Scalable Computing: Practice and Experience

Scientific International Journal for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 18(2) June 2017

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 18, Number 2, June 2017

TABLE OF CONTENTS

Special Issue on HPC Solutions for Complex Problems:

Introduction to the Special Issue iii

ARank: A Multi-agent Based Approach for Ranking of Cloud Computing Services 105
Arezoo Jahani, Farnaz Derakhshan, Leyli Mohammad Khanli

Automatic Tuning on Many-Core Platform for Energy Efficiency via Support Vector Machine Enhanced Differential Evolution 117
Zhiliu Yang, Zachary I. Rauen, Chen Liu

Integrating Generic FEM Simulations into Complex Simulation Applications 133
Robert Dietze, Michael Hofmann, Gudula Rünger

SaaS for Energy Efficient Utilization of HPC Resources of Linear Algebra Calculations 145
Hrachya Astsatryan, Wahi Narsisian, Georges da Costa

Towards HPC-Embedded. Case Study: Kalray and Message-Passing on NoC 151
Pedro Valero-Lara, Ezhilmathi Krishnasamy, Johan Jansson

Regular Papers:

Impact of Process Allocation Strategies in High Performance Cloud Computing on Azure Platform 161
Hanan A. Hassan, Aya I. Maiyza, Walaa M. Sheta

DR-SWDF: A Dynamically Reconfigurable Framework for Scientific Workflows Deployment in the Cloud 177
Khadija Bousselmi, Zaki Brahmi, Mohamed Mohsen Gammoudi

DOI 10.12694/scpe.v18i2.1282
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. iii–iv. http://www.scpe.org © 2017 SCPE

INTRODUCTION TO THE SPECIAL ISSUE ON HIGH PERFORMANCE COMPUTING SOLUTIONS FOR COMPLEX PROBLEMS

PEDRO VALERO-LARA[*], MAWUSSI ZOUNON[†], MAKSIMS ABALENKOVS[‡], AND FERNANDO L. PELAYO[§]

Scientific and engineering problems are becoming increasingly complex. Such complex problems include numerical simulations, molecular dynamics, computational fluid dynamics, bioinformatics, image processing, and deep-learning, to name a few. In addition to these complexities, improvements in high performance computing (HPC) compromise important modifications on computer architectures, bridging the gap gap between the general scientific user community (in need of an easy access to efficient high performance computations) and the HPC programmers community (in charge of the implementation of such a complex problems efficiently).

Today, HPC programmers and scientific applications have to deal with numerous computer architectures details to take advantage of modern computing systems. Thus, strategies and tools that can help us to adapt our codes over different computing architectures is of vital importance. However, previously, we must know and identify what are the efficient programming strategies and architectonic features. This is a difficult task, as it depends on the particular problem to be computed and the computational platform on which the problem must be computed.

While only a few computational architectures were available during the last decade, today the diversity of HPC systems is more extensive. This range from clusters of common processors to heterogeneous clusters equipped with accelerators (GPU) co-processors. These platforms usually use their own compilers, languages, etc., being the implementation of portable codes very complex.

This special issue provides several studies, which involve different applications and strategies to improve performance and to achieve better usage of modern HPC systems. It is composed by five works.

The work "ARank: A Multi-Agent based Approach for Ranking of Cloud Computing Services" by A. Jahani, F. Derakhsan, and L. Mohammad-Khanl, proposes a new multi-agent based method named ARank, which is applied for ranking algorithm to reduce the waiting time of users. ARank method uses intelligent agents which they choose some candidate services and rank these services based on the quality of service values. Furthermore, the agents of ARank include the satisfaction rate of the earlier users in ranking process. The results of this evaluation show reduction in the waiting time of the users using ARank method, compared to the existing related work, Analatic Hierarchical Process (AHP) and Singular Value Decomposition (SVD).

The paper "Automatic Tuning on Many-Core Platform for Energy Efficiency via Support Vector Machine Enhanced Differential Evolution" by Z. Yang, Z. I. Rauen, and C. Liu, proposes SVM-JADE, a machine learning enhanced version of an adaptive differential evolution algorithm (JADE). They monitor the energy and EDP values of different frequency and voltage combinations of the cores, or power islands, as the algorithm evolves through generations. By adding a well-tuned support vector machine (SVM) to JADE, creating SVM-JADE, they are able to achieve energy-aware computing on many-core platform when running multiple-program workloads. Their experimental results show that their algorithm can further improve the energy by 8.3% and further improve EDP by 7.7% than JADE.

In the next study "Integrating Generic FEM Simulations into Complex Simulation Applications" by R. Dietze, M. Hofmann, and G. Rünger, it is described the efforts for integrating alternative FEM codes into a complex simulation application from the area of engineering optimisation. The application area, as well as the software components, and their interactions are presented. The integration of two different FEM codes is demonstrated based on a dedicated FEM data conversion component.

The work "SaaS for Energy Efficient Utilization of HPC Resources of Linear Algebra Calculations" by H. Astsatryan and G. da Costa, focuses on one of the most important factor of High performance computing (HPC) systems nowadays, that is to limit or decrease the power consumption while preserving a high utilization. With the availability of alternative energy, which powers such systems, there is a need to maximize the usage of

[*]Barcelona Supercomputing Center (BSC), Barcelona, Spain. (pedro.valero@bsc.es)
[†]The University of Manchester, Manchester, UK. (mawussi.zounon@manchester.ac.uk)
[‡]The University of Manchester, Manchester, UK. (m.abalenkovs@manchester.ac.uk)
[§]The University of Catilla La-Mancha, Albacete, Spain. (fernandol.pelayo@uclm.es)

alternative energy over brown power. For now, the usage of alternative energy is varying in time due to different factors such as sunny days, the wind, etc. and it is crucial to have an energy-aware algorithm to maximize the usage of this energy. In this work, a SaaS service is presented to optimize a usage of alternative energy, to reduce the power consumption and to preserve a best possible percentage of resource utilization.

Today one of the most important challenges in HPC is the development of computers with a low power consumption. In this context, the paper "Towards HPC-Embedded. Case Study: Kalray and Message-Passing on NoC" by P. Valero-Lara, E. Krishnasamy, and J. Jansson, analyses one of the new "low-cost" parallel computers, Kalray. Unlike other many-core architectures, Kalray is not a co-processor (self-hosted). One interesting feature of the Kalray architecture is the Network on Chip (NoC) connection. Habitually, the communication in many-core architectures is carried out via shared memory. However, in Kalray, the communication among processing elements can also be via Message-Passing on the NoC. One of the main motivations of this study is to present the main constraints to deal with the Kalray architecture. In particular, memory management and communication. They assess the use of NoC and shared memory on Kalray. Unlike shared memory, the implementation of Message-Passing on NoC is not transparent from programmer point of view. The synchronization among processing elements and NoC is other of the challenges to deal with in the Karlay processor. Although the synchronization using Message-Passing is more complex and consuming time than using shared memory, it is achieved an overall speedup close to $6\times$ when using Message-Passing on NoC with respect to the use of shared memory. Additionally, we have measured the power consumption of both approaches. Despite of being faster, the use of NoC presents a higher power consumption with respect to the approach that exploits shared memory. This additional consumption in Watts is about a 50%.

Last, but not less, we would like to thank the editorial board of SCPE and reviewers for their effort and time.

DOI 10.12694/scpe.v18i2.1283
SCALABLE COMPUTING: PRACTICE AND EXPERIENCE ISSN 1895-1767
Volume 18, Number 2, pp. 105–116. http://www.scpe.org © 2017 SCPE

ARANK: A MULTI-AGENT BASED APPROACH FOR RANKING OF CLOUD COMPUTING SERVICES

AREZOO JAHANI, FARNAZ DERAKHSHAN, AND LEYLI MOHAMMAD KHANLI *

Abstract. Cloud computing enables access to computing, processing and storage resources as a service. These services offered to users through internet and based on payment obligations. There are various service providers and services for users, so users have the challenge of choosing appropriate service, matching their needs. Therefore, having a system which helps users to choose the best service based on their need is very important. In this paper, we propose a new multi-agent based method named ARank, which is applied for ranking algorithm to reduce the waiting time of users. ARank method uses intelligent agents which they choose some candidate services and rank these services based on the quality of service values. Furthermore, the agents of ARank include the satisfaction rate of the earlier users in ranking process. The results of our evaluation show reduction in the waiting time of the users using ARank method, compared to the existing related work, Analytic Hierarchical Process (AHP) and Singular Value Decomposition (SVD).

Key words: Cloud computing, Cloud services, Multi agent system (MAS), Ranking, Quality of service.

AMS subject classifications. 68M14, 68T42

1. Introduction.

Cloud computing enables using different resources as a service [1]. While there are various service providers with different services, the act of choosing one service from a set of services is a challenge [2]-[4]. Because users have different requirements, therefore, reaching the highest performance with the lowest cost depends on choosing an appropriate service. It can be said that if the best service is selected, then we can use the total capacity of the provider [5], [6].

Cloud computing offers three types of services: Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). The difference of these services is the way they grant access of resources and applications available to user. IaaS provides user with the highest level of flexibility and management control over her/his IT resources [7], [8]. PaaS allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app. In this type of service, user do not need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running users application. With a SaaS offered user do not have to think about how the service is maintained; user only need to think about how she/he will use that particular software [9]. Each specific service from these services is selected based on different attributes. For example, each service can be proposed with attributes like accountability, cost, security and usability [2].

In cloud environment, users have different requirements and the services have many kinds of attributes with variable quality. So selecting a service match with the requirement of users is difficult, and does not have a clear solution. But we can find an answer near to optimum one [10]- [12]. In other words, it would be a repetitive task to select some candidate services then compare them with users requirements to rank, while this task can be done in parallel [13]. Therefore, it can be found that ranking services is a problem that can be solved with the distributed problem solving. This kind of problems can be solved using evolutionary algorithms, parallel algorithms and multi agent systems (MAS [14]).

The main challenge of this paper is to select some services based on the request of users, then ranking them. Most existing ranking methods do not include all standard attributes, and they rank services based on some limited attributes [14]- [16]. Some methods include all of the existing services in the ranking [17]- [19]; while it is obvious that there is no need to rank all available services and in fact services with rank 100 and more are not usable for users [20]. Therefore, it can be concluded that it is necessary to have a ranking method including all attributes (or proposed methods must be flexible to add new attributes to it in the future). Also, such a system is not required to rank all services, and one can choose some candidates in advance [21]- [23]. This paper try to address all of these challenges.

*Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. (a.jahani@tabrizu.ac.ir, derakhshan@tabrizu.ac.ir, l-khanli@tabrizu.ac.ir)

In this paper, we use multi-agent systems, and called our proposed method ARank (Agent Rank). In fact, agents can do works in parallel and in distributed way. We use the concepts of cooperation and coordination of agents in our system. In ARank, agents found some candidate services that can satisfy the requirements of users. Then, after comparing the candidate services with requirements, rank the cloud computing services. The ranking method in ARank has steps as follows: at the first step, ARank chooses multiple candidates services that taking advantage of agents. Then, ranks the selected candidates based on quality of services and scores that given by earlear users. As a result, ARank can achieve higher performance during candidate selection phase, because of the cooperation and parallel operations of agents.

After this introduction, we continue this paper as follows: Section 2 reviews and analyses the related work. Section 3 explores the attributes of services for cloud computing. In Section 4, we present our proposed method and architecture for ranking cloud computing services using MASs which we called ARank. Section 5 introduces the ranking phases of ARank and shows the cooperation and negotiation between agents, followed by Section 6 that offers the analyses results. Finally, Section 7 gives a conclusion and future work ideas.

2. Related work. During recent years, several methods have been proposed for ranking cloud computing services. Most proposed methods included all the services in the ranking process [24]. Some of these methods rank all services and show k top rankings to user at the end of ranking, however, ranking all services is not needed. It is effective to choose the best choices of candidate services, in order to avoid ranking all of them. Our proposed ARank method select some candidate services in parallel with ranking them using agents intelligence and their ability in cooperation and coordination.

Chan and Chieu [25] proposed a data collection and analyses algorithm, based on Singular Value Decomposition (SVD) [25]. This mechanism determines a service among all the services for a specific application with a limited set of requirements with each runtime. The architecture of the proposed method is that a user enters his/her requirements and the system finds a service provider using the proposed method and allocates the users request to the service. So user does not interact with the service selection process, and all is done by the system. This method is useful for those kinds of users which are unaware of service types and their attributes.

Choudhury et al. [26] proposed a method named Service Ranking Systems (SRS) for Cloud Vendors. This method performs ranking in two ways: dynamic, and static. In static mode, SRS ranked all available services in cloud market by not noticing to users requirements. But in dynamic mode, it ranked services according to users requirements.

Zheng et al. [27] proposed a framework named QoS Ranking which ranks services by predicting quality of services values and finding similar users to recall exact value of services and avoid time wastage.

Qu et al. [14] proposed a method which used two types of information for service ranking. The information is gained by earlier users feedback from services and evaluating quality of services values or monitoring services. This info is processed using fuzzy methods and various evaluation tools and then ranks services.

Rajkumar et al. [6] proposed a framework named SMICloud (Service Measurement Index) [30] to rank services based on Analytic Hierarchical Process (AHP) [28] which ranks services in 4 phases: receiving users requirements hierarchically, receiving weight of attributes, finding relations between attributes weight and calculating decision number, and ranking services. SMICloud ranked services based on SMI (Service Measurement Index) attributes [30] that presented by Cloud Service Measurement Index Consortium (CSMIC) and was especially designed for evaluate and compare cloud computing services.

Jahani and Mohammad Khanli [29] (our earlier work) proposed a ranking system (NSGA_SR) that can select some candidate services and rank them based on multi-objective optimization problem. This method is so useful and need less time than other methods in literature. However ARank improve NSGA_SR; Similar to NSGA_SR, ARank selects some candidate services and ranks them based on users requirements, however ARank perform these works in parallel by agents abilities that be more efficient than NSGA_SR. NSGA_SR rank services based on all essential and non-essential requirements. But ARank uses only the requirements that presented by users.

It can be seen that, except the approach offered by Rajkumar [6] and Jahani [29], other proposed methods do ranking based on limited attributes and they do not consider SMI attributes, in their ranking approach. In this way, user often searches for a service with special attributes which the ranking process do not perform ranking based on those attributes. To have more information on SMI, SMI attributes will be explained in the next section.

3. Quality of service attributes . Each cloud service is identified by its quality attributes. Until 2012, there were not any standard way to compare quality of attributes. In 2012, universal consortium CSMIC announced a standard, named SMI (Service Measurement Index) attributes, for representing the quality of service attributes [30]- [32]. After that, these attributes become the basis for evaluation and comparison of cloud services.

SMI attributes are designed based on ISO standards and they are a standard method to compare cloud services. Generally, there are seven main attributes defined in SMI. Accountability, agility, financial, performance, assurance, security and privacy and usability, also these attributes include some sub-attributes. Each of these attributes is given in the following [30]- [32]:

1. **Accountability:** If a system or a service does not have accountability, no one or organization will attempt to use it, and they will not use these systems for assigning critical data. Because, it is possible that they cannot access their critical data when they need. The sub-attributes of this attribute are: audit, fill, data ownership and tolerability.
2. **Agility:** This attribute allows users to change or expand their service charge free. Compatibility, capacity, elasticity and extensibility are some of its sub-attributes.
3. **Financial:** It is important for users to know that whether the charge that they pay for a service is cost efficient or not. Pay as use is a sub-attribute of this one too.
4. **Performance:** Performance means the least time or energy use for the largest task done. This attribute measures the usability of services to the request of users. Having performance attribute helps to find the largest load that a system can resist against it. Accuracy and competency are the sub-attributes of this one.
5. **Assurance:** Assurance shows the success or failure probability of a system or cluster completing its tasks within a limited time without any system failure.
6. **Security:** The security of a system is depended by the protection of the data and creating security for the transferred data to the system.
7. **Usability:** This attribute represents the fast compromise of cloud service with various requests and different environment. This attribute allows users to use the service and easily delegate their tasks to the system. Our proposed method, ARank, which will be explained in detail in the following, uses all seven attributes and their sub-attributes for cloud service ranking.

4. Our Proposed Architecture. In this section, we present the architecture of our new proposal for ranking cloud computing services. There are two main features for our method: First, it is based on multi-agent system (MAS), so it takes the advantage of distribution of MASs. Second, our proposed method prevents to rank all cloud computing services and the ranking is based on ranking of the candidate services. Thus, the waiting time of users for ranking process will be decreased.

Our proposed method called ARank (Agent Rank), which is based on multi-agent systems and it uses multiple agents which have cooperation and coordination with others. ARank avoids ranking all the services, by choosing some candidates services based on the requirements of user. Then, it ranks candidate services. This architecture contains three types of agents: user agent, manager agent and zone agent. These agents are autonomous in the environment. The agents can negotiate with others when it is necessary, so they can rapidly reach general goals of the system. The general goal of the system has been assigned to agents as a distributed solvable problem for cloud services ranking. The architecture of ARank, which represents agents of the system and the communications between them, is shown in Fig. 4.1.

User agent is an agent which is assigned to use ranking service by the user, as shown in Fig. 4.1. So it is possible that some user agents exist simultaneously and they concurrently do their tasks for their users. Manager agent acts as a central coordinator in the ranking system and coordinates agents to reach maximum performance of the system. Zone agent has the specifications of service providers. These agents are able to communicate with others, to perform parallel tasks and also to get the best result in the least possible time. Each agent is described in detail as follows:

User agent: This agent is created when user requests for service ranking. So the number of this agent is dynamic in the system, and it depends on the number of active users in the ranking system. This agent sends

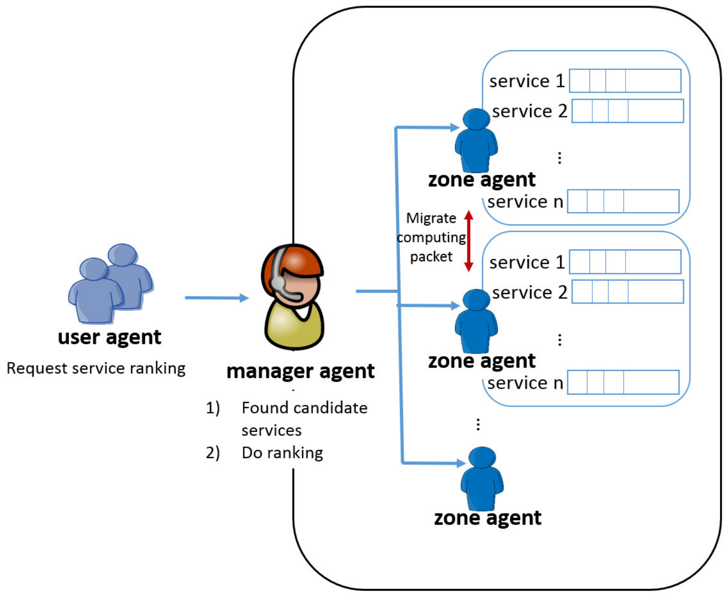

Fig. 4.1. *Multi-agent architecture of proposed ARank method*

user request to manager agent in the form of a message. User agent also has another important task: it asks user to enter his/her degree of satisfaction from used service(s) as a score, after it gives the user the ranking results. Then, user agent returns the scores to the zone agent related to each service. Because, the next levels of ranking is the consideration of the user satisfaction feedbacks and involves them in the ranking. This method makes a competition between service providers. Because, the scores are saved by zone agents (away from the accessibility of service provider), which denotes the history of service provider and record of those services. It is important that each request of the users includes some quality requirements of SMI attributes.

Manager Agent: This agent is responsible for the general coordination of the system. Manager agent receives the requests from user agents, and then for each request, it sends a message (containing information about the request) to every zone agent. In fact, it requests the zone agent to find some candidate for the received message. In response to this message, zone agent sends the name and sum of the scores (collected from earlier users) to manager agent.

Candidate services are chosen based on the quality of services that user requests. Therefore, all candidate services chosen are good for user. To determine how good candidate services are, it is important to rank them. Ranking the candidate services is a time-consuming task. Therefore, the manager agent creates a package, named *'Computing Package'*, which includes the data received from the zone agent, including the name of candidate services and their scores. Ranking must be done based on the content of these packages.

In our proposed method, the manger agent selects one of the zone agents to compute the package and takes the advantage of its computing capability. With this method, multiple *computing packages* are given to the zone agents simultaneously. If multiple users use the ranking system concurrently, the system can compute the *computing package* of each user simultaneously. One of the benefits of this method is the ability of coordination between agents, which causes fast result in ranking the systems that provides better and more accurate services to the users. In order to compute the *computing packages* for service ranking, we used the concept of domination, which will be explained completely in Section 5.3.

Zone agent: Each zone agent contains some services from multiple providers. It is necessary for service providers to register their services in zone agents to benefit the ranking system and introduce themselves to the users. Each zone agent can determine the kind of services that users need. For example, some zone agents only register PaaS type of services, some of them register only for SaaS or only for IaaS type of clouds. So, it is obvious that if the ranking system can do its job perfect, more providers will register in the system. Therefore, the ranking system will be powerful enough to offer best choices for the requests of users.

The other task of zone agents is the duty of collecting the request of users, and generating candidate services for each request. These agents send the value of attributes and scores of the chosen candidate services to the manager agents. Their next task is receiving *computing packages* and sending them to manager agents. Every zone agent can decide how to compute a *computing package* after receiving a compute message. If it is possible for the zone agent to do the computing and sending the result of a *computing package* by itself, so does it. However, if the zone agent cannot compute the package in the required time, the zone agent negotiates with other zone agents and allows other zone agent calculate the package. The computing result is sent to the manager agent. This agent also stores and uses scores given by user from the user agent. In the architecture, each agent has some exploratory functions for completing task, negotiation and coordination with other agents. Each of these functions are detailed in the next section.

5. Ranking with ARank.

The architecture of our proposed method (ARank) is shown in Fig. 4.1. This architecture includes the agents and all communications between them. The main purpose of this architecture is to find some candidate service and ranking them based on users requests in the shortest time. Also the system must have the ability to respond to more users in every time slot simultaneously.

5.1. Modeling the Input Variables of ARank.

The inputs for ARank method are the requirements of users on the basis of SMI, and also the priority of those attributes. According to Section 3, in SMI, there are 7 main attributes and some sub-attributes for recognizing each service. These attributes are classified into two types. Within the first type, values are presented with one number, so the value of the attributes must be the same with the users request.

However, the second type of values is presented with two numbers which are the upper bound and lower bound values of a range. Our ranking system has offered a method which is adaptable with both kinds. The system converts each single input into a two number range (with the starting and the ending of the range). For example, if the assurance attribute value is 14, given by user, the system converts it into range [14,14]. The set of requirements by the user is shown in Eq.(5.1).

$$R_u =< r_1, r_2, ..., r_i, ..., r_Q > \qquad u \in [1, m] \tag{5.1}$$

In Eq. (5.1), R_u is an ordered pair of requirements of user which is shown with r_i, and each r_i is the requirement for attribute i. Q is the number of all attributes. Variable u is the id number of system users, while the maximum of users would be m.

