processing.

Quite interestingly, although it is quite simple to develop and implement a specific field attuned formal language, using standardised tools, and although the productivity of writing in such a specialised language is much higher than in generic formal languages, this very valuable approach seems to be all but forgotten. Even the usage of different already developed and implemented formal languages which are specifically adapted for certain areas of problems, and can yield really much in productivity inside their specialisation, is presently not common. It seems that everybody wants to do everything in just the one (or very few) formal language(s) well known (and actually the popularity of those most popular - C and its family - is a consequence of the C compiler being part of all standard UNIX distributions up until recently, and not a merit per sui generi). Does it matter at all how hard certain things are to program, or how complex certain algorithms come out in those over-popular languages?

Actually almost all our computer linguistic efforts are generally oriented towards making a formal language which could be easy enough for a human to understand and learn, and still simple enough to be easily and efficiently translated into the scalar serial frame of mind of the prevailing computer architecture. They all somehow tend towards the ideal of mathematical notation, algorithmic notation, but also towards the ideal of a natural way of expressing. However, the mathematical and algorithmic notation used in inter-human communication is still too much away towards the natural languages to be realistically regarded as a formal language. These systems of notation, developed ages ago and constantly

enhanced, were never meant to be self-standing in communication, but were and are an "enhancement", a linguistic field-specific addition to the natural language, primarily in the area of the appropriate sociolects. Many other such notations as the mathematical do exist, in a wide variety of domains, with more or less internal formality, consistency and information content inter-fixation - which is the highest in mathematics. All of them are used in inter-human communication only inside natural language texts!

IV. Still Computers? Actually No... Ordinators!

We have seen that the major focus of our use of electronic computing machines has fully changed, from the time of being equipment to compute numerical results of mathematical algorithms to the present time of being equipment to process complex information structures and flows based on algorithms from diverse fields of sciences, arts and communications.

Though we still call our machines computers, actually they are much better described by the French word "ordinateur" (equally in e.g. Spanish, Italian...).

Or perhaps we shall keep this name, *the Ordinator*, for the next generation of machines which will actually process information and not data, and which will be completely different in their architectures, as to allow productive processing based on natural-like languages, where many algorithms will be automatically chosen and optimised based on the knowledge of the context on the lowest levels of hardware and software. And with which humans will communicate in a consistent and understandable way.

IV.1 The Hardware/Programming Languages Bottleneck

Once upon a time we had Simple (as in "then constructable") Hardware architecturally serial scalar and for that kind of hardware we developed Simple formal Languages, expanding them as necessary to cope with as much low level programming as possible in a reasonably "high" formal language environment.

70 years later we extensively use Simple formal Languages - whose development started once upon a time conceptually for serial scalar hardware architectures - to make programmes to be executed on Very complicated Simple Hardware, the direct descendants of hardware constructed once upon a time.

It seems that we got stuck in a vicious circle: The development of new hardware approaches, which would be radically different form present day architectures is not regarded as viable, as there is a lot of "software" which shall be directly compilable/executable on any new or old computer system. Therefore the programming languages keep the possible further hardware and computer architectures development in check. The development of new linguistical approaches, which would be radically different in the sense of much higher, towards the human oriented, expression power, is hampered by the necessity to be implemented using existing programming languages and programming paradigms and for them appropriate hardware. Therefore the hardware keeps possible the development of future alternate and novel human-computer communication languages in check by being very to extremely unfit for effective execution of such communication/programming paradigms.

IV.2 Drinking Water From a Glass

Let us, for a moment, pretend to be very obedient, and execute the following instructions: There is a glass of water on the table. If there is water in the glass do the following. Take a toothpick. With the toothpick acquire a drop of water from the glass. Put the toothpick between your lips. Swallow the drop of water. Return the toothpick. Execute again from the second instruction sentence.

Well, now do it 2 billion times a second, instruction by instruction, drop by drop. What a slow and tedious way to drink a glass of water! So what shall I do? Use a fork! So I will process four drops at once! - Still very slow. So let us try to use for example a hundred thousand toothpicks, or even forks! That shall do the trick! Well, it would... if the glass would not have been to small for so many forks, and if somehow we could solve that terrible problem of COORDINATION of 100.000 hands, some left, some right! And even with the two of them often the one does not know what the other does. (Well, the whole exercise with 100.000 forks picking up the drops 2 billion times per second has actually no sense, as the amount of water drops in a glass of water is only around 5000. So we did a monumentous overkill!)

But it would have been fair (and much much more productive) that somebody have said to you, instead of giving you this silly instructions text, to drink the water from the glass on the table. It would have been much easier for both of you. He could have said it in a simple and to him also easy understandable way, and you could have ignored the toothpick and the fork and used your hand to pick the glass and drink the water from it. This is the difference in productivity based on just the use of proper high level contextfull language (context here, inter alias, being the knowledge of the notions table, glass and drink on both sides of the communication channel).

(Any similarity to present day usage and construction of computers is purely coincident.)

V. Natural vs. Formal Languages

Every Language is in its basis Formal (otherwise we could not understand each other). However this formalness of the natural languages is not specified outside itself, as with strictly formal languages, but is inbuilt in the very essence of the language and the corresponding model of the World. Therefore it is flexible, and the tensions of these flexibilities spread over the society using the language pull certain notions, grammatical rules, phrases, words and pronounciations to new positions in the interrelations network, giving them new meanings or new expression forms.

However all this is possible because natural languages have Context which is mostly implicit in the use of words and their sequences, i.e. phrases, sentences, texts... Therefore it is completely appropriate to use the same lexical word in conveying different meanings, the so called homonyms, as the context allows (if there is enough of it) proper "decoding" of the meaning. Homonyms are two different words with different meanings but which look and sound the same. Something like *cool* in "This jacket is really cool" - i.e. 'Wow'; and "This water is really cool", i.e. 'Brrrr'. Programming languages do not have implicit knowledge of the context and therefore no possibility for usage of homonyms. Contrary, if using information context of what the objects (data) of a specific verb are, what they represent

First NESUS Workshop • October 2014 • Vol. I, No. 1

- that is directly usable to properly define the necessary algorithms to process the data according to the appropriate homonym.

Another important element of all natural languages are synonyms (although theoretically there is always the language efficiency principle acting, which on a longer run disallows complete, full synonyms - no two natural language words have exactly the same meaning).

Formal languages can not describe novel grammatical and semantic rules of themselves. They are never self-referential. Natural languages are a-priory self-referential, as the only possible definition of the language is inherently inside itself, i.e. inside the fractal knowledge of all speakers of that language.

There is also a third kind of languages, in between the formality and rigidity of formal languages, and the naturality and flexibility of natural languages, let's call this language family Natural-Like Languages. These languages, though formally defined in their basis, enable the expansion of their own linguistical rules, of the grammar, of the lexic and of the semantics and context-environment handling. The axiomatic language of mathematics is such a system, which can be expanded and contracted in grammar, in lexic and in semantics, by the proper use of already defined phrases (constructs). Unfortunately the consistent possibility to enhance and change the grammatical rules of itself is not existing in strict formal languages, as are our programming languages. This consequently also leads to the impossibility of algorithms written in "programming" languages to be meaningfully linguistically combined.

VI. An Approach Towards High Productivity Computing

It is obvious that presently we have a huge heap of more and more complex problems to solve if we want to continue our expansionist usage of *data* processing and communication equipment. It is also obvious that we actually need *information* processing and communication equipment, with a much higher productivity of the human-ordinator coordination.

VI.1 Present State of Affairs

When "programming" computers we talk really a lot in a language which can express very little with its grammar and lexis, so we have to talk so much to be able to define at all any even slightly complicated processes and define them well. In other words, we use languages which say very little with a lot of words. Using a language which says a lot with a few words it is possible to understandably express something not trivial in a very short text. Only than are we able to use a lot of words to say really much.

Yet we succeeded to develop enormous amounts of algorithms and by that gained huge quantities (and qualities) of knowledge how to implement our wishes in a formal way. This knowledge, gained in a very hard way, by programming those computing engines which know how to execute only a very few very elemental mathematical, logical and organisational operations, is extremely worthy. But we have to regard it as knowledge and experience, and we shall be very careful by taking those algorithms, those programmes, those applications literaly, as they are now, into the future. By regarding these historical developments in computer science primarily as knowledge founded on experience we are freed to use the same ideas we used to implement on classical computers in the future on completely novel ordinator architectures, expressing them in completely novel natural-like, human language similar language(s).

Much can be said about the rationale of the above-said. We presently use layer upon layer upon layer of extremely complex sequences of extremely simple "instructions" in those "modern" programming languages we use. This results in high levels of unnecessary processing, from the level of the operating system, to the level of the highest layers of user-land. We have scalar serial processors and try to connect them in clusters, grids and clouds, without a general linguistic notion of how to program uniformly those (heterogenous) processing elements. Some of our algorithms may work properly with 32 processing cores, but do not gain anything, or even lose, if we use 64 processors. How do we expand such algorithms to work on thousands of processors? We talk a lot about information processing (even the whole area of human effort is called Information and Communication Technologies), but most of the numbers (specifically on the processing level) we process are pure non-tagged data. Enormous amounts of bits in bytes representing something which could be either enormously overgrown so called Applications, or it could be pictures, films, cardiograms, parts of a Beethoven symphony, in miriads of formats, or anything else, as a matter of fact. The processing unit has absolutely no idea what the data processed represents. The storage unit neither. Nor do we, if we lose the "directory list". We almost know how to program single scalar serial processors, how do we do it when we want to program and coordinate hundreds, thousands, millions? ...and even more, we shall be productive when doing it.

Unfortunately the above short list of just some problems to be solved is far from being exhaustive, as we approach the era of peta-, exa-, zetta-, yotta-. (Yotta is the approximate estimated amount of stars in the presently Observable Universe - that is all stars in all galaxies we could observe by the most modern methods of observation. Our own galaxy, the Milky Way, consists of less than half a Tera of stars.)

VI.2 Data vs. Information

"10.77032961 10.44030651 10.19803903 10.04987562 10.04987562 10.1803903 10.44030651 10.77032961
10.81665383 10.29563014 10.29563014 10.81665383 10.63014581 10.63014581 10.63014581 10.63014581
10.81665383 10.81665383 10.29563014 10.29563014 10.77032961 10.77032961 10.44030651 10.44030651
10.19803903 10.19803903 10.04987562 10.04987562 10.04987562 10.04987562 10.19803903 10.19803903
10.44030651 10.44030651 10.77032961 10.77032961 10.29563014 10.29563014 10.81665383 10.81665383
10.63014581 10.63014581 10.63014581 10.63014581 10.81665383 10.29563014 10.29563014 10.81665383
10.77032961 10.44030651 10.19803903 10.04987562 10.04987562 10.1803903 10.44030651 10.77032961"

Cited here is some data. Obviously a list of numbers, all in an open interval between 10 and 11. This is quite apparent.

However, not much can be done with the above list of data. It is actually meaningless as such. It just happens that I personally know what those numbers represent, as I did generate them, but then, on the other hand, as I did not write any explanation, this list of numbers actually can have any "meaning" anybody gives it, as long as that "meaning" preserves the consistency of the internal data relationships. Or not... Perhaps it is just individual numbers which have no internal relationship at all inside the list? Or should it perhaps be a table, and not a list?

This example shows quite obviously that Information is not Data only. That no meaningful information can be extracted from the presented list without deep investigation, if even then. If we would

use these data as information(s) we have to have a context in which the data has some meaning, some sense. Therefore to talk about information it is necessary to have a context. For the area of data/information processing we could define the notion of information as being a combination of data and metadata - the value and the context, or, in the parlance of linguistics, the significator and the significant, the expression and the content. Though it may seem confusing, we can take as a premise that, for our purposes, the data is actually the expression and the metadata the content, as the data actually expresses a specific value of the same type of content. Therefore we could write Information: Data + Metadata.

It could be suspected, by not knowing the whole truth, that mathematical notation does not involve context. Actually it always does, in a crude way similar (but on a much higher model level) to the (formal) use of I-, J- and K- names in Fortran, which on the level of the (unspoken) language inherent model have a presuposed "context" of being integers, if not explicitly defined as something else by some sentence of the programme. The meanings of mathematical notation (the same as any other, e.g. musical notes) are deeply rooted in the human language and the model of the World, and directly contextually connected with the enveloping linguistic expressions. Humans never process data - only information. The context of this information is always deeply rooted in the perception of the environment, independent of the level of "education" or "knowledge" or "intelligence". And natural languages, consequently (and presequently) handle only Information - even the bees when communicating the location of usable flowers!

VI.3 An Approach Towards High Productivity Computing

Finally, given here are some indications of paths which shall be taken, or at least explored, and which can lead us towards a much higher level of productivity in our cooperation with Computers (or Ordinators of the future).

Firstly we shall use Information, being here defined as context-aware Data and Meta-Data, and not use Data alone. This is the prerequisite change on the paradigmatic level which enables the development of natural-like languages. Further highly important features of natural languages are that they are "eclectic", meaning that different styles, different synonyms and even different grammatical rules on different semantical fields can be seamlessly integrated into the Language. Well, otherwise we could not communicate in the society at all! The use of a contextful language, dependent on information and not data alone, enables the very important and productive usage of homonyms (a simple example would be the usage of a multiplication operator on two complex numbers, where the type of the numerical content - polar, cartesian or dimensionless - is known to the processor, so that actually we have the same word doing mathematically quite different operations, where the results may, as in this case, or may not be inter-compatible, as when multiplying a character by 0 or 1, taking it none times or one time). And, very importantly (and extremely easily implementable) - for the benefit of Humans the language has to have synonyms.

To raise the level of productivity of the human-computer cooperation, it is essential to try to level up the basic Ordinator (Computer) language understanding towards the level of Humans (and not vice versa!). This will enable to significantly widen the approach towards construction of basic hardware processing units and their architectures. As already stated, the present formal languages interlocked with present hardware architectures do not enable efficient implementations of natural-like languages. Naturally, as they are Turing-complete, it is conceivable and attainable to implement this kind of language understanding using present day languages and methods. This is very relevant for the interoperability of approaches towards High Productivity Computing, as the same linguistic system can be implemented in software, as a Virtual Executor, on any convenient spread of present day computing architectures, as well as on specially developed hardware and computer architectures. Naturally, as already explained, the software implementation is sub-optimal per se, and only full development of natural-like human-computer communication and interaction languages implemented in the lowest levels of hardware, microware and firmware will enable a full growth towards the aim of keeping the productivity of information processing and computing as high and as balanced as possible between the two actors in the process - the Human and the Ordinator (computer).

VI.4 An Experiment in High Productivity Computing - Virtue

At the Rudjer Boshkovich Institute in Zagreb we started tackling these problems systematically several years ago, and an investigation into basic principles of computer programming, as well as the principles by which we serialize the inherently extremely massively parallel universe around us into serial algorithms was performed - and a new approach taken.

The result of this approach is Virtue - the Virtual Interactive Resource-Tasking Universal Environment, an experimental implementation of an approach towards High Productivity Computing.

Imagine a mathematically simple and effective visualisation - a four-dimensional hypersphere to be rendered as simple asterisks showing all the dots inside the sphere's radius, and spaces outside, the third and fourth dimension to be shown as a sequence of two-dimensional slices. By just looking at such a very elementary visualisation the basic structure of the hypersphere can easily be deduced. Now find a C, C++ or Java programmer and ask him to make a short programme to show such a sphere to you... And now, after they have shown you this four-dimensional sphere, ask them to show to you how it would look like in 5 or 6 dimensions... Or, better, do not ask them to do any of that for you in such a language, as it would take quite a lengthy time even for the best. (A non-optimised solution which we prototyped in C for 4 dimensions has more then 130 lines of source code - non-paralelised! An optimized or parallelised version would take much longer to develop.) And it is a kind of "one-execution is enough" request and programme.

The definition of a new Virtue verb "sphere" which solves the above problem for any number of dimensions up to 8 [taking three scalar numerical objects, expressed as a real number for the radius, and two n-dimensional numbers (real - 1d, complex - 2d, quaternion - 3d or 4d, or octonion - 5d, 6d, 7d or 8d) for the centre and the space-size] is this sentence: TRIADIC SPACE CENTRE MAGNITUDE GREATER '*' MULTIPLY; OPERATOR @sphere SET. For example: 15 15i15j15k15 31i31j31k31 sphere.

What it says is that you want to have an asterisks wherever the radius is greater than the magnitude of the centered space elements;

First NESUS Workshop • October 2014 • Vol. I, No. 1

and otherwise a blank. The verb SPACE (synonymous to INTERVAL) with a scalar numerical object makes a space of indices from 1 (or 1i1, 1i1j1, 1i1j1k1, ...) up to the specified last indices, i.e. the indicated size of each dimension. The word CENTRE is a simple synonym for the verb SUBTRACT (scalar subtract of n from an index array actually centres it around that n).

Being based on such principles, Virtue is a language which proposes a different approach, by keeping the inherent parallel structure of natural algorithms, and doing the parallel processing by itself, if it is algorithmically possible and feasible. Virtue is a syntactically very simple, yet semantically extremely complex language, offering, inter alias: consistent application of any operation on any logically usable combination of data-types, no "reserved words" and almost no "reserved interpunctions", synonyms, automation of memoisation, multiple word contexts (and therefore homonyms), combined data types of anything Virtue supports (e.g. functions, symbol names, scalars, multidimensional sub-structured spaces etc.), stochastic processing, multivalued and multidimensional logic operations, multidimensional sub-structured file access structures, continuations etc., SIMD, MIMD, SISD and MISD programming models and furthermore also allows for the changes of its own grammar.

Hierarchically, by defining complex meanings for Virtue words (or, differently said, named functions defined as operators), as e.g., building on the previous example, by defining the "meanings" of 'small' 'big' and 'please' thus: NILADIC 5 5i5 11i11; OPERATOR @small SET. NILADIC 30 30i30j30k30 61i61j61k61; OPERATOR @big SET. NILADIC; OPERATOR @please SET., high level simple natural-language like expressions can be used: "small sphere please." or "please big sphere.". Being interactive the Virtue Environment enables easy and understandable development and debugging (including tracing, stepping, intervention, editing...). Used in batch mode, a programme written in Virtue may be used for computing input to other programmes, as for example PowRay for visualisations, or the Virtue environment may be used as a pipe-through between any existing programmes/applications.

Therefore, due to this semantic richness and grammatical simpleness, in Virtue, for example, the text of the algorithm for Conway's "Game of Life" necessitates only 12 language tokens (7 words, 14 numbers in 3 vectors and 2 delimiters) in one sentence: MONADIC (1 1 1 1 0.5 1 1 1 1) [3 3] MONADIC RAVEL SUM (2.5 3 3.5) IDENTICAL ANY; STENCIL;. (The STENCIL is a stenciling operator, synonymous to MASK. The array edge behaviour can easily be modified by optional modifier words REFLECT or WRAP. These operators will work on any-dimensional spaces.) This sentence (algorithm/programme) will work for any size of a "Game of Life" board, up to the size of the underlying hardware memory, and automatically using, if feasible, parallel processing (actually a higher level form of SIMD in this case).

The computer implementation of Virtue is presently in Alpha 0.7 state, and parallelisation is implemented on single-image systems. The experimental implementation is constantly parallely developed and tested on a very wide range of different computers, ranging from the mid-1980-ies Sun3 (16MHz/16MiB and 20MHz/24MiB) workstations up to modern day blades, with various operating systems and their generations (SunOS, Solaris, NetBSD, FreeBSD, Linux, MacOS/X, UWIN, Cygwin...), different compilers and compiler generations, various processors (M68k, MIPS, SPARC, PowerPC, AMD, Intel), on 32 and 64 bit architectures, 32 and 64 bit floating-point, in single-processor and SMD/NUMA multi-processor, multi-threading and multi-core computers. Such a wide range of computers, both historically and speed-wise, for the experimental Virtue implementations allows for development of a very easily adaptable system, and the behaviour of the old Sun3 systems shows that even on them the execution is fast for the amount of data which can be represented in the memory of those computers (as an example, executing the classical double-recursive fibonacci algorithm, a full list of all the *first 1500* Fibonacci series numbers will be produced on a 1986 20MHz Sun3, using the inbuilt Virtue memoisation word RESULT, in only *27.827s*, whereas the next time the same request is entered the results will come in just 4.461s - using the same basic algorithm in Virtue, a non-memoised recursive calculation of the (just) 32^{nd} Fibonacci series number on a modern day SunFire X4240 takes full *26.012s*!).

The internal speed measurements which the Virtue Executor has have provided us with quite a lot of important data on the behaviour of different computer systems and different processors, so an investigation into the "speed of a computer" is presently being performed, with some results to be presented soon.

Shortly presented in this subsection is an experiment in computer implementation of the necessities and possibilities of High Productivity Computing. Further development of this idea allows for definition of a more syntactically rich very high level human oriented machine interaction language, which, combined with additional artificial intelligence components, appropriate ergonomic human presentation/sensory interfaces and with the integration of user style association memory, we sincerely hope can help the future development of Computer Science and Usage Practice.

VII. Focus Group High Productivity Computing

Under the umbrella of Working Group 1 ("State of the art and continuous learning in Ultra Scale Computing Systems") of the NESUS Action (COST IC1305 - "Network for Sustainable Ultrascale Computing") the "Focus Group High Productivity Computing" (FG HProC) was established. The main aim of this Focus Group is to explore the possibilities of an integrative approach which would allow a significant shortening of the time lapse between the human ideas or needs and computer implementation solutions. As already quite thoroughly explained in this article, the major idea driving this Focus Group and its (present and future) work is that computers, as being more and more inseparable part of our whole society and civilization, have to be user friendly to any educational level of potential users, and the notion of High Productivity in this sense encompasses the productivity of any human necessitating help from information processing and computing equipment, whatever their needs be. This is the idealistic aim towards which this Focus Group will try to steer itself.

To achieve this global aiming, the Focus Group will focus on two major aspects of computer science: the past (and present) and the future. To learn from history: a comparative exploration of computer history and present day tools, methodologies, languages, algorithms and hardware in view of the information processing and computing needs and necessities perceived now and, as much as possible, projected into the future. Some of the preliminary investigation results are given in this article. An extremely important aspect of this is historical research into avenues of computer science taken, but not pursued. A huge amount of great ideas is actually

almost forgotten, ideas that can be very useful today, and for which in the time they were invented and thought out there was not a feasible possibility to actually be realized in their full potential. This exploration will be a very valuable addition to the NESUS WG1, regarding the State of the Art, as well as to the general computer history, as observed from the technical and technological side.

To be able to define the necessities of high productivity in the sense of the aim of this Focus Group, another objective is important, the exploration of the concept of High Productivity Computing, or, maybe better to say, High Productivity Information Processing and Computing. The objective of this Focus Group is to explore the development of necessities in the area of high productivity through the history of computer science to present day, and produce knowledgeable reports on possible future shifts of the productivity focus. Prime goals are the envisioned resulting knowledge gained from extended study, and a strict definition of the concept of High Productivity in Information and Computing Sciences.

The central objective of the Focus Group High Productivity Computing is the development, definition and standardization of a universal (linguistic) environment from the level of (new) hardware up to the level of algorithmic expression of complex algorithms involving a wide spectrum of available information processing and computing equipment, and based on the algorithmic and linguistic knowledge gained so far by our civilisation. Based on these the goal is to actually be able to realise a prototype of a High Productivity Information Processing and Computing Infrastructure by facilitating cooperation of different interested partners through the FET and other H2020 European Union science founding calls.

VIII. The End:

– Use courage to "Boldly go where no other computer scientist has gone before"! –

References

[1] C. Backus, "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs.," *Communications of the ACM*, vol. 21, no. 8, pp. 613-641, August 1978.

[2] Y. Bar-Hillel, *Language and Information*, Addison Wesley, Reading, MA; London, U.K, 1964.

[3] D. R. Hofstadter, *Goedel, Escher, Bach: An Eternal Golden Braid*, The Harwester Press, G.B, 1979.

[4] L. Floridi, "Is Semantic Information Meaningful Data?," *Philosophy and Phenomenological Research*, vol. LXX, no. 2, pp. 350-370, March 2005.

[5] G. Lerner, "The Necessity of History," in *Why History Matters: Life and Thought*, New York, NY, USA, 1997.

[6] J. S. Light, "When Computers Were Women," *Technology and Culture*, vol. 40, no. 3, pp. 455-483, July 1999.

[7] E. Lusk, "Languages for High-Productivity Computing: The DARPA HPCS Language Project," *Parallel Processing Letters*, vol. 17, no. 1, pp. 89-102, 2007.

[8] S. Marin, M. Ristic and Z. Sojat, "An Implementation of a Novel Method for Concurrent Process Control in Robot Programming," *Third International Symposium on Robotics and Manufacturing: Research, Education and Application*, ISRAM '90, Burnaby, BC/CA, 1990.

[9] J. Mrsic-Flogel, D. M. Reynolds, Z. Sojat, M. Bianchessi and S. Sala, *Data Communication*, International Patent, US Ref. No: 7565210, 2009.

[10] K. Skala and Z. Sojat, "Towards a Grid Applicable Parallel Architecture Machine," *Computational Science - ICCS 2004, 4th International Conference* KrakÃşw, Poland, 2004.

[11] Z. Sojat, "Principles of Selforganizing Learning and Their Application on a Computor Program," *Pan Arab Science Fair*, Kairo, Egypt, 1976.

[12] Z. Sojat, "Pri la problemo de lingvsignkontinudiskretigeco kaj la sia masxinrealigo," *10e Congres International de Cybernetique*, Namur, Belgium, 1983.

[13] Z. Sojat, "An Approach to an Active Grammar of (Non-Human) Languages," *27. Linguistisches Kolloquium*, Muenster, Germany, 1992.

[14] Z. Sojat, J. Mrsic-Floegel and D. Moritz, "The Facets of an Information," *II. Orwelian Symposium*, Karlovy Vary, CZ, 1994.