Other input from the user is the priority of each attribute. Because from the viewpoint of the user, some attributes are too important and they must be guaranteed at any cost. in cotrast, some attributes are not so important. The user input values for each attribute ranging from 0 to 1. Attributes with priority 1 must be guaranteed at highest quality by proposed services. The set of user priority is shown in Eq. (5.2).

$$C_u =< c_1, c_2, ..., c_i, ..., c_Q > \qquad u \in [1, m] \tag{5.2}$$

As shown in Eq. (5.2), C_u is an ordered pair of priorities of user u which is shown by c_i, each c_i is the value related to attribute i. Q shows the total quality attributes.

According to the presented architecture, service providers must register in zone agents and send them their services information. The services information is shown by SMI attributes quality which is represented in Eq. (5.3).

$$S_j =< q_1, q_2, ..., q_i, ..., q_Q > \qquad j \in [1, n] \tag{5.3}$$

As shown in Eq. (5.3), S_j is the mark for service j which is represented by an ordered pair of quality values q_i. If the total number of quality attributes is Q, variable j describes the id of the services of the corresponding zone agent. The number of services in each zone agent is equal to n. By the way, it is noticeable that the qualities of services values are measured using network observers or by the ranking system itself, so that service providers cannot violate registration phase.

5.2. Supplement of Candidate Services in ARank. A message is sent to a manager agent when a user agent gathered users required values. Following that, the manager agent sends the users required values to all zone agents. Each zone agent starts searching for the services which satisfy users requirements in its set of services after it receives the message from the manager agent. The search process is shown in Algorithm 1.

Algorithm 1 finding candidate services (by zone agents)

```
Require: $R_u, S_j$
Ensure: $candidateservices, feedback$;
1: for all $S_i$ do
2:     if $S_i$ is in range of $R_u$ then
3:         add $S_i$ to candidate services
4:     end if
5: end for
6: return candidate services to manager agent
7: return feedback for candidate services to manager agent
```

According to Algorithm 1, each service is analyzed by zone agent to find the matching services to users requirements (lines 1 and 2). If a service matched the requirements, then it is added to the candidate services set (line 3). At the end, zone agent returns the set of candidate services and scores of each service given by earlier users as an output (line 6 and 7).

5.3. Cooperation Protocol in ARank. When candidate services are found and sent to manager agent, in the next step, manager agent packs the list of candidate services and their scores (given from different zone agents) into a message called *computing package*. The ranking result will be revealed when the contents of *computing package* is computed.

Each manager agent is free to choose a zone agent as a destination of *computing package* and it is possible to use different methods to send the message. It might send the message in circular way, or it decides based on the load on each zone agent. But to be aware of other agents, it should always send a message to be informed of agents status. Sending this message will increase the load on connections, and results in execs of messages in inter-agent communications. So it is better to deal the problem like the load balancing problem in the cloud computing environment and use methods presented for them in [21]- [22].

Load balancing is used in service providers unit. Service providers use servers which might be in different geographical places. Each service provider tries to allocate each service to a low loaded server to increase quality of service. This process improves the quality of service and called load balancing [21].

When load balancing is being processed in cloud computing environments, a dynamic method is used to choose the best server for current requests of system [21]. The first allocation is done with random selection method or in circular turns. After that to prevent the incrementing the load of servers, it is allowed to servers that if they cannot handle the task, they search for a server with a lower load to autonomously assign the task to them. In this case, the load on the central service provider is decreased, and this agent does not get busy with complicated initial task allocation process [22].

Similarly, in order to decline the working load, manager agent get the task of computing the *computing package* to each zone agent randomly, or in circular turn way. Following that, each zone agent autonomously select to compute the *computing package* or assign it to another zone agent through negotiation. The protocol of sending packages and also sending the user requirements to find candidate services is shown in Fig. 5.1.

As shown in Fig. 5.1, manager agent sends a message to zone agents with each request from user. Zone agents choose some candidate services and send their name and scores to the manager agent. Followed by that, the manager agent creates a package including quality attributes of services and their scores and sends it to a zone agent using random algorithm or circular turn method. Next, the ranking result is given to the manager agent, and this agent guides the user to the selected service. Ranking steps (computing the *computing package*) and inter-agent negotiation protocol is described in the following.

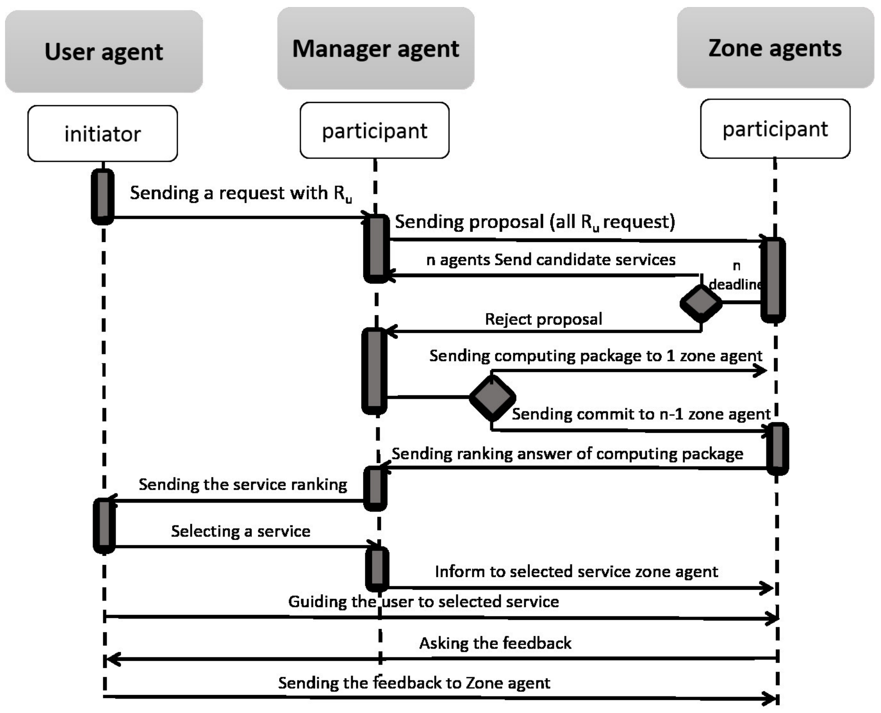
User agent
Manager agent
Zone agents
initiator
participant
participant
Sending a request with R_u
Sending proposal (all R_u request)
n agents Send candidate services
n
deadline
Reject proposal
Sending computing package to 1 zone agent
Sending commit to n-1 zone agent
Sending ranking answer of computing package
Sending the service ranking
Selecting a service
Inform to selected service zone agent
Guiding the user to selected service
Asking the feedback
Sending the feedback to Zone agent

Fig. 5.1. *Cooperation protocol between agents in ARank*

5.4. Ranking Candidate Services (*Computing Package*). The *computing package* is sent by manager agent. The content of this *computing package* is quality attributes and scores given by earlier users to the service. Computing of rankings is assigned to one of the zone agents. If only quality attribute of the services is received, ranking of services is done easily with sorting the services in anticlimactic order. But two types of information (the quality of services attributes, and scores given by earlier users) exist. Use of these two types of information is one of the main advantage of the proposed method. Ranking of the services must be done based on these two goals. Eq. (5.4) and Eq. (5.5) present computing functions.

$$F_1 = \sum_{i=1}^{7} c_i \times q_i \tag{5.4}$$

$$F_2 = \sum_{i=1}^{7} c_i \times {q_i}^{feedback} \tag{5.5}$$

As shown in Eq. (5.4), function $F1$ shows the sum of quality attributes of each candidate service, variable q_i shows the ith quality attribute of each candidate service. According to Eq. (5.5), function $F2$ presents the sum of users feedback or users scores for each quality attribute, variable presents the score given to each quality attribute for attribute i of each candidate service. The input of *computing package* is a number of candidate services with two goal functions. Input is shown in Fig. 5.2.

According to Fig. 5.2, the input of ranking algorithm is some candidate services found by zone agents. Each candidate services are presented with two values, sum of quality of attributes values ($F1$), and sum of scores given by previous users ($F2$).

Now, in order to compare services based on two separate goal functions, domination method is used. In the domination method, services are compared pairwise. Service 1 dominates and wins over service 2, if values of goal services are no worst (bigger, smaller, or any other comparing standard) then service 2, and is better from

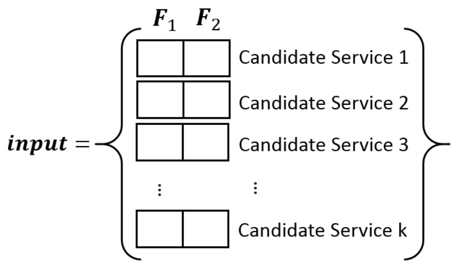

Fig. 5.2. *Input of computing package*

service 2 in one aspect at least. Number of goal functions is not important in domination method; this method is accountable to number of goal functions. This paper has two goal functions $F1$ and $F2$. Therefore, these two methods are compared in both services, and services dominate others will rank as first. The domination method of two is shown in Eq. (5.6) and Eq. (5.7).

$$S_1 \gg S_2 \Rightarrow \forall j \in [1,2] \qquad F_j^{S_1} \geq F_j^{S_2} \tag{5.6}$$

$$\exists j \in [1,2] \qquad F_j^{S_1} > F_j^{S_2} \tag{5.7}$$

In Eq. (5.6) and Eq. (5.7), $\gg$ means S_1 dominates S_2. and respectively show the value of jth function of service 1 and value of jth function of service 2. So, for ranking candidate services using domination of services (computing contents of *computing package*), algorithm 2 is presented.

Algorithm 2 Algorithm 2: Ranking candidate services (by zone agents)

```
Require: $F_1, F_2$
Ensure: Ranked services;
   R=1;
2: while there is not-Ranked candidate services do
      find all services that are dominated to other services and set there Rank to R
4:    if there is not any service which dominated to others then
         set the Rank of all services to R
6:    end if
      R=R+1;
8: end while
   return the Rank of candidate services
```

As shown in Algorithm 2, the goal functions $F1$ and $F2$ are used as inputs to compute *computing package*. The algorithm runs until there is no not-ranked services (line 2). Within each run, services which dominate other services, but are similar to each other, are given a rank (line 3). If the algorithm could not find any services that dominated to others, put all remained services in same rank (line 4). With each run, a unit is added to rank given to services for the next session (line 7). As a result, rank of services is returned (line 9).

5.5. Inter-agent negotiations protocol. In the previous section, it is said that ranking is done based on running some operations on *computing package*. And also each zone agent can choose between two options when it receives a package. In the first option, it can take care of operations, start ranking services and return the result to manager agent. Second option is that this agent for any reason (over load or busy ranking candidate services) cannot perform ranking, therefore, convinces one of other zone agents by negotiating to rank services and finally sends the *computing package* to that agent. Negotiation between each initial zone agent and others is shown in Fig. 5.3.

According to Fig. 5.3, the zone agent is not capable of computing a *computing package*, and starts negotiating with other zone agents by sending a message asking for computing the *computing package*. The

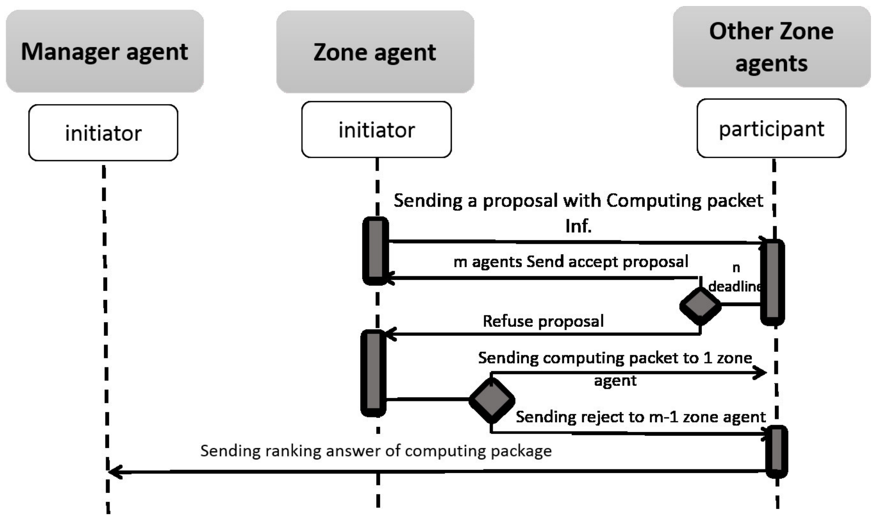
Manager agent
Zone agent
Other Zone agents
initiator
initiator
participant
Sending a proposal with Computing packet Inf.
m agents Send accept proposal
n deadline
Refuse proposal
Sending computing packet to 1 zone agent
Sending reject to m-1 zone agent
Sending ranking answer of computing package

Fig. 5.3. *Inter-agent negotiations protocol*

S_1= < 10 4 1 20 1 2 7 >	Feedback S_1= < 0,5 0.5 1 1 0.2 0,3 1 >
S_2= < 3 6 1 8 6 7 9 >	Feedback S_2= < 1 0.2 0.8 0.7 1 1 0,5 >
S_3= < 1 4 2 0 2 3 17 >	Feedback S_3= < 0.5 0.6 1 1 1 0.4 0,8 >
S_4= < 4 10 1 4 5 16 7 >	Feedback S_4= < 0.1 1 0.2 0.3 0.5 1 0.7 >
S_5= < 2 5 1 17 32 50 30 >	Feedback S_5= < 0.5 0.6 1 0 1 0.4 1 >

Fig. 5.4. *An example with five services in service marketing*

agents which are received the message respond by a bid message, if they can compute that message. Initial zone agents choose one of proposes randomly, then sends it the *computing package* or sends bid rejection to other agents. Any agent, which computes contents of the package, sends the result to manager agent, so that manager agent sends it to user agent.

5.6. A Sample Scenario. This section investigates an example and shows the ranking process. Supouse we have five services and only one active user in our ranking system. Each service has at least seven quality attributes which can be as Fig. 5.4. The previous users feedback represented in Fig. 5.4 too.

As shown in Fig. 5.4, we have five services that shown with their quality attributes. It should be noted that quality attributes for services measured by third section and we suppose these values as input our ARank method. Also Fig. 5.4 shows the feedback of services that intered by previously users. Now suppose we have one active user with inputs as follows:

$R_1 = \prec 3 \quad 4 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \succ$

$C_1 = \prec 0.5 \quad 0.1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \succ$

The R_1 shows user's requirments and C_1 shows user's priorities. In ranking process, at first zone agents try to found the services which can satisfy user's requirements (candidated services). In this example, services S_1, S_2 and S_4 can satisfy user's requirments. then the *computing package* includes only candidated services and their feedback and also the active user's inputs (requirments and priorities).

At the second step, one of the zone agents should calculate the *computing package* using Eq. 5.4 and 5.5 and make the results like Fig. 5.1. The result of computation will be as follow:

$S_1 \Rightarrow F_1 = 6.4$ and F_2=0.7

$S_2 \Rightarrow F_1$=3.1 and F_2=0.15

$S_4 \Rightarrow F_1$=4 and F_2=0.35

In the end, Algorithm 2 should be run to Rank the Candidated services. In this Example, service S_1

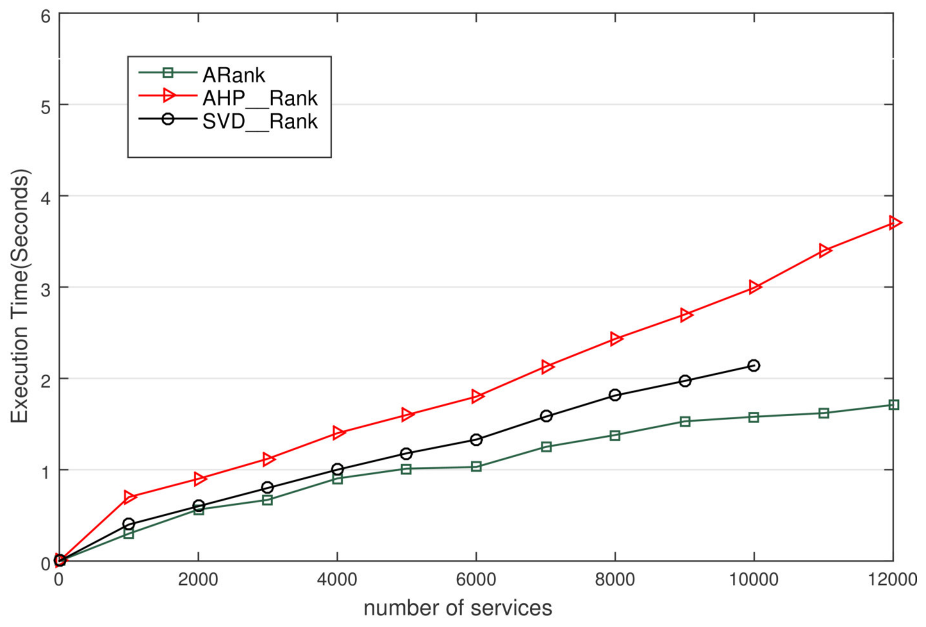

Fig. 6.1. *Response time of system by increasing number of services*

dominated to other two services and receives first rank. Service S_4 received second rank and service S_2 received third rank. This example represented ranking process in ARank.

5.7. Implementation of ARank. The proposed method is implemented on a system with CPU Intel Corei5 Duo 2.53 GHz, 4 GB of RAM, and Windows 7 x86 Enterprise OS using Jade Library (V. 4.3.3) along with earlier methods. In implementing ARank method, we created one manager agent, one user agent, and five zone agents. Values of user required attributes and also values of services attributes are 7 (SMI attributes only). Selection of 7 attributes in implementation is for comparing of proposed method with earlier ones, and also for the existence of 7 features in used dataset.

To analyze the propose method, ARank, version 2 of QWS [23] data set is used. Because QWS is a real dataset, from more than 2,500 web services. This dataset is the only one which includes all 7 SMI attributes. In addition, using these attributes help to rank and compare all proposed method with others.

6. Evaluation Analyses. This section analyzes the required time for getting suggestion from ranking service. Our proposed method is compared with methods based on SVD (Singular Value Decomposition) [25] and based on AHP (Analytic Hierarchical Problem) [28].

The experiment is based on response time of system by increasing number of services. Services are increased from 1 to 12,000 with 1,000 jump. With each increment in number of services, response time from system is measured. Experiment results are shown in Fig. 6.1. The methods based on SVD and based on AHP are respectively shown as SVD_Rank and AHP_Rank.

As shown in Fig. 6.1, by increasing the number of services, SVD_Rank method is capable of responding up to 10,000 services, because it uses Singular Value Decomposition method. As SVD method uses serial and central run, response time increases when the number of services increase. The AHP_Rank also ranks all the existing services; and this method also runs central and serialized. Therefore, it needs more time to respond. In return, the ARank method runs distributed as it uses agents and their cooperation ability has lower response time compared to previous methods.

7. Conclusion and Future Works. In this paper, we proposed a novel method for ranking cloud computing services based on mutiagent systems called ARank. The aim of our system is to decrease user waiting times to get a suggestion from ranking service. The ARank method prevented ranking all of services by choosing candidate services, And it took the advantage of capabilities of agents, by choosing and ranking services in distributed way which resulted in lower ranking time.

Our proposed method, ARank, prevented ranking all services by choosing some candidate services with zone agents simultaneously. And also this method could use all user requirements using the standard SMI attributes. In addition, ARank is able to use the feedback of users along values for each attribute quality, which inspires a competition between service providers.

For our future works, the requirements of users can be classified into essential and non-essential that will be increased ranking accuracy. In addition, we can create some new services with service composition that are more suitable for users requirements. Furthermore, it is possible to optimize the functions that are finding best matching zone agents to calculate the *computing package*.

REFERENCES

[1] R. Buyya, C. Vecchiola, S. T. Selvi, *Mastering cloud computing, cloud computing architecture*, Foundations and Applications Programming, chapter 4, 2013, pp. 111-140.

[2] S. Ding, S. Yang, Y. Zhang, C. Liang, C. Xia, *Combining QoS prediction and customer satisfaction estimation to solve cloud service trustworthiness evaluation problems*, Knowledge–Based Systems, 56 (2014), pp. 216-225.

[3] S. Garg, S. Versteeg, R. Buyya, *a Framework for ranking of cloud computing services*, Future Generation Computer Systems, 29 (2013), pp. 1012-1023.

[4] A. Quarati, A. Clematis, D. Agostino, *Delivering cloud services with QoS requirements: Business opportunities, architectural solutions and energy-saving aspects*, Future Generation Computer Systems, 55 (2016), pp. 403-427.

[5] B. Rajkumar, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, *Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th Utility*, Future Generation Computer Systems, 25 (2009), pp. 599-616.

[6] Y. Hao, Y. Zhang, J. Cao, *Web services discovery and rank: An information retrieval approach*, Future Generation Computer Systems, 26 (2010), pp. 1053-1062.

[7] L. Chen, Y. Feng, W. Jian, Z. Zheng, *An enhanced QoS prediction approach for service selection*, IEEE International Conference on Services Computing, Washington, DC, 2011, pp. 727-728.

[8] R. Zhang, K. Zettsu, Y. Kidawara, Y. Kiyoki, *Web service ranking based on context*, Second International Conference on Cloud and Green Computing, Xiangtan, 2012, pp. 375-382.

[9] F. Durao, J. Carvalho, A. Fonseka, V. Garcia, *A systematic review on cloud computing*, The Journal of Supercomputing, 68 (2014), pp. 1321-1346.

[10] M. J. Katchabaw, H. L. Lutfiyya, M. A. Bauer, *Usage based service differentiation for end-to-end quality of service management*, Computer Communications, 28 (2005), pp. 2146-2159.

[11] H. Fan, *An integrated personalization framework for SaaS-based cloud services*, Future Generation Computer Systems, 53 (2015), pp. 157-173.

[12] L. Qu, Y. Wang, M. A. Orgun, *Cloud service selection based on the aggregation of user feedback and quantitative performance assessment*, SCC '13 Proceedings of the IEEE International Conference on Services Computing, IEEE Computer Society Washington, DC, USA, 2013, pp. 152-159 .

[13] C. Mao, *Search-based QoS ranking prediction for web services in cloud environments*, Future Generation Computer Systems, 50 (2015), pp. 111-126.

[14] M. Wooldridge, *An introduction to multi-agent systems*, Department of Computer Science, University of Liverpool, 2009, pp. 200-450.

[15] L. Qu, Y. Wang, M. A. Orgun, *Cloud service selection based on the aggregation of user feedback and quantitative performance assessment*, SCC '13 Proceedings of the IEEE International Conference on Services Computing, IEEE Computer Society Washington, DC, USA, 2013, pp. 152-159.

[16] S. Yau, Y. Yin, *QoS-based service ranking and selection for service-based systems*, IEEE International Conference on Services Computing, 2011, pp. 56-63.

[17] M. Almulla, H. Yahyaoui, K. Al-Matori, *A new fuzzy hybrid technique for ranking real world Web services*, Knowledge-Based Systems, 77 (2015), pp. 1-15.

[18] D. Skoutas, D. Sacharidis, A. Simitsis, T. Sellis, *Ranking and clustering web services using multicriteria dominance relationships*, IEEE Transaction on Service Computing, 3 (2010), pp. 163-177.

[19] M. D. Dikaiakos, D. Zeinalipour Yazti, *A distributed middleware infrastructure for personalized services*, Computer Communications, 27 (2004), pp. 1464-1480.

[20] S. S. Yau, Y. Yin, *QoS–based service ranking and selection for service based systems*, IEEE International Conference on Services Computing, Washington, DC, 2011, pp. 56-63.

[21] J. Octavio, A. Ramirez, *Collaborative agents for distributed load management in cloud data centers using Live Migration of virtual machines*, IEEE Transactions on Service Computing, 8 (2015), pp. 916-929.

[22] J. Octavio, A. Ramirez, *Agent-based load balancing in cloud data centers*, Cluster Computing, 18 (2015), pp. 1041-1062.

[23] E. Al-Masri, Q. H. Mahmoud, *Investigating web services on the world wide web*, In Proceedings of the 17th international conference on World Wide Web, ACM, 2008, pp. 795-804.

[24] E. Badidi, *A Framework for software as a service selection and provisioning*, International Journal of Computer Networks & Communications, 5 (2013), pp. 189-200.

[25] H. Chan H, T. Chieu, *Ranking and mapping of applications to cloud computing services by SVD*, Network Operations and Management Symposium Workshops (NOMS Wksps), Osaka, 19-20 April, 2010, pp. 362-369.
[26] P. Choudhury, M. Sharma, K. Vikas, T. Pranshu, V. Satyanarayana, *Service ranking systems for cloud vendors*, Advanced Materials Research, 433 (2012), pp. 3949-3953.
[27] Z. Zheng, X. Wu, Y. Zhang, M. Lyu, J. Wang, *QoS ranking prediction for cloud services*, Journal of IEEE Transactions on Parallel and Distributed Systems, 24 (2012), pp. 1213-1222.
[28] A. Ishizaka, A. Labib, *Analytic hierarchy process and expert choice: benefits and limitations*, OR Insight, 22 (2009), pp. 201-220.
[29] A. Jahani, L. Mohammad Khanli, *Cloud service ranking as a multi objective optimization problem*, Journal of Supercomputing, 72 (2016), pp. 1897-1926.
[30] CSMIC, *CSMIC SMI overview diagram two point one*, Carnegie Mellon University Silicon Valley, Moffett Field, CA USA, 2011, pp. 1-10.
[31] J. H. Chen, *A hybrid model for cloud providers and consumers to agree on QoS of cloud services*, Future Generation Computer Systems, 50 (2015), pp. 38-48.
[32] L. Sun, *Cloud-FuSeR: Fuzzy ontology and MCDM based cloud service selection*, Future Generation Computer Systems, 57 (2016), pp. 42-55.

Edited by: Pedro Valero Lara
Received: Dec 23, 2016
Accepted: May 25, 2017

DOI 10.12694/scpe.v18i2.1284
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 117–131. http://www.scpe.org © 2017 SCPE

AUTOMATIC TUNING ON MANY-CORE PLATFORM FOR ENERGY EFFICIENCY VIA SUPPORT VECTOR MACHINE ENHANCED DIFFERENTIAL EVOLUTION

ZHILIU YANG, ZACHARY I. RAUEN, AND CHEN LIU*

Abstract. The modern era of computing involves increasing the core count of the processor, which in turn increases the energy usage of the processor. How to identify the most energy-efficient way of running a multiple-program workload on a many-core processor while still maintaining a satisfactory performance level is always a challenge. Automatic tuning on the voltage and frequency level of a many-core processor is an effective method to aid solving this dilemma. The metrics we focus on optimizing are energy usage and energy-delay product (EDP). To this end, we propose SVM-JADE, a machine learning enhanced version of an adaptive differential evolution algorithm (JADE). We monitor the energy and EDP values of different voltage and frequency combinations of the cores, or power islands, as the algorithm evolves through generations. By adding a well-tuned support vector machine (SVM) to JADE, creating SVM-JADE, we are able to achieve energy-aware computing on many-core platform when running multiple-program workloads. Our experimental results show that our algorithm can further improve the energy by 8.3% and further improve EDP by 7.7% than JADE. Besides, in both EDP-based and energy-based fitness SVM-JADE converges faster than JADE.

Key words: Machine learning, many-core processors, energy-aware computing

AMS subject classifications. 68M20, 68T05

1. Introduction.

In the modern computing era, the core count of the processor has continuously being on the rise in pursuit of higher computational throughput. One downside, however, is the increased power consumption and energy cost associated with core count increase. There have been several different methods to minimize the energy usage of the processor, one of the most popular options being dynamic voltage and frequency scaling (DVFS). With DVFS, the voltage and frequency level of the cores can be changed in an effort to reduce the overall energy usage. On the other hand, we can keep the cores running in a very low power state in order to save energy. But this will result in a very poor user experience as the performance in terms of computation speed is unsatisfactory. How to identify the most energy-efficient way of running the workload while still maintaining a satisfactory performance level is always a challenge. When running multiple programs on a many-core platform, one energy-aware approach is to use different voltage and frequency setting with different number of cores related to the inherent property of the program. Since it is not always feasible to search through all possible combinations of frequency, voltage, core count with a brute force method, automatic tuning becomes an effective and promising method to aid solving this dilemma.

Several algorithms exist to accomplish the task of automatic tuning such as Genetic Algorithm (GA) and Differential Evolution Algorithm (DE), which search for solution as the algorithm evolves through generations. In evolutionary computation, all of the individuals in a single iteration are called a generation. Similar to the evolution of living creatures, individuals in one generation can exchange their unique features, which provides the possibility for generating better offspring. Standard differential evolution (DE) can provide a good solution for auto-tuning [1], however, its convergence rate is still relatively slow. Convergence in a performance curve is the point the performance stops to change obviously. Fast convergence rate means the algorithm reaches or gets close to the optimal performance much faster. In the realm of auto-tuning for many-core platforms, there have been other algorithms that attempt to converge faster than standard DE, one of which being the adaptive differential evolution algorithm (JADE).

In [3], Jiang et al. did a comprehensive study between JADE and several other representative algorithms such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and the β algorithm. In [3], JADE is proven to converge faster than standard DE algorithm and still produce the best results among the comparable approaches, making it the new benchmark for this type of algorithm.

*Department of Electrical and Computer Engineering, Clarkson University, 8 Clarkson Ave, Potsdam, New York, 13699 (zhilyan@clarkson.edu, rauenzi@clarkson.edu, cliu@clarkson.edu)

Although it converges quicker than the normal DE algorithm, JADE still requires several generations of evolution that requires a large computation overhead. In order to decrease the overhead of JADE, we aim to decrease the number of generations required to achieve convergence. As shown in [2], Machine Learning Enhanced Differential Evolution (mDE) can result in a much quicker convergence. Since JADE is proven to be more effective than DE in energy-aware automatic tuning on many-core processors, and mDE converges quicker than JADE, we propose SVM-JADE, a new auto-tuning algorithm for fast DVFS auto-tuning, which can identify the optimal settings and number of cores per program to minimize energy and energy-delay product (EDP) while running multiple programs on many-core platforms.

The rest of the paper is organized as follows: A brief overview of background works can be found in Sect. 2, followed by the methodology for our specific version of differential evolution in Sect. 3. The implementation of scheme are introduced in Sect. 4 with the experimental results being discussed in Sect. 5. Our conclusions are presented in Sect. 6.

2. Background. In this section we introduce the hardware platform we use to conduct this research, as well as review some representative works on energy-aware auto-tuning.

2.1. Hardware Platform. The Single-chip Cloud Computer (SCC) [4] is an experimental processor created by Intel Labs for many-core based research. The architecture of SCC is shown in Fig. 2.1 [6]. The SCC has 48 Intel Pentium P54C cores and employs mesh communication across the entire core-set. These cores are divided into 6 power domains consisting of 8 cores each. Each of these power domains can be set to different voltage levels independent of one another. The individual cores are grouped into pairs within the power domain yielding 4 pairs per power domain. These core pairs form frequency domains that can have their frequency set individually, much like the voltage of the power domain. One example of these domains can be seen in Fig. 2.1. Cores 32 through 35, and cores 44 through 47 all belong to the same power domain as seen by the box encompassing them. The pairings of the cores seen within the power domain, such as cores 46 and 47, are all individual frequency domains.

Due to these abilities, the SCC becomes an ideal testing platform for energy-related many-core research. Also, due to the computing capability of the SCC, it is possible to deploy multiple programs to run on the SCC simultaneously.

With these characteristics, it is possible to not only run multiple programs on the SCC, but run the programs at different voltage and frequency levels. This allows for a dynamic and energy-aware system when running computations. However, there are many gears to choose from for each program, where a gear is defined as a specific voltage and frequency combination of the cores. Specifically on the SCC, the voltage of the core can range from 0.7V to 1.3V and the frequency can range from 100MHz to 800MHz [5]. An exemplary gear would be [1.1V, 800MHz] for a power domain of 8 cores. This leaves us an abundance of choices to choose from. As a result, an automatic tuning method needs to be developed.

2.2. Related Works. Auto-tuning with specific algorithms is a promising method to reduce the energy consumption of processors. Flautner et al. [7] applied the Differential Evolution algorithm to dynamic voltage scaling problem of processor first. After that, Hsu et al. [8] worked out an algorithm named β algorithm which focused on another power-aware run-time system. These two are early exploratory auto-tuning works. Some auto-tuning works based on SCC platform have been carried out as well. Berry et al. [9] adopted a manual approach to scale voltage and frequency levels on SCC. Later, Berry et al. [10] introduced the Differential Evolution algorithm to solve SCC power-aware computation problem, where a large program with several different phases was tested. Continuing the work on differential evolution, Roscoe et al. [1] expanded the workload on SCC to 3 programs. Recently, Jiang et al. [3] found a better energy configuration with Adaptive Differential Evolution Algorithm (JADE), with 4 programs being examined in their work. The results produced by above researchers are solely relied on the evolution computation algorithm techniques. With respect to the prediction ability of machine learning, we believe using machine learning techniques can enhance the searching ability of the previous evolutionary related algorithms. Yan et al. [2] put forward a method to enhance the DE algorithm by Adaptive LS-SVM method. Zhang et al. [11] performed a survey about machine learning enhanced evolutionary computation.

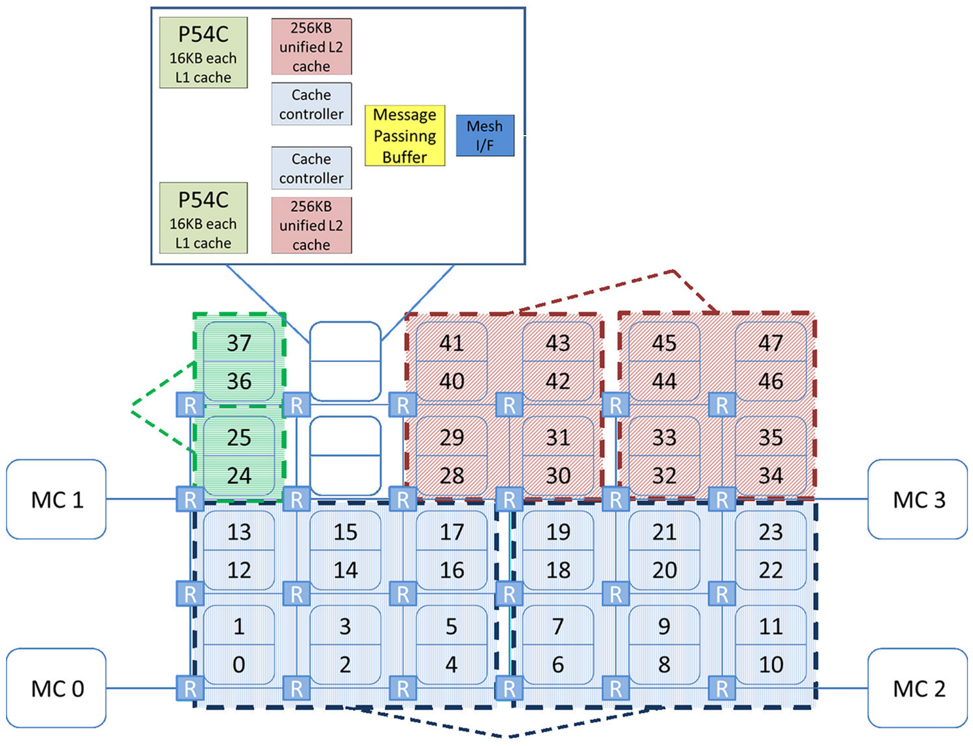

Fig. 2.1: Inner architecture of the SCC

3. Methodology. In order to achieve auto-tuning, the many-core platform needs to dynamically balance voltage, frequency, core numbers occupied by each program. These different variants construct a huge search space, which is difficult to deal with by manual or brute force methods. So the key aspect of finding the most energy-aware way of running the workload is transformed to find the method or algorithm which could converge in the least amount of searching time. In this work, a machine learning enhanced DE algorithm is adopted for this purpose. Classical DE algorithms and our new algorithm are discussed in this section.

3.1. Differential Evolution. Differential Evolution (DE) algorithm is a parallel direct search method. Fig. 3.1 [12] shows an example of the relationship between the searching space and searching vectors of DE.

This algorithm utilizes D-dimensional vectors, where NP vectors constitute the entire population in one generation. In other words, NP stands for the size of the population. The algorithm flow of DE is shown in Fig. 3.2.

The initial population vectors are:

$$x_{i,G}, i = 1, 2, ..., NP \tag{3.1}$$

Selecting one vector from current generation as target vector, randomly selecting other three vectors to perform the arithmetic operation as shown in Fig. 3.2 to get mutated vector which is represented as:

$$v_{i,G+1}, i = 1, 2, ..., NP \tag{3.2}$$

The arithmetic operation is defined by:

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}), i = 1, 2, ..., NP \tag{3.3}$$

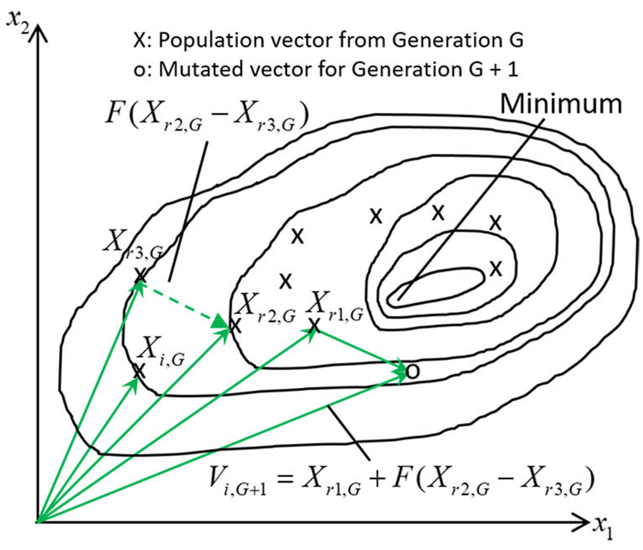
X: Population vector from Generation G
o: Mutated vector for Generation G + 1
$F(X_{r2,G} - X_{r3,G})$
Minimum
$X_{r3,G}$
$X_{r2,G}$
$X_{r1,G}$
$X_{i,G}$
$V_{i,G+1} = X_{r1,G} + F(X_{r2,G} - X_{r3,G})$
x_2
x_1

Fig. 3.1: Two-dimensional Function optimization showing contour line and its process for generating $v[i, G+1]$

Here, F is amplification value. Then randomly taking elements of target vector $x_{i,G}$ cross compare with $v_{i,G+1}$, determined by random-generated number and CR factor. This process simulated chromosomal crossover in creature's cell. After that, totally new vectors:

$$u_{i,G+1}, i = 1, 2, ..., NP \tag{3.4}$$

are generated, then these vectors get their fitness value by the evaluation process. In the context of genetic algorithms, fitness is any arbitrary metric that rates the quality of any given candidate. In this work, the fitness we use is energy and EDP. Finally, there is a selection between the parent and the offspring according to their corresponding fitness values. The above process should be applied to each individual in the generation at one time. In this way, the next generation:

$$x_{i,G+1}, i = 1, 2, ..., NP \tag{3.5}$$

is produced successfully.

3.2. JADE. Adaptive Differential Evolution Algorithm (JADE) [13] improved the performance of classical DE algorithm mentioned above. JADE is a greedy algorithm whose mutation strategy is mutating the top $p\%$ of current generation. So one difference between JADE and DE is that mutation operator is changed to:

$$v_{i,G+1} = x_{r_1,G} + F_i \cdot (x^p_{best,G} - x_{r_1,G}) + F_i \cdot (x_{r_2,G} - x_{r_3,G}), i = 1, 2, ..., NP \tag{3.6}$$

Amplification factor F_i is also changed. On the other hand, JADE changed its crossover factor into:

$$\mu_{CR} = (1 - c) \cdot \mu_{CR} + c \cdot mean_A(S_{CR}) \tag{3.7}$$

$$CR_i = randn_i(\mu_{CR}, 0.1) \tag{3.8}$$

where initial μ_{CR} is the mean of the normal distribution with 0.1 variance at first generation. c is a constant to balance the proportions of two mean values. S_{CR} is the set comprised by "successful" CR values of last generation. It is neglected at first generation. The mean here is the arithmetic mean. In this way, JADE introduces the experience from the previous successful generation. For example, classical DE algorithm only

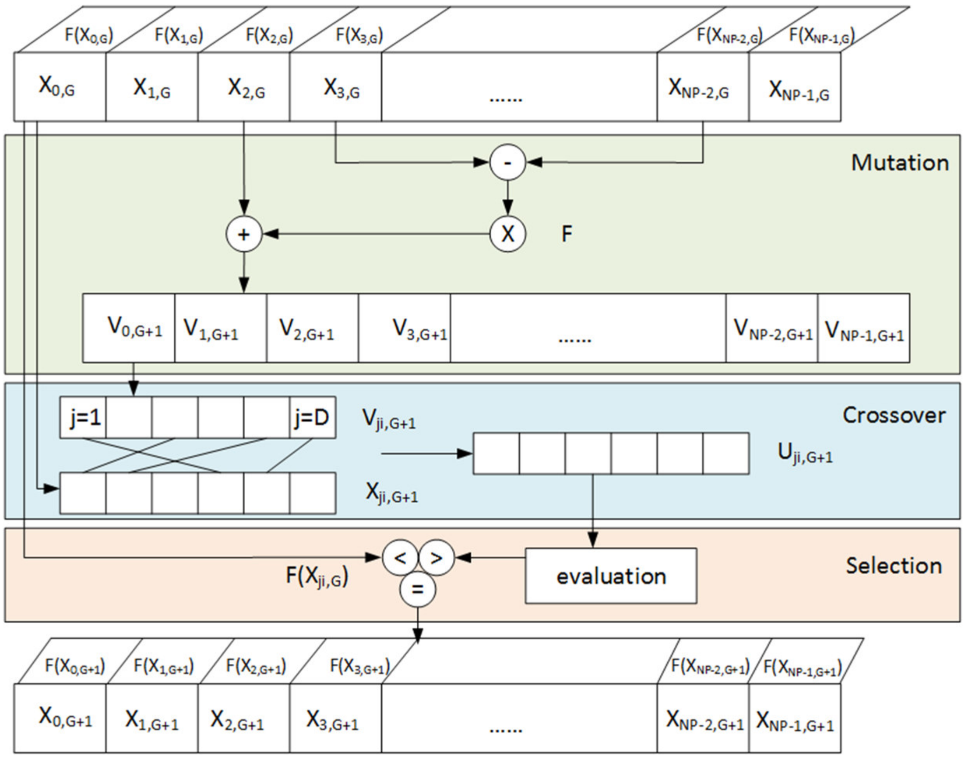

Fig. 3.2: Algorithm Flow of Differential Evolution

mutates one candidate, which is the best one in the generation including 40 candidates. JADE algorithm can mutate top p% candidates. If p is equal to 20, then 8 candidates will be mutated in the generation including 40 candidates. And if constant c is set to 0.1, the μ_{CR} in $(n+1)_{th}$ generation is made up by 90% μ_{CR} and 10% mean value of "successful" CR values in $(n)_{th}$ generation. And the general flow of JADE can be described in the following pseudo code:

Algorithm 1 Classical DE Algorithm

```
1: function DE(Initial Candidates, NP, F, CR)
2:     for (i to NP) do
3:         $P1, P2, ..., Pm \leftarrow Candidate\ Randomly\ Picking$
4:         $V \leftarrow P3 + F_i \cdot (P1 + P2) + F_i \cdot (P3 + P4)$          ▷ Mutation
5:         $CR_i \leftarrow Mean\ values$
6:         $U \leftarrow P5,\ M,\ CR_i$          ▷ Crossover
7:         $f(U) \leftarrow U$
8:         $Compare\ f(U)\ with\ f(P6)$          ▷ Selection
9:     return New Candidates
```

3.3. Support Vector Machines. Originally a lot of artificial intelligence algorithms were considered as case-by-case method. They performed well on a certain problem, but when the input condition or applied objects changed, the algorithm performed poorly. The creation of the Support Vector Machine (SVM) [14] algorithm largely improved this kind of drawback. Initially, SVM method was mainly used for classification problem. Later, it was extended to the case of regression. In our paper, we use the regression function of the SVM, which could also be called Support Vector Regression (SVR). The input of the SVM algorithm is training data sets, which could be presented as n-dimensional vector x:

$$x = \{x_i\}_{i=1}^{n} \tag{3.9}$$

and the classification function of the data set is:

$$f(x) = \sum_{i=1}^{n} w_i \langle (x_i), (x) \rangle + b \tag{3.10}$$

Training data set could be mapped into another high-dimensional space by mapping kennel. The SVM model in the high-dimensional space could be represented as follows:

$$f(x) = \sum_{i=1}^{n} w_i \langle \phi(x_i), \phi(x) \rangle + b \tag{3.11}$$

Here, with respect to simplification of description, we use a 2-dimensional training data set example to express the theory of mapping [17]. Assuming two 2-dimensional vectors $x_1 = (\eta_1, \eta_2)$ and $x_2 = (\xi_1, \xi_2)$, and $\phi(\cdot)$ is the mapping relationship from 2-dimension to 5-dimension. In this way:

$$\langle \phi(x_i), \phi(x) \rangle = \eta_1\xi_1 + \eta_1^2\xi_1^2 + \eta_2\xi_2 + \eta_2^2\xi_2^2 + \eta_1\eta_2\xi_1\xi_2 \tag{3.12}$$

In another point of view:

$$(\langle \phi(x_i), \phi(x) \rangle + 1)^2 = 2\eta_1\xi_1 + \eta_1^2\xi_1^2 + 2\eta_2\xi_2 + \eta_2^2\xi_2^2 + 2\eta_1\eta_2\xi_1\xi_2 + 1 \tag{3.13}$$

Equation 3.12 and Equation 3.13 share the same shape. It is not hard to get the expression of $\phi(\cdot)$, which is:

$$\phi(x_1) = (\sqrt{2}\eta_1, \eta_1^2, \sqrt{2}\eta_2, \eta_2^2, \sqrt{2}\eta_1\eta_2, 1) \tag{3.14}$$

Here, Equation 3.12 calculates in the high-dimension space, while Equation 3.13 calculates easily in the low-dimension space and avoids to express the mapping relationship explicitly. The method of Equation 3.13 is called kernel function method, which reduces the computation complexity by committing computation in the high-dimensional space of problem.