[15] Z. Sojat, T. Cosic and K. Skala, "Virtue - A different approach to human/computer interaction," *Information and Communication Technology, Electronics and Microelectronics (MIPRO); 37th International Convention on*, Opatija, Croatia, 2014.

[16] A. N. Whitehead and B. Russell, *Principia Mathematica*, Cambridge University Press, first published 1910.

[17] L. Wittgenstein, *Philosophical Investigations*, Blackwell, Oxford, U.K, 1953.

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)'.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop • October 2014 • Vol. I, No. 1

Efficient Parallel Video Encoding on Heterogeneous Systems

Svetislav Momcilovic, Aleksandar Ilic, Nuno Roma, Leonel Sousa

INESC-ID / IST-TU Lisbon, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
{Svetislav.Momcilovic, Aleksandar.Ilic, Nuno.Roma, Leonel.Sousa}@inesc-id.pt

Abstract

In this study we propose an efficient method for collaborative H.264/AVC inter-loop encoding in heterogeneous CPU+GPU systems. This method relies on specifically developed extensive library of highly optimized parallel algorithms for both CPU and GPU architectures, and all inter-loop modules. In order to minimize the overall encoding time, this method integrates adaptive load balancing for the most computationally intensive, inter-prediction modules, which is based on dynamically built functional performance models of heterogenous devices and inter-loop modules. The proposed method also introduces efficient communication-aware techniques, which maximize data reusing, and decrease the overhead of expensive data transfers in collaborative video encoding. The experimental results show that the proposed method is able of achieving real-time video encoding for very demanding video coding parameters, i.e., full HD video format, 64×64 pixels search area and the exhaustive motion estimation.

Keywords video coding, divisible load theory, load-balancing, CPU+GPU computing

I. Introduction

The newest video coding standards, such as H.264/AVC [17] and HEVC/H.265 [16], achieve high compression efficiencies, by relying on advanced encoding techniques (e.g. multiple partitioning modes, large search ranges, quarter-pixel precision). On the other hand, all these techniques dramatically increase the computational requirements, and make real-time encoding of High Definition (HD) video sequences hard to be achieved on any individual device available on modern desktops, such as multi-core Central Processing Units (CPUs) and Graphics Processing Units (GPUs).

To simultaneously employ several heterogeneous devices available on modern desktops for real-time video encoding, an efficient method for collaborative H.264/AVC inter loop on CPU+GPU systems is proposed herein. A unified execution environment was designed to ensure efficient cross-device execution and to guarantee the correctness of the video encoding process. It includes an extensive library of highly optimized parallel algorithms for all inter-loop video encoding modules, which are developed using the device-specific programming models and tools (e.g. CUDA, OpenMP, etc.). In order to minimize the over inter-loop encoding time, the proposed collaborative environment also integrates different scheduling, load balancing and data access management routines.

Highly efficient parallel algorithms are developed for both CPUs and GPUs and for all inter-loop modules, namely Motion Estimation (ME), Sub-Pixel ME (SME), Interpolation (INT), Motion Compensation (MC), Transform and Quantization (TQ), Inverse TQ (TQ^{-1}) and Deblocking Filtering (DBL). The integrated scheduling and load balancing routines allow efficient distribution of the workloads for these modules over all processing devices. For the most computationally intensive modules (i.e., ME, SME and INT), the proposed load balancing, based on Divisible Load Theory (DLT) [20], relies on realistic and dynamically built Functional Performance Models (FPMs) [7,10] of both communication and computation system resources. The workloads of the remaining modules are distributed at the module level by applying the optimal Dijkstra algorithm [4]. Furthermore, the proposed method also includes specific, communication-aware techniques to maximize data reuse, and to decrease the data transfers overhead. Because of a similar algorithmic structure of the video encoding inter-loop, many solutions provided herein can also be applied to HEVC/H.265 encoders.

The obtained experimental results show that the proposed method achieves a real-time inter-loop video encoding for full-HD (1080p) video sequences, when applying exhaustive ME and 64×64 pixels search area (SA) on a commodity desktop platform equipped with a multi-core CPU and two GPUs. To the best of the authors' knowledge, this is one of the first approaches that applies adaptive load balancing with dynamically built partial estimations of the FPMs to tackle efficient collaborative execution of complex multi-module problems, such as video encoding, in heterogenous environments.

II. Related Work

There are only few state-of-the-art approaches that deal with the efficient parallel implementation of the entire video encoder (or its main functional parts), namely, for multi-core CPU [21], GPU [22], or CPU+GPU [14,15] environments. In CPU+GPU systems, these approaches either *i)* simply offload a single inter-loop module in its entirety (mainly the ME) to the GPU, while performing the rest of the encoder on the CPU [9,19], or *ii)* exploit simultaneous CPU+GPU processing at the level of a *single* inter-loop module [15,23].

These approaches have a limited scalability (only one GPU can be employed) and cannot fully exploit the capabilities of CPU+GPU systems (since the CPU is idle, while the GPU processes the entire offloaded module) [19]. In [9] the pipelining granularity is decided through a large set of experiments, while in [23] the cross-device load distribution is found by intersecting the experimentally obtained fitted full performance curves. However, both approaches impose limited scalability over the number of processing devices and coding

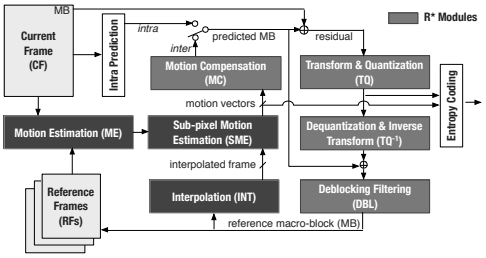


Figure 1: H.264/AVC encoder.

parameters, and, also, introduce huge preprocessing scheduling overheads. The method proposed in [15] use a single GPU and constant compute-only performance parametrization, while in [18] a simple equidistant data partitioning is applied for video encoding in multi-GPU systems, since the CPU is only used to orchestrate the execution across a homogenous set of GPUs.

The methods proposed herein span over three load balancing/scheduling classes for heterogenous environments, namely: simultaneous multi-module load balancing, static DAG-based scheduling and dynamic iterative load balancing. The proposed load balancing method relies on DLT [20] and there are only a few studies targeting the DLT scheduling in CPU+GPU systems either for general [7] or application-specific [1] problems. In [11], the authors apply DLT scheduling for a single-module load distribution in CPU-only cluster environments for a custom video encoder. In [5,13], we proposed load balancing methods that rely on constant performance models and linear programming for determining the cross-device load distributions, while in [12], only the inter-prediction modules were considered.

In this work, for the first time, the real-time video encoding on commodity CPU+GPU platforms is investigated for a complete H.264/AVC inter-loop, where the adaptive iterative load balancing with on-the-fly update of partial FPMs [6,7] is applied at the level of inter-prediction modules. The dynamically built partial estimations of the full FPMs allow realistic and simultaneous modeling of capabilities for communication links and heterogeneous devices during the execution of several inter-loop modules. Moreover, the proposed approach also considers the replication of certain computationally inexpensive encoding modules (i.e., INT) over several processing devices in order to reduce the communication overheads and achieve significant performance gains.

III. Parallel Inter-loop video encoding on CPU and GPU devices

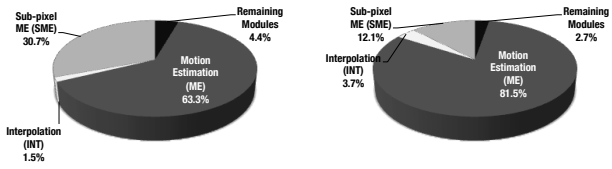
In order to allow an efficient parallelization and collaborative CPU+GPU execution, the inherent data dependences of H.264/AVC inter-loop encoder and the computational requirements of different encoding modules, must be considered. According to the H.264/AVC standard [17], current frame (CF) is divided in multiple square-shaped Macroblock (MB), which are encoded using either an intra- or an inter-prediction mode (see Fig. 1). This standard allows a further subdivision of the MB by considering 7 different partitioning modes, namely 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 and 4×4 pixels. In the most computationally demanding and most frequently applied inter-prediction mode, the prediction of each MB is obtained by searching within already encoded Reference Frames (RFs). This procedure, denoted as ME, is then further refined with previously interpolated Sub-pixel Frames (SFs) from INT module by applying the SME procedure. In the MC module, the residual signal is computed according to the selected MB subdivision mode, which is found as the best trade-off between the size of data required to encode the residual signal and the motion vectors (MVs). The residual is subsequently transformed and quantized in TQ modules, and entropy coded (alongside with the MVs and the mode decision data), before it is sent to the decoder. The decoding process, composed of the TQ^{-1} and DBL, is also implemented in the feedback loop of the encoder, in order to locally reconstruct the RFs.

Parallelization at the level of entire inter-loop imposes several hard-to-solve challenges, which must be explicitly taken into account to ensure the correctness of the overall video encoding procedure. In detail, an efficient parallelization requires the observance of **data dependencies** at several levels: *i)* between consecutive frames, *ii)* within a single video frame, and *iii)* between the inter-loop modules.

In the H.264/AVC inter-loop, the encoding of the CF can not start before the previous frames are encoded and the required RFs are reconstructed. Such a dependency prevents the encoding of several frames in parallel. Moreover, the inherent data dependencies between the neighboring MBs in certain inter-loop modules (such as DBL) also limit the possibility to concurrently perform the entire encoding procedure on different parts of a frame. Hence, efficient module-level pipelined schemes can hardly be adopted, either for parts of the frame or for the entire frame. Furthermore, the output data of one module is often the input data for another (e.g., the MVs from ME define the initial search point for the SME), which imposes additional data dependencies between the inter-loop modules. Hence, the data-dependent inter-loop modules have to be sequentially processed (within a single frame). The only exceptions are ME and INT modules, which can be simultaneously processed, since both of them use the CF and/or the RFs.

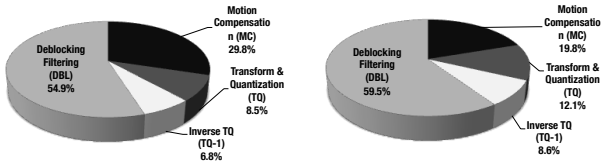
Architecturally different devices in modern CPU+GPU systems impose additional challenges to the **parallelization of individual inter-loop modules**. For example, GPUs require to exploit the fine-gained data-level parallelism suitable for simultaneous processing on hundreds of cores, while for CPU architectures with several general purpose cores the parallelism can be exploited at coarser-grained level. Therefore, it is required to parallelize the modules for each device in the system according to both per-module parallelization potentials and architectural characteristics of devices. For each inter-loop module, the detailed description of parallel algorithms and parallelization techniques applied herein can be found in [13,15].

In what concerns the ME, efficient CPU and GPU parallelizations and collaborative processing the Full-Search Block-Matching (FSBM) is adopted, since the execution pattern of adaptive algorithms highly depends on the video content, which makes the achievable performance hard to be predicted and prevents efficient load balancing. Moreover, the dependency on the video content causes branch divergence for matching candidates examined on different GPU cores, resulting in attaining very poor GPU performance. In fact, even the state-of-the-art approaches dealing with the GPU parallelization of the adaptive algorithms [3] (namely UMHexagonS [2]) were unable of achieving better performance than CPU implementations.



(a) Quad-core Intel i7 950 (CPU_N). (b) NVIDIA GeForce GTX580 (GPU_F).

Figure 2: Execution share of H.264/AVC inter-loop modules in the total encoding time for 1080p HD sequences, 4 RFs and SA of 32×32.



(a) Quad-core Intel i7 950 (Nehalem). (b) NVIDIA GeForce GTX580 (Fermi).

Figure 3: Execution share of remaining H.264/AVC inter-loop modules on different device architectures.

Furthermore, to exploit the fine-grained data-level parallelism required for efficient GPU parallelization and collaborative video encoding in CPU+GPU platforms, it is also required to provide a sufficient amount of data-independent computations that can be simultaneously processed on hundreds of GPU cores. Accordingly, to relax spatial data dependences imposed by the definition of SA center, a set of temporary dependent predictors was analyzed in [12]. It was observed that the best MV found for the 16×16 partitioning mode in the previous frame for the collocated MB represents a good compromise for the SA center predictor. In fact, this predictor is only used herein to compute the SA center, while the selected MVs are then post-computed according to real median vectors of the neighboring MBs.

In order to experimentally assess the **contributions of individual modules** to the overall video encoding time, the initial inter-loop encoding was performed for an 1080p HD video sequence on two different device architectures, namely on quad-core Intel i7 950 processor (Nehalem) and NVIDIA Fermi GeForce GTX580 (Fermi). During the initial evaluation on each device architecture, the parallelized modules were used and the inter-loop encoding was performed with 4 RFs, the SA size of 32×32 pixels, and FSBM. As it can be observed in Fig. 2, the *inter-prediction modules* (i.e., ME+INT+SME) participate with more than 95% in overall encoding time, for both CPU and GPU parallel implementations. Consequently, their efficient execution is crucial to achieve real-time video encoding on target desktop systems. It is worth noting that the participation of the ME in the overall inter-loop share highly depends on the selected encoding parameters, such as the number of RFs, SA size, and the search algorithm. However, the conclusions provided herein can be equally applied to any selected parameters, considering the clear dominance of the inter-prediction modules.

For simplicity, the *Remaining Modules*, i.e., MC, TQ, TQ^{-1} and DBL, are referred herein as R* modules and their share in the overall encoding time is typically less than 5% (see Fig. 2). In addition, Fig. 3 depicts the breakdown of the computational requirements for each R* module. As it can be observed, in both CPU and GPU parallel implementations the DBL represents the dominant module, with more than 50% in overall computational share.

Algorithm 1 Collaborative Inter-loop video encoding for heterogeneous CPU+GPU systems

1: define the initial distributions
2: **for** all inter frames **do**
3: perform inter-loop for the defined distributions
4: initialize/update the performance models
5: define module-level distribution for R* modules
6: perform the load balancing for inter-prediction modules
7: **end for**

IV. Collaborative Inter-loop Video Encoding

The collaborative inter-loop video encoding for heterogeneous CPU+GPU systems, proposed herein, provides unified execution environment that dynamically instantiates the parallel CPU and GPU algorithms for individual inter-loop modules. It is implemented in OpenMP and CUDA programming models in order to attain high execution control and to maximally exploit the parallelization potential of CPU+GPU system.

According to the analysis provided in Section III, the collaborative inter-loop video encoding is performed at a frame level in several steps. As presented in Algorithm 1, in the first step (Algorithm 1 line 1) the initial cross-device load distributions are defined for all inter-loop modules. In detail, for the ME, SME and INT modules, the equidistant data partitioning is performed, while the R* modules are assigned for execution on all processing devices in their entirety. This evaluation is conducted in order to build the initial FPMs for each device/inter-prediction module pair, as well as to assess the performance disparity among heterogeneous devices for each R* module. Based on this characterization, for each subsequent inter-frame, the inter-loop video encoding is performed on the target CPU+GPU system according to newly determined load distributions (line 3). Afterwards, the execution and data transfer times are recorded for each device/module pair and the corresponding performance models are updated (line 4).

In the proposed method, different scheduling approaches are applied for the inter-prediction (ME, SME and INT) modules and for the R* modules. For the R* modules, the cross-device distribution of entire modules is determined with Dijkstra algorithm [4], such that the overall encoding time is minimized (line 5). On the other hand, the computational load of the inter-prediction (ME+SME+INT) sequence is distributed among all the processing devices according to the dynamically built partial estimations of the full FPMs (line 6).

IV.1 Distribution for the R* modules

In this procedure, each of the least computationally intensive R* modules (MC, TQ, TQ^{-1} and DBL) is mapped to a processing device, such that the overall encoding time is minimized. This procedure also reflects the device-module execution affinities and the required

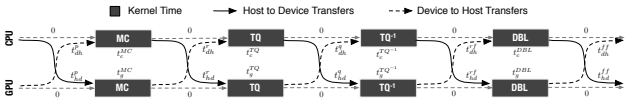


Figure 4: Data-flow diagram and weighted DAG structure of H.264/AVC encoding procedure for mapping the remaining R modules.*

data transfers for possible migration of the encoding procedure among the processing devices.

The implementation of this procedure is illustrated in Fig. 4 for a typical CPU+GPU system. Initially, a data-flow diagram is constructed, such that both processing and data transfer times (in each direction) are included for each R* module and for each devices in the system. In fact, such data-flow diagram is a weighted DAG that encapsulates all possible communication paths between the accelerators and the CPU. In detail, t^{MC}, t^{TQ}, $t^{TQ^{-1}}$ and t^{DBL} represent the time required to process each R* module, i.e. MC, TQ, TQ^{-1} and DBL, respectively, on different device architectures (where index c designates CPU and index g GPU device). In Fig. 4, the edges represent the input/output data transfers to/from a certain device, namely: *i*) t^p is the transfer time of needed MVs from SME and/or SFs from INT to perform the MC; *ii*) t^r is the transfer time of produced residual data to initiate TQ; *iii*) t^q is the time required to transfer quantization coefficients for TQ^{-1}; and *iv*) t^{rf} and t^{ff} represent the transfer times of reconstructed and filtered frames, respectively. The transfer direction is designated by *hd* or *dh* indices to represent the transfers occurring from the CPU (host) to the GPU (device) or from the GPU to the CPU, respectively.

As soon as the DAG is constructed, the minimal path between the first and last node is found. This path represents the optimal mapping of the modules to the processing devices in the CPU+GPU system. On the other hand, the sum of the weights of the edge within the path represents the prediction for the smallest R* encoding time achievable by applying the proposed method. Considering the fixed and limited number of DAG nodes, the optimal Dijkstra's algorithm [4] is applied to find the minimal path.

As it can be observed in Fig. 2, individual R* modules might have different device affinities, according to their data dependencies and parallelization potential, as well as the characteristics of target devices (number of cores, memory hierarchy etc.). However, the migration of R* sequence among devices rarely compensate the imposed data transfers, thus the entire R* sequence is usually performed on a single (fastest) device. For simplicity, in the remaining text, it will be assumed that the R* modules are processed on a single device. According to the selected device/architecture, the applied scheduling will be considered as *GPU-centric* and *CPU-centric*. However, it is worth emphasizing that this simplification is only introduced for presentation purposes and it does not influence the generality of the proposed load balancing approach.

IV.2 Load balancing for inter-prediction modules

The load balancing for inter-prediction (ME+INT+SME) sequence proposed herein relies on data parallelism at the level of the parts of the frame. In detail, it considers that each of the k CPU cores and w GPU accelerators (i.e. p_i processing devices, where i=$\{1,..,k+w\}$) performs the same algorithm on different parts of the input buffers.

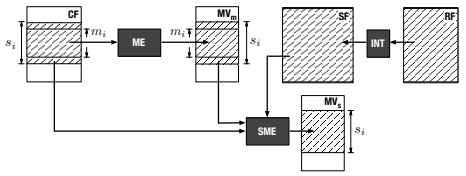


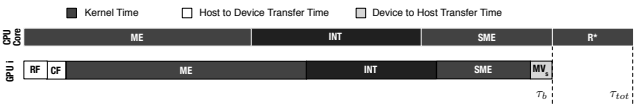
Figure 5: Data access management for collaborative processing of ME+SME+INT sequence.

As it is depicted in Fig. 5, the CF is partitioned among all CPU and GPU devices, such that the ME is collaboratively performed on the assigned CF portions, in order to produce the respective parts of MV_m buffer. The MV_m buffer is further partitioned among devices, to collaboratively perform the SME module. The simultaneously produced MVs by SME on different devices, are then collected in the MV_s buffer in the CPU main memory. It is worth noting that while the CPU can directly access the buffers from the main memory, for the GPUs explicit data transfers need to be performed.

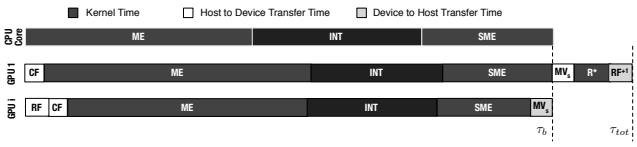
In the proposed method, the per-device load distributions are determined at the level of MB-rows. The major rationale behind adopting such granularity lies in the fact that it provides low scheduling overheads, while efficiently exploiting bandwidth of communication lines and device performance. In contrast, at the finer-grained level, the latency might dominate the execution, and the inevitable repacking is required of the original frame format from a matrix/array of pixels (in raster scan order) to an array of structures (MBs).

In order to eliminate the cost of expensive data-transfers for the SF (16 times larger than CF and RF), the execution of INT module is replicated on all devices. In fact, since the INT procedure is much faster than the corresponding data transfers, this replication also allows minimization of the overall inter-prediction time. Hence, the list of complete SFs is kept updated on each processing device. Accordingly, the distribution of the SME workload, considers only the input and output transfers of the full-pixel and quarter-pixel MVs, respectively. In order to minimize the memory requirements, both lists of SFs and RFs are updated in the form of FIFO circular buffers, where the newest SF/RF replaces the oldest one.

In Fig. 6, the proposed scheduling method is presented in two variants, i.e., *CPU-centric* (Fig. 6(a)) and *GPU-centric* (Fig. 6(b)). In



(a) CPU centric.



(b) GPU centric.

Figure 6: Scheduling strategy for collaborative inter-loop video encoding.

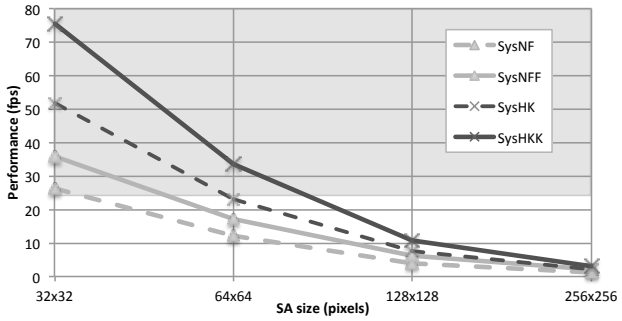


Figure 7: Performance of the proposed method for different SA sizes, single RF and 1080p resolution.

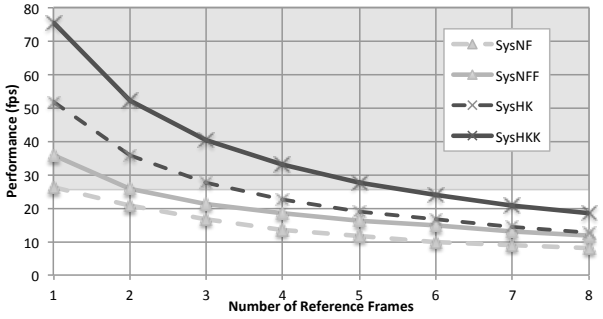


Figure 8: Performance of the proposed method for different number of RFs, 32×32 pixels SA and 1080p resolution.

both cases, there are only two synchronization points for video encoding inter-loop, namely: *i*) τ_b at the end of collaborative processing of inter-prediction sequence; and *ii*) τ_{tot} at the end of inter-loop.

For both variants, the inter-prediction processing on the GPU that does not perform R* modules (i.e., GPU_i) also requires the initial transfer of the CF and RF input buffers, and the output transfer of the produced part of MV_s buffer (see Fig. 6 and 5). However, for the GPU that processes the R* modules (i.e., GPU_1 in Fig. 6(b)), the output transfer of the MV_s buffer is not required, since the MVs are only needed on the device that performs the MC module. However, since the MVs produced on the other devices must be collected, the input transfer for the remaining part of the MV_s buffer is required after the τ_b point. At the end of R* sequence, the produced RF must be returned to the CPU, in order to allow processing of the next inter-frame on other devices. Due to the replication of the INT module, the transfer of SF buffer is not required.

In each iteration (inter-frame), the distribution vectors $m=\{m_i\}$ and $s=\{s_i\}$ are determined, where the m_i and s_i represent the number of MB-rows assigned to the ME and SME, respectively, for each processing device p_i. The m and s distribution vectors are determined by the application of the MSLBA algorithm [7] and by relying on dynamically built FPMs for each device-module pair. The partial estimations of the full FPMs are constructed by applying piece-wise linear approximation on a minimum set of points, i.e., by considering the performance obtained in previous iterations, and the asymmetric bandwidth of communication lines. In this algorithm the distributions are firstly found in the real domain, while the final integer distributions (m and s) are obtained in a refinement procedure [7].

V. Experimental Results

For the evaluation purposes, the proposed method is integrated in JM 18.6 reference coder [8]. The Baseline Profile was applied, and two different quantizer values (28 and 33). Also, two different 1080p sequences are tested, namely *RollingTomatoes* and $Sunflower$. Presented values are the average performance obtained for these parameters/sequences. However, it is worth noting that the obtained performance does not significantly depend on the video content. The tests were executed on different desktop platforms composed of the following devices: Intel i7 4770K (Haswell) CPU, Intel Core i7 950 (Nehalem) CPU, NVIDIA Tesla K40c (Kepler) GPU and NVIDIA GeForce GTX 580 (Fermi). The system composed of single Haswell CPU and single Kepler GPU is assigned as *Sys_HK*, while the system composed of the same CPU and two Kepler GPUs is *Sys_HKK*. On the other hand, the system composed of a single Nehalem CPU and a single Fermi GPU is assigned as *Sys_NF*, while the system composed of the same CPU and two Fermi GPUs is *Sys_NFF*.

Figure 7 presents the average performance in frames per second (fps) obtained with the proposed method, when considering different SA sizes, single RF and full HD (1080p) video sequences. As it can be observed, a real time encoding (more than 25 fps) was achieved in all tested systems for 32×32 pixels SA. In *SysHKK* platform, a real-time encoding (more than 33 fps) was achieved even for more demanding 64×64 pixels SA, while in *SysHK* a near real-time performance was achieved (more than 22 fps) with the same coding parameters.

Figure 8 shows the experimentally achieved average performance (in fps) with the proposed method in different systems and for different number of RFs and 32×32 pixels SA. As it can be observed, all the systems except the *Sys_NF* were able of achieving a real-time performance of more than 25 fps for multiple RFs. Moreover, in *Sys_HKK* a real-time performance for up to 5 RFs was achieved.