3.4. SVM-JADE. When multiple programs are running simultaneously on the SCC, energy consumption will differ by using different number of cores and gears of voltage and frequency. Among them, finding the configuration vector x_D which make SCC consume the lowest energy-delay product (EDP) or energy is meaningful. The vector x_D is made up of gears and number of cores. The G_i represent the gear level of i_{th} program, and the C_i represent the number of cores of i_{th} program. Considering 4 programs are running at the same time in our experiment, our x_D could be represented as:

$$x_D = (G_1, G_2, G_3, G_4, C_1, C_2, C_3, C_4)^T \tag{3.15}$$

with constraints:

1. The total number of cores used by all 4 programs cannot exceed 48 at any time, which is the total number of cores of the SCC platform.
2. One individual program must be run under one gear, even if it occupies more than one power domain.
3. Number of possible gears are regulated down to 5, one of which is an idle state to be used on a domain when no cores from it are in use. The different gears can be seen in Table 3.1. This limitation on gears means that each voltage level will be using the maximum possible frequency for that power level, which allows no power to be wasted.

Table 3.1: Different gear combinations in JADE

Gear	Frequency	Voltage
1	800 MHz	1.1 V
2	533 MHz	1.0 V
3	400 MHz	0.9 V
4	320 MHz	0.8 V
Idle	100 MHz	0.7 V

The third constraint reduces the search space for brute force searching to a certain degree, as it removes some unnecessary state. But even under this scenario, assuming a 4-program case where each program has one of 5 possible gears and is limited to a maximum of 8 cores, we still have a search space of size 2.56×10^6. With such a search space, brute force searching is still too labor intensive to be feasible.

Yan et al. [2] put forward a method to enhance the DE algorithm by Adaptive LS-SVM method. Synthesizing the advantages of above-mentioned algorithm, we adopt a method named SVM-JADE to accelerate the converge speed of finding optimum low-power combination in SCC auto-tuning system. The training set of this problem could be presented as: $\{x_i, f(x_i)\}_{i=1}^{n}$. The flowchart of SVM-JADE is given in Fig. 3.3.

The SVM-JADE embedded an SVM prediction procedure after the selection procedure of each generation. At the end of one generation, current population status is defined as:

$$s_{i,G+1}, i = 1, 2, ..., NP \tag{3.16}$$

while selecting the n best individuals among Equation 3.16 and their corresponding fitness values as input into SVM model to fit a regression hyper-plane for predicting Energy or EDP value of SCC platform. Next, using regression hyper-plane performing the vicinity search around the best performance vector in Equation 3.16. The vicinity method is shown in Fig. 3.4. Here, with respect to the difficulty in visualizing a D-dimensional space when $D > 3$, we use 3-dimensional vector as an example to explain the search procedure.

With vicinity search, we obtain the optimum vector. Optimum vector is predicted by the SVM regression model which compares across the fitness values of individual in current generation. Specially, the optimum vector should be feasible, i.e., could be executed on the SCC platform. Then, the configuration of optimum vector will be tested on the SCC to get the actual fitness value, being energy or EDP in our case. Finally, compare the fitness of optimum vector with lowest fitness of vectors in Equation 3.16, the candidate with better fitness value stays. Then we get the generation in Equation 3.5 of SVM-JADE. At the end of every generation, repeat these procedures. The evolution is usually stopped either after a certain number of generations or by reaching a specified threshold.

4. Implementation.

The general diagram of the implementation of SVM-JADE is shown on Fig. 4.1. Our initial approach in implementing our SVM was to test the SVM in MATLAB using the libSVM library [15] as a quick prototyping environment. After each generation, a population of 40 individuals is generated. We input them into SVM model with different size of data-sets, varying from D to NP. We find that the lowest mean squared error (MSE) between actual value and regression value of individual is obtained when the size of training set is D, which is 8 in this case. That is to say, the squared correlation coefficient is almost equal to 1.0 in this size of training data set.

Our implementation here used both the theories of JADE and SVM as discussed in Section 3. We started with a base implementation of JADE as laid out in [3]. This implementation used the Inspyred [16] python library to assist in the evolutionary computations. In accordance with the theory of JADE, a modified version of the mutation and crossover steps were used in place of the standard DE versions. Other things we followed were running four programs simultaneously and the same fitness tests. The two different fitness types used here are energy and energy-delay product (EDP). Aside from adding an SVM portion to JADE, there were a few other modifications made as well. SVM-JADE has an auto-initialization at the beginning to ensure that all tests

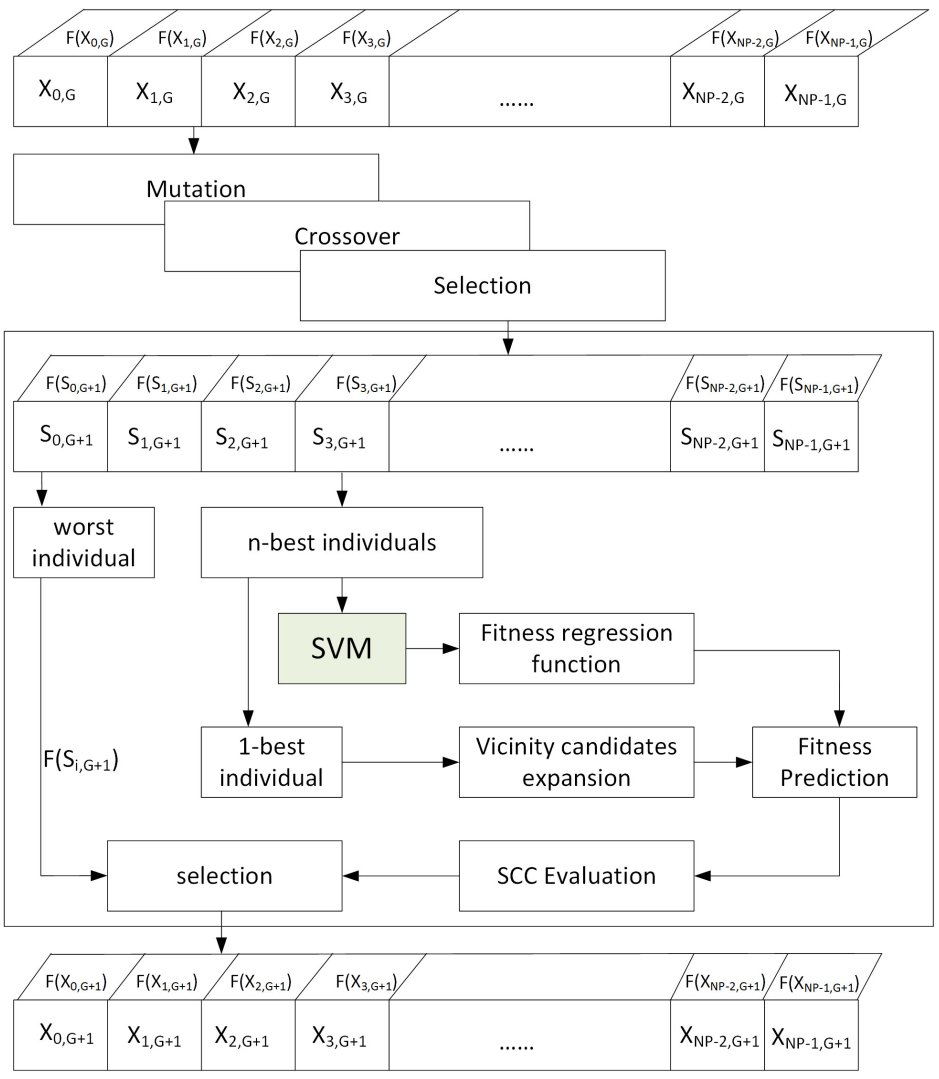

Fig. 3.3: Algorithm Flow of SVM-JADE

begin with the same initial conditions. Another change is that the initial population can be predetermined, which allows for more specific testing including continuing testing in the case of a failure during experiment.

The four programs used in our experiments are *mmul*, *conv*, *cpi*, and *seqpi*, respectively. *mmul* is a matrix multiplication program which can be partially paralleled without error. *conv* is a program similar to *mmul* by operating on matrices, however, *conv* finds the convolution of the two matrices which is much easier to parallelize. *cpi* is a highly parallel program that calculates the value of π, stopping after a given number of iterations. The iterations number used here is one billion, which is the same as used in JADE. Lastly, we have *seqpi* which is a very sequential program. This accomplishes the same task as *cpi* but in sequential fashion. We use these four programs to form our multi-program workload for the many-core processor, the same as described in [3].

In order to add the SVM module into JADE, we again used the libSVM library [15]. However, this time we used the python extensions to make combining the algorithms easier. This also made organizing our implementation easier by designing our SVM as an object that is able to perform all necessary computations on its own.

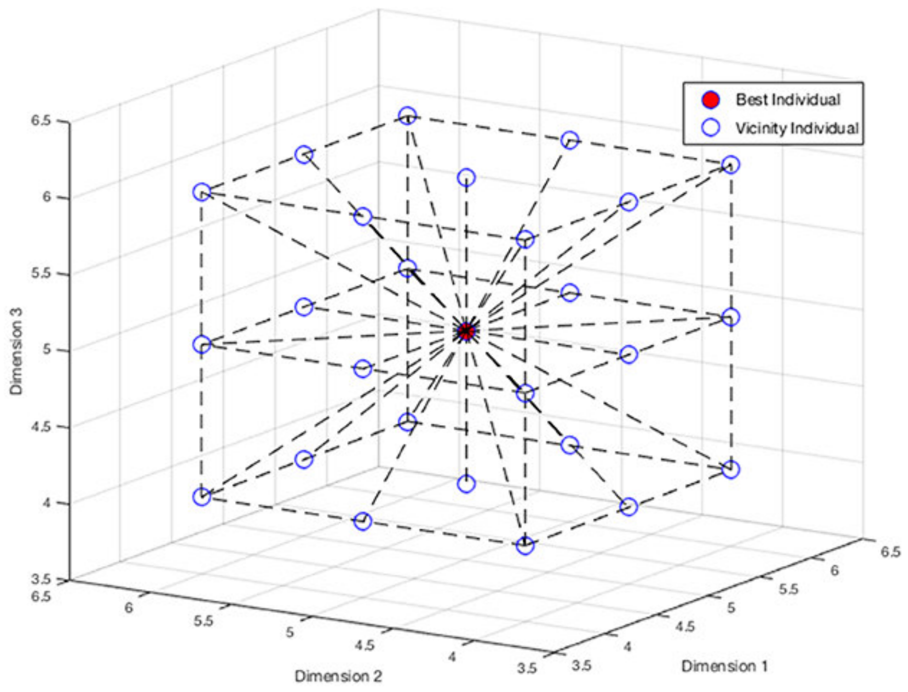

Fig. 3.4: Demo of vicinity Search

Our SVM was inserted into the selection phase, being trained on the candidates of the current population based on the current fitness. After being trained, the SVM then generates its own search space for both gears and core count. The search space generated follows the algorithm laid out below:

Algorithm 2 Search Space Generation for Gears

```
1: function GEARSPACE(candidate, dimension)
2:     $value \leftarrow candidate[dimension]$
3:     if $value = 1$ then
4:         return $[1, 2]$
5:     else if $value \in [2, 3]$ then
6:         return $[value - 1, value, value + 1]$
7:     else if $value = 4$ then
8:         return $[3, 4]$
```

As can be determined from the algorithm, for the search space of gears we end up with a search size of 2 or 3. Similarly, when generating the search space for the core count, we end up with possible sizes of 3 and 5. These are a bit larger because there is a larger set of values to choose from.

In summary, our algorithms form small search space around the best candidate of the offspring population and uses the SVM decision function to predict the different fitness values for each candidate in the search space. The smallest value is then chosen as the SVM predicted candidate, because we are trying to minimize the fitness value, such as energy usage. This candidate is then evaluated against the worst candidate of the offspring population, replacing it if the SVM prediction is better. This modified offspring population becomes

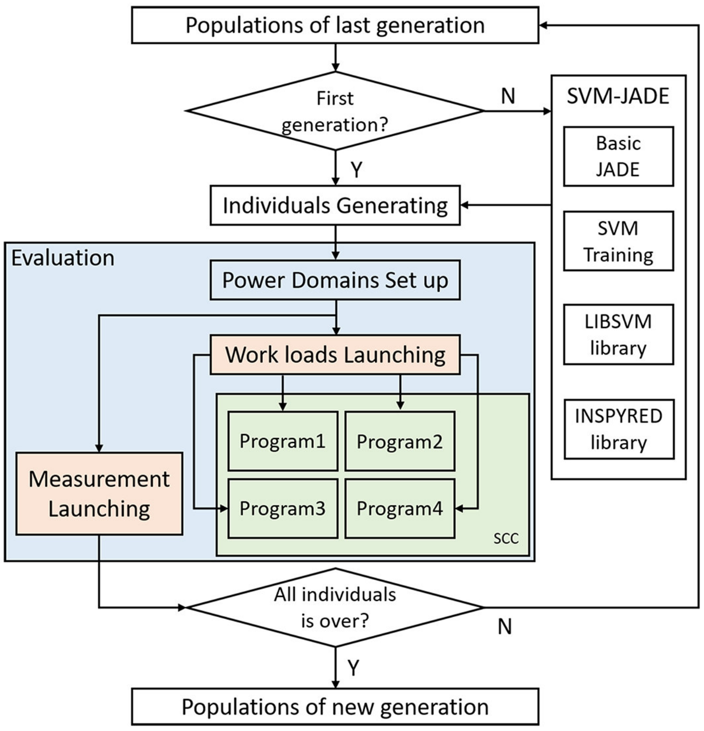

Fig. 4.1: Implementation of SVM-JADE

Algorithm 3 Search Space Generation for Core Count

```
1: function CoreSpace(candidate, dimension, max)
2:     $value \leftarrow candidate[dimension]$
3:     if $value \in [1, 2]$ then
4:         return $[1, 2, 3, 4, 5]$
5:     else if $3 \leq value \leq max - 2$ then
6:         return $[value - 1, value, value + 1]$
7:     else if $value] \in [max - 1, max]$ then
8:         return $[max - 4, max - 3, max - 2, max - 1, max]$
```

the new population for the next generation and continues to follow DE's flow of operations shown in Fig. 3.2.

5. Experimental Results. SVM-JADE described in Section 4 was implemented onto the SCC with the Intel SCCkit v1.4.0. The controlling computer, the MCPC, runs Ubuntu 10.04 64-bit with 8GB of RAM. The initial setup conditions for all the tests were that no single program could use more than 32 cores at any given time. This limitation is in place to ensure each program has at least one power domain (group of 8 cores). The SVM is trained on the top 8 candidates of the population as opposed to the entire population, and the four programs are run simultaneously. As we mentioned previously, energy and energy-delay product (EDP) are used as fitness values. EDP is a measure of both performance as well as energy as it takes both the energy it takes to execute the program as well as the user response time, e.g., the time to execute the program into consideration. The different fitness values were tested individually 3 times and the results are aggregated by average value. Since SVM-JADE and JADE are based on DE, the important part of the data is the fitness value of the single best candidate from each generation. This also corresponds to the data analysis performed in [3]. In order to conduct a fair comparison between SVM-JADE and Jade, the random candidate generator for the initial generation (Generation 0) was fixed to the same seed.

Our first experiment was to use the EDP fitness type. The results are presented in Fig. 5.1, which represents the EDP from the best single candidate of each generation averaged across all runs. It can demonstrate the overall performance of SVM-JADE vs. Jade including the convergence rate.

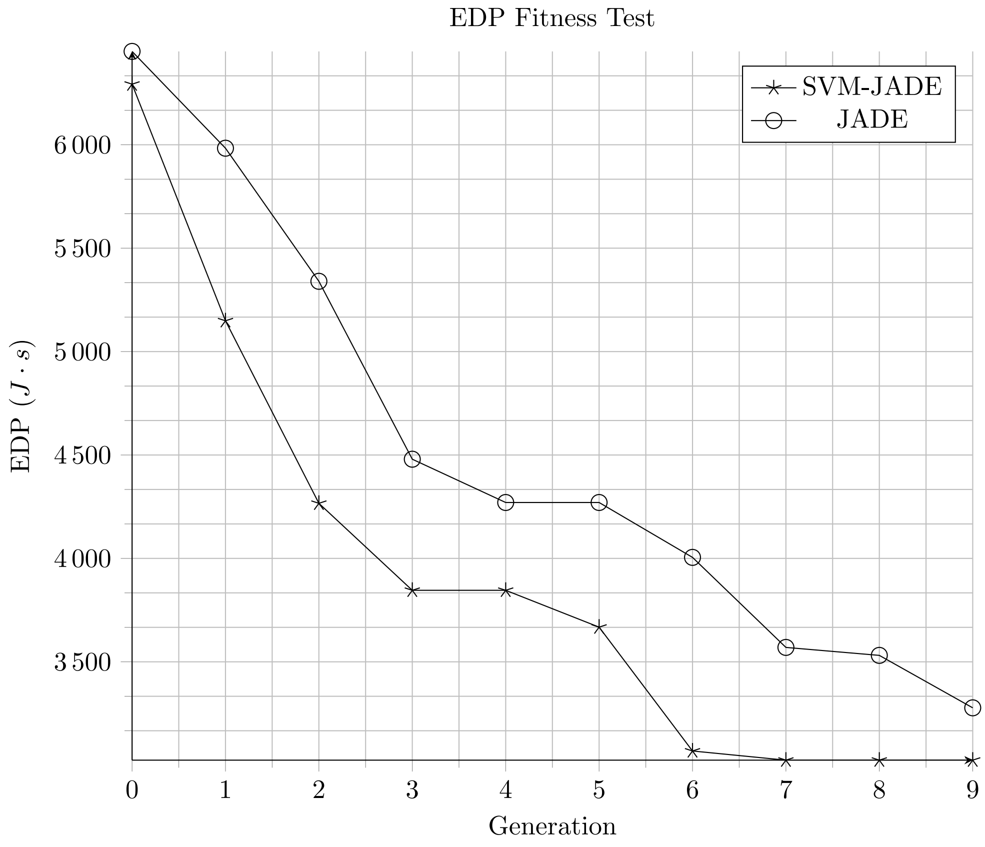

Fig. 5.1: Algorithms using EDP fitness

From Fig. 5.1, SVM-JADE achieved an EDP of 3024.499 Joule-seconds in Generation 9 while Jade ended up only decreasing the EDP to 3277.522 Joule-seconds in Generation 9. SVM-JADE is able to further reduce the EDP by 7.7% than JADE. Past Generation 6, SVM-JADE decreases by less than 1.5% showing convergence. Jade, however, is still continuing to decrease even through Generation 9.

By evaluating the difference across generations, we find that SVM-JADE results in an EDP changing rate at about −322.952 Joule-seconds per generation while Jade results an rate of −352.646 Joule-seconds on average. This point can be exemplified through a linear regression across the convergence. That is to say SVM-JADE should be regressed from Generations 0 to 6, while Jade should be regressed from Generations 0 to 9. The point of doing a linear regression across the convergence is to show the rate at which the algorithms converged since they ideally converge before the end. The regressions shows that SVM-JADE has a trend of $EDP = -427.49X + 5963.4$ and JADE has a trend of $EDP = -339.42X + 6384.6$. As can be seen from the slopes, SVM-JADE decreased to its convergence point at a quicker rate than Jade. The actual values for both SVM-JADE and JADE for EDP-based fitness can be seen in Table 5.1 with SVM-JADE in subtable (a) and JADE in (b).

The best overall result using EDP fitness from a single run was with vector $[4, 4, 4, 4, 11, 12, 23, 1]$ yielding 2997.477 Joule-seconds. This means that all four programs ran on Gear 4 and Programs 1 through 4 ran on 11, 12, 23, and 1 cores, respectively. Recall that the four programs used were *mmul*, *conv*, *cpi*, and *seqpi*.

cpi, the most parallel of all the programs, used the most cores of 23. *mmul* and *conv* are both fairly parallel programs and they used 11 and 12 cores, separately. The last program, *seqpi*, is a very sequential program and consequently only used a single core, showing the effectiveness of our proposed approach.

Table 5.1: Tabular data for EDP based fitness

Generation	EDP $(J \cdot s)$
0	6 292.992
1	5 148.874
2	4 266.357
3	3 845.74
4	3 845.74
5	3 667.679
6	3 068.848
7	3 024.499
8	3 024.499
9	3 024.499

(a) SVM-JADE

Generation	EDP $(J \cdot s)$
0	6 451.335
1	5 983.302
2	5 339.798
3	4 479.627
4	4 270.295
5	4 270.295
6	4 004.921
7	3 569.608
8	3 531.114
9	3 277.522

(b) JADE

The next form of experiment was to use energy as the fitness metric in SVM-JADE and JADE. The results for the energy fitness test are presented in Fig. 5.2.

We can see again that SVM-JADE stops significant changes in Generation 6. Jade, however, still continues to decrease even through to Generation 9. SVM-JADE achieved an energy value of 392.945 Joules in Generation 9 while Jade ended up only decreasing the energy to 428.632 Joules in Generation 9. SVM-JADE is able to reduce the energy consumption further by 8.3% than JADE. Past Generation 6, SVM-JADE continues to improve at a slowed rate as before, but JADE does not converge even by Generation 9. In this case, we have reached our near-ideal value very quickly.

Evaluating the difference across generations, we can find that SVM-JADE is changing at a rate about -16.277 joules per generation while Jade results a rate of -17.658 joules per generation on average. This can be shown via a linear regression across the convergence. That is to say SVM-JADE should be regressed from Generations 0 to 6 while Jade should be regressed from Generations 0 to 9. The regressions shows that SVM-JADE has a trend $Energy = -22.349X + 550.6$ and Jade has a trend $Energy = -17.615X + 587.55$. As can be seen from the slopes, SVM-JADE decreased to its convergence point at a quicker rate than Jade. The actual values for both SVM-JADE and JADE for energy-based fitness can be seen in Table 5.3 with SVM-JADE in subtable a and JADE in b.