In addition to the ability of the proposed method to achieve a real-time performance for very demanding coding parameters, Fig. 7 and 8 also show that the proposed method is scalable over both the SA size and the number of RFs. This is achieved mainly due to the high efficiency of the proposed load balancing approach and the developed highly optimized parallel CPU and GPU algorithms.

VI. Conclusions

In this study, an efficient method for collaborative H.264/AVC inter-loop in heterogeneous CPU+GPU systems was proposed. In order to cope with the inherent data-dependencies and computational complexity of inter-loop modules, a unified execution environment was designed to ensure efficient cross-device execution and to guarantee the correctness of the video encoding process. It also includes an extensive library of highly optimized parallel algorithms for all inter-loop video encoding modules, which are developed using the device-specific programming models and tools. The integrated scheduling and load balancing routines allow efficient distribution of the workloads for these modules across all processing devices. For the most computationally intensive inter-prediction modules,

the proposed adaptive load balancing relies on realistic and dynamically built FPMs of both communication and computation system resources. The workloads of the remaining modules are distributed according to their module-device affinities by applying the optimal Dijkstra algorithm. In order to minimize the overall inter-loop encoding time, the proposed collaborative encoding method also integrates data access management and specific, communication-aware replication techniques to maximize data reuse, while decreasing the data transfers overheads. The experimental results shown that the proposed method is able of achieving real-time video encoding for full HD resolution, with a 64×64 pixels SA and exhaustive ME on the state-of-the-art commodity CPU+GPU platforms. Moreover, the scalability of the proposed method over the SA size, number of RFs and number of processing devices was experimentally shown.

Acknowledgment

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects PEst-OE/EEI/LA0021/2013, PTDC/EEI-ELC/3152/2012 and PTDC/EEA-ELC/117329/2010.

References

[1] G. Barlas, A. Hassan, and Y. Al Jundi. An analytical approach to the design of parallel block cipher encryption/decryption: A CPU/GPU case study. In *Proceedings of the Int. Conf. on Par., Dist. and Network-Based Proc. (PDP)*, pages 247–251, 2011.

[2] Z. Chen, J. Xu, Y. He, and J. Zheng. Fast integer-pel and fractional-pel motion estimation for H.264/AVC. In *Journal of Visual Communication and Image Representation*, pages 264–290, October 2005.

[3] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung. Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2):79–89, 2010.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Num. Math.*, 1(1):269–271, Dec. 1959.

[5] A. Ilic, S. Momcilovic, N. Roma, and L. Sousa. FEVES: Framework for efficient parallel video encoding on heterogeneous systems. In *Proceedings of the International Conference on Parallel Processing*, pages 165–174, 2014.

[6] A. Ilic and L. Sousa. Scheduling divisible loads on heterogeneous desktop systems with limited memory. In *Proceedings of the Euro-Par Workshops*, pages 491–501, 2012.

[7] A. Ilic and L. Sousa. Simultaneous multi-level divisible load balancing for heterogeneous desktop systems. In *Proceedings of the IEEE Int. Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 683–690, 2012.

[8] ITU-T. *JVT Reference Software unofficial version 18.6*. http://iphome.hhi.de/suehring/tml/, 2014.

[9] Y. Ko, Y. Yi, and S. Ha. An efficient parallelization technique for x264 encoder on heterogeneous platforms consisting of CPUs and GPUs. *Journal of Real-Time Image Proc.*, pages 1–14, 2013.

[10] A. Lastovetsky and R. Reddy. Distributed data partitioning for heterogeneous processors based on partial estimation of their functional performance models. In *Proceedings of the Euro-Par*, pages 91–101, 2010.

[11] Ping Li, B. Veeravalli, and A.A. Kassim. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Trans. on Circuits and Systems for Video Technology*, 15(9):1098 – 1112, September 2005.

[12] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa. Collaborative inter-prediction on cpu+gpu systems. In *Proceedings of the IEEE International Conference on Image Processing*, 2014.

[13] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa. Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems. *IEEE Transactions on Multimedia*, 16(1):108–121, Jan 2014.

[14] S. Momcilovic, N. Roma, and L. Sousa. Multi-level parallelization of advanced video coding on hybrid CPU+GPU platforms. In *Proceedings of the Euro-Par Workshops*, pages 165–174, 2012.

[15] S. Momcilovic, N. Roma, and L. Sousa. Exploiting task and data parallelism for advanced video coding on hybrid cpu+ gpu platforms. *Journal of Real-Time Image Proc.*, pages 1–17, 2013.

[16] J. Ohm and G.J. Sullivan. High efficiency video coding: the next frontier in video compression [standards in a nutshell]. *Signal Processing Magazine, IEEE*, 30(1):152–158, Jan 2013.

[17] J. Ostermann et al. Video coding with H.264/AVC: tools, performance, and complexity. *IEEE Circuits and Systems Magazine*, 4(4):7–28, April 2004.

[18] B. Pieters, C. F. Hollemeersch, P. Lambert, and R. Van De Walle. Motion estimation for H.264/AVC on multiple GPUs using NVIDIA CUDA. In *Proceedings of the Society Photo-Optical Instrumentation Engineers*, page 12, 2009.

[19] R. Rodríguez-Sánchez, J. L. Martínez, G. Fernández-Escribano, J. L. Sánchez, and J. M. Claver. A fast GPU-based motion estimation algorithm for H. 264/AVC. In *Advances in Multimedia Modeling*, pages 551–562. Springer, 2012.

[20] B. Veeravalli, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6:7–17, 2003.

[21] H. Wei, J. Yu, and J. Li. The design and evaluation of hierarchical multi-level parallelisms for H.264 encoder on multi-core architecture. *Comput. Sci. Inf. Syst.*, 7(1):189–200, 2010.

[22] N. Wu, M. Wen, J. Ren, H. Su, and D. Huang. High-performance implementation of stream model based H.264 video coding on parallel processors. In *Multimedia and Signal Processing*, volume 346, pages 420–427. Springer Berlin Heidelberg, 2012.

[23] J. Zhang, J. F Nezan, and J.-G. Cousin. Implementation of Motion Estimation Based on Heterogeneous Parallel Computing System with OpenCL. In *Proceedings of the IEEE Int. Conf. on High Perf. Comp. and Comm.*, pages 41–45, 2012.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop. October. 2014

On Efficiency of the OpenFOAM-based Parallel Solver for the Heat Transfer in Electrical Power Cables

Raimondas Čiegis, Vadimas Starikovičius, Andrej Bugajev

Vilnius Gediminas Technical University, Lithuania
raimondas.ciegis@vgtu.lt, vadimas.starikovicius@vgtu.lt, andrej.bugajev@vgtu.lt

Abstract

In this work, we study the efficiency of the OpenFOAM-based parallel solver for the heat conduction in electrical power cables. The 2D benchmark problem with three cables is used for our numerical tests. We study and compare the efficiency of conjugate gradient solver with diagonal incomplete Cholesky (DIC) preconditioner and generalized geometric-algebraic multigrid solver (GAMG), which is available in OpenFOAM. The convergence and parallel scalability of the solvers are presented and analyzed. Parallel numerical tests are performed on the cluster of multicore computers.

Keywords OpenFOAM, parallel algorithms, domain decomposition, preconditioner, multigrid

I. Introduction

The knowledge of heat generation and distribution in and around the high-voltage electrical cables is necessary to optimize the design and exploitation of electricity transferring infrastructure. Engineers are interested in maximum allowable current in different conditions, optimal cable parameters, cable life expectancy estimations and many other engineering factors.

Presently applicable IEC standards for the design and installation of electrical power cables are often based on the analytical and heuristic formulas. Obviously, these formulas cannot accurately account for the various conditions under which the cables are actually installed and used. They estimate the cable's current-carrying capacity (so-called *ampacity*) with significant margins to stay on the safe side [1]. The safety margins can be quite large and result in 50-70% usage of actual resources. A more accurate mathematical modelling is needed to meet the latest technical and economical requirements and to elaborate new, improved, cost-effective design rules and standards.

When we need to deal with mathematical models for the heat transfer in various media (metals, insulators, soil, water, air) and non-trivial geometries, only the means of parallel computing technologies can allow us to get results in an adequate time. To solve numerically selected models, we develop our numerical solvers using the OpenFOAM package [2]. OpenFOAM is a free, open source CFD software package. It has an extensive set of standard solvers for popular CFD applications. It also allows us to implement our own models, numerical schemes and algorithms, utilizing the rich set of OpenFOAM capabilities [3]. However, application of OpenFOAM libraries for solving specific problems still requires an appropriate theoretical and empirical analysis and a nontrivial selection of optimal algorithms. Examples of such problems are described in [4] and [5].

The important consequence of this software development approach is that obtained application solvers can automatically exploit the parallel computing capabilities already available in the OpenFOAM package. Parallelization of OpenFOAM is based on MPI (Message Passing Interface) standard. However, the modular structure of this package allows the development of parallel applications for modern hybrid and heterogeneous high-performance computing (HPC) platforms. See, for example, [6], for CUDA applications on GPU.

In recent years, scalability and performance of parallel OpenFOAM solvers are actively studied for various applications and HPC platforms. In [7] it is noted that the scalability of parallel OpenFOAM solvers is not very well understood for many applications when executed on massively parallel systems.

We note that an extensive experimental scalability analysis of selected OpenFOAM applications is one of the tasks solved in PRACE (Partnership for Advanced Computing in Europe) project, see [8], [9]. In [8] are presented results on IBM BlueGene/Q (Fermi) and Hewlett Packard C7000 (Lagrange) parallel supercomputers for a few CFD applications with different multi-physics models. The presented experimental results are showing a good scaling and efficiency with up to 2048–4096 cores. It is noted that such results are expected when balancing between computation, message passing and I/O work is good. Obviously, the next generation of ultrascale computing systems will cause additional challenges due to their complexity and heterogeneity.

In this work, we study and analyze the performance of OpenFOAM-based parallel solver for the heat conduction in electrical power cables. Two linear system solvers are considered and compared, namely, the conjugate gradient solver with diagonal incomplete Cholesky (DIC) preconditioner and generalized geometric-algebraic multigrid solver (GAMG), which is available in OpenFOAM. We study the convergence and parallel scalability of the linear solvers, the sensitivity of parallel preconditioners with respect to the grid size and the number of processes. Another goal is to study the ability of OpenFOAM to efficiently deal with the heterogeneity of the computer cluster.

In Section II, we shortly describe the problem, mathematical model and benchmark problem used for our parallel numerical tests. In Section III, we describe our OpenFOAM-based solver and discuss the parallelization approach employed in the OpenFOAM package. In Section IV, we present and analyze the obtained results on convergence and scalability of the considered linear solvers for our application. Finally, some conclusions are drawn in Section V.

II. Benchmark problem

As a benchmark problem in this research we solve the heat conduction problem for electrical power cables directly buried in the soil. It is also assumed that the thermo-physical properties of the soil remain constant, i.e. the moisture transfer in the soil is not considered. Such a simplified problem is described by the following well-known mathematical model:

$$\begin{cases} c\rho\dfrac{\partial T}{\partial t} = \nabla\cdot(\lambda\nabla T) + q, & t\in[0,t_{max}], \vec{x}\in\Omega, \\ T(\vec{x},0) = T_b, & \vec{x}\in\Omega, \\ T(\vec{x},t) = T_b, & \vec{x}\in\partial\Omega, \\ T, \lambda\nabla T \ \text{ are continuous}, & \vec{x}\in\Omega, \end{cases} \tag{1}$$

where $T(\vec{x},t)$ is the temperature, $c(\vec{x}) > 0$ is the specific heat capacity, $\rho(\vec{x}) > 0$ is the mass density, $\lambda(\vec{x}) > 0$ is the heat conductivity coefficient, $q(\vec{x},t,T)$ is the heat source function due to power losses, T_b is the initial and boundary temperature. Coefficients $\lambda(\vec{x})$, $c(\vec{x})$, $\rho(\vec{x})$ are discontinuous functions. Their real values can vary between metallic conductor, insulators and soil by several orders of magnitude [1].

In this work, we have used 2D geometry for our benchmark problem. Three cables are buried in the soil as shown in figure 1. The red area is metallic conductor, the blue area is an insulator and the gray area marks the soil.

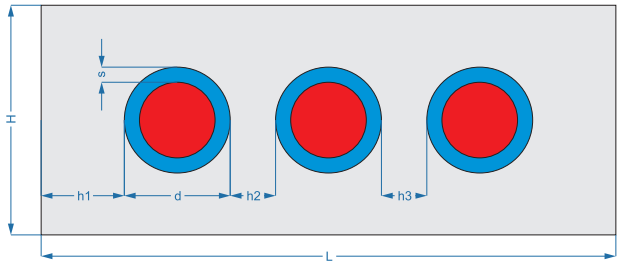


Figure 1: 2D geometry of benchmark problem: three cables in the soil.

III. Parallel OpenFOAM-based solver

OpenFOAM (Open source Field Operation And Manipulation) [2] is a C++ toolbox (library) for the development of customized numerical solvers for partial differential equations (PDEs). For numerical solution of PDEs OpenFOAM uses the Finite Volume Method (FVM) with co-located arrangement of unknowns [3].

We obtain a numerical solver for our benchmark problem (1) by the slight modification of the standard *laplacianFoam* solver: adding variable problem coefficients $c(\vec{x})$, $\rho(\vec{x})$, $\lambda(\vec{x})$ and a nonlinear source term $q(\vec{x},t,T)$ dependent on temperature T. Note that a proper interpolation should be used for calculation of discontinuous coefficient $\lambda(\vec{x})$ on conductor-insulator-soil interface walls of according finite volumes. We employ the *harmonic* interpolation to ensure the continuity of heat flux $\lambda\nabla T$.

We generate the finite volume mesh in problem domain Ω using OpenFOAM preprocessing tool and obtain FVM discretization with the four point stencil for the 2D problem. Resulting systems of linear equations with symmetric matrices can be solved by preconditioned conjugate gradient method with various preconditioners and multigrid method.

Parallelization in OpenFOAM is robust and implemented at a low level using the MPI library. Solvers are built using high level objects and, in general, don't require any parallel-specific coding. They will run in parallel automatically. Thus there is no need for users to implement standard steps of any parallel code: decomposition of the problem into subproblems, distribution of these tasks among different processes, implementation of data communication methods. A drawback of such automatic tools is that the user has very limited possibilities to modify the generated parallel algorithm if the efficiency of the OpenFOAM parallel code is not sufficient.

OpenFOAM employs a common approach for parallelization of numerical algorithms – domain decomposition. The mesh and its associated fields are partitioned into sub-domains, which are allocated to different processes. Parallel computation of the proposed finite FVM algorithm requires two types of communication. First type is the local communication between neighboring processes for approximation of the Laplacian term on the given stencil and matrix-vector multiplication. Second type is the global communication between all processes for computation of scalar products in iterative linear solvers.

OpenFOAM employs a zero-halo layer approach [6], which considers cell edges on sub-domain boundaries as boundary and applies a special kind of boundary condition.

OpenFOAM supports four methods of domain decomposition, which decompose the data into non-overlapping sub-domains: simple, hierarchical, scotch and manual [2]. For our parallel tests the mesh is partitioned by using Scotch library [10]. Scotch is a library for graph and mesh partitioning, similar to well-known Metis library [11]. It requires no geometric input from the user and attempts to minimize the number of boundary edges between subdomains. The user can specify the weights of the sub-domains, what can be useful on heterogeneous clusters of parallel computers with different performance of processors. We will use this feature solving our benchmark problem on heterogeneous cluster.

IV. Parallel performance tests and analysis

In our previous work [12], we have investigated the theoretical complexity and scalability of our OpenFOAM based parallel solver with preconditioned conjugate gradient method. To validate the theoretical model in numerical tests, we have fixed the number of iterations for solving systems of linear equations – 1000. In this way, we have ensured that the same amount of work was done in all parallel tests (10 time steps with 1000 iterations), despite the possible differences in convergence due to parallel preconditioning and different round-off errors, due to data communication.

In this study we want to investigate the convergence of conjugate gradient linear solver with diagonal incomplete Cholesky preconditioner (DIC/CG) and compare to the convergence of generalized geometric-algebraic multigrid solver (GAMG). We want to study the scalability of both linear solvers and the influence of the mesh partitioning on parallel preconditioners. So, now the tolerance of linear solvers is fixed and set to 10^{-6}. In each test, we do 10 time steps with different number of iterations, which is dependent on convergence.

	p	1×1	2×1	4×1	8×1	8×2
Mesh size - 1018488						
DIC/CG	N_p^{av}	721.9	839.3	865.4	942.4	1009.7
	T_p	265.1	161.7	77.0	41.1	28.5
	T_p^{av}	0.0367	0.0192	0.0089	0.0044	0.0028
GAMG	N_p^{av}	20.4	22.6	25.2	37.0	59.2
	T_p	49.7	30.7	20.3	25.3	44.8
	T_p^{av}	0.2436	0.1356	0.0807	0.0684	0.0757
Mesh size - 2048000						
DIC/CG	N_p^{av}	1015.6	1140.1	1142.6	1448.2	1401.6
	T_p	799.3	437.1	237.0	133.4	87.2
	T_p^{av}	0.0787	0.0383	0.0207	0.0092	0.0062
GAMG	N_p^{av}	19.4	27.0	27.7	39.0	39.3
	T_p	94.6	70.8	41.5	39.7	39.2
	T_p^{av}	0.4874	0.2622	0.1497	0.1017	0.0997
Mesh size - 4102264						
DIC/CG	N_p^{av}	1427.0	1452.0	1928.4	1939.4	1717.8
	T_p	2119.4	1114.9	872.2	414.6	265.8
	T_p^{av}	0.1485	0.0768	0.0452	0,0214	0,0155
GAMG	N_p^{av}	24.6	25.9	41.0	40.3	47.0
	T_p	243.9	137.0	120.2	68.9	69.7
	T_p^{av}	0.9913	0.5291	0.2932	0.1708	0.1482
Mesh size - 8192000						
DIC/CG	N_p^{av}	-	2468.9	2463.9	2535.1	2635.1
	T_p	-	3933.5	2165.9	1126.8	829.6
	T_p^{av}	-	0.1593	0.0879	0,0445	0,0315
GAMG	N_p^{av}	-	38.2	33.8	39.0	41.3
	T_p	-	402.5	193.3	123.67	100.6
	T_p^{av}	-	1.0536	0.5720	0.3171	0.2436

Table 1: The average number of iterations N_p^{av} with p processes, the total wall time T_p of 10 time steps with p processes, the average time of one iteration - $T_p^{av} = T_p/N_p^{av}/10$ with p processes for DIC/CG and GAMG linear solvers with tolerance 10^{-6}. Here $p = n_d \times n_c$ is the number of parallel processes using n_d nodes with n_c cores per node.

Parallel numerical tests were performed on the Vilkas cluster of Vilnius Gediminas technical university. We have used eight nodes with Intel Core i7-860 processors, 4 cores (2.80 GHz) per node. Computational nodes are interconnected via Gigabit Smart Switch.

In table 1, we present the average number of iterations N_p^{av}, the total wall time T_p of 10 time steps, the average time of one iteration - $T_p^{av} = T_p/N_p^{av}/10$ obtained with p processes. Results are presented for both linear solvers and increasing finite volume mesh. Size of the mesh (the number of finite volumes) was almost doubled for each next test.

Note that the case $p = 1\times1$ provides data on convergence and elapsed wall time for the sequential algorithms. There are no data for the mesh size 8192000, because this case does not fit into memory available on the single node. As we can see, the number of iterations of DIC/CG solver is increasing as $\sqrt{2}$ when the mesh size is doubled. This is in accordance with the theory. The number of iterations of GAMG solver is also growing with the mesh size, but not as much as in the case of DIC/CG, compare $24.6/20.4 \approx 1.21$ to $1427/721.9 \approx 1.99$ (2 expected).

Comparing the elapsed times T_p we see that the GAMG solver is a faster alternative that DIC/CG solver. The GAMG solver achieves the biggest advantage over the DIC/CG solver at sequential tests. For parallel tests, the advantage is constantly decreasing with increasing number of processes - p. It seems that there are two reasons for this. First reason is the degradation in performance of parallel preconditioner. For parallel DIC preconditioner with conjugate gradients, the number of iterations is quite gradually increasing up to 40% going from 1 to 16 processes. At the same time the number of GAMG iterations exibits significant jumps, altogether up to 2-3 times going from 1 to 16 processes.

Another reason for the significant degradation of parallel performance of GAMG solver is algorithmic scalability of its parallel algorithm. Comparing the average times of one iteration with p processes - T_p^{av}, we see that the parallel algorithm of DIC/CG solver scales much better than the parallel algorithm of GAMG solver.

Next, we want to study the ability of our OpenFOAM-based parallel solver to utilize the full power of heterogeneous cluster made of computational nodes of different speeds. For these numerical tests we have used eight additional nodes with Intel Quad Q6600 processors, 4 cores (2.4 GHz) per node.

Computing nodes with Intel Core i7-860 processors are up to 1.6-1.7 times faster than the Intel Quad Q6600 processors solving our benchmark problem (see [12]). To achieve the load balancing between *i7* and q nodes, we employ the weights in the mesh partitioning algorithm from the Scotch library [10]. The performance results of the tests on heterogeneous cluster are presented in table 2.

Solving our biggest case in this work with 8192000 finite volumes, we obtain a real speedup only with DIC/CG linear solver. The performance of GAMG linear solver is further degrading. It is interesting to note that slight changes in the weighting factor w, can cause significant changes in the convergence of GAMG: from 43.2 to 51.7 iterations per time step with 16×1 processes.

V. Conclusions

We have tested the parallel performance of the conjugate gradient solver with diagonal incomplete Cholesky preconditioner (DIC/CG) and generalized geometric-algebraic multigrid (GAMG) solver in OpenFOAM-based application for the heat conduction in

Mesh size - 8192000							
	p	16×1	16×1	16×2	16×2	16×4	16×4
	$i7(w)$	1.6	1.7	1.6	1.7	1.6	1.7
DIC/CG	N_p^{av}	2340.7	2589.5	2790.7	2659.9	2630.2	2643.9
	T_p	647.2	728.6	574.6	536.6	540.7	524.7
	T_p^{av}	0.0277	0.0281	0.0206	0.0202	0.0206	0,0199
GAMG	N_p^{av}	43.2	51.7	53.8	55.2	56.2	60.7
	T_p	144.8	170.7	135.4	135.6	229.0	256.3
	T_p^{av}	0.3351	0.3301	0.2517	0.2457	0.4075	0.4223

Table 2: The average number of iterations N_p^{av} with p processes, the total wall time T_p of 10 time steps with p processes, the average time of one iteration - $T_p^{av} = T_p/N_p^{av}/10$ with p processes for DIC/CG and GAMG linear solvers with tolerance 10^{-6}. Here $p = n_d \times n_c$ is the number of parallel processes using n_d nodes with n_c cores per node, $i7(w)$ is the weighting factor used in mesh partitioning algorithm for faster i7 nodes.

electrical power cables. The best running times in our tests were obtained with GAMG solver. However, DIC/CG linear solver has shown to be less sensitive to the parallel preconditioning degradation. It has also shown better algorithmic scalability.

DIC/CG linear solver is to be recommended for the parallel computations on parallel computing systems with large number of processors and cores. Additional options of GAMG solver need to be investigated to improve its scalability. Parallel performance of the conjugate gradient solver with GAMG preconditioner (instead of DIC) is to be studied in future work.

The weighting factors in mesh partitioning algorithm allow efficient utilization of heterogeneous computing nodes for our parallel application. More research is needed on the influence of different graph partitioning algorithms.

Acknowledgment

The work of authors was supported by Eureka project E!6799 POWEROPT "Mathematical modelling and optimization of electrical power cables for an improvement of their design rules".

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] I. Makhkamova, *Numerical Investigations of the Thermal State of Overhead Lines and Underground Cables in Distribution Networks*, Ph.D. thesis, Durham University, Durham, United Kingdom, 2011.

[2] OpenFOAM, *Open source Field Operation And Manipulation*, CFD toolbox, http://www.openfoam.org.

[3] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A Tensorial Approach to Computational Continuum Mechanics Using Object-oriented Techniques," *Journal of Computational Physics*, vol. 12, no. 6, pp. 620-631, 1998.

[4] P. Higuera, J. Lara, and I. Losada, "Realistic wave generation and active wave absorption for Navier–Stokes models: Application to OpenFOAM," *Coastal Engineering*, vol. 71, no. 1, pp. 102-118, 2013.

[5] O. Petit, A. Bosioc, H. Nilsson, S. Muniean, and R. Susan-Resigo, "Unsteady simulations of the flow in a swirl generator using OpenFOAM," *International Journal of Fluid Machinery and Systems*, vol. 4, no. 1, pp. 199-208, 2011.

[6] A. AlOnazi, *Design and Optimization of OpenFOAM-based CFD Applications for Modern Hybrid and Heterogeneous HPC Platforms*, Master thesis, University College Dublin, Ireland, 2013.

[7] O. Rivera, K. Furlinger, and D. Kranzimuller, "Investigating the scalability of OpenFOAM for the solution of transport equations and large eddy simulations," *Lecture Notes in Computer Science*, vol. 7017, pp. 121-130, 2011.

[8] P. Dagna, "OpenFOAM on BG/Q porting and performance", PRACE Report, CINECA, Bologna, Italy, 2012.

[9] M. Culpo, "Current Bottlenecks in the Scalability of OpenFOAM on Massively Parallel Clusters", PRACE White Papers, CINECA, Bologna, Italy, 2012.

[10] C. Chevalier and F. Pellegrini, "PT-Scotch: A Tool for Efficient Parallel Graph Ordering," *Parallel Computing*, vol. 34, no. 6-8, pp. 318-331, 2008.

[11] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359-392, 1999.