The best overall result using energy fitness from a single run was with vector $[1, 4, 1, 1, 10, 10, 21, 1]$, yielding an energy consumption of 399.753 Joules. This means that *mmul*, *cpi*, and *seqpi* all ran on Gear 1 and *conv* ran on Gear 4. This is reasonable as *conv* is less sequentially heavy than programs such as *mmul* or *cpi*. *mmul* and *conv* both ran on 10 cores as they are both mainly parallel programs. *cpi*, the most embarrassingly parallel program, again has the highest core count of 21. Lastly, *seqpi*, the purely sequential program, was again dedicated just a single core, as was the case with EDP-based fitness.

It is also useful to observe how the candidate evolves across generations based on our algorithms, mapping to the program descriptions as seen in Sec. 4. Table 5.5 shows example candidates from a single run using EDP-based Fitness.

Please note here each gear and core combination in the table lines up with the program *mmul*, *conv*, *cpi* and *seqpi* respectively. Looking at $Gear_1$ and $Cores_1$, we can see that in the final generation it ends with 1 and 9 respectively. Using Gear 1 allows for heavy computations, and 9 cores allow for a semi-parallel program, which matches the previous description. *conv* obtains a gear of 4 and 16 cores because the computations are not too difficult, but the program runs much faster on multiple cores. *cpi*, a very parallel program ends with a

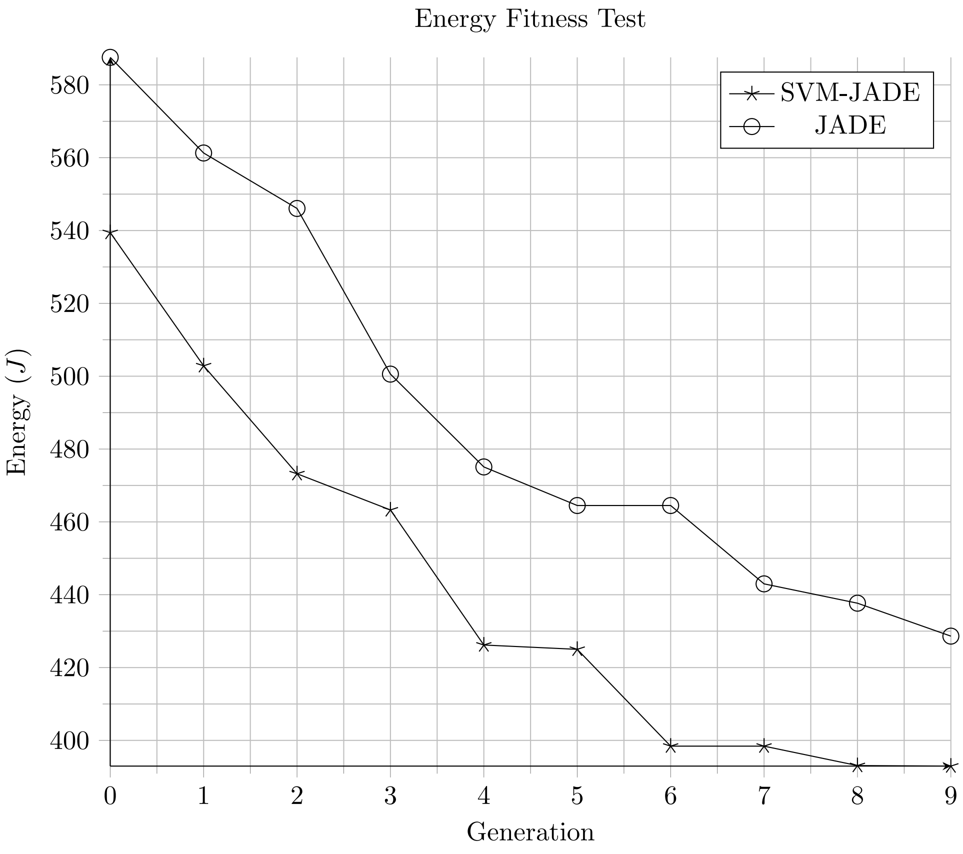

Fig. 5.2: Algorithms using Energy fitness

gear of 1 and 17 cores. Gear 1 is caused by the complexity of the integral computation, and 17 cores because the program is embarrassingly parallel. Finally, *seqpi* receives gear 1 and only 1 core. This is because it has complex computations but is a very sequential program and cannot be easily run with multiple cores.

Lastly, it is important to see the savings provided by SVM-JADE over other algorithms such as Jade. In terms of computation overhead, because SVM-JADE converges in Generation 6 as opposed to Generation 9 for Jade, in practice we are able to save the calculation and testing of over 120 candidates. This results a reduction in total computation overhead by one third.

6. Conclusions. Working in many-core computing adds great computational horsepower but also brings in added energy consumption. How to identify the most energy-efficient way to run a specific workload while still maintain a satisfactory performance level is always a challenge. The work we presented here shows that a brute force search for an ideal DVFS configuration can be avoided by using an automatic tuning algorithm. In this work, we were able to successfully apply Support Vector Machine (SVM) to an Adaptive Differential Evolution algorithm (JADE), creating a new automatic tuning algorithm SVM-JADE. SVM-JADE is more effective than JADE, as the final EDP value achieved by SVM-JADE is 7.7% less than that of JADE and the final energy value achieved by SVM-JADE is 8.3% less than that of JADE, respectively, after running 10 generations. Besides, SVM-JADE, was able to minimized desired fitness levels quicker than JADE with faster convergence speed. This is a significant improvement since each candidate from these generations gets mutated and compared. The overhead brought by adding an SVM evaluation becomes negligible when considering the potential savings in calculations and energy saving it provides. SVM-JADE is a step forward in DVFS automatic tuning on many-core platforms by minimizing fitness values lower and quicker than a representative comparable

Table 5.3: Tabular data for Energy based fitness

Generation	Energy (J)
0	539.435
1	502.859
2	473.217
3	463.278
4	426.189
5	425.005
6	398.424
7	398.424
8	393.088
9	392.945

(a) SVM-JADE

Generation	Energy (J)
0	587.553
1	561.293
2	546.062
3	500.576
4	475.089
5	464.496
6	464.496
7	442.984
8	437.675
9	428.632

(b) JADE

Table 5.5: Candidate examples from a single run

Generation	$Gear_1$	$Gear_2$	$Gear_3$	$Gear_4$	$Cores_1$	$Cores_2$	$Cores_3$	$Cores_4$
0	1	3	1	1	10	11	11	11
1	1	3	1	2	8	8	12	9
2	1	3	1	2	8	8	12	9
3	1	1	1	1	14	11	18	5
4	1	1	1	1	14	11	18	5
5	1	1	1	1	14	11	18	5
6	1	1	1	1	12	11	18	6
7	1	4	1	1	9	16	17	1
8	1	4	1	1	9	16	17	1
9	1	4	1	1	9	16	17	1

algorithm JADE. Our future work includes adjusting our SVM, its training set, as well as the decision function to continue improve the accuracy of support vector regression when selecting candidates.

7. Acknowledgement. The authors would like to thank Intel Labs for providing us with the SCC platform to conduct this research. The authors would also like to thank the anonymous reviewers for their feedbacks that greatly improved the quality of this paper. This material is based upon work supported by the National Science Foundation under Grant No. ECCS-1301953. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of either Intel or the National Science Foundation.

REFERENCES

[1] B. Roscoe, M. Herlev, C. Liu, *Auto-tuning multi-programmed workload on the scc*, 2013 International Green Computing Conference Proceedings, 06 2013.

[2] X. Yan, M. Wu, B. Sun, *An adaptive ls-svm based differential evolution algorithm*, 2009 International Conference on Signal Processing Systems, 2009.

[3] Y. Jiang, X. Qi, C. Liu, *Energy-aware automatic tuning on many-core platform via differential evolution*, The 5th International Workshop on Power-aware Algorithms, Systems, and Architectures, 2016.

[4] I. Labs, *Scc platform overview*, Intel Many-Core Application Research Community, 2010.

[5] I. Labs, *The SCC Programmer's Guide*, Intel Many-Core Application Research Community, 2012.
[6] G. Torres, J. Chang, F. Hua, C. Liu, S. Schuckers, *A Power-Aware Study of Iris Matching Algorithms on Intel's SCC*, Proceedings of the 2013 42Nd International Conference on Parallel Processing, 2013.
[7] K. Flautner, S. Reinhardt, T. Mudge, *Automatic performance setting for dynamic voltage scaling*, pp. 260–271, 2001.
[8] C.-h. Hsu and W.-c. Feng, *A power-aware run-time system for high-performance computing*, pp. 1–, 2005.
[9] K. Berry, F. Navarro, C. Liu, *A manual approach and analysis of voltage and frequency scaling using scc*, pp. 1–4, 2012.
[10] K. Berry, F. Navarro, C. Liu, *Application-level voltage and frequency tuning of multi-phase program on the scc*, pp. 1:1–1:7, 2013.
[11] J. Zhang, Z. h. Zhan, Y. Lin, N. Chen, Y. j. Gong, J. h. Zhong, H. S. H. Chung, Y. Li, Y. h. Shi, *Evolutionary computation meets machine learning: A survey*, IEEE Computational Intelligence Magazine, vol. 6, pp. 68–75, 2011.
[12] R. Storn, K. Price, *Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces*, Journal of global optimization, vol. 11, no. 4, pp. 341–359, 1997.
[13] J. Zhang, A. C. Sanderson, *Jade: adaptive differential evolution with optional external archive*, IEEE transactions on evolutionary computation, vol. 13, no. 5, pp. 945–958, 2009.
[14] C. Cortes, V. Vapnik, *Support-vector networks*, Machine learning, vol. 20, no. 3, pp. 273–297, 1995.
[15] C.-C. Chang, C.-J. Lin, *LIBSVM: A library for support vector machines*, ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.
[16] A. Garrett, *Inspyred: A framework for creating bio-inspired computational intelligence algorithms in python.* Software available at https://aarongarrett.github.io/inspyred/.
[17] CSDN, *Support Vector Machine Learning Notes–Three Levels*, 2012. Available at http://blog.csdn.net/v_july_v/ article/details/7624837.

Edited by: Pedro Valero Lara
Received: Dec 23, 2016
Accepted: May 24, 2017

DOI 10.12694/scpe.v18i2.1285
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 133–144. http://www.scpe.org © 2017 SCPE

INTEGRATING GENERIC FEM SIMULATIONS INTO COMPLEX SIMULATION APPLICATIONS

ROBERT DIETZE[†], MICHAEL HOFMANN[†], AND GUDULA RÜNGER[†]

Abstract. In this article, the efforts for integrating alternative FEM codes into a complex simulation application from the area of engineering optimisation are described. The application area, its participating software components, and their interactions are presented. The integration of two different FEM codes is demonstrated based on a dedicated FEM data conversion component. Performance results are shown to investigate the overhead caused by the data conversion.

Key words: FEM simulations, data conversion, component-based development, distributed simulations

AMS subject classifications. 68U20, 65M60

1. Introduction. The development of scientific applications for complex problems often leads to complicated program codes that are hard to maintain and less portable in terms of their performance on different hardware platforms. Component-based development approaches can provide means to handle this complexity, especially if several independently developed application programs have to be integrated into a single complex simulation [12]. Important aspects of these approaches are the implementation of new components or the integration of existing components, the efficient data exchange between components based on various communication methods, and the flexibly distributed execution of components on different hardware platforms. A further major challenge is to ensure the interchangeability of specific components, for example, to perform compute-intensive tasks, such as finite element method (FEM) simulations, with different application programs.

Having the ability to choose flexibly between different application programs provides several advantages. Program codes with different application functionalities, for example, might provide alternative solution methods that lead to different result precision or support specific simulation conditions. If the program codes provide the same application functionality, then it might be preferable to use codes that reduce costs, for example, in terms of execution time, hardware utilisation, or software licenses. The flexible integration of FEM codes into a component-based scientific simulation needs support by a generic FEM simulation component that can invoke different FEM codes. This approach would allow users of complex simulation applications, e.g. domain specialists such as mechanical engineers, to focus on the design of their simulation problems and solution methods. The usage and integration of specific program codes as well as their efficient execution on dedicated platforms, such as high performance computing clusters, is then left to the application developers.

The complex simulation which we consider is an application from mechanical engineering for optimising lightweight structures based on numerical simulations. This class of applications is extensively studied in the research project MERGE[1]. The goal is to perform simulation-based optimisations of the design and the manufacturing of fibre-reinforced plastics [10]. The overall complex simulation application consists of various program components, such as computationally intensive numerical simulations, control programs for implementing the optimisation process as well as data-oriented programs for the generation, management, and visualisation of the simulation data. FEM codes are used to simulate the cooling of the manufactured parts, which leads to residual stresses and deformations, and for the characterisation of their mechanical properties with specific operating load cases. To achieve an interchangeability of the FEM simulation component, it is required to implement data conversions for the FEM input data formats and to utilise different FEM codes and their specific execution platforms. The FEM input data formats are usually text-based and comprise information, such as the geometry and the material properties of the part to be simulated and the boundary conditions of the desired solution. The FEM codes range from open source codes to closed commercial or proprietary codes while as execution platforms usually both Linux/Unix-based or Windows-based systems are used.

In this article, we present the integration of a generic FEM simulation component into a component-based simulation application for the optimisation of lightweight structures. The main contributions are as follows:

[†]Department of Computer Science, Technische Universität Chemnitz, Chemnitz, Germany
[1]MERGE Technologies for Multifunctional Lightweight Structures, http://www.tu-chemnitz.de/merge

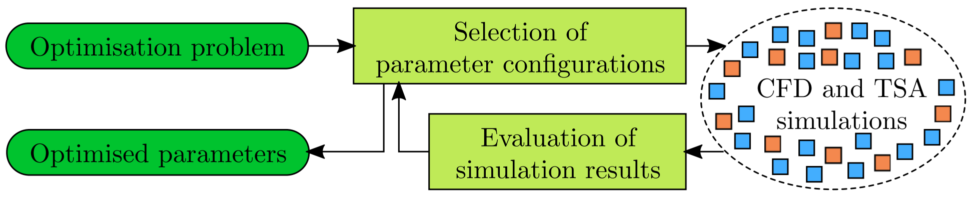
Optimisation problem
Selection of parameter configurations
CFD and TSA simulations
Evaluation of simulation results
Optimised parameters

Fig. 2.1. *Overview of the coarse structure of the optimisation process for designing lightweight structures.*

We describe the major components of the complex simulation application as well as the interactions between these components based on a service-oriented approach. The integration of generic FEM simulations is mainly achieved with a dedicated FEM data conversion component. We present the design of this conversion component and describe its support for two different FEM input data formats. Finally, we describe the implementation of the program components of the simulation-based optimisation application and their interactions based on the Simulation Component and Data Coupling (SCDC) library [12]. The SCDC library supports different data exchange methods, such as direct function calls, inter-process communication, and network communication. Thus, using the SCDC library allows for a flexibly distributed execution of the program components among different hardware platforms without requiring additional programming efforts.

The rest of this article is organised as follows. Section 2 gives an overview of the complex simulation application for optimising lightweight structures. Section 3 presents the data conversion between different input formats of FEM codes. Section 4 describes the component-based approach for implementing the distributed execution of the program components of the complex simulation. Section 5 presents results for estimating the overhead of the FEM data conversion in comparison to the execution of the FEM codes. Section 6 discusses related work and Section 7 concludes the article.

2. A complex simulation application for the optimisation of lightweight structures. The optimisation of lightweight structures to be developed in the project MERGE is performed with numerical simulations for manufacturing and using fibre-reinforced plastics. The simulation-based approach and the component-based implementation of the resulting complex simulation application is described in the following.

2.1. Optimising lightweight structures. The lightweight structures considered in the project MERGE are plastic parts that are manufactured by injection moulding. Fillers, such as short glass or carbon fibres, are mixed into the plastic to improve the mechanical properties of the parts. A computational fluid dynamics (CFD) simulation is used to simulate the manufacturing process of injecting the molten plastic and the fibres into a mould. The density and orientation of the fibres within the manufactured parts have a strong influence on their mechanical properties. The results of the CFD simulation characterise the fibre distribution within the parts and are used to model the material properties of such short fibre-reinforced plastics.

The CFD simulation is finished when the mould is filled with the molten material, thus leading also to a temperature distribution within the manufactured part. A thermal analysis simulation uses the temperature distribution as a starting point to simulate the cooling to room temperature. This cooling process can lead to residual stresses in the solid material and to deformations of the manufactured part. The resulting information about the geometrical and material properties of the manufactured part are finally used to simulate its behaviour in different operating load cases using structural analysis simulations. Both thermal and structural analysis (TSA) simulations are performed with finite element method (FEM) programs.

The overall goal of the complex simulation application is to optimise the properties of the lightweight structures. Figure 2.1 shows a coarse overview of this optimisation process. Based on an optimisation problem defined by the user of the complex simulation application, an optimisation method selects specific values for the parameters to be optimised. For short-fibre reinforced plastics, material and manufacturing parameters, such as the fibre percentage or the injection position, are varied. Each selected parameter configuration is then used to simulate the manufacturing, the cooling, and the usage of the corresponding plastic part. In this step, usually about 10–100 simulation tasks have to be computed with dedicated CFD and TSA simulation programs. However, the specific number of tasks depends on the number of parameters to optimise, the utilised optimisation method, and the load cases to consider. The simulation results with the different parameter configurations are then evaluated, such that the optimisation method can either select further parameter configurations to simulate or finish with the optimised parameters. A Kriging metamodel approach is used for the global optimisation [13]. A more detailed overview of the simulation and optimisation approaches developed in the research project MERGE is given in [8].

In this work, we concentrate on the utilisation of FEM codes for the simulation of operating load cases. This can be used, for example, to perform parameter studies with successively changing loads or to compare different solution methods supported by FEM programs. As a primary example of an FEM program, we use an in-house adaptive FEM code called SPC-FEM [5] which has been further developed according to the needs of the project MERGE. The input data for the FEM simulations to perform is generated in the custom data format of the SPC-FEM code. Additional commercial FEM codes, such as ANSYS[2], should be employed as alternative or complementary FEM methods. Thus, it is required to integrate appropriate data conversions and program executions into the complex simulation application.

2.2. Component-based application with generic FEM integration. The optimisation of lightweight structures described in the previous section involves a variety of different application programs. To achieve a sustainable solution that allows a continuous adaption to new usage scenarios and a flexible utilisation of distributed execution platforms, the overall complex simulation application is separated into individual software components as follows:

- The **optimisation** component contains the modelling of the optimisation problem with its parameters as well as the optimisation method to be used. This component acts as a driver for the simulation application and generates the input data for the FEM simulation tasks to be performed. The execution is usually not computationally intensive and might be based on user interactions. The optimisation component is thus executed on a desktop platform.
- **Simulation** components execute the FEM simulation tasks. Separate FEM components are available to perform the simulations either with the SPC-FEM or the ANSYS-FEM code. For the integration of a generic FEM simulation, an additional generic FEM component that invokes the application-specific FEM components is provided. Since FEM simulations can be computationally intensive, it might be advantageous to execute them on HPC platforms. However, the generic FEM component might also be executed in close cooperation with the optimisation component on a desktop platform.
- Domain-specific problems that are not directly related to the simulation-based optimisation of lightweight structures are solved by auxiliary components. This includes a dedicated **scheduling** component that determines schedules for the efficient utilisation of HPC platforms. In [8], we have presented several scheduling methods for assigning simulation tasks to compute resources of a heterogeneous compute clusters such that the total time for executing all simulation tasks is minimised.
- The data conversion between different FEM input data formats is performed by a dedicated **conversion** component. The input data formats used by the SPC-FEM and ANSYS-FEM codes and the implementation of the data conversion is described in Section 3. Both the scheduling and the conversion component are mainly used by the generic FEM simulation component and, thus, are executed close to their execution platform.
- A **storage** component provides separate locations for storing the input and output data of each FEM simulation. This component remains passive and does not perform any computations. However, to support high numbers of FEM simulations with large amounts of simulation data, the storage component might be executed on a dedicated server with high storage capacities.

A service-oriented approach is used to model the interactions between the different components. Within this model, each component can be both a client that accesses other service components and a service that is accessed by other client components. Figure 2.2 illustrates the service-oriented implementation of the simulation application for the optimisation of lightweight structures. The optimisation component is invoked by the user of the simulation application and generates the FEM simulations to be performed. Thus, this component represents a client that is activated first and then accesses other components. The execution of several FEM

[2]`www.ansys.com`

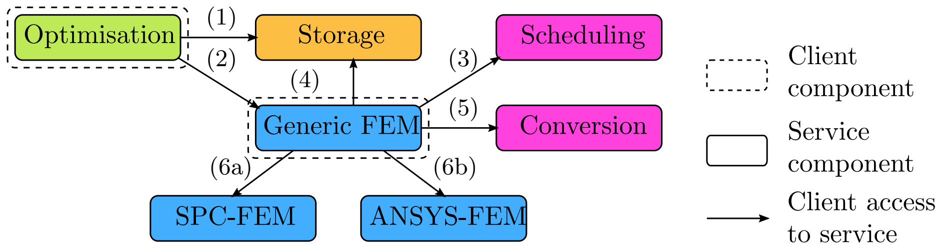

Fig. 2.2. *Overview of the components for the service-oriented implementation of the simulation application for lightweight structures. The interactions (1)–(6) represent client accesses to services.*

simulations (e. g., with different parameters, see Section 2.1) is then performed as follows:

- The input data of the FEM simulations is first transferred from the optimisation component to the storage service (1) and then the FEM simulation tasks are submitted to the generic FEM service (2).
- The generic FEM service uses information about the FEM simulation tasks and the available compute resources to set up a scheduling problem. Solving this problem is performed by a dedicated scheduling service that is accessed by the FEM service (3).
- The generic FEM service retrieves the input data of each FEM simulation task from the storage service (4) and performs the required data conversions by accessing the conversion service (5).
- The generic FEM service submits the FEM simulation tasks with its converted input data and the scheduling information about the compute resources to be used to either the SPC-FEM (6a) or the ANSYS-FEM (6b) service. The utilised service then executes its specific FEM code on the given compute resources. The FEM simulation results are gathered by the generic FEM service and then transferred to the storage service. After the generic FEM service finished all FEM simulation tasks, the submission of the FEM simulation tasks performed by the optimisation component (2) is completed.

In [12], we have presented the SCDC library that is specifically designed for implementing complex simulation applications with a component-based service-oriented approach as described in this subsection. The usage of the SCDC library for implementing the simulation application for the optimisation of lightweight structures including the flexible use of FEM codes introduced in this work is given in Section 4.