[12] R. Čiegis, V. Starikovičius, and A. Bugajev, "On Parallelization of the OpenFOAM-based Solver for the Heat Transfer in Electrical Power Cables," in *Euro-Par 2014: Parallel Processing Workshop*, Lecture Notes in Computer Science, vol. 8805, Porto, Portugal, August 25-29, 2014.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop. • September 2014 • Vol. I, No. 1

On the Performance Of the Thread-Multiple Support Level In Thread-Based MPI

Juan-Carlos Díaz-Martín
University of Extremadura, Spain
juancarl@unex.es

Juan-Antonio Rico-Gallego
University of Extremadura, Spain
jarico@unex.es

Abstract

Exascale systems are likely to have orders of magnitude less memory per core than current systems (though still large amounts of memory). As the amount of memory per core is dropping, going to thread-based models might be an unavoidable step towards the exascale milestone. AzequiaMPI is a thread-based open source full conformant implementation of MPI-1.3 for shared memory. We expose the techniques introduced in AzequiaMPI that, first, simplify the implementation and second, make the thread-based model to significantly improve the bandwidth of process-based implementations. Current version is also compliant with the MPI_THREAD_MULTIPLE thread-safety level, a feature of MPI-2.0 standard. The well known Thakur and Gropp MPI_THREAD_MULTIPLE tests show that both latency and bandwidth figures of AzequiaMPI significantly improve that of MPC-MPI, MPICH and Open MPI in an eight cores Intel Xeon E5620 Nehalem machine.

Keywords MPI performance, Thread-based MPI, MPI_THREAD_MULTIPLE, Multicore architectures

I. Introduction

MPI [4] is an industry de-facto parallel programming standard based on the message-passing paradigm. As new languages and alternatives as Unified Parallel C (UPC) and OpenMP are being proposed for supporting more efficiently multicore architectures, MPI keeps facing the challenge. The fact is that MPI is still used on shared memory due to its better portability and its good data locality due to data partitioning. An MPI application is composed by a set of independent program instances that communicate by message passing. Traditional mainstream MPI implementations as MPICH and OpenMPI build each instance as a full-fledged process. It entails some disadvantages in shared memory because message passing between two processes must go through a per-pair intermediate shared buffer, and copying degrades the communication efficiency.

Two trends drive the performance of current HPC clusters. One is the strong increase in the amount of memory-per-node, and the other the rising number of cores per processor. However, if the count of cores per machine is doubling approximately every 2 years, the DRAM DIMM capacity is just doubling about every 3 years [5]. This means that the memory capacity per core is expected to drop by 30 % every two years. As a result, exascale systems are likely to have orders of magnitude less memory per core than current systems. Against this backdrop, going to thread-based models might be an unavoidable step towards the exascale milestone [3], because it should reduce the application memory footprint [7]. Not only that, thread-based MPI improves the MPI performance in shared memory because enables optimised communication mechanisms, as the single copy. This paper is about this issue.

It is possible building the MPI instance as a thread. Thread-based MPI has been around for years, but the fact is that no thread-based MPI implementation has been widely adopted in practice. The reasons are not fundamental, but rather practical concerns imposed by the prevalence of the OS-level process. One of them is the issue of the global variables of a MPI rank, that get shared by them all under the thread-based case, giving place to faulty programs. Though privaticing them through program transformation techniques is a well studied requirement, and some partial efforts have been undertaken in this direction [[11], [10]], the fact is that no explicit tool is currently available, or at least provided by a main stream tool chain to transparently circunvent the problem. The other argumented weakness of thread-based MPI is the thread-safety of third-party software, either library code or device drivers. It is true that non thread-safe software is unusable by theaded MPI ranks, but the same happens to the threads created under the MPI_THREAD_MULTIPLE ability of the MPI-2 standard in a process-based implementation. All in all, a not thread-safe library should be labelled as quite limited software product nowadays, whether used in an MPI context or not.

AzequiaMPI [9] is thread-based but still an open source full conformant implementation of the MPI-1.3 standard. Its original version was targeted to embedded distributed signal processing platforms of DSP and FPGAs supporting a single address space, which forced us to implement MPI upon threads. This work presents performance evaluation of current AzequiaMPI, targeted to standard HPC multicore machines. The design is rooted on the lock-free queue structure used in MPICH2-Nemesis, but exploiting the advantage of sharing a common address space. Our tests show a relevant improvement against MPICH, Open MPI and MPC-MPI in an 8 cores Nehalem machine. The rest of the paper is as follows. Related work is presented in section 2. In section 3 we introduce the core design of the system and comparative performance figures. Section 4 presents the extensions to the desing that support the MPI_THREAD_MULTIPLE option in an efficient way. Section 5 concludes.

II. Related Work

Implementing a MPI node as a thread is not a new concept, but it has received very limited attention in the literature. Seminal work [2] discusses TOMPI (for Thread Only MPI), an early proof of concept prototype that implements just a handful of MPI primitives, not even in a conformant way.

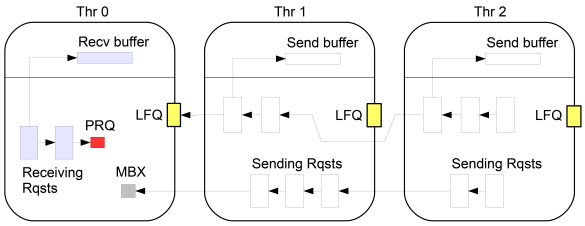


Figure 1: Requests and queues in AzequiaMPI.

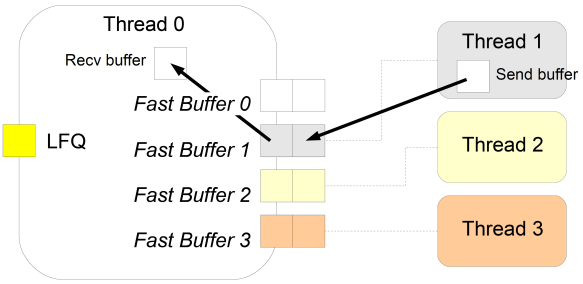


Figure 2: FastBuffers in a configuration of four threads.

TMPI (for threaded MPI) [11] is a more solid and deeper research on thread-based MPI. TMPI is also a partial implementation of MPI-1 (29 functions) addressed to clusters of multiprocessors. Authors claimed that, on the average, TMPI is 46% faster than MPICH, but note that the further Nemesis library made MPICH up to twice faster for short messages and shows an up to 1.4 factor of improvement for large messages. Unfortunately, TMPI only provides the source code of an early version based in mutexes.

MPI Actor (for MPI Accelerator) [6] is a middleware that maps MPI nodes to threads in multicore machines. It claims to patch any existing process-based MPI implementation, so that a MPI process runs as a thread in shared memory. Its big advantage would be avoiding the huge work of building a new fully conformant thread-based implementation. Unfortunately, MPI Actor source code is not publicly available. Anyway, the performance of AzequiaMPI overcomes by large that achieved by MPI Actor.

MultiProcessor Communications environment (MPC) [8], aims an efficient runtime system unifying the MPI, POSIX Threads and OpenMP. The key idea is to use user-level threads instead of processes to increase scheduling flexibility, to better control memory allocations and optimize scheduling of the communication flows with other nodes.The scheduler and memory allocator modules cooperate to preserve data locality, a crucial issue when dealing with NUMA nodes.

III. Thread-Single: Design and Performance

Fig. 1 shows the basic design of AzequiaMPI. Each MPI rank (a thread) has three queues, known as PRQ, MBX and LFQ. PRQ is the queue of pending receive requests, and MBX (for mailbox) is the queue of unexpectedly arrived send requests. Both are ordinary double-linked lists. Only its owner rank accesses to them. LFQ is a lock-free queue, the same used by MPICH2-Nemesis [1]. It allows a single receiver (its owner) and many senders. All of them access it without locking. To receive a message, the MPI rank r allocates a receiving request R from its per-thread pool. R is initialised so that a field points to the receive user buffer. Next, r explores MBX, looking for a send request S that matches R. On success, r performs the copy, updates S as satisfied, and finally dequeues and liberates R. If no matching happens, r enqueues R in its PRQ queue. If the receive primitive is blocking (MPI_Recv) r enters the progress routine, that basically polls LFQ for new events. If the primitive is non-blocking (MPI_Irecv) r returns.

To send a message, an MPI rank s allocates a sending request S from its per-thread pool. A field of S points to the send user buffer as Fig. 1 shows. Next, S is enqueued in the receiver lock-free queue LFQ. A blocking send primitive makes s entering the progress engine, that polls the state field of S until it is satisfied. Then s liberates S. A non-blocking send simply makes s to continue and defers the polling to subsequent test/wait invocations.

The progress routine basically polls LFQ for new events. When the invoking rank t dequeues a send request S from LFQ, t matches S against its PRQ. If a matching receiving request R is found, t dequeues and liberates R, makes the copy from the send buffer to the receive buffer and set S as satisfied. If a matching receive request is not found, S is enqueued in MBX.

MPICH2-Nemesis provides a mechanism called fastbox to accelerate small messages [1]. A fastbox is a small buffer associated to each pair of MPI ranks. It allows a sender to by-pass the LFQ by copying directly its message to the fastbox. After the copy is done, an integer flag acting as a turn is switched for reception. The receiver copies from the fastbox to its user buffer, switching the turn for sending. AzequiaMPI extends fastboxes to fastbuffers, as Fig. 2 illustrates. In AzequiaMPI we have observed in the respective left and right scenarios of Fig. 3 that a fastbuffer, with either one or two fastboxes, diminish the latency of small messages by half, and that a fastbuffer of two fastboxes multiplies the bandwidth of small messages by four.

Fastboxes, however, come with a price. They pose the issue of message ordering because introduce a second path between a sender and a receiver. If the receiver checks the fastbox before the LFQ or viceversa, it may receive the messages in the wrong order. This situation is handled so that every local message carries on a sequence number that is matched on reception, as it is done with source and tag. Messages up to 1 KB are delivered this way if the fast buffer is not full. Management of sequence numbers is tricky, but cheap. The true downside of fastBuffers is memory compsumtion, of order $O(Q^2)$ where Q is the number of cores per node. If the size of the fastBuffer is 2KB, the implementaton would allocate 2KB x 128 x 128 = 32 MB only to this resource on a machine of $Q = 128$ cores. To save memory AzequiaMPI creates fastbuffers on the fly, and only when really needed. Look at Fig. 2. The four fast buffers of rank 0 do not come into existence when the application starts up. Instead fastbuffer 1, for instance, is created by rank 1 when it tries to send its first small message to rank 0.

Fig. 3 shows the setup produced by the single-threaded latency and bandwidth tests from the well known Thakur and Gropp benchmark [12], in the eight cores of the Nehalem. The benchmark is included in the source distribution *gim.unex.es/azequiampi* of AzeqioaMPI. Latency is measured on a ping-pong setup, and likely bandwidth on a stream. Fig. 4 and Fig. 5 show the performance

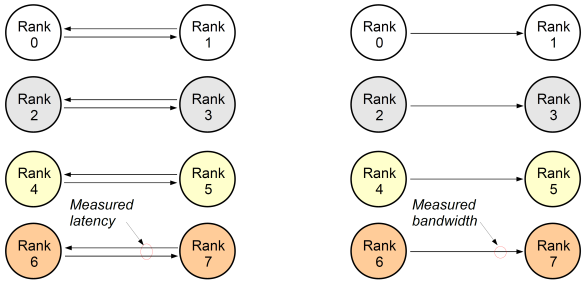


Figure 3: Thakur and Gropp single threaded tests. Experimental setup.

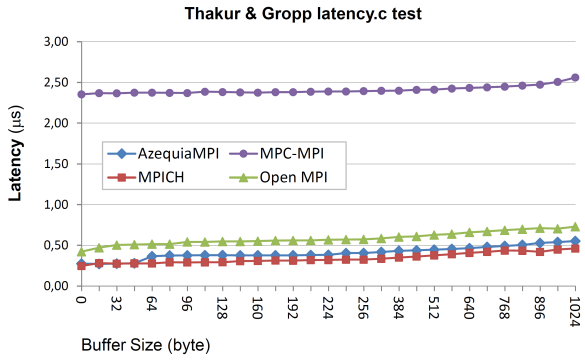


Figure 4: Latency measurements of four MPI implementations in the setup at the left side of Fig. 3.

achieved by this design, compared to that of MPICH, OpenMPI and MPC-MPI. AzequiaMPI threads, and MPICH and Open MPI processes are core bound in round-robin. It is currently not possible to bind tasks in MPC-MPI. The eight MPC-MPI threads, designed to run unbound in oversubscripted scenarios, have the eight cores available. Note that MPC-MPI is almost one order of magnitud slower in terms of latency than AzequiaMPI and MPICH. MPICH produces the best outcomes in this respect. AzequiaMPI shows the best bandwidth, partly due to an optimisation technique only possible when the send and receive user buffers share memory: both sender and receiver cooperate in the copy, so that, for instance, sender copies lower half of send buffer to lower half of receive buffer, and likely the receiver on the upper halves. We call this method "split copy" and it should be clear that it constitutes a significant advantage with respect to process based implementations. It seems that MPC-MPI does not currently exploit this technique.

IV. Thread-Multiple: Design and Performance

AzequiaMPI, being MPI-1.3 conformant, also supports the highest level of thread safety for user programs, MPI_THREAD_MULTIPLE, an MPI-2 feature that allows concurrent MPI calls from multiple threads. Turning thread-safe a MPI implementation is a problem of introducing the right set of mutexes around the critical message queues, what inevitably leads to a performance breakdown. Minimizing this impact is a challenging task. This section discusses the

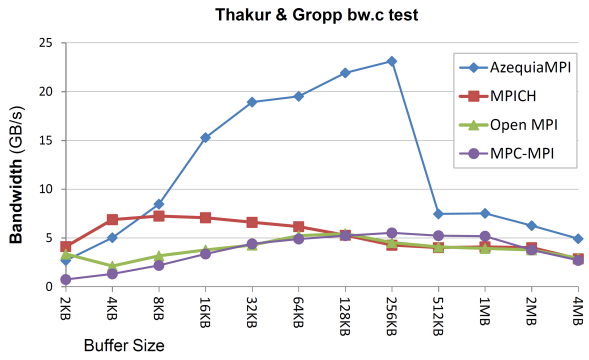


Figure 5: Bandwidth measurements of four MPI implementationsin the setup at the right side of Fig. 3.

MPI_THREAD_MULTIPLE design in AzequiaMPI.

The standard says that all the threads of a rank s can send a message to another rank r, and that any thread of rank r can handle the message, but a specific thread is not an addressable object. On other hand, the standard sets out that a pair of messages from the same rank but emitted by different threads are considered as concurrent events (even in the case that the messages had been emitted "physically" one after the other). This means that it is not possible to establish a temporal order relation between them and, as a result, they may be collected by the receiver rank in any order. Obviously, the messages sent by the same thread do keep the order. AzequiaMPI enforces such order using the simple concept of *flux*, which will be addressed below.

On the reception side, the standard literally states that "if two receive operations that are logically concurrent receive two successively sent messages, then the two messages can match the two receives in either order." We understand that this statement frees the implementation from ordering receiving requests from different threads r_i of a rank r, and as a result, each thread r_i may set up its own PRQ receiving queue .

A flux is a sort of connexion, an object with a pair source-destination $[s_i, r]$ where s_i is the thread i of source rank s and r

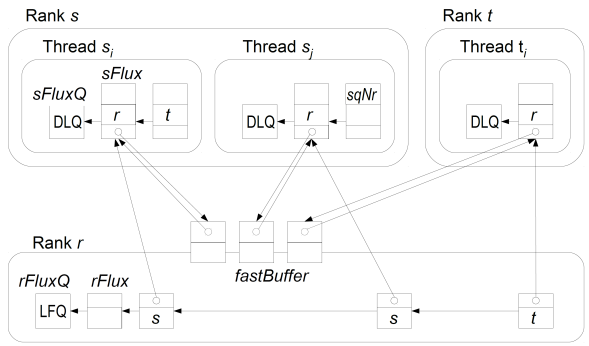


Figure 6: Design of fluxes. Any thread sending to a rank creates its private fastbuffer

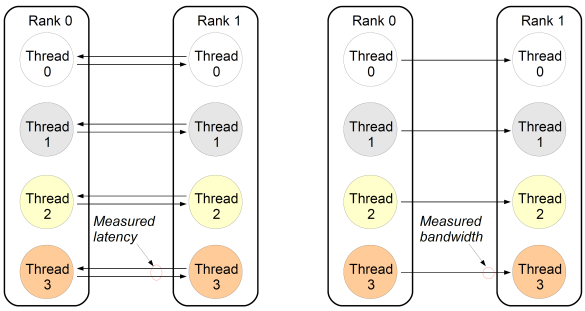


Figure 7: Thakur and Gropp multiple threaded tests. Experimental setup.

the destination rank. The source and destination extremes of a flux are internal objects of types $sFlux$ and $rFlux$ respectively. The first time the thread s_i invokes MPI_Send to send to a rank r, s_i allocates a $sFlux$ object, initialises it with the name $[s_i, r]$ and enqueues it in a private single-linked list. Next, at the destination extreme, s_i allocates the counterpart $rFlux$ object, initialises it with the handle of its corresponding $sFlux$ object and enqueues it in a lock-free queue of r named $rFluxQ$. Concurrent insertions in $rFluxQ$ are expected from any thread of any local rank. Every flux has an associated fastbuffer, a technique that improves the latency of the implementation (see Fig. 6). Fasbuffers introduce a second path from source to destination, what imposes a sequence number in the messages of a flux. The flux object keeps the current sequence number.

When a thread s_i invokes MPI_Send to send a message to rank r, it begins by looking up the $[s_i, r]$ object in its $sFluxQ$ queue. If r is not there then the flux object is created. As before mentioned, the first message of the flux under 1KB arranges the creation of the fastBuffer, whose handle is registered in $sFlux$. These small messages are sent by copying to the fastBuffer (Fig. 2) and then MPI_Send returns. Greater messages follow another path. A request object S is allocated per message, and enqueued in the LFQ of r, as shown in Fig. 1. Once enqueued, MPI_Send polls a flag of S. When the flag is set, the message has been received, then thread s_i frees S and returns. This process is done concurrently in a thread-safe way by all the s_i threads of process s without using any mutex.

When a thread r_i invokes MPI_Recv for a small message from rank s, it looks up all the fastbuffers associated to the threads of s. To this end r_i runs through the LFQ of $rFlux$ to access these fastbuffers. If a matching takes place the fastbuffer content is copied to the r_i user buffer and MPI_Recv returns. It can be thought that the $rFluxQ$ lock-free queue may be accessed for insertion while it is being explored. It can be shown, however that both operations are safe when they take place simultaneously, what avoids the introduction of a costly mutex to protect $rFluxQ$. For messages greater than 1KB, and also when no matching happens in the fastbuffer, MPI_Recv allocates a receiving request, inserts it in the private queue of receiving pending requests (PRQ) of r_i and enters the progress engine in an infinite loop.

Each iteration of the progress engine works in two stages. The first stage runs through the private PRQ queue of the invoking thread, let be t_j from rank t. For each request R found in PRQ the pair [s, keyTag] is obtained, where s is the desired source rank. Then

1. The set of fastBuffers from s (see Fig. 6) is probed against the pair [s, keyTag], as above discussed. If a matching happens R becomes satisfied and the progress engine returns.

2. If no matching takes place in the fastbuffers, the private unexpected queue (PMBX) of invoker t_i is probed against the pair [s, keyTag]. If a matching source request S is found, it is satistied, as well as R, and the progress engine returns.

3. If no matching takes place in PMBX, the global unexpected queue MBX is probed against the pair [s, keyTag]. MBX is a double-linked queue of owner t, concurrently read and written by all the t_i threads. It hence needs a mutex.

If no matching happens, the iteration of the progress engine enters the second stage. It consists of a loop of dequeue operations on the LFQ of rank t (see Fig. 1) until it becomes empty. This type of queue allows two or more concurrent enqueuers, but a single dequeuer, what imposes a mutex m to protect dequeueing. A thread t_j acquires m before a dequeue operation. Each dequeued request S is proved against all the pending requests of the private PRQ of t_j. If a matching of source and tag happens, but not of sequence

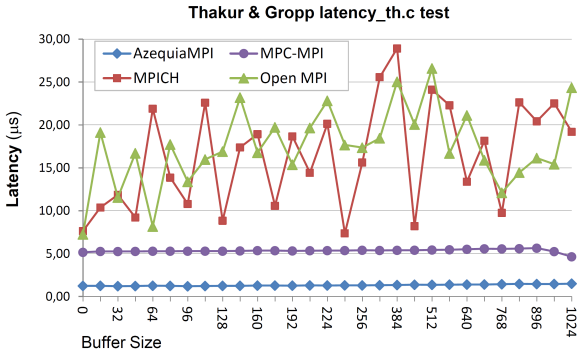


Figure 8: Latency measurements of four MPI implementations in the setup at the left side of Fig. 7.

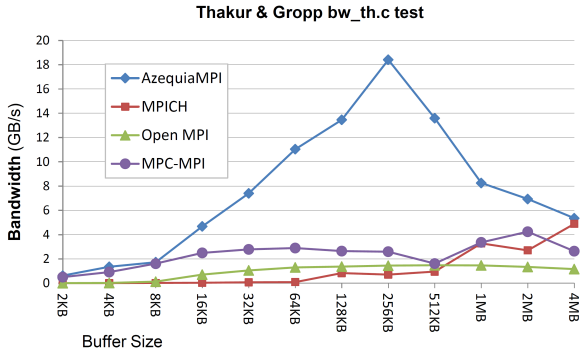


Figure 9: Bandwidth measurements of four MPI implementations in the setup at the right side of Fig. 7.

number, S is enqueued in the private unexpected queue PMBX of t_j; else is enqueued in the global unexpected queue MBX. The attained efficiency of this MPI THREAD MULTIPLE design has been measured with the Thakur and Gropp benchmark, and compared to that of MPICH, Open MPI and MPC-MPI. Fig. 7 shows the benchmark setup and Fig. 8 and Fig. 9 the obtained results. It can be appreciated that the AzequiaMPI figures considerably improve that of the rest of implementations.

V. Conclusions and further work

The thread-based design of AzequiaMPI and further optimizations based on its common address space makes it to outperform other MPI distributions in a significant manner. AzequiaMPI has shown that the thread-based approach opens opportunities to implement the MPI_THREAD_MULTIPLE thread safety level in a more efficient way than current popular MPI process-based libraries do. We aim to explore the implementation of the recent extensions of the MPI-3 standard to support shared memory at the ligth of these experiences in a thread-based framework.

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] Darius Buntinas, Guillaume Mercier, and William Gropp. Design and evaluation of nemesis, a scalable, low-latency, message-passing communication subsystem. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 10–pp. IEEE, 2006.

[2] Erik Demaine. A threads-only mpi implementation for the development of parallel programs. In *In: Proceedings of the 11th International Symposium on High Performance Computing Systems*, pages 153–163, 1997.

[3] Jack Dongarra, Pete Beckman, and Terry Moore et al. The international exascale software project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, February 2011.

[4] MPI Forum. Mpi: A message-passing interface standard, version 3.0., September 2012.

[5] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. *SIGARCH Comput. Archit. News*, 37(3):267–278, June 2009.

[6] Zhiqiang Liu, Kaijun Ren, and Junqiang Song. Mpiactor - a multicore-architecture adaptive and thread-based mpi program accelerator. In *Proceedings of the 2010 IEEE 12th International Conference on High Performance Computing and Communications*, HPCC '10, pages 98–107, Washington, DC, USA, 2010. IEEE Computer Society.

[7] Marc Perache, Patrick Carribault, and Hervé Jourdren. MPC-MPI: An MPI Implementation Reducing the Overall Memory Consumption. In *EuroPVM/MPI 2009*, pages 94–103, Helsinki, Finlande, September 2009. Springer-Verlag.

[8] Marc Pérache, Hervé Jourdren, and Raymond Namyst. Mpc: A unified parallel runtime for clusters of numa machines. In *Proceedings of the 14th International Euro-Par Conference on Parallel Processing*, Euro-Par '08, pages 78–88, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Juan-Antonio Rico-Gallego and Juan-Carlos Díaz-Martín. Performance evaluation of thread-based mpi in shared memory. In Yiannis Cotronis, Anthony Danalis, DimitriosS. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface*, volume 6960 of *Lecture Notes in Computer Science*, pages 337–338. Springer Berlin Heidelberg, 2011.

[10] Eduardo R. Rodrigues, Philippe O. A. Navaux, Jairo Panetta, and Celso L. Mendes. Preserving the original mpi semantics in a virtualized processor environment. *Sci. Comput. Program.*, 78(4):412–421, April 2013.

[11] Hong Tang, Kai Shen, and Tao Yang. Program transformation and runtime support for threaded mpi execution on shared-memory machines. *ACM Transactions on Programming Languages and Systems*, 22:673–700, 2000.

[12] Rajeev Thakur and William Gropp. Test suite for evaluating performance of multithreaded {MPI} communication. *Parallel Computing*, 35(12):608 – 617, 2009. Selected papers from the 14th European PVM/MPI Users Group Meeting.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
• October.2014 •

Content Delivery and Sharing in Federated Cloud Storage

J.L.Gonzalez[1], Victor J. Sosa-Sosa[1], Jesus Carretero[2], Luis Miguel Sanchez[2]

Cinvestav-Tamps, Mexico
[1]joseluig@ac.upc.es,[1]vjsosa@tamps.cinvestav.mx
University Carlos III, Spain
[2]luismiguel.sanchez,jesus.carretero@uc3m.es

Abstract

Cloud-based storage is becoming a cost-effective solution for agencies, hospitals, government instances and scientific centers to deliver and share contents to/with a set of end-users. However, reliability, privacy and lack of control are the main problems that arise when contracting content delivery services with a single cloud storage provider. This paper presents the implementation of a storage system for content delivery and sharing in federated cloud storage networks. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network. We developed a prototype based on this scheme as a proof of concept. The experimental evaluation shows the benefits of building content delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

Keywords Content delivery, Virtualization, Storage federation, fault-tolerant

I. Introduction

Space agencies, hospitals, government instances, news agencies and scientific centers not only are producing huge amount of contents but also they need to distribute them to different communities or population segments through the Internet [1].

Cloud storage approach has becoming a cost-effective solution for building dynamic and elastic online content delivery(CD) systems[2]. Organizations can contract a CD service with a provider through self-service and self-organizing web applications and their users can retrieve the contents in any time from anywhere by using almost any device.

However, organizations and users still have concerns about unavailable service[3], lost data risks [4] and lack of controls over data management [5] when using cloud storage. Organizations and users are quite justified in expressing their concerns about data storage and management in the cloud because a user rejects her legitimate expectation to privacy when she voluntarily relegates private content to a *third-party* [6].

Vendor dependence lock-in is another problem that has caused a big concern among organizations. This problem arises when one organization contracts with a single cloud provider the storage and management of all the data. This becomes a real problem when the organization decides to change the cloud provider or the cloud provider pulls out of the market[7]. In the first scenario it is not clear that other provider could handle the content delivery in its current state because of software dependencies. In the latter, the organization depends on the window offered by the provider to the clients for migrating their data to another provider. In both scenarios, the more data stored in the cloud the more economic impact on the costs suffered by the organizations.

The federated cloud model enables organizations to create a shared cloud storage based on strategic partnership policies[8]. In this model, a set of organizations cooperates for building a shared storage space to serve requests of other members that are in either failure or saturation circumstances. This model improves the reliability of the CD services as a federation member can withstand failures of their site by using the resources of partners for serving their user requests. Federation also enables the organizations to preserve autonomy and privacy even when a portion of their infrastructure has been used by the partners experimenting site failures.

This paper presents the implementation of a system for content delivery and sharing in Federated Cloud Storage. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network.

We developed a prototype based on this system as a proof of concept and its performance was compared with a fault-tolerant distributed web storage as well as public and private File Hosting Services.

A case study based on data obtained from the European Space Astronomy Center (ESAC) for the Soil Moisture Ocean Salinity [9] is presented as experimental evaluation. We distribute satellite images to a set of organizations, from two countries spanning two continents by using a federated Storage System (FSS). The end-users retrieved images by using a FSS client.

The experimental evaluation shows the benefits of building con-

tent delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

II. Related Work

Content Delivery Networks (CDN) such as Akamai [10], Coral [11] or Globule [12] cache small pieces of information and distribute them to locations near who requests them. The final end-user observes a reduction of the latency and overhead in the content delivery process.

The cloud-based storage services enable organizations to create catalogs of contents based on URLs. The end-users can access the catalog without simultaneous download restrictions by using any of the web browsers, synchronizer based on HTTP streams or (S)FTP applications. However, studies show that users prefer local storage solutions than public solution when managing sensitive data [13]. Moreover, cloud-based storage services are based on a pay-as-you-go pricing model, which apply rates based on the monthly stored contents plus the penalizations stated at the service level agreement (SLA) contracted. These conditions could lead to long-term costs. For instance, EUMETSAT and EOSDIS transfer around 1 TB of meteorological images per day, which might press organizations to consider an alternative service.

In addition, Vendor lock-in problem could arise when the organization decides to change the cloud provider or the cloud provider pulls out of the market[7](Nirvanix provides cautionary tale for cloud storage).

In order to face up system failures, in this kind of approaches the servers split the contents into chunks by applying a given codification[13] and they distributes them according a given fault-tolerant strategy. Nevertheless, the users and the organizations send the whole contents to the storage services, which could produce concerns about the way in which the privacy data is managed. In addition, the encoding/decoding data and its distribution both represent an extra work to be performed by the organizations servers.

The distribution of data on several providers have been proposed to avoid this type of problems [14]. Nevertheless, this kind of solutions are only available for public providers and solutions taking both the user and organization concerns into account are currently required by organizations.

This system can configure federations by using either private or public storage resources for organizations to deliver contents to end-users. Moreover, the codification applied to content dispersion by our system takes advantage of continuous flows and the multiple cores available in current computers.

III. A storage system for content delivery and sharing in Federated Cloud

A traditional content delivery system commonly includes stages such as source, formatting, deliver and acquisition of contents. In the first stage providers send a set of contents in raw format to the formatting stage, in which a set of users performs a set of annotation tasks for achieving manufactured contents, which are sent to a storage system. In the phase of acquisition, the end users retrieves the contents from the content delivery service by using a web application.

Our federated storage system (FSS) takes advantage of thee types of technologies to improve the effectiveness of the aforementioned content delivery process. The first is a cloud storage federation technology [9] applied to a metadata manager for keeping the control of content delivery in-house and enabling the organization to preserve autonomy in failure scenarios. The second are publish-subscribe patterns applied to a storage synchronization monitor, which enables FSS to inform the user about new available contents and to register the consume of each storage resource. The last one is the multi-core technology applied to a redundancy layer based on a multi-threaded engine for taking advantage of multiple cores of end-users and providers computers to improve the performance of the codification and dispersion of the federated fault-tolerant schemes used by FSS in the CD service.

Figure 1 shows an example of content delivery based on FSS. Raw data is sent to providers, which perform a set of annotation tasks for achieving manufactured contents. These contents are sent in the form of a catalog to the metadata manager by using an agent of our system. The metadata performs a push operation to split the $|F|$ into $n = C1..Cn$ chunks, which are distributed to a set of storage synchronization monitors that are members of the storage federation. When the provider shares that content by publishing the catalog, the end-users are notified and can retrieve the contents by using a FSS client APP. This APP obtains the locations to retrieve the chunks once the metadata manager verifies that the credentials of the client are valid. The APP retrieves the chunks from the federated storage and reconstructs the file in-house. As a result, the members of the federation has no possibility to reconstruct any file without either the authorization of the metadata manager or the collaboration of other members of the federation.

III.1 A metadata management layer

This layer is a cloud image in charge of the metadata flows, which includes the following modules for establishing management rules for the content flows:

- *Catalog manager*. This module enables the organization to create and manage catalog of contents. This module includes an attribute-based policy for managing the access to the contents listed in the catalog. It also enables authorized users, called providers, to add new contents to the catalog.

- *Multi-tenant module*. It manages end-users and providers accounts and is also in charge of the controls of the catalogs property. In this module, the contents of a given user account are isolated and remains invisible to other accounts.

- *Publish and Subscribe server*. It is in charge of the catalog publication and the subscriptions of contents. It serves the contents orders sent by end-users and controls the location of each catalog in the federation. This module includes an alert system that notifies to providers who is subscribing their contents and delivers the links to get access to published contents. It also includes push and pull RESTful functions for storing and retrieving contents form the federation.

This critical component is installed in-house by using cloud instance placed at private cloud of the organization.

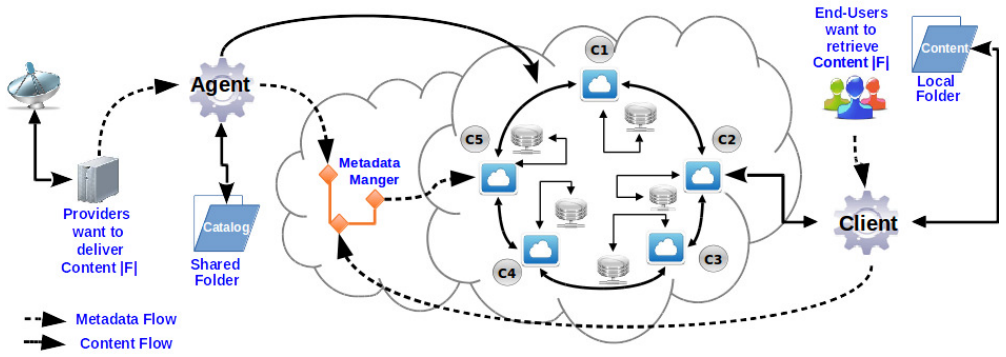


Figure 1: Content Allocation/Location in SkyStorage System.

III.1.1 A storage synchronization monitor

This module creates a database for managing the paths and access methods of each storage resource assigned by an organization to the federated cloud in the content delivery service. In order to observe the conditions of the agreement of the federation, this database also includes the consume metrics of each storage location. For instance, the storage quotas of received contents from each partner and the agreement characteristics for determining when a partner is ready to receive redundancy.

This module registers the operations performed by FSS, providers and end-users with each storage resource while an operational and safety monitor keeps updating the statistics in the database.

An agent in this module manages the metadata of the storage locations stores/retrieves contents by using the the redundancy layer.

III.2 A redundancy layer based on a multi-threaded engine

We add redundancy to the original content by using a dispersion algorithm called IDA [15]. This algorithm basically splits a given file $|F|$ into n redundant chunks, which must be distributed to n different storage locations. In scenarios where some original locations are unavailable and the user retrieves $|F|$, the algorithm recovers any of m number of chunks from the available storage locations with which the engine can reconstruct $|F|$. This means, it is granted that $|F|$ can be reconstructed when $n > m$ and the unavailable locations are $n - m$.

This algorithm can be implemented with different combinations of m and n parameters. This combination determines the codification costs in terms of storage space and computation time. The size of each resultant chunk is $|F|/m$, which results in a percentage excess of redundancy equal to $(n - m)/m$. Let us consider an IDA implementation with parameters $(n = 5; m = 3)$, in this case $|F|$ is transformed into five chunks and it can be reconstructed by retrieving at least three chunks from any three different locations; as a result, the system produces 66.7% of redundancy overhead, which is less than one replica.

Table 1 shows the amount of extra capacity spent for n servers when requiring m chunks (at least) to support fault-tolerance.

Table 1: IDA Parameters Combination m (chunks required for recovering contents) and n (servers)

n(servers)	m=1	m=2	m=3	m=4	m=5	m=6
n=2	100%					
n=3	200%	50%				
n=4	300%	100%	33.3%			
n=5	400%	150%	**66.70%**	25%		
n=6	500%	200%	100%	50%	20%	
n=7	600%	250%	133.3%	75%	40%	16.7%

III.2.1 Parallelism and Continuous Workflows

In order to save storage space, we use this algorithm in the content delivery process instead of using several replicas. In terms of latency, the client is retrieving $(|F|/m)$ chunks of data, which is similar to retrieve the whole file.

Nevertheless, the codification of the redundant chunks produces computation overhead while the distribution of chunks produces latency, which could be a problem depending on the network characteristics.

In order to reduce the effects of overhead and latency on the encoding/decoding procedures, we proposed and implemented an IDA codification technique based on parallel and continuous flows called Continuous Workflows.

We defined a *distribution workflow* based on the IDA encoding procedure. This workflow has been designed for providers to deliver contents to the federated storage by using push operations. This workflow includes *an acquisition stage* that receives a *service key* as input parameter. This key reports the construction of this workflow to metadata manager and enables the engine to obtain n relative URLs mapping n different containers. This engine stores each chunk that will produce this workflow by using a relative URL, which includes an anonymized name that will be the identifier of that chunk in the assigned container. This means that this chunk is managed as a file by the container/provider. This stage starts when reading the content that will be distributed by the workflow and ends up when sending both the content and the URLs to the next stage.

The transformation stage creates as many process as cores in the computer where the engine has been launched to split the content into n redundant chunks. This stage adds redundancy to each chunk and sends the obtained results to the transport stage. This stage

Table 2: The characteristics of Agents Clients

	PCs and Cloud Instances	*Cores*	*RAM*
Agents			
UC3M-Cloud	5 Instances	2	4GB
UC3M-Colme	2 PCs	4 (i7) and 2 (i5)	4GB
Cinvestav	5 Instances	4	8GB
Clients			
UC3M-Cloud	2 Instances	4	4GB
UC3M-Colme	2 PCs	7 (i7) and 4 (i5)	4GB
Cinvestav	5 Instances	2(3) and 3(2)	2(1GB) and 3(4GB)

writes the results, sent by the previous stage, in n streams created by using the relative URLs. This stage closes the streams when the encoding of each chunk is done and reports to the engine the found errors if any.

We also defined a *retrieving workflow* based on the IDA decoding for end-users to retrieve contents from the storage federation.

In this workflow, *the acquisition stage* receives as input parameter the name of the content that will be retrieved by the workflow, creates one file $|F|$ whit this name by using the local file system and sends m relative URLs to the pipeline. *The transformation stage* creates as many process as cores available and sends the URLs to the transport stage which creates m streams by using the URLs, reads the chunks and sends the results to the pipeline. *The Transformation stage* receives data, starts the reconstruction of the content and sends the results to the pipeline.*The acquisition stage* writes the received data in the file $|F|$.

The goal of this technique is to use all the processing power available in the computer where the engine is placed for enhancing the performance of the content workflow as it represents the highest costs in a CD service.

This multi-threading version improves the performance of encoding and decoding tasks by taking advantage of multiple cores commonly found in current devices. This technique allows the engine to reduce the codification overhead making feasible to introduce a fault-tolerant scheme in the CD process. The implementation of CD as continuous flow allows the engine to avoid writing chunks in the local disks.

We implemented the transformation stage by using TBB technology [16] and the transportation stage by using Curl libraries [17].

IV. The prototype

We consider a scenario where a set of organizations build a CD service in a cloud federated network by using our FSS. The federation includes three members that are represented by the following acronyms *UC3M-Colme*, *UC3M-Cloud* and *Cinvestav*. *UC3M-Cloud* located in Leganes, *UC3M-Colme* located in Colmenarejo (both cities near to Madrid, Spain). *Cinvestav* is located in Northeastern Mexico.

The *UC3M-Cloud* is in charge of the metadata manager and all the members have installed a image of the storage monitor and including a set of agents of the multi-threaded engine. We have launched a set of client images in the infrastructure of all the members with which the end-users retrieve the contents published by a source.

Table 2 shows the features of the storage infrastructure shared by each organization as well as the agents and clients included in this prototype.

V. Experimental Evaluation

We defined two evaluation scenarios: In the first we evaluated the performance of the federated storage system(FSS) with a synthetic workload while in the second we conducted a study case in which apply our FSS to the content deliver by using a set of real images.

We developed an *IO_Launcher* for producing publish, subscribe, pull (Download) and push (Upload) operations. The API sends these operations to the metadata manager. It assumes this artificial load comes from real and valid users and captures the response time, which is the only metric evaluated so far.

We captured the response time per each performed operation, which helps us to determine the degree of satisfaction or end-users. This time is measured from the publication/subscription moment of a given content until the time point in which the storage service retrieves that content, meaning that the request has been successfully dispatched. This time includes the streaming time, the network round trip latency and the write/read time in the temporal paths. This time also includes the time spent in subscription synchronization. Finally, this time also could include codification time when using redundancy.

VI. Experiments per evaluation scenarios and Results

The preliminary results included in this section consider the evaluation of the two scenarios previously defined.

VI.1 Performance evaluation of Federated storage system

In this scenario, we evaluated a storage network created with *UC3M-Colme* and *UC3M-Cloud* for testing the performance of FSS when delivering and retrieving contents with different size file.

We have designed a synthetic workload scenario to test the redundancy engine in which *IO_Launcher* sends an incremental load by duplicating the file size from 512KB to 1GB.

We defined the following configurations in this scenario:

- *Phoenix*: In this configuration, users store and retrieve contents by using an Online Distributed Web Storage System called Phoenix[18]. We implemented *Phoenix* in the *UC3M-Colme* and *UC3M-Cloud* organizations and it has been configured to apply a fault-tolerant strategy to the contents based on dispersion of information by using the IDA algorithm.
- *Private FHS*: In this configuration, users store and retrieve contents by using a Private Web File Hosting System. We have implemented a *Private FHS* in *UC3M-Colme* organization in which the contents are stored without producing and distributing redundancy.
- *Amazon*: In this configuration, users store and retrieve contents by using an AWS Amazon instance with the standard redundancy associated to a free account.
- *FSS*: We implemented our file distributor and acquisitor agent in the *UC3M-Colme* and *UC3M-Cloud* organizations. This configuration allows us to measure all the stages of the workflow produced by our storage system.

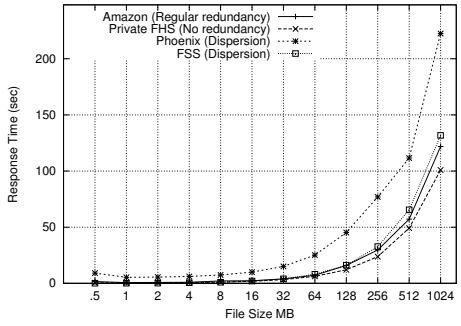


Figure 2: Response Time of Uploads

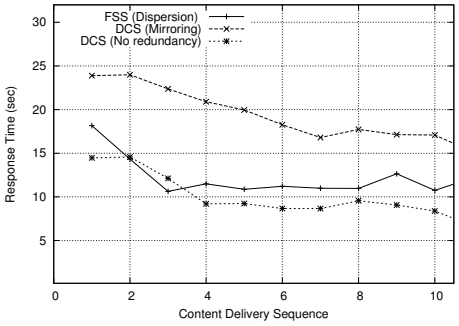


Figure 3: The response times of push operations for FSS and Distributed Cloud Storage configurations

Fig. 2 shows, in vertical axis, the mean response time obtained when a user sends contents of different sizes (horizontal axis) to the online storage system.

Fig. 2 also shows that *Private FHS* yields the best performance because it returns the control to the user when the file arrives to the cloud. This means *Private FHS* does not generate redundancy overhead, which improves the response time observed by the user. Moreover, *Private FHS* is situated in Colmenarejo city while the common users of this service are from both Madrid and Comenarejo. This reduces latency because both cities are geographically close to each other. *Amazon* configuration performs the same procedure as *Private FHS* but it produces more delay because its servers are located at Ireland.

Phoenix configuration produces the worst delay because it solves the vulnerability window in which the user could loss data by immediately splitting contents into chunks and distributing them to cloud storage locations of the two organizations. Once this has been performed, *Phoenix* returns the control to the synchronizer. As a result, the user obtains data assurance and she can retrieve that file even when the site of her organization is down.

The *FSS* performance is better than *Amazon* configuration (44% in mean) when the file size <4MB because the locality compensates the overhead of codification and, when the file size >4MB, *Amazon* configuration is better than *FSS* (9.14% in mean) because *FSS* distributes five chunks per I/O request (66.7% more data than *Amazon*). Nevertheless, this is a small increment because of *FSS* optimizes the storage stages on the client-side. In addition, that increment can be reduced or even eliminated by reducing the amount of distributed chunks when applying the information assurance policies.

As we can see, *FSS* offer the same reliability as *Phoenix* at reasonable cost. Moreover, *FSS* preserves the original file in-house, which means the contents can not be reconstructed by the administrators of a given organization without obtaining some chunks from either other organizations or the user device. When the users retrieving contents by using FSS, we observed the same behavior that the upload operations.

VI.2 Case study: Satellite Images delivery

This case study is based on data from the European Space Astronomy Center (ESAC), located at Villafranca del Castillo (Spain), which is in charge of the Soil Moisture Ocean Salinity or SMOS mission from 2009. This mission has generated a vast amount of contents, that will be used to create the first Earth global salinity map. Our system distributes images of the Sun of SMOS mission in FITS format to end-users through the Internet with a mean size of 44MB. A set of organizations and end-users, from two countries spanning two continents, get the contents by using a client APP of our system.

The following three configurations were defined in this case study:

- *DCS*: In this configuration, the end-user can access the contents by using an online distributed cloud storage system or DCS that does not include any redundant information.
- *DCS-Mirroring*: We implemented DCS in the UC3M-Colme and UC3M-Cloud organizations that includes a fault-tolerant strategy based on simple mirroring. This means two replicas are sent to the federated storage. The organization can withstand the failure of one cloud storage site by using this configuration.
- *FSS*: In this configuration, the organization delivers contents to interested users by using a FSS system, which was configured with an IDA combination $(n = 5, m = 3)$. This means the FSS client retrieves three chunks each time the end-users subscribe a published content. This IDA configuration allows the system to withstand 2 failures.

FHS and Phoenix configurations are not considered in this study as the first exposes security issues and the second produces more overhead than its version multi-threaded version FSS.

In this scenario we configured a storage network federation including UC3M-Cloud, UC3M-Colme and Cinvestav organizations. UC3M-Cloud was in charge of metadata manager and master of the content distribution role. In this scenario, a source sends images to the content delivery service by using publish and subscribe patterns. In this evaluation we define a push pattern in which the source sends the contents to its near members (UC3M-Cloud, UC3M-Colme). We defined two pull patterns. The first invoked by member that are not near to the source (Cinvestav). This pattern has been configured as a hot standby scheme in which each storage monitor worker of the Cinvestav member has a partner in the UC3M-Cloud and UC3M-Colme. Each worker of Cinvestav retrieves a chunk from its partner when the master receives a publish pattern. The service is accessed by the users of the members as single domain by using subscribe

patterns. Based on this unified interface, all users of all members are supported even when the servers of their organization are not available.

Figure 3 shows the mean response times observed by organizations when they distribute contents to end-users by using the mentioned configurations. Each point in this graph represents the mean response time of ten push operations. This means that 120 contents, with a mean size of 44MB, were sent to the cloud storage by the content delivery configurations. Figure 3 also shows that the response times of *DCS* configuration are better than the *FSS* configuration. Nevertheless, *FSS* configuration allows organization to withstand the failure of one cloud site and the source, while *DCS* is a single backup that can tolerate the failure of the source.

DCS-Mirroring improves the reliability of *DCS* configurations but it introduces a considerable overhead in the response times. Moreover, this configuration sends the whole file, so privacy and legal problems could arise. The *FSS* system distributes anonymized and encoded chunks, which means that the *FSS* agent and client know the locations of the chunks and are the only ones that can reconstruct the contents. In addition, *FSS* only produces up 66.7% of redundancy overhead while this overhead for DCS mirroring is of 100%.

VII. Conclusions

This paper presented the implementation of FSS: a system for content delivery and sharing in Federated Cloud Storage. This system virtualizes the storage resources of a set of organizations as a single federated system, which is in charge of the content storage. The architecture of this system includes a metadata management layer to keep the content delivery control in-house and a storage synchronization worker/monitor to keep the state of storage resources in the federation as well as to send contents near to the end-users. It also includes a redundancy layer based on a multi-threaded engine that enables the system to withstand failures in the federated network. The experimental evaluation shows the benefits of building content delivery systems in federated cloud environments, in terms of performance, reliability and profitability of the storage space.

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) Nesus-members mailing list Nesus-members@arcos.inf.uc3m.es https://www.arcos.inf.uc3m.es/cgi-bin/mailman/listinfo/nesus-members

References

[1] T. J. Bittman and L. Leong. Worldwide archival storage solutions 2011-2015 forecast: Archiving needs thrive in an information-thirsty world. pages 1-21, *IDC. Market Analysis*. October 2011

[2] http://aws.amazon.com/es/cloudfront/ Access 29 Sep 2014

[3] Google: Software bug caused gmail deletions (2011). http://www.pcmag.com/article2/0,2817,2381168,00.asp, Access 09 Sep 2014.

[4] Storm clouds ahead. http://www.networkworld.com/news/2009/030209-soa-cloud.html?page=1. Access 09 Sep 2014.

[5] http://www.itproportal.com/2012/07/04/power-outage-generator-failure-responsible-for-instagram-netflix-blackout/, Access 09 Jul 2013

[6] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. "Controlling data in the cloud: outsourcing computation without outsourcing control". *In Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85-90, 2009.

[7] Report: Nirvanix customers have two weeks to get data out of cloud: http://www.networkworld.com/article/2170916/cloud-computing/report–nirvanix-customers-have-two-weeks-to-get-data-out-of-cloud.html

[8] J. L. Gonzalez, J. C. Perez, V. Sosa-Sosa, J. F. Rodriguez Cardoso, and R. Marcelin-Jimenez. "An approach for constructing private storage services as a unified fault-tolerant system". *J. Syst. Softw.*, 86(7):1907-1922, July, 2013.

[9] smos: Soil moisture and ocean salinit.

http://www.esa.int/esalp/lpsmos.html. Access date: 23 jul 2013.

[10] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. "Globally distributed content delivery". *Internet Computing, IEEE*, 6(5):50-58, 2002.

[11] M. J. Freedman, E. Freudenthal, and D. Mazieres. "Democratizing content publication with coral". In NSDI, volume 4, pages 18-18, 2004.

[12] G. Pierre and M. Van Steen. "Globule: a platform for self-replicating web documents". *In Protocols for Multimedia Systems*, pages 1-11. 2001.

[13] R. Seiger, S. G, and A. Schill, Seccsie: A secure cloud storage integrator for enterprises, *in Proceedings of the 2011 IEEE 13th CEC2011*, pp .252-255.

[14] J. Spillner, G. Bombach, S. Matthischke, J. Muller, R. Tzschichholz, and A. Schill, "Information dispersion over redundant arrays of optimal cloud storage for desktop users,"*in Fourth IEEE International Conference on Utility and Cloud Computing*, 2011,pp.1-8.

[15] M. O. Rabin. "Efficient dispersal of information for security, load balancing, and fault tolerance". *J. ACM*, 36(2):335 348, 1989.

[16] A. Robison, M. Voss, and A. Kukanov, "Optimization via reflection on work stealing in tbb," in IPDPS. IEEE, 2008, pp. 1-8.

[17] libcurl, http://curl.haxx.se/libcurl/ Access date: 23 Sep 2014.

[18] J. L. Gonzalez and R. Marcelin-Jimenez, "Phoenix: Fault tolerant distributed web storage." *in Journal of Convergence*, Section C:Web and Multimedia, Vol.2, No.1, 2011.