3. Data conversions for generic FEM simulations. The integration of generic FEM simulations into the optimisation application for lightweight structures is based on a data conversion between different FEM input data formats. In the following, a generic input data of FEM simulations as well as two specific data formats of FEM codes are described. Furthermore, the conversion between the two specific formats and their integration in a flexible conversion tool is presented.

3.1. Generic FEM input data. The FEM input data consists of three main parts: the geometry of the structure to be simulated, its material properties, and the boundary conditions of the desired solution. The geometry part describes the shape of the structure to be simulated and is assumed to be given as a mesh. The smallest units of a mesh are nodes given by their coordinates in a three dimensional space. Pairs of nodes can be connected to create edges. Multiple edges together form faces, e. g. four edges form a quadrilateral face. The composition of multiple faces leads to elements. Finally, the whole structure is represented by a set of elements.

The material part of the FEM input data specifies the properties of the materials the structure to be simulated consists of. Each material is described by a set of parameters of a corresponding material model. Currently, a linear elastic model is supported and the elements of the structure can use different materials (i. e., with different sets of parameter). Simulating the behaviour of a structure requires to determine the solution of an equation system that describes the state of the structure. The boundary conditions usually define constraints that need to be fulfilled by the desired solution. For example, in the FEM input data, the boundary conditions can be used to provide information about the occurring mechanical forces in an operating load case. In this case, the FEM simulation can determine a solution that describes the resulting deformation of the structure. Further boundary conditions can be set to prevent that specific parts are deformed, for example, to simulate a clamped structure.

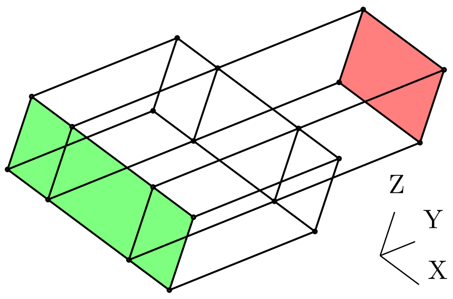

Fig. 3.1. *Geometry of an example structure for an FEM simulation where one side of the structure (green) is clamped and on the opposite side (red) a load is applied.*

Figure 3.1 illustrates an example structure consisting of two connected cuboids with different size in x-direction. The corresponding mesh consists of 20 vertices, 36 edges, 21 faces and 4 elements. This example structure is used in the following to demonstrate the different FEM data formats. The boundary conditions are chosen such that one side of the structure (green) is clamped and on the opposite side of the structure (red) a load is applied in z-direction.

3.2. SPC-FEM format. The SPC-FEM code [5] is an in-house development of an adaptive FEM solver. The content of the corresponding input format can be separated into parts for geometry, material, and boundary conditions. Furthermore, the SPC-FEM format contains a header with additional information, such as the degrees of freedom to be used. In general, the format uses a `#` character followed by a keyword to start the input data of the different parts.

The geometry part contains nodes, called vertices. After the line starting with the keyword `VERTEX`, each vertex is given by an index and its coordinates in a three dimensional space. Edges are listed after the line starting with the keyword `EDGE`. Each edge is given by an index, a type (e. g., a straight line), and the two indices of the corresponding vertices. Faces are listed after the line starting with the keyword `FACE`. Each face is given by an index, a type (e. g., a plane face), and the number of associated edges followed by the indices of these edges. Elements, called solids, are listed after the line starting with the keyword `SOLID`. Each element is given by an index, a type that specifies its material, and the number of associated faces followed by the indices of these faces. Materials are defined after the line starting with the keyword `MATERIAL`. Each single material definition comprises an index and the number of material parameters followed by the parameters to describe the material.

The SPC-FEM format supports two different types of boundary conditions. The first type of boundary conditions represents Dirichlet boundary conditions which are given after the keyword `DIRICHLET`. Dirichlet boundary conditions can be used to pre-define some values of the solution, e. g. to fix the position of some nodes. The second type of boundary conditions represents Neumann boundary conditions which are given after the keyword `NEUMANN`. Neumann boundary conditions can be used, for example, to define loads on specific parts of the structure. Each boundary condition refers to a face that is given by its index followed by one line for each degree of freedom. Each of these lines contains a type that specifies how the boundary condition is represented (e. g., with constant values) followed by the specific data of this representation.

Figure 3.2 (left) shows an example of FEM input data in the SPC-FEM format. The geometry part contains vertices (i. e., nodes), edges, faces, and solids (i. e., elements) as shown for the example structure in Fig. 3.1. The Dirichlet boundary conditions fix the coordinates of the faces of one side of the structure (e. g., face with index 8) and the Neumann boundary conditions define a load in the z-direction on the face of the opposite side of the structure (i. e., face with index 4).

3.3. MAPDL format. ANSYS is a commercial engineering analysis software including ANSYS Mechanical, a tool for FEM analysis. The Mechanical ANSYS Parametric Design Language (MAPDL) [2] is a scripting language that is used to describe the input data of ANSYS Mechanical. The MAPDL format can be used to

```
#VERSION: 2.0
#DATE: Monday, December 19, 2016
#DIMENSION: 3D
#EQN_TYPE: any equation
#DEG_OF_FREE: 3
#HEADER: 9
  20  36  21  4  0  3  1  1  0
#VERTEX: 20
   1  0.5    0.0  0.0
   ⋮
  20  0.375  0.0  0.2
#EDGE: 36
   1  1   1   2
   ⋮
  36  1  13  16
#FACE: 21
   1  1  4   1  22  11  21
   ⋮
  21  1  4  13  14  15  36
#SOLID: 4
   1  1  6  1  2  11  10  14  18
   ⋮
   4  1  6  3  4   5  12  17  21
#MATERIAL: 1
   1  5   10000   0.30   0   0   0
#DIRICHLET: 3
   8
      1  0.0
      1  0.0
      1  0.0
   ⋮
#NEUMANN: 1
   4
      1   0.0
      1   0.0
      1  10.0
```

```
/prep7

et, 1, 185

mp, ex, 1, 10000
mp, prxy, 1, 0.3

n,  1,   0.5, 0.0, 0.0
⋮
n, 20, 0.375, 0.0, 0.2

e, 1, 2, 3, 10, 11, 12, 13, 20
⋮
e, 3, 4, 5,  6, 13, 14, 15, 16

d, 8, UX, 0.0
d, 8, UY, 0.0
d, 8, UZ, 0.0
⋮
d, 19, UX, 0.0
d, 19, UY, 0.0
d, 19, UZ, 0.0

f,  4, FZ, 10.0
⋮
f, 15, FZ, 10.0

/solu
solve
finish
```

FIG. 3.2. *Example of FEM input data in the SPC-FEM format (left) and in the MAPDL format (right).*

define the mesh of the structure to be simulated, the properties of the materials to be used, and the boundary conditions for the desired solution. Additionally, instructions for preprocessing steps and for the solution method can be given.

The mesh of the structure to be simulated can be described by nodes and elements. The ANSYS-FEM code supports different types of elements with various shapes and material properties. The command `et` is used to select the current element type that is used for all following definitions of elements. The material properties of the elements are defined with the command `mp` followed by specific material parameters, such as the elastic modulus `ex` and the Poisson ratio `prxy`. After the current element properties are defined, the mesh of the structure is given as nodes and elements. A node is defined with the command `n` followed by an index and its coordinates in a three dimensional space. Elements represent collections of nodes and are defined with the command `e` followed by the indices of the nodes. With the MAPDL format, the boundary conditions can be defined for nodes. The command *d* is used to define displacement constraints of nodes and the command *f* is used to define force loads at nodes. Both commands require to specify the index of the node, a label that identifies the degree of freedom, and the value to be set.

Figure 3.2 (right) shows an example of FEM input data in the MAPDL format that corresponds to the FEM input data in the SPC-FEM format presented in Fig. 3.2 (left). The FEM input data begins by calling the preprocessor. Afterwards, the element type and the material properties are set and the nodes and elements are created. The boundary conditions are defined, by setting the displacement of the node with index 8 (and several other nodes) and by setting a force load in the z-direction for the node with index 4 (and several other nodes).

```
comment = Suppress(Literal("##")+restOfLine)

inum = Regex(r'-?\d+')
fnum = Regex(r'-?\d+(\.\d*)?([eE]-?\d+)?')

config_keys = oneOf("VERSION DATE DIMENSION EQN_TYPE DEG_OF_FREE ...")
config_info = Group(Suppress("#")+config_keys+Suppress(": ")
                +restOfLine).setParseAction(fill_info)

header_info = Group(Suppress("#")+Literal("HEADER")+Suppress(": ")
                +restOfLine).setParseAction(fill_info)
header_data = Group(OneOrMore(inum)).setParseAction(fill_data_header)

vertex_info = Group(Suppress("#")+Literal("VERTEX")+Suppress(": ")
                +restOfLine).setParseAction(fill_info)
vertex_data = Group(OneOrMore(fnum)).setParseAction(fill_data_vertex)

...

spc_data = OneOrMore(comment+lineEnd
                |config_info+lineEnd
                |header_info+lineEnd+LineStart()+header_data+lineEnd
                |vertex_info+lineEnd+LineStart()+vertex_data+lineEnd
                |...)
```

FIG. 3.3. *Definition of the Pyparsing grammar for the SPC-FEM format.*

Finally, the solver is started with the command `solve` and after finishing the computations, the ANSYS-FEM code is terminated with the command `finish`.

3.4. SPC-FEM to MAPDL format conversion. For parsing the source format of the FEM input data, we have used the Pyparsing library [16]. Pyparsing is an open source Python module for creating grammars and parsing text according to those grammars. For each FEM data format, a separate grammar has to be created. The given FEM input data is then parsed with the grammar of the source format.

Figure 3.3 shows a Python code example that defines the Pyparsing grammar for the SPC-FEM format. The grammar is build up hierarchically by defining separate parsing objects for each entry of the SPC-FEM format. Comment lines starting with the literal `##` are matched as entries whose occurrence is ignored. Commonly occurring entries, such as numbers, are defined once with a parsing object (e. g., `inum` and `fnum`) and then used in the definition of further entries. The definition of each specific FEM data entry uses its keyword as a literal and a corresponding parsing action that is executed when the entry is matched by the parser. The utilised parsing actions (e. g. `fill_info` or `fill_data_vertex`) store the parsed data in a dictionary-based data structure. Finally, the entire grammar object `spc_data` is defined by listing all previously defined entries that can occur within the source format.

After parsing the FEM input data given in the SPC-FEM format, the corresponding MAPDL format has to be generated. Since the two formats can use different representations for the mesh of the structure, an appropriate data transformation has to be applied. For example, the MAPDL format defines elements as collections of nodes, whereas the SPC-FEM format defines elements based on faces. Converting an element from the SPC-FEM format into the MAPDL format thus requires to determine all nodes of the element. For each element, the corresponding data transformation iterates over all participating faces, edges, and vertices to build the required collection of nodes. The resulting information is also stored within the dictionary-based data structure. After all data transformations are performed, the MAPDL format is generated by iterating over the required entries of the dictionary-based data structure.

3.5. Flexible FEM data conversions. The conversion of FEM data formats described in the previous subsections is closely tied to the SPC-FEM and MAPDL format. To provide a more flexible FEM data conversion for the complex simulation application developed in Section 2, we design the FEM data conversion as follows.

A dictionary-based data structure is used as a central data pool that stores all information involved in the conversion. Strings are used as keys for the dictionary entries such that both format-independent and format-specific data fields can be stored. In general, a data conversion is then performed by executing the following operations:

Source format parsing: These operations import the given FEM input data and fill in the corresponding data fields of the central data pool. For each supported FEM data format, a separate parser operation is developed as demonstrated for the SPC-FEM format. Furthermore, parsing operations might also be implemented for importing only specific parts of the FEM input data, such as the mesh of the structure.

Data transformation: These operations read the existing data fields of the central data pool, translate or combine them into new information, and write them into the corresponding data fields of the central data pool. For example, determining the nodes for each element as described for the SPC-FEM to MAPDL format conversion is implemented as a transformation operation. A further example of a transformation operation is the refinement of the given mesh of the structure by dividing each element into several smaller elements.

Target format generation: These operations use the existing data fields of the central data pool and export them with a specific data format. For each supported FEM data format, a separate generator operation is developed as demonstrated for the MAPDL format. Furthermore, generator operations might also be implemented for exporting only specific parts of the FEM input data, for example, to visualise the mesh of the structure with separate tools.

Performing a specific FEM data conversion is achieved by flexibly composing the required operations. The usage of a dictionary-based data structure avoids any limitations that might exist in specific FEM data formats. Additional FEM data formats can be easily integrated by providing the implementations of the corresponding parser or generator operations. The memory requirements of the conversion are mainly determined by the size of the dictionary-based data structure where the geometry information usually represent the largest entries. Currently, all operations of the conversion are performed sequentially and one after another. While parsing the source format and generating the target format are inherently sequential operations, data transformations might also be performed in parallel.

4. Component-based implementation for distributed systems.

The individual software components of the complex simulation application for the optimisation of lightweight structures need to be executed on dedicated hardware platforms, such as desktop computers, HPC clusters, or storage servers. We utilise the Simulation Component and Data Coupling (SCDC) library to implement the interactions between these components. In the following, we give an overview of the SCDC library and describe their usage for implementing the client and service components described in Section 2.2. A more detailed description of the SCDC library is given in [12].

4.1. Simulation Component and Data Coupling (SCDC) library.

The SCDC library is a programming library that can be used by an application programmer to implement service-oriented interactions between client and service components. In general, the interactions supported by the SCDC library are application-independent and proceed according to the following scheme: A service component provides access to *datasets* that are managed by *data providers*. A client component interacts with service components by executing *commands* on their provided datasets. The execution of a command allows to transfer input data from the client to the service and output data from the service to the client. The SCDC library provides a C and a Python interface for its library functions. Service-side functions are used to set up the data providers, to configure the data exchange methods to be used for data transfers, and to keep a service component running. Client-side functions are used to execute commands on datasets.

The datasets of an SCDC service are identified with an URI-based addressing scheme. This address identifies the specific SCDC service to interact with, the data access method to be used for data exchanges, and the specific dataset to be accessed. The following data exchange methods are supported: direct function calls between components of a single process, inter-process communication with Unix Domain Sockets between components in separate processes of a single host system, and network communication with TCP/IP sockets between components in separate host systems. The implementation details of these different data access methods are hidden in the SCDC library. Thus, an application can easily switch between different components and their specific data access methods without additional programming efforts.

The functionalities of datasets and commands depend on their specific data providers. The SCDC library contains data providers with pre-defined functionalities as well as with functionalities that can be defined by the programmer. In this work, the following data providers are used:

- The **jobrun** data provider implements a job-oriented execution of arbitrary programs. Datasets represent the jobs to be executed and commands are used to submit jobs with their input data as well as to wait for the completion of jobs and to retrieve their output data. The program to be executed for each job has to be defined by the programmer either as a shell command or as a handler function. A list of hosts can be provided for executing the jobs. The assignment of jobs to hosts is either performed in a round-robin way or according to a determined schedule.
- The **store** data provider implements a nonhierarchical folder-oriented storage within the local file system. Datasets represent the folders and the commands are used to store and retrieve the data items of the folders.
- The **hook** data provider implements a mechanism for executing arbitrary functions whenever a dataset is accessed. All functionalities of datasets and commands have to be defined by the programmer. This data provider represents a generic mechanism to set up a service with arbitrary functionality while the accessibility to the service is still handled by the SCDC library.

4.2. Implementation of client and service components of the optimisation application. Each software component of the complex simulation application for the optimisation of lightweight structures is implemented in Python as a separate module. Thus, it is possible to execute the components flexibly combined, for example, in a single Python program on one hardware platform or in separate Python programs on dedicated hardware platforms. Since all interactions between the software components are performed with the SCDC library, only the addresses used for accessing the service components have to be changed if their execution platforms are changed. The software components and their interactions as described in Section 2.2 are implemented with the SCDC library as follows:

- The **optimisation** component is implemented as a client that is started by the user of the simulation application. The SCDC library is used to execute commands for storing the FEM input data of the FEM simulation tasks on the storage component, for submitting FEM simulation tasks to the generic FEM component, and for retrieving the result data of the FEM simulation tasks from the storage component.
- The **generic FEM** component is implemented as both a service and a client. The SCDC library is used to set up a jobrun data provider that executes the submitted FEM simulation tasks as jobs. The program to be executed for each job is defined by a handler function that uses the SCDC library as a client to execute further commands. These commands access the storage component to retrieve the FEM input data, the conversion component to perform the conversion of the FEM data formats, and the SPC-FEM or ANSYS-FEM component to execute the FEM simulation task with the requested FEM code. Additionally, the SCDC library is also used as a client to execute commands that access the scheduling component to determine a schedule for the execution of the FEM simulation tasks.
- The **SPC-FEM** and **ANSYS-FEM** components are both implemented as services. The SCDC library is used to set them up with jobrun data providers that execute the submitted FEM simulation tasks as jobs with the corresponding FEM code configured as a shell command.
- The **scheduling** component is implemented as a service. The SCDC library is used to set up a hook data provider that computes a schedule when an accessing client requests it by executing a corresponding command. Command parameters are used to select a specific scheduling method. The input data of the command contains the information about the scheduling problem and the output data returns the determined schedule.
- The **conversion** component is implemented as a service. The SCDC library is used to set up a hook data provider that performs a data conversion when an accessing client requests it by executing a corresponding command. Command parameters are used to select the specific parser, data transformation, and generator operations as described in Section 3.5. The input data of the command contains the FEM input data in the source format and the output data returns the converted result in the target

Table 5.1

File sizes in KB for the FEM input data of the example structure with the SPC-FEM and ANSYS-FEM formats (without multiple white spaces).

Number of elements	SPC-FEM	ANSYS-FEM
4	1.5	1.6
32	7.6	4.0
256	54.4	22.6
2048	454.4	175.8
16384	3969.3	1491.1

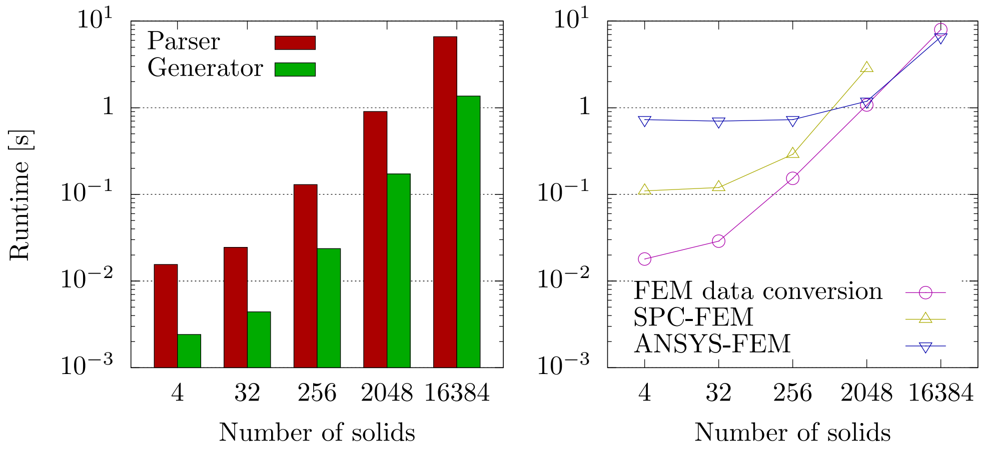

Fig. 5.1. *Runtimes of the parser and the generator for the FEM data conversion (left). Runtimes of the FEM data conversion and the SPC-FEM and ANSYS-FEM codes (right).*

format.

- The **storage** component is implemented as a service. The SCDC library is used to set up a storage data provider that stores the data within a configured folder of the local file system.

5. Performance results. In this section, we present performance results of the FEM data conversion described in Section 3 and investigate the resulting overhead in comparison to the FEM codes.

5.1. Experimental setup. The measurements are performed on a single compute node with a 4-core Intel Core i5-4440 processor with 3.10 GHz, 8 GB main memory, and a 240 GB solid-state drive. The FEM data conversion is implemented as a Python module and uses files in the local file system for reading and writing the FEM input data with the different data formats. The SPC-FEM code is an in-house FEM program implemented in Fortran that performs adaptive mesh refinement based on residual type error indicators to achieve high precision solutions. The ANSYS-FEM code is part of the commercial software package ANSYS Workbench, version 16.2. Both FEM codes use multithreading to exploit the available cores of the compute node. Thus, only shared-memory parallelism is used in the experiments. All software programs are executed under the Scientific Linux 7 (64-bit) operating system and using Python interpreter of version 2.7.5.

5.2. Data conversion performance and overhead. The example structure shown in Section 3 is used as FEM input data for the following benchmark measurements. The original mesh of the structure consists of four elements. A refinement operation that subdivides each element into eight smaller elements has been used to increase the data sizes. Thus, with four refinement steps, the original mesh with four elements is refined into geometries with 32, 256, 2048, and 16384 elements. The data conversion is performed from the SPC-FEM format to the ANSYS-FEM format. Table 5.1 lists the resulting file sizes with the two formats.

Figure 5.1 (left) shows runtimes for parsing the source format and generating the target format during the FEM data conversion depending on the number of elements in the FEM input data. The results show that parsing the source format requires significantly more runtime than generating the target format. The high computational costs of the Pyparsing module correspond to about 80–90% of the overall runtime for the FEM data conversion. The runtimes of both operations depend on the size of the mesh and, thus, increase strongly for increasing numbers of elements.

Figure 5.1 (right) shows runtimes for the FEM data conversion as well as for executing the SPC-FEM and ANSYS-FEM codes depending on the number of elements in the FEM input data. The overhead of the FEM data conversion in comparison to the FEM codes depends on the size of the mesh as well as on the specific FEM code. Up to about 256 elements, the runtime of the FEM data conversion is significantly lower and does not lead to a noticeable overhead. However, with 2048 and more elements, the runtime of the FEM data conversion increases strongly and is about equal or even higher than the runtime of the ANSYS-FEM code. As already shown, this high overhead is mainly caused by the parsing of the source format with the Pyparsing module. A direct comparison of the runtimes of the two FEM codes demonstrates their different approaches: The SPC-FEM code has a low overhead and is faster for small geometries. These are the preferred use cases, because its adaptive method is designed for starting with a coarse mesh and then refining the mesh adaptively where it is necessary. Results with 16384 elements could not be obtained for the SPC-FEM code, because an initial mesh of this size is not supported.

6. Related work. In scientific computing, the interchangeability of single software components represents a widely used concept that is especially used for compute-intensive and numerical computations. Being able to switch between different software components is used to achieve different goals. For standard libraries, such as BLAS [6] or LAPACK [1] for linear algebra computations, various implementations that are specially adapted to specific hardware platforms exist. These different implementations represent interchangeable software components that can be used to achieve performance improvements. Domain-specific libraries, such as the parallel graph algorithm library PT-SCOTCH [7] or the ScaFaCoS library for the computation of Coulomb interactions [3], provide different solution methods. These different methods represent interchangeable software components that can be used, for example, to exploit the properties of specific solution methods or to improve the accuracy of the results. The flexibility achieved with these approaches is based on the fixed programming interface of the software libraries and, thus, requires that the functional properties of all software components are the same. The approach proposed in this work is not limited to such a fixed functionality, because the individual composition of arbitrary transformation operations allows more flexible and extensible data conversions.