First NESUS Workshop • October 2014

Improvement of Heterogeneous Systems Efficiency Using Self-Configurable FPGA-based Computing

Anatoliy Melnyk, Viktor Melnyk

Lviv Polytechnic National University, Ukraine
aomelnyk@polynet.lviv.ua vamelnyk@lp.edu.ua

Abstract

Computer systems performance is is being improved today using two major approaches: general-purpose computers computing power increase (creation of multicore processors, multiprocessor computer systems, supercomputers), and adaptation of the computer hardware to the executed algorithm (class of algorithms). Last approach often provides application of the ASIC-based and FPGA-based hardware accelerators, also called reconfigurable, and is characterized by better performance / power consumption ratio and lower cost as compared to the general-purpose computers of equivalent performance. However, such systems have typical problems. The ASIC-based accelerators: 1) are effective for certain classes of algorithms only and 2) algorithms and software require adaptation for effective application. The FPGA-based accelerators and reconfigurable computer systems (that use FPGAs as a processing unit): 1) in the process of writing require a special program to perform computing tasks balancing between the general-purpose computer and FPGAs; 2) require designing the application-specific processor soft-cores; and 3) are effective for certain classes of problems only, for which application-specific processor soft-cores were previously developed. In this paper, we consider an emerging type of high-performance computer systems called self-configurable FPGA-based computer systems, which are deprived of specified challenges. We have analyzed the background of self-configurable computer systems creation, presented current results of our research, and introduced some ongoing works. Self-configurable computer systems are being developed within the project entitled "Improvement of heterogeneous systems efficiency using self-configurable FPGA-based computing" that is the part of the NESUS Action.

Keywords Field programmable gate arrays, high performance computing, reconfigurable computing, self-configurable computer systems.

I. Introduction

Computer systems performance increase continues to be a determinative factor for the progress in science and engineering branches, and also have become a catalyst of emerging a range of new spheres of human activity. It is hard to find a field today that does not depend on this characteristic of the computer systems. However in the beginning of the 21^{st} century the pace of performance increase of the general-purpose processors that make a base of personal computers and multiprocessor computer systems, fast until now, has begun getting slower. This trend is even more notable today, when the general-purpose processors clock rate practically remains the same within several last years. The reasons for this are the fundamental boundaries in the energy efficiency for the CMOS technology used today in the integrated circuits industry and limitations that a traditional method of information processing imposes on the general-purpose processor architecture, particularly, the sequential execution of instructions in the program, and the sequential access to instructions and data in the memory. Trying to provide the instruction set compatibility for each next processor generation with the previous one and to support the existing software by the processors of different generations, the hardware and software manufacturers do not introduce principal changes neither into the single-processor architecture nor into the method of information processing in it.

The task of computer systems performance increase is being solved today mostly by creating the multi-core processors, where the number of cores in the chip ranges from a few to tens of complex and to hundreds of simpler ones, and tends to grow. However, an approach to increase the number of cores in the chip in the system has principal disadvantages as well. Thus, having the parallel data processing, it is also necessary to investigate and develop much more energy-efficient computer systems on each technological level, including devices, hardware architecture and software.

Therefore, today one of the most perspective directions in high-performance computing is creation of the heterogeneous computer systems. These systems combine one or more general-purpose processors and hardware accelerators, including those built on the basis of reconfigurable environments - field-programmable gate arrays (FPGA).

The problems related to designing the heterogeneous computer systems with the use of the hardware accelerators, primarily the reconfigurable ones, are considered in the book [1]. In this paper, the design issues and the development trends of the heterogeneous computer systems - from those built on the ASIC-based hardware accelerators to the reconfigurable and self-configurable ones - are being discussed.

II. Improving the Computer Systems Performance Using the ASIC-based Hardware Accelerators

The ASIC-based hardware accelerator (AHA) is a device that contains a high-performance programmable processor with architecture dedicated to the specific class of algorithms and intended to increase the computer system performance on that class of algorithms. Typ-

ically, AHAs are used to execute the complex algorithms applied to big data arrays that cannot be processed by the general-purpose processor during the acceptable period of time.

Today the most widely used AHAs are [1]: the CELL processors developed jointly by IBM, Toshiba and Sony; the accelerator boards manufactured by ClearSpeed (Great Britain-USA); the GPGPU boards produced by ATI and Nvidia; and the GRAPE boards developed jointly by the scientists from the University of Tokyo and the National Astronomical Observatory of Japan.

Usually, when using AHA, the time of task execution is shortened by one/two orders of magnitude as compared to that spent for this task by the general-purpose processor. However, there are several problems that significantly reduce AHA effectiveness, namely:

1. AHAs are very complex devices. Their development requires years of work of a number of engineers (four years and nearly four hundred of engineers for the CELL processor creation).

2. AHAs have got the parallel architecture. In order to write the parallel programs for them, first the algorithms should be adapted to this architecture that practically means their re-designing and afterwards the parallel programs should be developed using special programming languages, and this requires a great amount of intellectual work. Moreover, not for all algorithms this architecture is effective.

3. An approach to combine a general-purpose computer with AHA gives an opportunity to get high performance for executing the tasks, on which the algorithms, that AHA is dedicated to, are based, but not an arbitrary task.

Having a goal to provide the computer systems with the opportunity to achieve high performance characteristics while executing arbitrary tasks, the accelerators are being built on the basis of programmable logic devices. Such hardware accelerators, as well as the computer systems with them, are called reconfigurable.

III. Reconfigurable Computer Systems

Reconfigurable computer systems (RCCSs) compete with other types of high-performance computer systems due to the high characteristics of modern field-programmable gate arrays (FPGAs) - a hardware base of reconfigurable computing environment (RCE) of RCCS, and due to advances in design technology of application-specific processors to be synthesized in RCE of RCCS.

Co-functioning of the computer system based on the general-purpose processors with application-specific processors synthesized in RCE, whose structure considers an executed algorithms features, allows its productivity to be increased by 2-3 orders of magnitude. Reconfigurability and ability to synthesize an application-specific processor (ASP) with a new structure and functions in RCE allow one to change the functional commitment of RCCS created thereby with preserving its high performance at the new class of problems.

The RCCS architecture and organization on different levels are described in [2]-[4]. By analyzing these studies one may conclude difficulty of information processing in such systems, which is a consequence of a need to perform the ASPs design and synthesis in RCE before being used. The ASP design is performed by describing its architecture in the VHDL or Verilog hardware design languages (HDL) with the use of the register transfer level design tools, or even by specifying its characteristics and algorithm to be executed in the high-level programming language, as it is provided by the advanced electronic system level design tools, for example, System-C [5] from Celoxica, Catapult-C [6] from Calypto Design Systems, DIMETalk [7] from Nallatech, CoDeveloper [8]. These tools enable creation of the HDL-descriptions of computing devices on the register transfer level from the programs written in a high-level programming language, often a modified C language. The Chameleon Technology and tools [9] from Intron, as well as SPARK developed at the University of California, are intended to design ASPs using the algorithm description in the ANSI C language. The basic platform for ASPs creation here is a configurable processor architecture, which provides creation of its desired configuration with the use of the following configuration parameters: the number of functional units, the instruction set of each functional unit, the capacity of the program and data memories, the number of inputs and outputs of the communication network. These parameters together with the specification of the processor's interface should be submitted in addition to the algorithm description.

IV. Information Processing Method in RCCS. Challenges and Solutions

Information processing in RCCS can be represented as a sequential execution of the four stages [10]. At the first stage, the user creates the program P_{in} written in the high-level programming language, divides this program into the computer subprogram P_{GPC} and the RCE subprogram P_{RCE}, performs the P_{GPC} subprogram compilation, generates its executable file *obj* and stores it in the computer memory. At the second stage, the user develops (or uses a ready-made solution) an HDL-model $ASPM$ of ASP intended to perform the P_{RCE} subprogram of RCE, performs the logical synthesis of the ASP and loads the configuration files $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$, where K_{FPGA} is the number of FPGAs forming RCE, and, thus, creates an ASP in RCE. At the third stage, after the program initialization, the operating system loads the executable file *obj* of the computer subprogram to its main memory. At the fourth stage, RCCS executes the P_{in} program. Computer executes its own subprogram P_{GPC}, RCE executes its own subprogram P_{RCE}.

By analyzing the above-described method of information processing in RCCS, one can conclude the problems that significantly impede the improvement of its efficiency, namely:

- the need in the process of writing a program to perform computing tasks balancing between the general-purpose computer and the reconfigurable environment;

- the need of designing the application-specific processor soft-cores to be implemented in RCE that requires specialists and software packages for the IP Cores designing, testing, synthesis and implementation;

- in RCCS, there is an unsolvable problem caused by the inability to develop the application-specific processor HDL-models to be implemented in RCE that will be effective for the series of problems. This challenge imposes a fatal limitation on RCCS: these systems are effective only for certain classes of problems,

for which the application-specific processor HDL-models have been developed previously.

Automation of all stages of information processing in RCCS is a key approach to solve the above problems. The development of the ASP's HDL-model at the second stage requires from the user a significant amount of time and knowledge of the system-level design technology. However, as mentioned above, today the software tools are available allowing automatic creation of the ASP's HDL-models from high-level description of the algorithm to be implemented in. This software tools transform the algorithm described in the high-level programming language into the HDL-model of ASP. By linking the operations of ASP's HDL-model generation, ASP's logical synthesis and RCE configuring in automatically executable sequence, the computer system can load the configuration codes into RCE automatically with no user intrusion.

It should be noted that the use of generation tools imposes condition of availability of a high-level algorithm description to be implemented in ASP. The user creates this description at the first stage during dividing the input program into two subprograms, and this also requires from the user a significant amount of time. Automation of load balancing, besides reduction of amount of time, will allow one to:

1. link operations of load balancing and compilation of computer subprogram in one startup sequence, and, as a result, to obtain a subprogram executable file without user intrusion;

2. link operations of load balancing, generation of ASP HDL-model, ASP's logical synthesis and RCE configuration in one startup sequence, and, as a result, to load configuration codes into RCE automatically without user intrusion.

Consequently, automation of steps performed at the first two stages of information processing method in RCCS, i.e. automatic obtaining of computer subprogram executable file and automatic creation and loading of ASP configuration files into RCE for RCE subprogram execution, will allow one to reduce:

- the execution time for information processing;

- the complexity of information processing since the user has no longer to perform the systems analysis, ASPs architecture design and ASPs logical synthesis.

However, automation of the two first stages execution does not solve another problem mentioned above - namely, the list of problems that RCCS is effective on remains short and depends on the functional characteristics of ASPs implemented in RCE. Their change requires, at least, repeating the second stage's steps of the above-mentioned method of information processing, which should be done by the user.

This challenge can be solved by improving the method of information processing in RCCS in the way that loading configuration files obtained after the logical synthesis into RCE is carried out not by the user but by the operating system, and not at the second stage but at the third one, in parallel with loading the computer subprogram executable file into its main memory after the program initialization. This implies that configuration files should be stored in the computer memory after the logical synthesis. Thus, because the configuration files are formed automatically in parallel with the computer subprogram executable file and stored in its memory, the entire sequence of actions from the beginning of the load balancing up to obtaining the executable file and the configuration files should be treated as a single stage of program compiling.

V. Self-Configurable Computer Systems

We suggest to call self-configurable the computer system with reconfigurable logic, where program compilation includes automatically performed actions of configuration creation, and which acquires that configuration automatically during the program loading for execution [10].

In the Self-Configurable Computer System (SCCS), 1) execution of the computational load balancing between the general-purpose computer and RCE and 2) creation of the ASP's programming model are automated, and the method of information processing is improved in the way that loading the configuration files obtained after the logical synthesis into RCE is carried out not by user but by the operating system in parallel with loading the computer subprogram executable file into its main memory after the program initialization [10].

The diagram of the method of information processing in the SCCS is shown in figure 1. This method can be represented as a sequential execution of three stages: program compiling, loading and execution.

The user creates a program P_{in} written in a high level programming language and submits it into SCCS. SCCS during compiling automatically performs the following actions: divides this program into the computer subprogram P_{GPC} and the RCE subprogram P_{RCE}, performs computer subprogram P_{GPC} compilation, generates its executable file obj, creates ASP's HDL-model $ASPM$ to perform the RCE subprogram P_{RCE}, performs the ASP's logical synthesis, and stores the obtained executable file obj and the RCE configuration files $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$ in computer memory, where K_{FPGA} is the number of FPGAs that form RCE.

In order to perform these actions the SCCS has to contain the following means:

1. A computational load balancing system for load balancing between the computer and RCE. This system should automatically select from the program P_{in} fragments, whose execution in RCE reduces its execution time, and divide the program P_{in} into the computer subprogram P_{GPC}, replacing the selected fragments in it by instructions for interaction with RCE, and the RCE subprogram P_{RCE}, formed from the selected fragments. An example of such system is described in [11]. This system creates the RCE subprogram in the x86 assembly language, thus it must be supported by the means for the assembly language code translation into the high-level language to be used in SCCS. The tool of this type is available on the market, for example Relogix Assembler-to-C Translator [12] from MicroAPL.

2. A generating system for the ASP HDL-model creation, which should automatically generate a model $ASPM$ from the RCE subprogram P_{RCE}, like Chameleon system from Intron, Agility Compiler and DK4 Design Suite from Celoxica, CoDeveloper from Impulse.

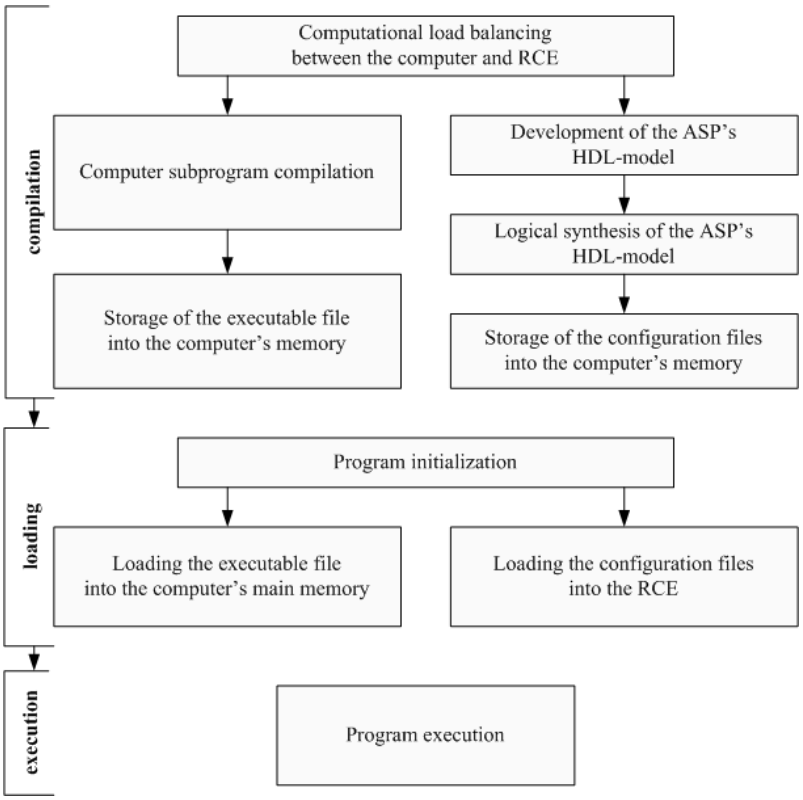


Figure 1: The diagram of the method of information processing in the SCCS.

3. The tools that are used in RCCS for performing the computer subprogram compilation and creation of its executable file, and for the logical synthesis of the ASPs.

At the stage of the program loading after its initialization, SCCS loads the executable file *obj* of the computer subprogram into its main memory using the standard loader and, at the same time, loads the configuration files $\mathbf{conf} = \{conf_q, q = \overline{1...K_{FPGA}}\}$ into RCE and, thus, creates an ASP in it. Then, the stage of the program execution is performed in the same way as in the RCCS. In order to perform these actions the same tools can be used in SCCS as in RCCS.

The structure of the self-configurable computer system that implements the proposed method of information processing is shown in figure 2. The term "self-configurable" against the FPGA-based computer system implies that here the computer system performs by itself all the steps of information processing after the program initialization, ranging from the load balancing between the computer and RCE up to obtaining the executable file of RCE, as well as the RCE configuring.

VI. SCCS Benefits

- **Ensured effective use of reconfigurable logic to perform arbitrary tasks** - loading of executable files and configuration files into the computer main memory and into RCE is, respectively, performed by the operating system after program initialization.

- **Shortened information processing time** - all the actions, starting from the load balancing and till obtaining the executable file and the configuration files, are executed automatically at the stage of program compiling without user's intrusion.

- **Reduced information processing complexity** - requirements to the user experience are simply reduced to knowing the high level programming language.

- **Simplified programming** - the user utilizes the ANSI C language that requires no additional constructions (parallel operators, directives etc.) and works with SCCS in the same manner as with the conventional personal computer.

VII. Background of SCCS

SCCS creation is the result of years of engineering and research work of the authors in the field of application-specific processors development, their ASIC-and FPGA-implementation, and creation of the methods and means for their high-level design. The background of SCCS is built on the:

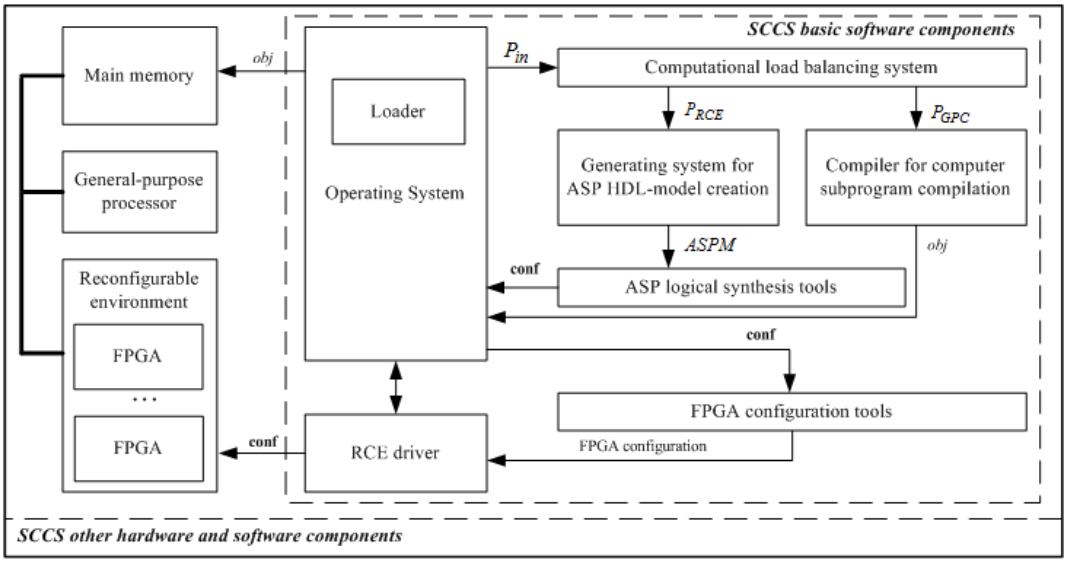


Figure 2: The structure of the self-configurable computer system.

1. ESL Design Tools and Solutions (developed in cooperation with Intron ltd (http://intron-innovations.com)):

   - Chameleon - System-Level Design Solution intended for the ASIC design automatic generation from the algorithm described in the ANSI C language. The developer, specifying an algorithm of the data processing on ANSI C, gets on return 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. Obtained VHDL design may be further implemented in FPGA by any FPGA design solution, e.g. the Xilinx ISE Web-PACK.
   - OSCAR - System-Level Design Solution intended for the VHDL design automatic generation from the algorithmic representation. The customer may select one of the following options for synthesizing the algorithmic computing device (ACD):
     (a) a single-stage ACD;
     (b) a multiple-stage ACD;
     (c) a pipeline ACD;
     (d) an ACD with the scalable layers/operators.
   - IP Cores Generator -high-level software tools, which allow IP Core with required characteristics to be obtained automatically on the basis of the scalable IP modules' sources. Some of them are the tools for generating: 1) the Fast Fourier Transformation IP Cores; 2) the Fast Cosine Transformation IP Cores; 3) the Data Encryption Standard IP Cores; 4) the Multiple-Block RAM Access Controller IP Cores [13], [14].

     Generator's input consists of the scalable parameterizable IP modules' sources and user-defined IP Core parameters from the given list. The generator output consists of the IP Core source HDL files; the IP Core test bench with the test patterns; text documentation etc.

2. Our new computer architecture.
3. Our new multiport computer memory with the parallel conflict-free data access [15].

VIII. What We Have Done Within the Project

- The theoretical principles of the SCCS design and operation have been developed, the characteristics of information processing duration in SCCS have been investigated and the SCCS's fundamental advantages over the reconfigurable ones have been proven.
- The method of information processing in SCCS for their single- and multi-processor implementations has been developed.
- The principles of the SCCS structural organization and the conceptual bases of their components design have been developed.
- The theoretical foundations of approaches to computational load balancing system design in SCCS and the method of computational load balancing between the general-purpose computer and the reconfigurable environment have been investigated. An instance of the computational load balancing system with the use of the LLVM compiler has been created.
- The high-level design tools have been developed allowing the application-specific processor VHDL IP-Cores to be generated on the basis of the ANSI C descriptions of their algorithms of

operation that are the advanced products for the system-level design.

IX. Current and Future SCCS Development

In our belief, SCCS represents definitely an emerging direction in developing the heterogeneous computer systems particularly and in the high-performance computing in general, and there is a need in further works in developing the SCCS theoretical basis, software and hardware design and implementation, testing, optimization etc. Some ongoing works on the SCCS creation are as follows:

- **Development of the SCCS operating system.** The goal of this work is to develop the theoretical background and the appropriate system software means for the SCCS operation during program compilation, loading for execution, and execution.
- **Load balancing in the SCCS.** The goal of this work is to develop the theoretical background and the appropriate software tools for balancing the computational algorithms specified by the input program between the general-purpose programmable processor and reconfigurable environment that fully explore the spatial and the temporal properties of the algorithm.
- **Automatic mapping of the soft-cores synthesized by the HLL2HDL tools into the target FPGA architecture** (the HLL - high-level programming language). The goal of this work is to develop the theoretical background and the appropriate software means for adaptation of the structure of the soft-cores synthesized by the HLL2HDL tools to that of the target FPGAs, as well as to integrate these tools with other software SCCS components.
- **SCCS based on the partially reconfigurable FPGAs.** The goal of this work is to improve the SCCS efficiency by developing the methods of the parallel execution of multiple computing tasks in their reconfigurable environment, to implement and investigate the self-configurable FPGA-based computer systems based on the partial dynamic reconfigurable FPGAs.

X. Conclusions

In this paper, we have described an emerging type of the heterogeneous computer systems, namely, the self-configurable FPGA-based computer systems.

The method of information processing in SCCS has been presented and the rules of application of the computer software and hardware tools necessary for its implementation have been described.

SCCS benefits are as follows: 1) an ensured effective use of the reconfigurable logic to perform arbitrary tasks; 2) a shortened information processing time; 3) a reduced information processing complexity; and 4) a simplified programming.

We have highlighted the background of SCCS, which is built on: 1) our ESL Design Tools and Solutions; 2) our new computer architecture; and 3) our new multiport computer memory with the parallel conflict-free data access.

Finally, we have shown some ongoing works on the SCCS creation, targeted on the development of the SCCS theoretical basis and their software and hardware components design and implementation.

References

[1] A. Melnyk, V. Melnyk, *Personal Supercomputers: Architecture, Design, Application*, Lviv Polytechnic National University Publishing, 2013. - 512 p.

[2] S. Hauck, A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*, Morgan Kaufmann, 2008. - 944 p.

[3] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications*, Springer, 2010. - 352 p.

[4] T.J. Todman, G.A. Constantinides, S.J.E. Wilton, O. Mencer, W. Luk and P.Y.K. Cheung, "Reconfigurable Computing: Architectures and Design Methods," *IEEE Proceedings: Computer and Digital Techniques*, vol. 152, no. 2, March 2005, pp. 193-208.

[5] *IEEE Standard for Standard SystemC Language Reference Manual*, IEEE Std 1666-2011, 9 January 2012. - 638 p.

[6] "Catapult LP for a Power Optimized ESL Hardware Realization Flow," [Online]. *Available: http://calypto.com/en/blog/2012/11/10/catapult-lp-for-a-power-optimized-esl-hardware-realization-flow/* - 11.10.2012.

[7] "DIMETalk. FPGA Development Tools," [Online]. *Available: http://www.nallatech.com/FPGA-Development-Tools/dimetalk.html*

[8] "C-to-FPGA Tools form Impulse Accelerated Technologies. Impulse CoDeveloper C-to-FPGA Tools," [Online]. *Available: http://www.impulseaccelerated.com/products_universal.htm*

[9] "Chameleon - the System-Level Design Solution," [Online]. *Available: http://intron-innovations.com/?p=sld_chame*

[10] A. Melnyk, V. Melnyk, "Self-Configurable FPGA-Based Computer Systems," *Advances in Electrical and Computer Engineering*, vol. 13, no. 2, pp. 33-38, 2013.

[11] 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.

[12] "Relogix Assembler-to-C translator," [Online]. *Available: http://www.microapl.co.uk/asm2c/*

[13] A. Melnyk, "Newest Computer Devices Design Technology on a Base of Configurable Models," in *Proceedings of First International Conference "Advanced Computer Systems and Networks"*, Lviv, Ukraine, Setember 2003, pp. 10-12.

[14] A. Melnyk, V. Melnyk, "IP Cores Generators in SoC Design," in *Proceedings of the 5th international Conference for Students and Young Scientists "Telecommunication in XXI Century"*, Wolka Milanowska, Poland, 24-26 November 2005, pp. 23-28.

[15] A. Melnyk, *Ordered Access Memory*, Lviv Polytechnic National University Publishing, 2014. - 296 p.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
Book paper template • September 2014 • Vol. I, No. 1

Data parallel algorithm in finding 2-D site percolation backbones

Biljana Stamatovic*, Roman Trobec+

*University of Donja Gorica, Montenegro biljana.stamatovic@udg.edu.me
+Institut Jozef Stefan, Slovenia roman.trobec@ijs.si

Abstract

A data parallel solution approach formulated with cellular automata is proposed with a potential to become a part of future sustainable computers. It offers extreme parallelism on data-flow principles. If a problem can be formulated with a local and iterative methodology, so that data cell results always depend on neighbouring data items only, the cellular automata could be an efficient solution framework. We have demonstrated experimentally, on a graph-theoretical problem, that the performance of the proposed methodology has a potential to be for two orders of magnitude faster from known solutions.

Keywords Sustainable computing, Data parallel, Data-flow, Cellular automata, Percolation backbone, Cluster

I. Introduction

Increasing data and processing requirements of applications are constantly pushing for further increases in computational capabilities. Today, we have reached a point where computational systems are confronted with physical barriers that limits a significant further increase of system frequencies. Additionally, excessive energy consumption, limited scalability and complex data management are obstacles that have to be solved nowadays to enable further increase of the computing performances.

It seems that massively parallel computing devices on all levels of computing, connected in heterogeneous computing systems, could counterbalance these difficulties. Multicore systems, graphic processing units (GPU), dedicated FPGA accelerators are already widely implemented and investigated options in contemporary high performance computers. Adaptive combinations of control-flow and data-flow approaches seems to be able to offer general purpose and sustainable ultrascale computers that will be able to cope the applications either with demanding processing power or with complex analysis of big data sets.

The cellular automata (CA) can be considered as an alternative way of computation based on local data-flow principles. The concept of CA was first proposed by John von Neumann in 1950s through self-reproducing systems (published later in [1]). The formalization has been improved by various authors [2, 3, 4, 5], emphasizing different theoretical and practical perspectives. Various application areas of CAs, ranging from ecology [6], biology [7], diffusion through soil pores [8], image processing [9], wafer diagnostics [10], sociology [11] etc., have been investigated later.

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.

The CA can be considered 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. The theoretical studies of the computational capabilities of CA [3, 4] have shown that there exists CAs that are equivalent to the Turing machine and therefore can be used as general purpose computers. Our work is relevant to the practical backward approach in the design of CAs [5], where transition rules are searched for that result in CA states that match the physical system.

Many efficient applications have been developed with data-flow approaches and CAs on problems that have been previously solved with local numerical methods [12, 13]. The CAs can compete with classical computers in computational performance and efficient use of energy because of massive parallelism and relatively low system clock. The hardware resources of CAs can be implemented with Systems-on-Chips (SoC) on the wafer level [14], with more general Field programmable logic arrays(FPGA) [12], with emulation on manycore systems or on hybrid architectures mixing all above options.

A CA can be mapped on the physical system with the methodology already established in heterogeneous computing: the initial problem data and possibly transition rules are mapped from the host CPU onto the computing array which implements the CA. CA cells compute in parallel for the required number of computational steps. After a stopping criteria is met the results are read from the CA into the host memory and further analysed or visualized on the CPU. The procedure can be implemented in a loop with an eventual reprogramming of the CA. It is evident that such a data-flow approach could have a striking advantage [15] over the classical crunching machines in the significantly lower consumption of energy per a computing operation, which could contribute to the sustainability of future computers.

The CAs rely on the discreteness, locality, regularity, and synchrony. Their simple definition has several advantages, e.g. CAs are not limited by the number of elements, their evolution is inherently parallel, they have a strong resemblance to many important principles in the nature like: cells that are building blocks for large systems, elementary particles, etc. However, to approach to real problems, some additional properties of CAs are necessary, e.g. global com-

Book paper template • September 2014 • Vol. I, No. 1

munication (for loading of initial data, implementation of stopping criteria or reading of final results) and non-homogeneity (for an adequate treatment of boundary conditions).

The rest of the work is organised as follows. In the next Section a principal description of the proposed system architecture is described focusing on global operations related to data movements. Section III is an overview of the work related to the identification of infinite clusters and their percolation backbones. This problem can be considered as an example of a large class of demanding tasks that can be solved by parallel iterative operations on local data. In Section IV a new efficient CA algorithm is proposed that can efficiently cope with this graph-theoretical problem. The proposed approach is experimentally tested and evaluated on finding 2-D site percolation backbones. It was shown that the performance of the proposed solution has a potential to be for two orders of magnitude faster than previous known solutions.

II. System architecture

To objectively evaluate a CA algorithm the time needed for data manipulation has also be considered. Fore example, initial data, e.g. pixels of images, sites of percolation array, etc. must be loaded into CA cells. Beside data management capability, each CA cell must comprise also a processing and memory unit that are tailored to algorithm requirements. All cells usually execute the same operations with the same system clock.

We can suppose that a CA array is build from rectangular tiles that incorporate $N = (L \times M)$ CA cells. For simplicity, we suppose that $M = L$. Each cell is connected directly with a small number of nearest neighbours d, e.g. $d = 4$ or $d = 8$ with a simple communication switch built in each CA cell [16].

A set of l routers with radix r is connected to a subset of r CA cells. The radix is here a number of communication ports to the lower level. Each router has also a single, possibly faster, port to the higher level. If the number of CA cells is large, p intermittent levels of routers can be introduced. The routers communicate with a host computer that manage data transfer to lower layer routers, or to CA cells if a router is positioned on the lowest level.

The communication among direct d connected cells is performed in a single communication step (hop). Also the routers can connect two levels of routers in a single hop. In the case of a single level of l routers, the data can be transferred from the host computer to all N CA cells in three hops. For example, if we have a grid of $N = 10.000$ CA cells arranged in a (100×100) grid with $d = 8$ and $r = 36$ then $dr^2 = 10.368$ CA cells can be reached with a single level of r routers, so the whole CA grid can be filled with initial data in three hops.

Note that a certain degree of pipe-lining in the communication is possible that can further decrease the total communication time. The ratio between the communication and calculation time limits the scalability and speed-up of the execution, as usual in all parallel systems.

III. Related work and problem definition

In percolation theory, one of the fundamental task is to find a spanning cluster (termed also infinite cluster) of the same sites that connect two opposite borders of a simulated square domain. Such clusters appear if the probability that a site or CA cell is coloured black is higher than the percolation probability p_c [17]. The recursive Hoshen-Kopelman algorithm [18] is a popular algorithm for the identification of infinite clusters that has been also successfully parallelized [19] on distributed memory computers. Several other approaches to the paralellization of graph algorithms are presented in [20].

The next task, which is computationally more complex and therefore still a bottleneck, is the backbone identification in infinite clusters. Informally, all sites in dead ends or loops that can not contribute to the "transport" of a matter trough the infinite cluster has to be identified and removed to determine the backbone. Tarjan's recursive depth-first-search (DFS) algorithm [21] is well known and often used for the backbone identification. The key to finding the backbone is to recognize the articulation sites of the infinite cluster. A local procedure for recognizing articulation sites along with an improved, almost four times faster, algorithm for the backbone identification was proposed in [22].

An interesting approach for the backbone identification in an infinite cluster, based on the principle of direct electrifying solved by FEM methodology, is presented in [23]. All of the backbone finding algorithms are recursive with a risk of stack overflow in large systems.

As an alternative to the above listed approaches, we propose a CA algorithm for identification of infinite clusters and their backbones in 2-D grids with open boundary conditions. The algorithm relays on local rules and can resolve the problem of stack overflow in large systems. The inherent data-parallel approach of CA can improve efficiency and speed-up of the execution.

We consider a CA as a two dimensional lattice network of unite squares (cells) whose centres are in integer lattice (grid). For simplicity, we suppose that the grid has $N = L^2$ cells (i, j) with positions determined by indices $i, j = 0, ..., L-1$ in x and y directions, where $L \geq 3$. Each cell can exist in a finite number of states. Cells of the lattice network change their states in discrete moments in time. Cell's next state is defined by *local transition function*, which manages with altering the states of each cell, based on the present cell state and states of neighbourhood's cells. We use Moore neighbourhood with twenty-four neighbours. A cell has four nearest neighbours (*nn*) and four next-nearest neighbours (*nnn*) and sixteen not next - nearest neighbours. Two cells (i_1, j_1) and (i_2, j_2) are *nn*-neighbours if $|i_1 - i_2| + |j_1 - j_2| \leq 1$, *nnn*-neighbours if $(i_1 - i_2)^2 + (j_1 - j_2)^2 = 2$ and *nnnn*-neighbours if $2 < (i_1 - i_2)^2 + (j_1 - j_2)^2 \leq 4$.

For a CA A applying the local transition function φ_A to all cells of a configuration $Conf$ simultaneously, we get the sequence of configurations $conf_A(t, Conf)$, where $t = 0, 1, 2, ...$ is a *time step*.

The initial configuration is represented by a 2-D grid of square cells and each cell can exist in two different states, white or black. On the top and bottom boundaries of the grid are black cells only while the left and right boundaries are white. All remaining cells of the grid, are coloured black with probability p and white with probability $1 - p$. These probabilities are independent for each cell. Two examples of grids with $p = 0.5 < p_c$ and $p = 0.6 > p_c$ are shown in Figures 2 and 1, respectively. We can see that an infinite cluster appears in the example from Figure 1.

Two cells (i_1, j_1) and (i_n, j_n), which are in the same state are *nn*-connected if there exists a sequence of cells (i_k, j_k), $2 \leq k \leq n$ which are in the same state, such that each pair (i_{k-1}, j_{k-1}) and (i_k, j_k) are *nn*-neighbours. Similarly, they are *nnn*-connected if the sequence of

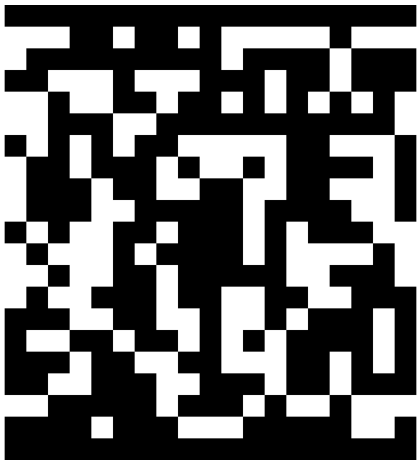


Figure 1: Initial grid configuration generated with $p = 0.5$ without infinite clusters.

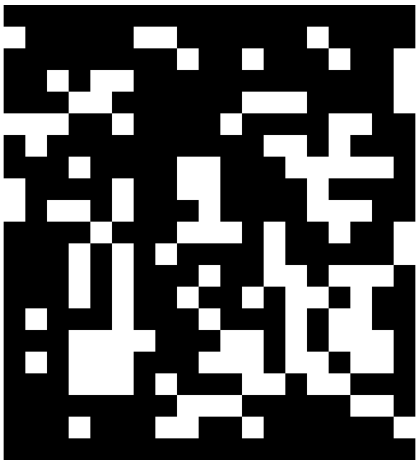


Figure 2: Initial grid configuration generated with $p = 0.6$ with a single infinite cluster.

the cells (i_k, j_k), $2 \leq k \leq n$ which are in the same state, such that each pair (i_{k-1}, j_{k-1}) and (i_k, j_k) are *nn*- or *nnn*-neighbours.

A black *nn*-cluster is a group of black cells which are *nn*-connected. The *infinite cluster* is a black cluster which spans from the top to the bottom row of a grid. The *backbone* is a subset of the infinite cluster, which cells are *nn*-connected with the top or bottom row cells with at least two disjoint chains. A cell is an *articulation cell* of an infinite cluster if removing the cell (i.e. changing its state to white) splits the infinite cluster into two or more parts, at least one part being connected to neither the top nor the bottom row.

In the same way, an *nnn*-cluster is a group of cells in the same state that are *nnn*-connected. In particular, white *nnn*-cluster is a group of white cells that are *nnn*-connected.

The proposed CA algorithm for identification of infinite cluster and its backbone is implemented in four steps. In the Step 1, white *nnn*-clusters are labelled with different colors. After finishing this task the algorithm identified the cells that belong to the infinite cluster. In Step 2, the algorithm recognizes articulation cells of the infinite cluster. Some of them are permanently removed, but some of them are marked by white color. In Step 3, the parts of backbone are labelled. Finally, in Step 4, some of previously marked white cells become a part of the backbone and the backbone is identified.

IV. Solution algorithm

Let $i, j \in \{0, 1, 2, ..., L-1\}$ and $t \in N, t \geq 0$. Denote by $c_{(i,j)}(t)$ the state of a cell (i, j) in time step t and by $c\,(t)$ an argument of a local transition function, which is an ordered collections of *nn*- , *nnn*- and *nnnn*- 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 five CAs A_i by their local transition functions φ_{A_i}, $i = 1, 2, ..., 5$. For a CA A_i applying the local transition function φ_{A_i} to all cells on a configuration $Conf$ simultaneously, we get the sequence of configurations $conf_{A_i}(t, Conf)$, $i \in \{1, 2, ..., 5\}$, where $t = 0, 1, 2, ...$ is a time step.

Step 1: CA A_1 will change states of some white cells in each white *nnn*-cluster. These new states will be all different because their colors are related to their positions. Let a and b are the lowest left and lowest right cells on the left and right boundaries of the grid in initial configuration C_0. Let $LR = \{a, b\} \bigcup$ $\Big\{(i, j) \in C_0 | c_{(i,j)}(0) = 0 \wedge c_{(i+1,j-1)}(0) = c_{(i+1,j)}(0) = c_{(i,j-1)}(0) = 1\Big\}$. Then $\varphi_{A_1}(c(t)) = c^1_{(i,j)}(t+1)$ where:

$$c^1_{(i,j)}\,(t+1) = \begin{cases} i * L + j + 2, \ t = 0 \wedge (i, j) \in LR \\ c_{(i,j)}\,(t)\,, \text{ otherwise.} \end{cases}$$

and $t = 0, 1, 2, ...$ is a time step.

Colour $i * L + j + 2$ of a cell (i, j) depends on the position of a cell in the grid. Hence, the CA A_1 changes the state of a cell in a new unique state, different from all other cells. Also, lowest left cells of any white *nn*- and *nnn*-cluster has black *nnn*-neighbours $(i+1, j-1), (i+1, j), (i, j-1)$. Hence, in every white *nnn*-cluster at least one cell will have its unique color, different from white. Note, for $t > 1$ the CA A_1 will remain idle.

CA A_2 will colour every white *nnn*-cluster with unique color, different from black and white. Let $Cols_{(i,j)}(t) = \Big\{c_{(k,l)}(t) | k \in \{i-1, i, i+1\}, l \in \{j-1, j, j+1\}\Big\}$ be the set of all colors of *nn*- and *nnn*-neighbours of a cell (i, j). Then $\varphi_{A_2}(c(t)) = c^2_{(i,j)}(t+1)$ is defined by:

$$c^2_{(i,j)}\,(t+1) = \begin{cases} max(Cols_{(i,j)}(t)), \ c_{(i,j)}(t) \neq 1 \wedge max(Cols_{(i,j)}(t)) > 1 \\ c_{(i,j)}\,(t)\,, \text{ otherwise.} \end{cases}$$

where $t = 0, 1, 2, ...$ For the CA A_2, after some time step, no cell will be changed. In implementation of the CA A_2 a global variable is used to identify this.

If cells a and b have the same colors then left and right boundaries are in same *nnn*- cluster. Hence, if cells a and b have the same colors initial configuration doesn't have an infinite cluster.

Step 2: CAs A_i, $i = 3, 4, 5$ will implemented the part of the algorithm from [22].

CA A_3 will locally identify articulation cells and color them with white color, in the first time step. In the second time step, the CA will label some *nn*-neighbours of white cells, because some of the white cells may belong to the backbone. Note, that after *Step 1* no cell remains white. Definition of CA A_3 is obtained from discussion and the corollary in [22].:

Corollary 1 *Let a be a cell of the black infinite cluster, and let G_a be a set of a and its nn and nnn cells then a is an articulation cell if and only if there are in G_a at least two white cells, referred to as b and c, that belong to the same nnn-cluster but cannot be connected by the white cells from G_a.*

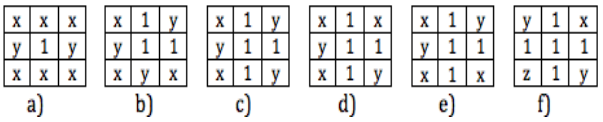


Figure 3: Cases of articulation cells: y is any color different from black and x and z are any colors with a restriction that z must be different from y.

In figure 3 some cases of G_a are shown. The set of the shown cases together with their rotated configurations for 90, 180, 270 degrees, are denoted by Ar. Let Tag is the set of black *nn*-neighbours of white cells which are tagged as contact couple. For example, for central cell (i, j) in figure 3 d) the contact couple is $\{(i+1, j), (i, j+1)\}$, in figure 3 e) the contact couple is $\{(i+1, j), (i, j-1)\}$ and in figure 3 f) contact couples are $\{(i, j+1), (i+1, j)\}$ and $\{(i-1, j), (i, j-1)\}$. In implementation we will have twelve cases for the tagging and we will use *nnnn*-neighbours. Here, we only use one color $tag = L^2 + 2$, because of a simplified explanation. Now, we define the local transition function $\varphi_{A_3}(c(t)) = c^3_{(i,j)}(t+1)$ by:

$$c^3_{(i,j)}(t+1) = \begin{cases} y,\ t = 0 \wedge (i,j) \in Ar\, a), b), c \\ 0, t = 0 \wedge (i,j) \in Ar\, d), e), f) \\ tag, t = 1 \wedge (i,j) \in Tag \\ c_{(i,j)}(t),\ \text{otherwise.} \end{cases}$$

where $t = 0, 1, 2, ...$ is a time step and y is from figure 3 a), b), c) . Note, for $t > 2$ the CA A_3 will remain idle.

Step 3: CA A_4 will color black cells from the infinite cluster that are in its backbone, except some articulation cells.

Let $m = L^2 + 3$. The local transition function φ_{A_4} will label a part of backbone by a color m. Let $TB = \left\{(i,j) | c_{(i,j)}(t) = 1 \wedge (j = 0 \vee j = L-1)\right\}$ be the set of top and bottom boundary black cells. Let Nn be a set of black and tag cells whose at least one of its *nn*-neighbours is in state m. Let $Ntag$ be a set of tag cells which have one white *nn*-neighbour and one different tagged *nnn*-neighbour or two white *nn*-neighbours and two m *nnn*-neighbours. Now, the local transition function $\varphi_{A_4}(c(t)) = c^4_{(i,j)}(t+1)$ is defined by:

$$c^4_{(i,j)}(t+1) = \begin{cases} m,\ (i,j) \in TB \bigcup Nn \bigcup Ntag \\ c_{(i,j)}(t),\ \text{otherwise.} \end{cases}$$

where $t = 0, 1, 2, ...$ is a time step. For the CA A_4, after some time step, no cell will be changed. In implementation of the CA A_4 a global variable is used to identify this.

Step 4: Here, we will define CA A_5 who decide which white cell from *Step3* is in the backbone. We will use the fact "a white cell should be restored to be a cell of the backbone if either (i) it has at least two *nn*-neighbours belonging to the infinite cluster, or (ii) it has an *nn*-neighbour of the infinite cluster and a white *nn* cell which are *nn*-connected by another cell of the infinite cluster" from paper [22].

Let B be a set of white cells with two or more m *nn*-neighbours and cells with an m *nn*-neighbour and a white *nn*-neighbour which is *nn*-connected by another m cell. The local transition functions $\varphi_{A_5} = c^5_{(i,j)}(t+1)$ is defined by:

$$c^5_{(i,j)}(t+1) = \begin{cases} m,\ (i,j) \in B \\ c_{(i,j)}(t),\ \text{otherwise} \end{cases}$$

where $t = 0, 1, 2, ...$ is a time step. Note, for implementation of the CA A_5 we will use *nnnn* - neighbours.

Using the described CAs in a loop we can form algorithm 1 for the identification of infinite clusters and their backbones.

Data: Initial configuration C_0
Result: If the infinite cluster does not exist then the algorithm stops else the obtained set of m cells is backbone.

```
φ_{A_1};
while exist cell which change its state do
    φ_{A_2};
end
if state of lowest left cell == state of lowest right cell then
    stop;          //no infinite cluster
else
    φ_{A_3};   // articulation cells become white
    φ_{A_3};   // nn-neighbours some of white cells are tagged
    Let m be an unique color;
    while exist cell which change its state do
        φ_{A_4};
    end
    φ_{A_5};   // some articulation cells are in backbone
end
```

Algorithm 1: Algorithm for identification of an infinite cluster and its backbone.

V. Experimental results and discussion

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 percolation probabilities, i.e. densities of black cells in initial configuration. In Figures 4 and 5 the final configurations are shown after running the Algorithm 1 on initial configurations from Figures 1 and 2, respectively.

It is evident from figure 4 that there is no infinite cluster in the initial configuration from figure 1. However, figure 5 indicates a

backbone of an infinite cluster, which is obviously present on the initial configuration from figure 2.

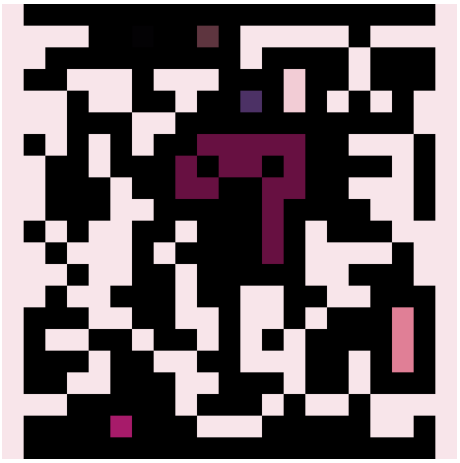


Figure 4: Final result after the application of proposed algorithm on the initial configuration from figure 1.

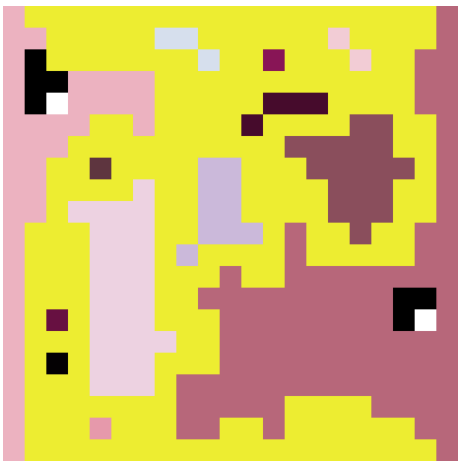


Figure 5: Yellow backbone for initial configuration from figure 2.

The preliminary numerical results for different sizes of grids and probability of black cells $p = 0.592745$ are shown in Table 1. We calculated the average time steps required for 100 initial configurations of each for grid size. The slope in the number of time steps can be estimated as $\Delta L^2/\Delta M$ and is significantly smaller as in [22]. However, the current implementation of the algorithm with NetLogo is limited with grid sizes. The results still indicate that the proposed algorithm can be faster than known algorithms for the backbone identification. For definite confirmation we need a data-flow implementation of the algorithm, which is a part of our future investigation.

Advantages of the proposed approach are in using data-flow approaches in *Step 1* and *Step 3* [24] while the well known algorithms for labelling components are based on DFS approach, which is recursive and hard to parallelize [25]. The proposed CA definition of the proposed data parallel 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.

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.

Dimesions of grid (L)	Mean time steps (M)
21x21	26.3
41x41	49.61
61x61	58.81
81x81	73.12
101x101	81.59
121x121	85.49
141x141	93.74
161x161	91.38
181x181	95.6
201x201	97.86

Table 1: Average number of time steps M as a function of grid size L.

References

[1] J. Neumann, Theory of Self-reproducing Automata, University of Illinois PressNeumann, 1966.

[2] J. Thatcher, Universality in the von neumann cellular model, Tech. rep., University of Michigan,. 03105-30-T (1964).

[3] E. Codd, Cellular Automata, Academic Press, NewYork, 1968.

[4] E. Burks, Essays on Cellular Automata, University of Illinois Press, 1966.

[5] S. Wolfram, Theory and Applications of Cellular Automata, World Scientific Publication, Singapore, 1986.

[6] P. Hogeweg, Cellular automata as a paradigm for ecological modeling, Appl. Math. Comput. 27 (1988) 81Ű100.

[7] G. Ermentrout, L. Edelstein-Keshet, Cellular automata approaches to biological modeling, J. Theor. Biol. 160 (1993) 97–133.

[8] G. Horgan, B. Ball, Simulating diffusion in a boolean model of soil pores, European Journal of Soil Science 45 (1994) 483–491.

[9] T. Ikenaga, T. Ogura, A DTCNN universal machine based on highly parallel 2-D cellular automata CAM2, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 45 (1998) 538–546.

[10] R. Trobec, I. Jerebic, Local diagnosis in massively parallel systems, Parallel Computing 23 (1997) 721–731.

[11] J. Epstein, Generative Social Science, Princeton University Press, 2006.

[12] E. Motuk, R. Woods, S. Bilbao, Implementation of finite difference schemes for the wave equation on FPGA, in: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Vol. III, 2005, pp. III237–III240.

[13] G. Kosec, P. Zinterhof, Local strong form meshless method on multiple graphics processing units, Computer Modeling in Engineering and Sciences 91 (2013) 377–396.

[14] R. Trobec, Evaluation of d-mesh interconnect for SoC, in: Proceedings of the International Conference on Parallel Processing Workshops, 2009, pp. 507–512.

[15] M. J. Flynn, O. Mencer, V. Milutinovic, G. Rakocevic, P. Stenstrom, R. Trobec, M. Valero, Moving from petaflops to petadata, Commun. ACM 56 (5) (2013) 39–42.

[16] R. Trobec, Two-dimensional regular d-meshes, Parallel Computing 26 (13-14) (2000) 1945–1953.

[17] D. Stauffer, A. Aharony, Introduction to Percolation Theory, Taylor and Francis, London, 1992.

[18] J. Hoshen, R. Kopelman, Percolation and cluster distribution: I. cluster multiple labeling technique and critical concentration algorithm, Phys. Rev. B 14 (1976) 34–38.

[19] J. Teuler, J. Gimel, Direct parallel implementation of the hoshen-kopelman algorithm for distributed memory architectures, Computer Physics Communications 130 (2000) 118–129.

[20] S. Hong, T. Oguntebi, K. Olukotun, Efficient parallel graph exploration on multi-core cpu and gpu, in: Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on, 2011, pp. 78–88.

[21] T. Robert, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.

[22] W.-G. Yin, R. Tao, Algorithm for finding two-dimensional site percolation backbones, Physica B 279 (2000) 84–86.