Environmental research is one of the most prominent areas for the development of complex simulation applications, as it involves a variety of models from different disciplines, such as atmospheric sciences, hydrology, geology, chemistry, and ecology. These applications often consist of independently developed software components for the different models. The interoperability and therefore also the interchangeability of the software components is achieved through the usage of common frameworks and toolkits. For high performance computing, this includes, for example, the Earth System Modelling Framework [11], the Common Component Architecture [4], and the Model Coupling Toolkit [14]. These approaches require that all interchangeable components implement the same interface, which often involves additional programming efforts as well as a limitation of the component functionalities. In comparison, our component-based approach is less invasive to existing application codes and allows to flexibly exploit the varying functionalities of different FEM codes.

Performing explicit data conversions as proposed in this work might also be achieved with dedicated data conversion tools. However, these tools usually support only specific parts of FEM data formats, such as the geometry of the structure. For example, the mesh generator Gmsh [9] supports the conversion between different mesh formats. The parser codes of these tools are either manually constructed or automatically created with parser generators, such as Lex and Yacc [15]. For standard formats, such as XML or VTK, existing programming libraries, such as libxml [18] or the Visualization Toolkit [17], can be used for parsing the source format or even for generating the target format. The proposed approach for the FEM data conversion can incorporate these existing tools and libraries for the implementation of specific parser, data transformation, or generator operations.

7. Conclusion. We have demonstrated that an integration of generic FEM simulations can be performed with different FEM codes and built into a complex simulation application. A component-based approach was presented to achieve a flexible implementation with interchangeable software components. A dedicated conversion component was developed and the FEM data conversion between data formats of two specific FEM codes was shown. The proposed method allows a flexible data conversion based on the composition of individual operations for parsing the source format, performing additional data transformations, and generating the target format. We demonstrated the overall approach with a complex simulation application for the simulation-based optimisation of lightweight structures. Performance results were shown to investigate the computational effort for the FEM data conversion and to compare their overhead with the FEM codes. The results have shown that especially the parsing of the source format can lead to a high overhead. Thus, it is highly required to provide a flexible data conversion approach where single operations can be easily replaced or optimised.

Acknowledgements. This work was performed within the Federal Cluster of Excellence EXC 1075 MERGE Technologies for Multifunctional Lightweight Structures and supported by the German Research Foundation (DFG). Financial support is gratefully acknowledged.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 1999.

[2] ANSYS, Inc., *ANSYS Parametric Design Language Guide*, 2015.

[3] A. Arnold, F. Fahrenberger, C. Holm, O. Lenz, M. Bolten, H. Dachsel, R. Halver, I. Kabadshow, F. Gähler, F. Heber, J. Iseringhausen, M. Hofmann, M. Pippig, D. Potts, G. Sutmann, *Comparison of scalable fast methods for long-range interactions*, Physical Review E 88 (2013), p. 063308.

[4] D. Bernholdt, B. Allan, R. Armstrong, F. Bertrand, K. Chiu, T. Dahlgren, K. Damevski, W. Elwasif, T. Epperly, M. Govindaraju, D. Katz, J. Kohl, M. Krishnan, G. Kumfert, J. Larson, S. Lefantzi, M. Lewis, A. Malony, L. Mclnnes, J. Nieplocha, B. Norris, S. Parker, J. Ray, S. Shende, T. Windus, S. Zhou, *A component architecture for high-performance scientific computing*, Int. J. High Performance Computing Applications 20 (2006), 163–202.

[5] S. Beuchler, A. Meyer, M. Pester, *SPC-PM3AdH v1.0 - Programmer's manual*, Preprint SFB/393 01-08, TU-Chemnitz (2001).

[6] L. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, R. Whaley, *An updated set of basic linear algebra subprograms (BLAS)*, ACM Transactions on Mathematical Software 28 (2002), 135–151.

[7] C. Chevalier, F. Pellegrini, *PT-Scotch: A tool for efficient parallel graph ordering*, Parallel Computing, 34 (2008), 318–331.

[8] R. Dietze, M. Hofmann, G. Rünger, *Water-level scheduling for parallel tasks in compute-intensive application components*, J. of Supercomputing, Special Issue on Sustainability on Ultrascale Computing Systems and Applications, 72 (2016), pp. 4047–4068.

[9] C. Geuzaine, J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, Int. J. for Numerical Methods in Engineering, 79 (2009), pp. 1309–1331.

[10] S. Hannusch, R. Herzog, M. Hofmann, J. Ihlemann, L. Kroll, A. Meyer, F. Ospald, G. Rünger, R. Springer, M. Stockmann, L. Ulke-Winter, *Efficient simulation, optimization, and validation of lightweight structures*, in Procs. 2nd Int. MERGE Technologies Conference for Lightweight Structures (IMTC), 2015, pp. 219–227.

[11] C. Hill, C. DeLuca, V. Balaji, M. Suarez, A. da Silva, *The architecture of the earth system modeling framework*, Computing in Science & Engineering, 6 (2004), pp. 18–28.

[12] M. Hofmann G. Rünger, *Sustainability through flexibility: Building complex simulation programs for distributed computing systems*, Simulation Modelling Practice and Theory, Special Issue on Techniques And Applications For Sustainable Ultrascale Computing Systems, 58 (2015), pp. 65–78.

[13] J. Kleijnen, W. van Beers, I. van Nieuwenhuyse, *Expected improvement in efficient global optimization through bootstrapped Kriging*, J. of Global Optimization, 54 (2012), pp. 59–73.

[14] J. Larson, R. Jacob, E. Ong, *The Model Coupling Toolkit: A new Fortran90 toolkit for building multiphysics parallel coupled models*, Int. J. of High Performance Computing Applications, 19 (2005), pp. 277–292.

[15] J. Levine, T. Mason, D. Brown, *lex & yacc*, O'Reilly & Associates, Inc., 2nd ed., 1992.

[16] P. McGuire, *Getting Started with Pyparsing*, O'Reilly Media, Inc., 2007.

[17] W. Schroeder, K. Martin, B. Lorensen, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, Kitware, 2006.

[18] D. Veillard, *libxml – The XML C parser and toolkit of Gnome.*

Edited by: Pedro Valero Lara
Received: Dec 23, 2016
Accepted: May 24, 2017

DOI 10.12694/scpe.v18i2.1286
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 145–150. http://www.scpe.org © 2017 SCPE

SAAS FOR ENERGY EFFICIENT UTILIZATION OF HPC RESOURCES OF LINEAR ALGEBRA CALCULATIONS

HRACHYA ASTSATRYAN, WAHI NARSISIAN[*] AND GEORGES DA COSTA[†]

Abstract. The most important factor of High performance computing (HPC) systems nowadays is to limit or decrease the power consumption while preserving a high utilization. And with the availability of alternative energy, which powers such systems, there is a need to maximize the usage of alternative energy over brown power. For now, the usage of alternative energy is varying in time due to different factors such as sunny days, the wind, etc. and it is crucial to have an energy-aware algorithm to maximize the usage of this energy. In this paper a SaaS service is presented to optimize a usage of alternative energy, to reduce the power consumption and to preserve a best possible percentage of resource utilization.

Key words: Linear algebra, HPC, Energy efficient utilization, DVFS, Cloud service

AMS subject classifications. 15A06, 15A24, 15A30

1. Introduction. Over the past few years, resilience has become a major issue for HPC systems, especially for large petascale and future exascale systems [1, 2], taking into account different challenges in this area varying from the deployment, maintenance and the shifting of more heterogeneous resources to a public and private cloud computing infrastructures. HPC infrastructures, such as HPC clusters, consume a lot of energy, not only by their computing elements, but also for cooling, networking and other facilities. The power consumption of data centers is increasing due to several aspects, such as increasing the data volume to deal with, the need for more calculation facilities etc [3], which in its term leads to serious environmental issues (including e-waste and CO2 emission). Thus, reducing the energy consumption of such HPC resources will play an important role to decrease the total energy consumption of these centers.

Taking into account the vast number of applications running on these systems, it is crucial to optimize or find ways to improve the energy efficiency for HPC applications and to build an automatic scheduling environment for optimization of the usage of such infrastructures in the context of keeping the performance regardless of energy consumption [4].

There has been a big amount of investigations and research to resolve these environmental challenges, especially by adopting renewable energy supply techniques (such as solar panels), data centers partially powered with green energy and it is very important to maximize the utilization of this energy when it is available [5]. In this work, a SaaS service is introduced that consist of the following two steps to reduce the power consumption of HPC clusters:

- To add more computing resources to the submitted jobs, if they are available.
- To suggest the usage of alternative energy period, which, in turn, will decrease the power consumption cost, because the price of brown energy is much higher than the price of alternative energy.

The studies have been carried out in the domain of linear algebra simulations lying at the heart of most calculations in scientific computing.

The remainder of this paper is divided into the following sections: section 2 is a related work, section 3 shows the methodology and the service, section 4 illustrates implemented experiments and their results, section 5 represents the case studies and, finally, section 6 is the conclusion.

2. Related work. The HPC clusters have crucial roles in the context of energy consumption because the energy consumption of the networking, data and computational infrastructures increases exponentially over the time. Therefore, it is important to study and analyze the power consumption of such infrastructures, as they incur tremendous energy costs and CO2 emissions. For instance, the power consumption of U.S. data centers is expected to grow to 140 billion kWh by 2020 [6].

[*]Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic of Armenia, P. Sevak 1, Yerevan 0014, Armenia (hrach@sci.am, wahi@sci.am).

[†]Institute of Computer Science Research, University of Toulouse, France (dacosta@irit.fr).

Several methods and techniques have been suggested and implemented to save energy at data centers and HPC resources. At the hardware level, the Dynamic voltage and frequency scaling (DVFS) method mainly is used, which is a well-known and efficient technique for reducing energy consumption in modern processors. For instance, the paper [7] analyzes the impact on power consumption of two DVFS-control strategies when applied to the execution of dense linear algebra operations on multi-core processors. Another example is to adopt DVFS method to decrease a power consumption of an application without affecting the performance of HPC resources [8, 9]. Reduces clock frequencies for MPI ranks that lack computational work, this technique showed an average measured energy savings of 10.6% and a most of 21.0% over regular application runs [10].

There are several studies related to measuring and control of the power and energy consumption of HPC systems by various components in the software stack [16, 12, 13]. In the cloud environments, the efficient computing resource utilization, and power consumption reduction are being addressed through the intelligent workload, server consolidation, and other mechanisms [14].

In recent years, significant amount of studies have been devoted to optimizing HPC applications for green energy sources, which expose rapid changes in power's availability due to the use of local renewable energy. There are studies, for instance, that discuss the issue of how deploying a scheduler, which can predict available renewable energy and can reschedule jobs preserving their deadlines [15], based on information gathered from the HPC load offer power-aware scheduling [16]. Our approach is to focus on combining the energy consumption of a real scientific application running on HPC production system by not only scheduling the jobs during alternative energy period, but also increasing more computational resources to the jobs in term of CPU cores. These approaches have been taken into account in the SaaS service for a maximum utilization of resources.

3. Service and Methodology. Nowadays, it is a challenge to balance the use and performance of HPC systems. In the meantime, periodical increase of energy costs is forcing infrastructure providers to operate the resources within an energy budget or to decrease energy usage. Therefore, resource management in the context of HPC refers to the process of assigning and scheduling workloads to resources. In the case of alternative energy sources, it is a necessity to maximize the utilization of alternative energy over brown energy.

Brown energy is energy that comes from conventional fossil fuels, such as oil or coal. The combustion of these fuels releases harmful emissions into the environment. Renewable or “green” energy comes from clean sources such as the wind, the solar cell that are more sustainable and are better for the environment.

The provided service or the scheduler is considered as a SaaS or “software as a Service” because it provides an easy way to handle job submission into Linux cluster, SaaS can be considered as a thin client model for software provision, where it is providing the user a point of access to software running on servers.

The total power consumption of HPC cluster is given by the following equation, where T_e is the total energy consumption of the system

$$T_e = \sum_{i=1}^{n} E_i \tag{3.1}$$

where E_i is the energy consumption per each job; n is the number of jobs completed per unit time. We consider two prices of the electricity: one is for the brown power and the second one is for the alternative energy. The energy consumption of each job is given by the following equation:

$$E_i = Time_i \cdot Price_i \cdot Nodes_i \tag{3.2}$$

The $Time_i$ is given in seconds; the $Price_i$ is given for one watt and the $Nodes_i$ is a crucial factor, because, depending on the number of nodes there are different numbers for power consumption.

There is a single restriction that the jobs must be completed during a single day, in other words, there is a number of tasks, which need to be completed during a 24 hours time period. The ultimate goal of the work is to reduce the total energy consumption T_e by the suggested methodology (see Fig. 3.1).

The job script checks the number of idle nodes to decide the free resources, then, after the submission, the script, based on the size of the job, allocates more resources, if they are available. The crucial factor is the job priority and the resources available; in case the job has a higher priority, the script will automatically alerts SLURM scheduler to pause the jobs on the required resources to run the higher priority job first. The “Period”

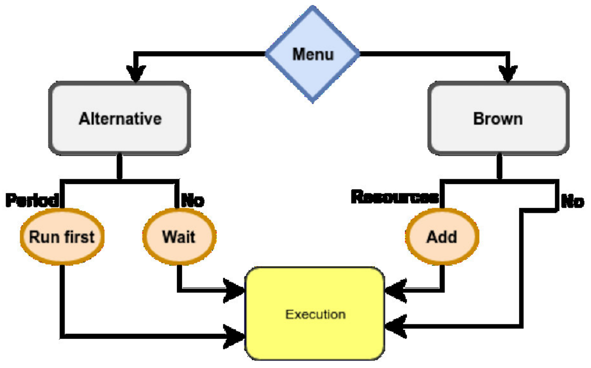

FIG. 3.1. *Schematic Illustration of Methodology.*

word on the schematic illustration refers to the jobs, which are coming in the alternative energy period, so they have priority to go first, the “Resources” on the other hand shows that during the brown energy period there is a possibility to add more resources if they are available. The main drawback here is that when there are many high priority jobs, the other jobs will get delayed for much longer than expected, but, as mentioned before, there is a restriction of a quantity of completed jobs for each day, so, on average, the time of delay will be acceptable.

The command line interface is used by users to submit jobs, the small matrix indicate 4096 size matrix, the medium 8192 size, and finally the large is 12288 size.

The following two scenarios are available after the submission:

- If the user chooses to submit a job during brown energy, the script automatically checks, if there are any available computational resources and adds them to the job. If there are no resources, the job is being executed as requested. This procedure is hidden from the user, so the user doesn’t know about it.
- If the submission is selected to be done during the alternative energy period, the script checks the time (there are two periods of alternative energy), if it is in those periods, the job will be executed at once, and all other jobs on the selected computing element will be put on hold.

The above-mentioned results are accomplished by increasing the priority of the alternative energy queue to be higher than the others. If the execution is not in the alternative energy period, the jobs stay on hold till that time. The approach gives some benefits, such as to hide the complexity of creating scripts to submit jobs, to increase the time execution, and to maximize the usage of alternative energy by finishing the maximum number of jobs.

4. Experiments Results. Scientific computing [17] aims at constructing mathematical models and numerical solution techniques for solving problems arising in science and engineering. The solution of linear system of equations lies at the heart of most calculations in scientific computing. For the past twenty years, there has been a great deal of activity in algorithms and software for solving linear algebra problems. For this reason, a specific case of the linear algebra problems has been chosen, namely, the matrix multiplications.

The PBLAS library of ScaLAPACK (Scalable Linear Algebra PACKage) [18] package has been used in the experiments, which is a library of high-performance linear algebra routines for distributed memory message-passing computers and networks of workstations supporting Parallel Virtual Machine and/or Message Passing Interface. PBLAS can be seen as a parallel version of the BLAS (Basic Linear Algebra Subprograms) [19]. Using the PBLAS, three separate levels of operations (vector-vector, matrix-vector and matrix-matrix) can be performed in parallel, which is widely used by various scientific applications in the domain of High-performance linear algebra. The PBLAS Level 3 routines, which are the most computing-intensive, perform distributed matrix-matrix operations.

The PDGEMM double precision routine of the PBLAS is used for experiments, as it is one of the most widely used layer three routine. The Linux cluster consists of a single controller node and four computational nodes (each node has 2 CPU cores Intel Xeon E5420, 1 GB RAM, Ubuntu 14.04 X86 64 OS)is used as a testbed

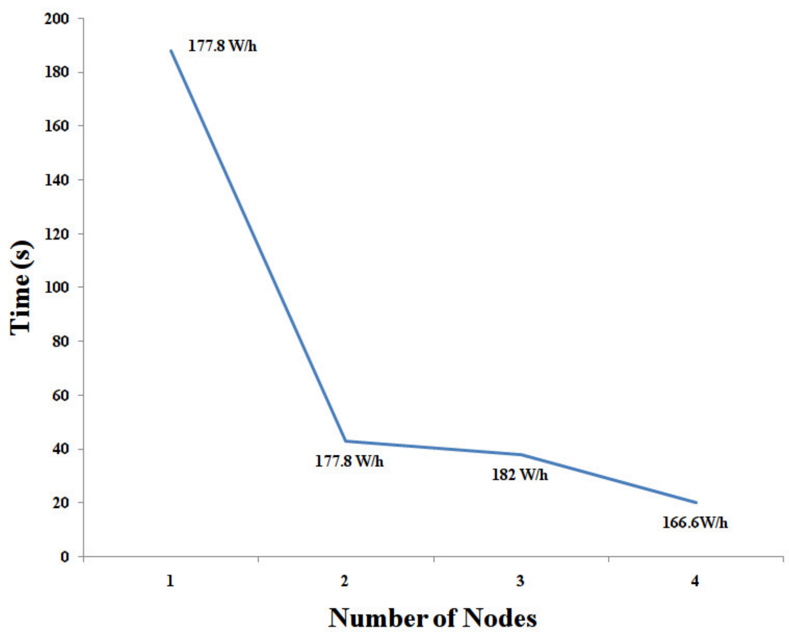

Fig. 4.1. *Experiments results for small matrices*

infrastructure for conducting the experiments. Taking into account the testbed infrastructure's parameters (mostly RAM), three different sizes of matrices are studied: small (4096×4096), medium (8192×8192) and large (12288×12288). The Simple Linux Utility for Resource Management (Slurm) [21] used for jobs scheduling which is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system both for large scale and small Linux clusters. To ensure that the results were statistically sound and the value of the execution time and power consumption of each matrix is reliable, for each random execution the experiment is run ten times and the arithmetic mean is taken.

The results of the experiments are shown in the Figure 4.1.

The 12288 matrix is only fitted on the whole cluster, and the execution time is: 11 minutes, the power consumption is: 191.8 W/h. Based on these results:

- For small size matrices, the power consumption is becoming almost the same if the number of nodes is increased.
- For medium matrices, there is an execution time improvements in case of using four nodes, and the power consumption remains almost the same.
- For large matrices, the best result is taken when the job is executed on four nodes in term of execution time and power consumption.

5. Use Case. As a use case randomly generated jobs have been submitted received that were from two different applications [22]. The power consumption of the server on the idle state has been measured in order to check the power consumption during jobs execution, because the power meter is showing the overall power consumption of the server. The total energy consumption for each hour on the idle state was 149 watt. The total energy consumption obtained for a single day was about $T_e = 1385$ kilowatt with the following breakdown results:

- 168 small matrix jobs (mix on 2, 3 and 4 nodes).
- 167 medium matrix jobs (mix on 2, 3 and 4 nodes).
- 72 large matrix jobs.

The energy consumption costs per day were about 126.7€(the mentioned price is for the corresponding Kilowatt). The same amount of jobs have been executed with the same node numbers and with the following scenarios in

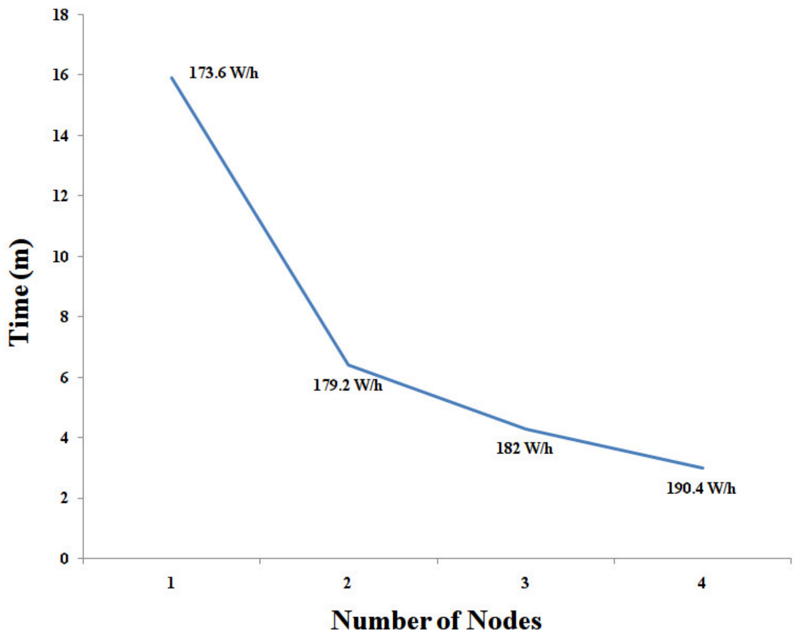

Fig. 4.2. *Experiments results for medium matrices*

order to study the effectiveness of the suggested model. In the case of only big matrix jobs are executed during the alternative energy period, the total energy consumption was about 1039 kilowatt The price for this amount is: Price = 87.7€ (the mentioned price is for the corresponding KW), and the same number of jobs is executed only in eighteen hours.

The same mix of matrices are executed during the alternative energy period and in this case, the total energy consumption is the same: T_e =1120€.

As stated from the suggested model there is a reasonable amount of reduced energy consumption and in the meanwhile, the utilization of the resources is increased by cutting out the time for the same number of jobs to be executed.

6. Conclusion. By using the suggested methodology there is an effective gain in two separate domains:

- The same amount of jobs can be run in a shorter time, which gives a possibility to increase the number of completed jobs per day.
- The price of consumed electricity is less due to effective usage of alternative energy period.

For future work, it is planned to enhance the methodology by taking into account the weight of each submitted job depending on several factors such as power consumption, time etc, which will give an opportunity to deploy the mentioned methods for various scientific applications.