[23] C. Li, T.-W. Chou, A direct electrifying algorithm for backbone identification, J. Phys. A: Math. Theor. 40 (2007) 14679–14686.

[24] L. Biehl, Forest fires, oil spills, and fractal geometry, Mathematics teacher 92 (1999) 128.

[25] J. H. Reif, Depth-first search is inherently sequential, Information Processing Letters 20 (1985) 229–234.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing
First NESUS Workshop • October 2014 • Vol. I, No. 1

Exploiting data locality in Swift/T workflows using Hercules

Francisco Rodrigo Duro,† Javier Garcia Blas,∗ Florin Isaila,+ Jesus Carretero*

University Carlos III, Spain † ∗ + *
frodrigo@arcos.inf.uc3m.es†, fjblas@arcos.inf.uc3m.es∗, fisaila@inf.uc3m.es+, jesus.carretero@uc3m.es*

Justin M. Wozniak†, Rob Ross‡

Argonne National Laboratory, USA † ‡
wozniak@mcs.anl.gov†, rross@mcs.anl.gov‡

Abstract

The ever-increasing power of supercomputer systems is both driving and enabling the emergence of new problem-solving methods that require the efficient execution of many concurrent and interacting tasks. Swift/T, as a description language and runtime, offers the dynamic creation and execution of workflows, varying in granularity, on high-component-count platforms. Swift/T takes advantage of the Asynchronous Dynamic Load Balancing (ADLB) library to dynamically distribute the tasks among the nodes. These tasks may share data using a parallel file system, an approach that could degrade performance as a result of interference with other applications and poor exploitation of data locality. The objective of this work is to expose and exploit data locality in Swift/T through Hercules, a distributed in-memory store based on Memcached, and to explore tradeoffs between data locality and load balance in distributed workflow executions. In this paper we present our approach to enable locality-based optimizations in Swift/T by guiding ADLB to schedule computation jobs in the nodes containing the required data. We also analyze the interaction between locality and load balance: our initial measurements based on various raw file access patterns show promising results. Moreover, we present future work based on the promising results achieved so far.

Keywords Locality, In-memory storage, Swift/T, workflows

I. Introduction

Storage systems represent one of the main bottlenecks in modern high-performance systems and are expected to pose a significant challenge when building the next-generation *exascale* systems [2]. Large-scale storage systems are likely to be hierarchical [4], a configuration that will probably be achieved by exploiting data locality and asynchronously moving data among hierarchy levels [7].

In order to extract maximum performance from the new hardware, exascale systems will require new problem-solving approaches. One of the most promising candidate approaches is the many-task paradigm relying on a workflow model. In this paper we propose a solution using Swift/T, a programming model and runtime developed at Argonne National Laboratory that simplifies the development and deployment of many-task applications on large-scale systems. To expose and exploit data locality in Swift/T, we use Hercules, a distributed in-memory store based on Memcached. Hercules offers Swift/T workers a shared storage in which I/O nodes can be dynamically deployed for increasing data locality, while scaling better than traditional shared file systems. Additionally, the performance can be isolated from the shared file-system load peaks, a feature that will be especially important on exascale systems where several applications can be running concurrently.

II. Swift/T

Swift [11] is a programming model and runtime engine that permits users to easily express the logic of many-task applications by using a high-level language called Swift. The latest Swift implementation, called Swift/T [12], can quickly launch tasks in any of the available workers using the dataflow model Turbine.This model can be deployed and distributed and can generate tasks with the throughput required by next-generation exascale systems [13].

As seen in Figure 1, Swift/T comprises three main components: the Swift compiler, the Turbine engines, and the Asynchoronous Dynamic Loud Balancing (ADLB) module [8]. The first step consists in converting the Swift code into Turbine code. The Swift language is a scripting language that can easily used to describe parallel algorithms. A Swift program specifies different leaf tasks with their input and output clearly characterized. These tasks can be written in the Swift language or can be independent programs written in any other language, treated by Swift as black boxes.

The second step is the identification of dependencies in the Turbine code. Independent tasks can be run in parallel, while data-dependent tasks will be held by the Turbine engines until every dependency is fulfilled. When a task is ready to run, it is dispatched to the ADLB load-balancing module. The Turbine engines can run on any number of nodes for additional load balancing.

In the third step, the ADLB module schedules the tasks to be

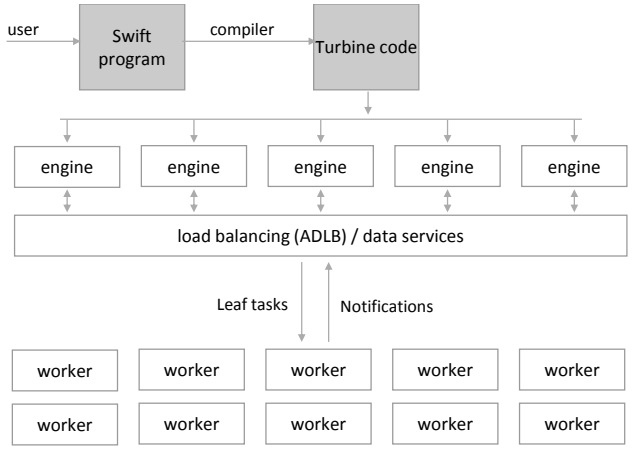


Figure 1: Architecture of the Swift/T runtime. Swift code is compiled into the Turbine code evaluated by the engines; workers execute leaf task applications.

launched on the available workers. When a task finishes, ADLB collects the results and notifies any Turbine engines subscribed to the outputs from that task. Upon notification, the Turbine engines update the data dependencies and release any remaining tasks that are ready to run.

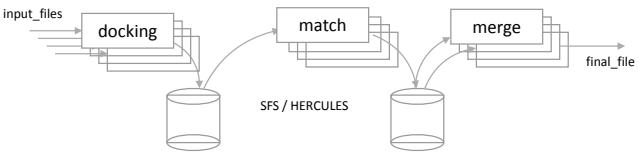


Figure 2: Scheme of a typical many-task workflow application: a protein-docking simulation. The output of one task is the input of the next task.

For a better understanding of the applications representative for Swift/T, Figure 2 shows an example of a workflow application. The application is a typical protein-docking workflow, in which a massive number of files describing protein characteristics are to be processed in order to evaluate how each combination of files docks. For this purpose, three different modules are applied consecutively, represented in the figure as boxes.

The *docking* module, written in C, evaluates two protein files and generates a temporary output file containing their combination. The *match* module, written in Java, takes the output of the previous file and determines whether the combination is correct, classifying it with a label and writing it to an output file. The *merge* application, written in Swift, uses all the files generated by the *match* application and merges them in a single file counting the number of labels of each kind. This file is the final output of the application. A snippet of the source code sample of this application is shown in Figure 3.

A typical protein-docking scenario involves thousands of different proteins producing millions of possible combinations. Manually launching these combinations is tedious work, and classical scripting languages do not have the capability to run independent

```
1  import string;
2  import files;
3
4  app (file f_output) dock (file f_in1, file f_in2)
5  {
6     "./docking" f_in1 f_in2 f_output;
7  }
8
9  app (file f_output) match (file f_input)
10 {
11    "java match" f_input f_output;
12 }
13
14 (file f_results) merge(file f_input[])
15 {
16    foreach f in f_input {
17       // Merge algorithm
18    }
19 }
20
21 main
22 {
23    file fin[];
24    file f_match[];
25    foreach fin1,i in fin{
26       foreach fin2,j in fin{
27          file f_tmp = dock(fin1,fin2);
28          index = i+j + j;
29          f_match[index] = match(f_tmp);
30       }
31    }
32    file f_out<"results.out"> = merge(f_match);
33 }
```

Figure 3: Example of Swift source code.

tasks in parallel. With the simple code from Figure 3, Swift/T evaluates the dependencies and runs millions of instances of existing programs without requiring any change in the source code—not even a recompilation if the code was previously compiled for the machine on which it is supposed to run. File reads and writes are made through a file system shared by every Swift/T worker node.

Currently, Swift/T uses distributed memory to store basic variables; but it requires a shared file system to store files, relying on the distributed memory mechanism only for solving file dependencies using file paths. As shown in Figure 3, this mechanism is used even when the files are going to be part of a workflow: the output of a task generates a temporary file used as the input of the next task. Hercules can alleviate this I/O bottleneck by storing the files in the main memory of the worker nodes. In addition to this problem, the original Swift/T scheduler was not able to exploit data locations, which prohibited the use of systems like Hercules. As shown in Section IV, the ADLB scheduler, driven by Swift, can be guided to exploit data locality and access Hercules files without network communication, colocating the computation in the worker that contains the data and obtaining performance improvements.

III. Hercules

The distributed memory space of Hercules [5] can be used by applications as a virtual storage device for I/O operations. We have adapted this space for use as in-memory shared storage for the Swift/T workers. Our approach relies on an improved version of the Memcached [6] servers, which provide a storage solution for the worker.

As can be seen in Figure 4, our solution consists of two levels: client library and servers. On top is the client user-level library with a layered design. Back-ends are based on the Memcached server, extending its functionality with persistence and tweaks. Main ad-

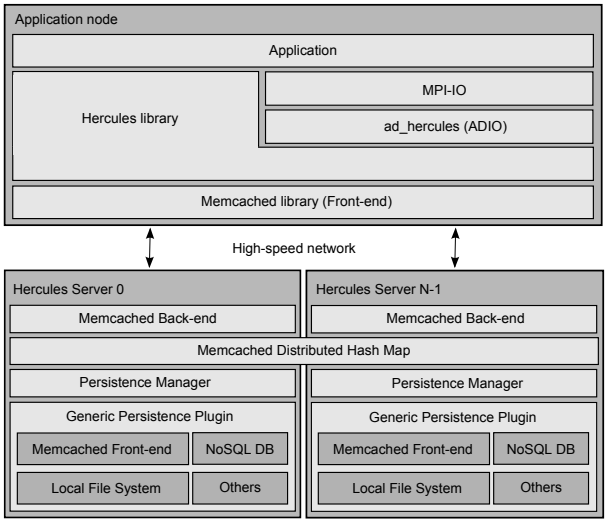


Figure 4: Hercules architecture. On the top is the client side, a user-level library. On the bottom is the server side with the Hercules I/O nodes divided in modules.

vantages offered by Hercules are scalability, ease of 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, in cases of high metadata accesses load. Data and metadata placement is calculated on the client side by an algorithmic hashing. The servers, on the other hand, are completely stateless.

Ease of deployment and flexibility are tackled at the client side by using a POSIX-like user-level interface (open, read, write, close, etc.) in addition to the classic put/get approach in current NoSQL databases. Existing software requires minimal changes to run with Hercules. Servers can be deployed in any kind of Linux system at the user level, and persistence can be easily configured by using existing plugins or developing new ones.

Performance is achieved by exploiting the parallel I/O capabilities of Memcached servers. Hercules is easy to deploy on as many nodes as necessary, and every node can be accessed independently, multiplying the total throughput peak performance. Furthermore, each node can serve requests in a concurrent way thanks to a multithreading approach. The combination of these two factors results in full scalability, in both the number of nodes and the number of workers running on each node.

In addition to the default algorithmic hashing provided by Memcached, we have designed two new custom placement algorithms. The first algorithm is a locality-aware placement, capable of placing all the items related with one specific file in the same node. Thus, if a task needs to access an existing file and is running on the same node as the Hercules server containing the data, it can access the whole file locally. Combining data locality and concurrent request serving, our solution can achieve intranode scalability, serving requests in parallel from different workers running on the same node—a common approach in current and future multicore compute nodes—and avoiding the usual single network interface bottleneck. The second algorithm has been designed but is not yet fully developed; its objective is to take into account load factors (capacity, CPU load, burst peaks) when selecting the data placement.

IV. Integrating Swift/T and Hercules

The objective of this work is to combine the Swift/T many-task runtime with our Hercules storage system in order to perform I/O operations in-memory instead of using default systemwide shared file systems. The integration of Hercules and Swift/T takes advantage of two features offered by each solution: (1) Hercules offers an ad hoc distributed storage shared among all available workers, using their main memory for storing data; and (2) Swift/T has an experimental feature, called *@location* [14], that can be used to override the default scheduling, placing a specific task on any desired worker node.

To integrate both features, we have developed a mechanism that spawns one Hercules server on each of the worker nodes available for Swift/T. We have implemented a function to easily determine where a specific file is located or where it will be located if it has not yet been written to expose data locality. We have used the *@location* experimental feature to schedule read operations in nodes containing the required data and write operations in the nodes that are going to contain the data to exploit data locality. The combination of these three techniques enables users to perform any kind of read/write file operation querying the Hercules server running in the same node, without needing network communication or disk operations.

Our solution also can be used as an ad hoc distributed in-memory storage, resulting in easier deployment and better scalability than conventional shared file systems provide. Furthermore, our ad hoc storage can avoid the peak load performance penalties that occur from sharing storage resources between different applications running on the same system, thus reducing the shared file system noise in I/O operations.

V. Evaluation

To evaluate our integration of Swift/T and Hercules, we ran a series of tests on the Fusion cluster at Argonne National Laboratory. This cluster has 320 nodes, composed of dual-socket boards with quad-core 2.53 GHz processors and 36 GB of main memory. The intercommunication networks are InfiniBand QDR (4 GB/s) and Gigabit Ethernet.

For the evaluation, we developed a synthetic application with 64 tasks consisting of 32 tasks performing a file write of 2 GB each and 32 tasks performing reads of the previously created 2 GB files. We compared raw file reads/writes with GPFS, Hercules without locality, and Hercules using data locality in all the tasks (all the I/O operations were done inside the node, without using the network). Hercules used the InfiniBand network over TCP/IP for I/O; the GPFS file system has a peak performance of 3200 MB/s over the InfiniBand network. We focused on two cases: scalability in the number of nodes (launching one worker per node) and scalability in the number of workers per node (with a fixed node setup).

As can be seen in Figure 5 and Figure 6, our solution scales better than GPFS, especially when contention is high. In other words,

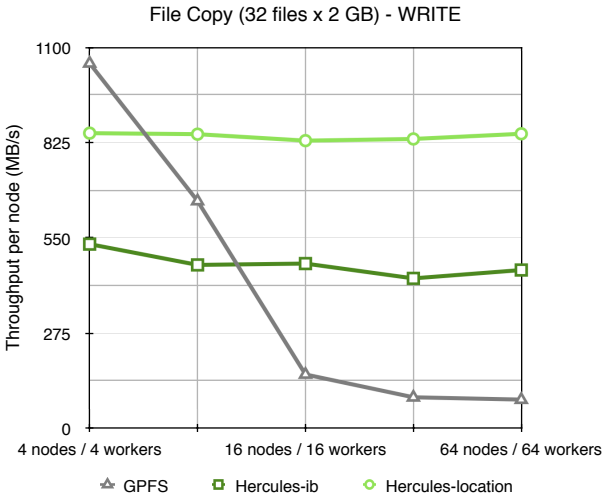


Figure 5: Raw file write throughput per node comparison between our proposed solution and GPFS, when scaling the number of nodes, running one worker per node.

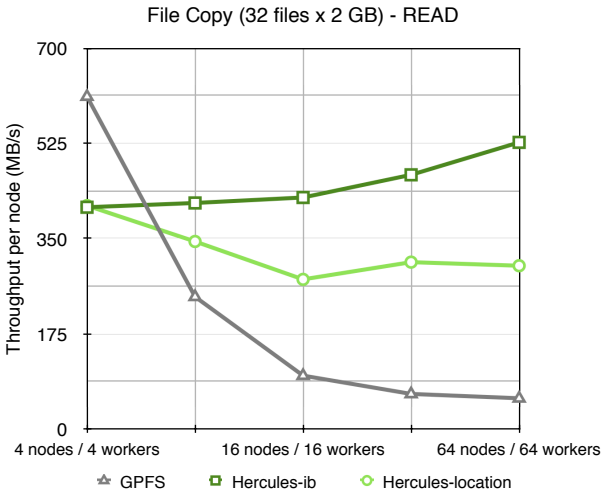


Figure 6: Raw file read throughput per node comparison between our proposed solution and GPFS, when scaling the number of nodes, running one worker per node.

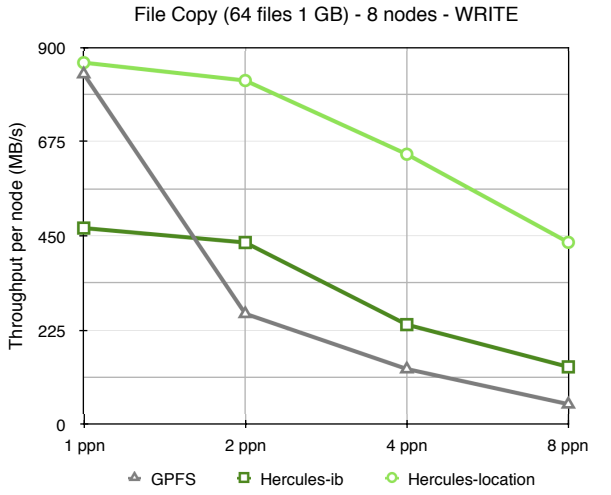


Figure 7: Raw file write throughput per node comparison between our proposed solution and GPFS scaling the number of workers per node running in a fixed 8-node setup.

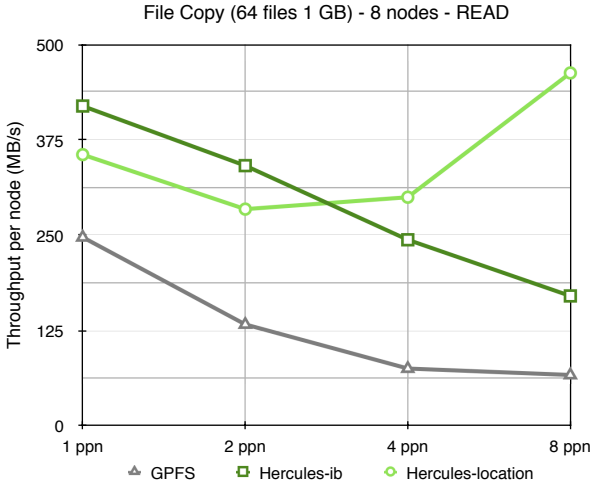


Figure 8: Raw file read throughput per node comparison between our proposed solution and GPFS scaling the number of workers per node running in a fixed 8-node setup.

when many workers are trying to access the file system concurrently, Hercules takes advantage of the increased number of I/O nodes launched and colocated with each worker to perform I/O accesses in a parallel way, whereas GPFS shares its maximum bandwidth between all the nodes. The throughput represented in both figures corresponds with the throughput per node, resulting in an aggregated throughput that scales with the number of worker nodes. Hercules performs similarly over the network and locally, probably because of the overhead of the TCP/IP stack even for local accesses.

Figure 7 and Figure 8 show how when we increased the number of workers per node, fixing the number of nodes to 8, the results are similar to the previous case: our solution scales better with the number of workers, whereas GPFS performance is affected by contention. Again, we note that the represented throughput is measured per worker, which explains the performance hit of GPFS, which has to share the available bandwidth among all the workers.

In this case, the locality-aware version performs better, because it can serve more workers in parallel without contention. That behavior is explained by the ability of the Hercules I/O nodes to serve various workers in parallel thanks to the multithreaded implementation. When the queries are done inside the same node, there is

First NESUS Workshop • October 2014 • Vol. I, No. 1

no need for sharing the network interface, resulting in an improved performance over remote queries using the InfiniBand network interface over TCP. Requests can be served by the multithreaded Hercules I/O nodes using the loopback TCP stack, avoiding sharing the available bandwidth.

These tests demonstrate how our proposed solution scales better than current state-of-the-art parallel shared file systems for I/O-bound applications. We have also demonstrated how our solution suffers less from contention and offers a more stable performance when different applications share the same parallel file system. In contrast, however, real-life applications usually mix computation and I/O operations. This behavior results in less contention to the shared file system and should be evaluated in the future. Another issue that was exposed when evaluating our solution is related to the overridden scheduling. Currently the *@location* functionality allows only one specific node to be selected; but since the load is not balanced among workers running on the same node, a load imbalance can result, reducing the performance gains produced by the improved throughput.

VI. Related Work

The increasing popularity of many-task computing and workflow engines and the I/O bottleneck in such scenarios have led several researchers to investigate this problem.

Parrot [9] is a tool for attaching existing programs to remote I/O systems through the POSIX file-system interface, and Chirp [10] is a user-level file system for collaboration across distributed systems such as clusters, clouds, and grids. They are usually combined to easily deploy a distributed file system ready to use with current applications through a POSIX API. Many characteristics are shared with Hercules: user-level deployment without any special privileges, transparency through the use of a widely used interface, and easy deployment using a simple command to start a new server. Hercules, however, is designed to achieve high scalability and performance by taking advantage of as many compute nodes as possible for I/O operations. Moreover, Hercules uses main memory for storage improving performance in data-locality-aware accesses.

Costa et al. proposed using extended file attributes in MosaStore [1, 3] to provide communication between the workflow engine and the file system through the use of hints about the data. The workflow engine can provide these hints directly to the file system, or the file system can infer the patterns by analyzing the data accesses. The MosaStore approach is radically different from Hercules, using a centralized metadata server instead of the fully distributed, easy-to-use, and flexible deployment approach of our proposed solution.

The AMFS shell [15] offers programmers a simple scripting language for running parallel scripting applications in-memory on large-scale computers. The objective of this solution is similar to the combination of Swift/T and Hercules, but Swift/T can automatically solve data dependencies and launch tasks to workers in a more efficient way by using distributed Turbine engines. AMFS and Hercules also share the distributed metadata approach; the main difference is that AMFS shell programs can explicitly specify in-memory or persistent storage, whereas Hercules can be deployed with persistence enabled in a transparent way for the programmer.

HyCache+ [16] is a distributed storage middleware that allows I/O to effectively leverage the high bisection bandwidth of the high-speed interconnect of massively parallel high-end computing systems. HyCache+ acts as the primary place for holding hot data for the applications (e.g., metadata, intermediate results for large-scale data analysis) and only asynchronously swaps cold data on the remote parallel file system. Similarities between HyCache+ and Hercules include their fully distributed metadata approach, use of compute network instead of the shared storage network, and the high scalability capabilities. HyCache+ relies on POSIX, however, whereas Hercules offers the possibility of using a POSIX-like interface and get/set operations. Moreover, HyCache+ focuses on enhancing parallel file systems in a generic way, whereas Hercules has been designed to work specifically with a many-task engine, exposing and exploiting data locality in current applications. HyCache+ and Hercules thus share similar ideas, but Hercules is ready to improve many-task I/O performance by focusing on easy and flexible deployment options.

VII. Conclusions

In this paper we have presented the integration of Swift/T and Hercules in order to expose and exploit data locality in many-task workflows. We have evaluated the capabilities of our solution for raw file access. The approach achieves a substantial improvement of throughput performance over that of the GPFS file system. In addition, our solution can deploy as many I/O nodes as Swift/T workers running the application, achieving better scalability than possible with traditional static parallel file systems. Another advantage of our solution is isolation from shared file system noise. In the increasingly common case of various applications running at the same time on the same system, our solution ensures isolation of the I/O performance, independent of the file system load at any specific instant.

To tackle the load imbalance issue, we are working on two new approaches that can be combined. The first one will try to improve the load balance of workers inside the same node. An improved scheduler has been implemented in Swift/T, and we are evaluating it with Hercules. The second approach focuses on load balance among nodes. We are developing a new placement policy to map data in a load-aware way, placing data in the less-loaded nodes or in the nodes with more memory/capacity available. Moreover, to better demonstrate the capabilities of our solution, we will evaluate it with CCTW, a real MapReduce-like application.

Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. Computing resources were provided by the Argonne Leadership Computing Facility. The work presented in this paper was supported by the COST Action IC1305, "Network for Sustainable Ultrascale Computing (NESUS)." The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 328582.

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] G. Bell, J. Gray, and A. Szalay. Petascale computational systems. *Computer*, 39(1):110 – 112, jan. 2006.

[3] 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.

[4] Jack Dongarra, Pete Beckman, Terry Moore, and Aerts. The International Exascale Software Project roadmap. *Int. J. High Perform. Comput. Appl.*, 25(1):3–60, February 2011.

[5] 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.

[6] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[7] Florin Isaila, Javier Garcia Blas, Jesus Carretero, Robert Latham, and Robert Ross. Design and evaluation of multiple-level data staging for blue gene systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):946–959, 2011.

[8] E. L. Lusk, S. C. Pieper, R. M. Butler, and Middle Tennessee State Univ. More scalability, less painâĂť: A simple programming model and its implementation for extreme computing. *SciDAC Rev*, 17, Jan 2010.

[9] Douglas Thain and Miron Livny. Parrot: Transparent user-level middleware for data-intensive computing. *Scalable Computing: Practice and Experience*, 6(3), 2005.

[10] 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.

[11] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A language for distributed parallel scripting. *Parallel Comput.*, 37(9):633–652, September 2011.

[12] J.M. Wozniak, T.G. Armstrong, M. Wilde, D.S. Katz, E. Lusk, and IT. Foster. Swift/T: Large-scale application composition via distributed-memory dataflow processing. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 95–102, May 2013.

[13] Justin M. Wozniak, Timothy G. Armstrong, Ketan Maheshwari, Ewing L. Lusk, Daniel S. Katz, Michael Wilde, and Ian T. Foster. Turbine: A distributed-memory dataflow engine for extreme-scale many-task applications. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies*, SWEET '12, pages 5:1–5:12, New York, NY, USA, 2012. ACM.

[14] Justin M. Wozniak, Michael Wilde, and Ian T. Foster. Language features for scalable distributed-memory dataflow computing. In *Proc. Data-Flow Execution Models for Extreme-Scale Computing at PACT*, 2014.

[15] 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.

[16] Dongfang Zhao, Kan Qiao, and Ioan Raicu. Hycache+: Towards scalable high-performance caching middleware for parallel file systems. In *IEEE/ACM CCGrid*, 2014.

List of Authors