Acknowledgments. The European Commission through the VI-SEEM (Virtual Research Environment in Southeast Europe and the Eastern Mediterranean), EaPConnect Projects, the COST Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)', and The Armenian State Thematic Project entitled "Towards the Green e-Infrastructures" supported the work presented in this paper.

REFERENCES

[1] J. Dongarra, P. Beckman, and et.al , *The International Exascale Software Roadmap*, International Journal of High Performance Computing, 25:1 (2011), pp. 3–60.

[2] T. Hey and S. Tansley, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, First ed., p. 284, 2009.

[3] E. H. D'Hollander, J. J. Dongarra, I. Foster, L. Grandinetti, and G. R. Joubert, *Transition of Hpc Towards Exascale Computing*, IOS Press, p. 232, 2013.

[4] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, *United States Data Center Energy Usage Report*, LBNL-1005775 p. 257, June 2016.

[5] C. Li, A. Qouneh and T. L, *iSwitch: Coordinating and optimizing renewable energy powered server clusters*, Proc. of ACM International Symposium on Computer Architecture, 2012, pp. 512–523.

[6] E. R. Masanet, R. E. Brown, A. Shehabi, J. G. Koomey, and B. Nordman, *Estimating the Energy Use and Efficiency Potential of U.S. Data Centers*, Proc. of IEEE 99 volume 8, August 2011, pp. 1440-1453.

[7] M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Ort, *DVFS-control techniques for dense linear algebra operations on multi-core processors*, Computer Science - Research and Development, 27:4 (2012), pp. 289-298.

[8] H. Astsatryan, W. Narsisian, A. Kocharyan, G. da Costa, A. Hankel, and A. Oleksiak, *Energy Optimization Methodology for e-Infrastructure Providers*, Concurrency and Computation: Practice and Experience, (2016), doi: 10.1002/cpe.4073.

[9] H. Astsatryan, W. Narsisian, G. da Costa, and T. Gurout, *Dynamic Voltage and Frequency Scaling for 3D Classical Spin Glass*, Proc. of IEEE Computer Science and Information Technologies (CSIT), September 2015, pp. 121-124.

[10] A. Tiwari, M. Laurenzano, J. Peraza, L. Carrington, and A. Snavely, *Green queue: Customized large-scale lock frequency scaling*, Proc. of 2012 Second International Conference on Cloud and Green Computing, November 2012, pp. 260-267.

[11] Y. Georgiou, T. Cadeau, D. Glesser, D. Auble, M. Jette, and M. Hautreux, *Energy Accounting and Control with SLURM Resource and Job Management System*, Proc. of 15th International Conference, ICDCN 2014, January 2014, Volume 8314 of the series Lecture Notes in Computer Science, pp. 96-118.

[12] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji, *Designing energy efficient communication runtime systems: a view from PGAS models*, Journal of Supercomputing, 63:3 (2013), pp. 691-709.

[13] H. Shoukourian, T. Wilde, A. Auweter, and A. Bode, *Monitoring Power Data: A first step towards a unified energy efficiency evaluation toolset for HPC data centers*, Environmental Modelling & Software, 56 (2014), pp. 13-26.

[14] Md. H. Ferdaus and M. Murshed, *Energy-Aware Virtual Machine Consolidation in IaaS Cloud Computing*, Cloud Computing, Part of the series Computer Communications and Networks, October 2014, pp. 179-208.

[15] I. Goiri, Md. E. Haque, K. Le, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, *Matching renewable energy supply and demand in green datacenters*, Ad Hoc Networks, 25 (2015), pp. 520-534.

[16] C. Dupont, *Renewable Energy Aware Data Centres: The Problem of Controlling the Applications Workload*, Proc. of Second International Workshop, E^{2}DC 2013, Volume 8343 of the series Lecture Notes in Computer Science, 2014, pp. 16-24.

[17] A. Petitet , H. Casanova , J. Dongarra , Y. Robert , and R. C. Whaley, *Parallel and Distributed Scientific Computing: A Numerical Linear Algebra Problem Solving Environment Designer's Perspective*, Handbook on Parallel and Distributed Processing, 1999.

[18] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK: A Linear Algebra Library for Message-Passing Computers*, Proc. of SIAM Conference on Parallel Processing for Scientific Computing, March 1997, pp. 1-15.

[19] S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, and K. Remington, *An Updated Set of Basic Linear Algebra Subprograms (BLAS)*, ACM Transactions on Mathematical Software, 28:2 (2002), pp. 135-151.

[20] H. Astsatryan, V. Sahakyan, Yu. Shoukouryan, M. Daydé, A. Hurault, R. Guivarch, A. Terzyan, L. Hovhannisyan, *Services Enabling Large-Scale Linear Systems of Equations and Algorithms based on Integrated P-Grade Portlal*, Grid Computing, 11:2 (2013), pp. 239-248.

[21] A. B. Yoo, M. A. Jette, M. Grondona, *SLURM: Simple Linux utility for resource management*, Proc. of 9th International Workshop, JSSPP 2003, June 2003, Volume 2862 of the series Lecture Notes in Computer Science, pp. 44-60.

[22] H. Astsatryan, T. Gevorgyan, and A. Shahinyan, *Web Portal for Photonic Technologies Using Grid Infrastructures*, Software Engineering and Applications, 5:11 (2012), pp. 864-869.

Edited by: Pedro Valero Lara
Received: Jan 9, 2016
Accepted: May 24, 2017

DOI 10.12694/scpe.v18i2.1287
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 151–160. http://www.scpe.org © 2017 SCPE

TOWARDS HPC-EMBEDDED. CASE STUDY: KALRAY AND MESSAGE-PASSING ON NOC

PEDRO VALERO-LARA,[*] EZHILMATHI KRISHNASAMY[†] AND JOHAN JANSSON [‡]

Abstract. Today one of the most important challenges in HPC is the development of computers with a low power consumption. In this context, recently, new embedded many-core systems have emerged. One of them is Kalray. Unlike other many-core architectures, Kalray is not a co-processor (self-hosted). One interesting feature of the Kalray architecture is the Network on Chip (NoC) connection. Habitually, the communication in many-core architectures is carried out via shared memory. However, in Kalray, the communication among processing elements can also be via Message-Passing on the NoC. One of the main motivations of this work is to present the main constraints to deal with the Kalray architecture. In particular, we focused on memory management and communication. We assess the use of NoC and shared memory on Kalray. Unlike shared memory, the implementation of Message-Passing on NoC is not transparent from programmer point of view. The synchronization among processing elements and NoC is other of the challenges to deal with in the Karlay processor. Although the synchronization using Message-Passing is more complex and consuming time than using shared memory, we obtain an overall speedup close to 6 when using Message-Passing on NoC with respect to the use of shared memory. Additionally, we have measured the power consumption of both approaches. Despite of being faster, the use of NoC presents a higher power consumption with respect to the approach that exploits shared memory. This additional consumption in Watts is about a 50%. However, the reduction in time by using NoC has an important impact on the overall power consumption as well.

Key words: Karlay, Embedded Architectures, High Performance Computing, Jacobi Method, OpenMP, Power Measurements.

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

Advanced strategies for the efficient implementation of computationally intensive numerical methods have a strong interest in industrial and academic community. In the last decade, we have lived a spectacular growth in the use of many-core architectures for HPC applications [8, 14, 15, 9, 10]. However, the appearance of other (low-power consumption) embedded many-core architectures such as Kalray [1] has created new challenges and opportunities for performance optimization in multiple applications. In this work, we have explored some of these new opportunities towards a supercomputing on a chip era.

Kalray integrates its own OS and is not in need of a co-processor as in the case of other many-core processors [4, 1]. In Karlay, highly expensive memory transfers from host main memory to co-processor memory are not necessary, as in other architectures, such as NVIDIA GPUs [13] or Inel MIC [12]. Besides, this architecture offers the possibility to communicate each of the processing elements via a Network on Chip (NoC) connection composed by links and routers [4, 1]. Kalray has been previously used for video encoding and Monte Carlo applications [2]. However, these works lack information of how to implement these applications and what are the most efficient programming strategies and architectonic features to deal with our embedded processor. The NoCs have been recently used as a level in-between the computing cores and shared memory [5, 17, 7]. The NoCs in these systems can be configurable depending on the particular needs of the applications. However, the NoC in Kalray is completely different. In Kalray, there are two different and independent inter-connectors, one bus which connects each of the processing elements to shared memory and one NoC which connects the different processing elements (clusters) among them.

We have chosen as a test case a widely known and extended problem, that is Jacobi method [16]. The main motivation of this work is twofold. While, on one hand, this work presents the main challenges to deal with the Kalray architecture. On the other hand, we present two different approaches to implement the communication among the different processing elements of our Kalray processor, one based on using shared memory and other based on using a Network on Chip, which works as interconnection among the set of processing cores. We detail and analyze deeply each of the approaches, presenting theirs advantages and disadvantages. Moreover, we include measurements for power consumption in both approaches.

[*]Barcelona Supercomputing Center, Spain. (pedro.valero@bsc.es).
[†]Basque Center for Applied Mathematics (BCAM), Bilbao, Spain (ekrishnasamy@bcamath.org.)
[‡]Basque Center for Applied Mathematics (BCAM), Bilbao, Spain, and KTH Royal Institute of Technology, Stockholm, Sweden (jjansson@bcamath.org).

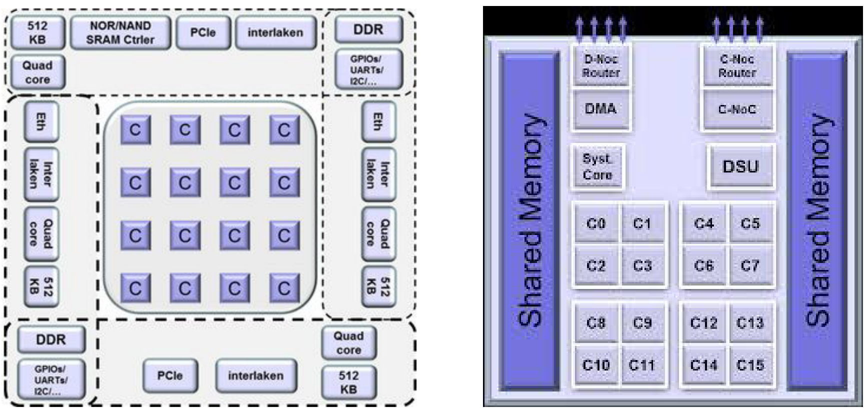

FIG. 2.1. *Kalray MPPA many-core (left) and compute cluster (righ) architecture [2].*

This paper is structured as follows. Section 2 briefly introduces the main features of the architecture at hand, Kalray. Then, we detail the techniques performed for an efficient implementation of the Jacobi method on Kalray processor in Section 3. Finally, In Section 4, it is carried out the performance analysis of the proposed techniques in terms of consuming time, speed-up and power consumption. At the end of this work, we outline some conclusions.

2. Kalray Architecture. Kalray architecture [2] is an embedded many-core processor. It integrates 288 cores on a single 28 nm CMOS chip with a low power consumption per operation. We have 256 cores divided into 16 clusters which are composed by 16+1 cores each. 4 quad-core I/O subsystems (1 core per cluster) are located at the periphery of the processing array (Figure 2.1-left). They are used as DDR controller for accessing to up to 64GB of external DDR3-1600. These subsystems control a 8-lane Gen3 PCI Express for a total peak throughput of 16GB/s full duplex. The 16 compute clusters and the 4 I/O subsystems are connected by two explicitly addressed Network on Chip (NoC) with bi-directional links, one for data and the other for control [2, 3]. NoC traffic does not interfere with the memory buses of the underlying I/O subsystem or compute cluster. The NoC is implemented following a 2-D torus topology.

The compute cluster (Figure 2.1-right) is the basic processing unit of our architecture [2]. Each cluster contains 16 processing cores (C0, C1, C2, . . . , C15 in Figure 2.1-right) and one resource management (Syst. Core in Figure 2.1-right) core, a shared memory, a direct memory access (DMA) controller, a Debug & System Unit (DSU), and two routers, one for data (D-NoC) and one for control (C-NoC). The DMA is responsible to transfer data among shared and the NoC with a total throughput of 3.2GB/s in full duplex. The shared memory compromises 2MB organized in 16 parallel banks, and with a bandwidth of 38.4 GB/s. The DSU supports the debug and diagnosis of the compute cluster.

Each processing or resource management core is a 5-way VLIW processor with two arithmetic and logic units, a multiply-accumulate & floating point unit, a load/store unit, and a branch & control unit [2]. It enables up to 800MFLOPS at 400MHz, which supposes almost 13 GFLOPS per cluster and almost 205GFLOPS in total by using the 16 clusters. These five execution units are connected to a shared register file which allows 11 reads and 4 writes per cycle. Each core is connected to two (data & instruction) separate 2-way associate caches (8KB each).

Kalray provides a software development kit, a GNU C/C++ & GDB development tool for compilation and debugging. Two programming models are currently supported. A high level programming model based on data-flow C language called $\sum$C [6], where programmers do not care about communication, only data dependencies must be expressed. The other programming model supported is a POSIX-Level programming model [4, 1]. It distributes on I/O subsystems the sub-processes to be executed on the compute clusters and pass arguments through the traditional *argc*, *argv*, and *environ* variables. Inside compute clusters, classic shared memory programming models such as POSIX threads or OpenMP pragmas are supported to exploit more than one processing core. Specific IPC takes advantage of the NoC connection. Unlike $\sum$C, the POSIX-

Algorithm 1 Jacobi OpenMP Algorithm.

```
1:  $jacobi(A, Anew, NX, NY)$
2:  $float\ err;$
3:  $\#pragma\ omp\ parallel\ for$
4:  for $int\ i = 1 \rightarrow NY - 1$ do
5:      for $int\ j = 1 \rightarrow NX - 1$ do
6:          $Anew[i * NX + j] = 0.25 * (A[i * NX + (j - 1)] + A[i * NX + (j + 1)]$
7:                                 $+ A[(i - 1) * NX + j] + A[(i + 1) * NX + j]);$
8:          $err = maxf(err, fabs(A_{new}[i * NX + j] - A[i * NX + j]));$
9:      end for
10: end for
11: $\#pragma\ omp\ parallel\ for$
12: for $int\ i = 1 \rightarrow NY - 1$ do
13:     for $int\ j = 1 \rightarrow NY - 1$ do
14:         $A[i * NX + j] = Anew[i * NX + j];$
15:     end for
16: end for
```

Level programming model presents more important challenges from programmer side, however it allows us to have more control on hardware and optimize both, communication and computation. In the present work, the authors have followed the programming model based on POSIX.

3. Jacobi Method Implementation on Kalray. We have chosen as test case the Jacobi method [16]. This is a good example, which allows us to study and evaluate different strategies for communication. The parallelization is implemented following a coarse-grained distribution of (adjacent) rows across all cores. This implementation is relatively straightforward using a few OpenMP pragmas on the loops that iterate over the rows of our matrix (see Algorithm 1).

One of the most important challenges in Kalray is the communication and memory management. To address the particular features of Kalray architecture, we use the Operating System called *NodeOs* [1], provided by Kalray. *NodeOs* implements the Asymmetric Multi-Processing (AMP) model. AMP takes advantage of the asymmetry found in the clusters between the Resource Management Core (RMC) and the Processing Element Cores (PEC). RMC runs the operating system (kernel and NoC routines) on the set of RM (single-core). PEC are dedicated to run user threads, one thread per PEC. PEC can also call functions, such as *syscall* that are in need of OS support, which are received and compute by RMC. When a PEC executes a *syscall* call, it sends an event and it is locked until it receives an event from the RMC. This process is necessary to know that the *syscall* has been processed. Data and parameters are exchanged using shared memory. We have two codes, one executed by RMC (*IO* code) and other (*cluster* code) executed by PECs. The work is distributed following a master/slave model that is well suited to Kalray architecture. The *IO* code is the master. It is in charge of launching the code and sending data to be computed by slaves. Finally they wait for the final results. Otherwise, the *cluster* code are the slaves. They wait for data to be computed and send results to IO cluster.

The POSIX-Level programming model of Kalray (*NodeOs*) allows us to implement the communication among different clusters in two different ways. While shared memory (accessible by all clusters) is used for the communication in the first approach (*SM*), in the second approach (*NoC*), we use channels (links) and routers. For sake of clarify, we include several algorithms in which we detail the main steps of each of the approaches. Algorithms 2 and 3 illustrate the *IO* and *cluster* pseudocodes for the *SM* approach and Algorithms 4 and 5 for the *NoC* approach respectively.

The communication is implemented by using some specific objects and functions provided by *NodeOS*. Next, we explain each of these objects and functions. The transfers from/to global/local memory are implemented via *portals*. These *portals* must be initialized using specific paths (one path per cluster) as *A_portal* in Algorithm 2. Then, they must be opened (*mppa_open*) and synchronized (*mppa_ioctl*) before transferring (*mppa_pwrite* in Algorithm 2 and *mppa_aio_read* in Algorithm 3) data from/to global/local memory. The slaves are launched from master via *mppa_spawn* which include parameters and name of the function/s to be computed by cluster/s.

Algorithm 2 *Shared Memory* I/O pseudocode.

```
1: $const\ char\ *cluster\_executable = "mainCLUSTER";$
2: $static\ float\ A[SIZE]; static\ float\ Anew[SIZE];$
3: $\textbf{int mainIO(int argc , char } *\textbf{argv[] )}$
4: $long\ long\ dummy = 0; long\ long\ match = -(1 << CLUSTER\_COUNT);$
5: $const\ char\ *root\_sync = "/mppa/sync/128:1";$
6: $const\ char\ *A\_portal = "/mppa/portal/"CLUSTER\_RANGE":1";$
7: $const\ char\ *Anew\_portal = "/mppa/portal/128:3";$
8: $// - -\textbf{OPENING\_FILES} - -//$
9: $int\ root\_sync\_fd = \textbf{mppa\_open}(root\_sync, O\_RDONLY);$
10: $int\ A\_portal\_fd = \textbf{mppa\_open}(A\_portal, O\_WRONLY);$
11: $int\ Anew\_portal\_fd = \textbf{mppa\_open}(Anew\_portal, O\_RDONLY);$
12: $// - -\textbf{PREPARE\_FOR\_RESULT} - -//$
13: $status| = \textbf{mppa\_ioctl}(root\_sync\_fd, MPPA\_RX\_SET\_MATCH, match);$
14: $mppa\_aiocb\_t\ Anew\_portal\_aiocb[1] = \{\textbf{MPPA\_AIOCB\_INITIALIZER}$
15:                                  $(Anew\_portal\_fd,\ Anew, sizeof(Anew[0]) * SIZE)\};$
16: $\textbf{mppa\_aiocb\_set\_trigger}(Anew\_portal\_aiocb, CLUSTER\_COUNT);$
17: $status| = \textbf{mppa\_aio\_read}(Anew\_portal\_aiocb);$
18: $// - -\textbf{LAUNCHING\_SLAVES} - -//$
19: $char\ arg0[10], arg1[10];$
20: $const\ char\ *argv[] = arg0, root\_sync, A\_portal, Anew\_portal, 0;$
21: for $int\ rank = 1 \to CLUSTER\_COUNT$ do
22:     $sprintf(arg0, "\%d", rank);$
23:     $status| = \textbf{mppa\_spawn}(rank, NULL, cluster\_executable, argv, 0);$
24: end for
25: // Wait for the cluster portals to be initialized.
26: $status| = \textbf{mppa\_read}(root\_sync\_fd, \&dummy, sizeof(dummy));$
27: // Distribute slices of array A over the clusters.
28: for $int\ rank = 0 \to CLUSTER\_COUNT$ do
29:     $status| = \textbf{mppa\_ioctl}(A\_portal\_fd, MPPA\_TX\_SET\_RX\_RANK, rank);$
30:     $status| = \textbf{mppa\_pwrite}(A\_portal\_fd, (A + rank * SIZE\_LOCAL) - (NX\_LOCAL * 2),$
31:     $sizeof(float) * SIZE\_LOCAL, 0);$
32: end for
33: // Wait for the cluster contributions to arrive in array |Anew|.
34: $status| = \textbf{mppa\_aio\_wait}(Anew\_portal\_aiocb);$
35: $return\ status < 0;$
```

The communication among cluster via links (*NoC*) is implemented by using of *channel*. Similar to the use of *portals*, *channels* must be initialized using one path per channel (see $C0\_to\_C1\_channel$ in Algorithm 2).

On the other hand, the synchronization is implemented by using of *sync*. They are used to guarantee that some resources are ready to be used or cluster are ready to start computing (for instance, see *mppa_ioctl* in Algorithm 2, 3, 4 and 5).

In order to minimize the number of transfers among main and local memory (*SM* approach) as well as among clusters through links (*NoC* approach), the matrix is divided into rectangular sub-blocks (Figures 3.1 and 3.2). In particular, the distribution of the workload and communication implemented in the *NoC* approach avoid multi-level routing, connecting each of the cluster with its adjacent clusters via a direct link.

Although, the ghost cells strategy is usually used for communication in distributed memory systems [11], we have used this strategy in Kalray processor to avoid race conditions among each of the sub-blocks assigned to each clusters. The use of ghost cells consists of replicating the borders of all immediate neighbors blocks. These ghost cells are not updated locally, but provide stencil values when updating the borders of local blocks. Every ghost cell is a duplicate of a piece of memory located in neighbors nodes. To clarify, Figures 3.1 and 3.2 illustrate a simple scheme for our interpretation of the ghost cell strategy applied to both approaches, *SM* and *NoC*, respectively.

Algorithm 3 *Shared Memory* CLUSTER pseudocode.

```
int mainCLUSTER(int argc, char * argv[])
$int\ i, j, status, rank = atoi(argv[0]);$
$const\ char\ *root\_sync = argv[1], *A\_portal = argv[2], *Anew\_portal = argv[3];$
$float\ A[SIZE\_LOCAL], Anew[SIZE\_LOCAL]; long\ long\ slice\_offset;$
$slice\_offset = sizeof(float) * (CHUNK * NX\_LOCAL+$
                              $((rank - 1) * (CHUNK - 1) * NX\_LOCAL));$
// Each cluster contributes a different bit to the root_sync mask.
$long\ long\ mask = (long\ long)1 << rank;$
//− − OPENING_PORTAL − −//
$int\ root\_sync\_fd =$ mppa_open$(root\_sync, O\_WRONLY);$
$int\ A\_portal\_fd =$ mppa_open$(A\_portal, O\_RDONLY);$
$int\ Anew\_portal\_fd =$ mppa_open$(Anew\_portal, O\_WRONLY);$
//− − PREPARE_FOR_INPUT − −//
$mppa\_aiocb\_tA\_portal\_aiocb[1] =$
        MPPA_AIOCB_INITIALIZER$(A\_portal\_fd, A, sizeof(A));$
$status| =$ mppa_aio_read$(A\_portal\_aiocb);$
//− − UNLOCK_MASTER − −//
$status| =$ mppa_write$(root\_sync\_fd, \&mask, sizeof(mask));$
// Wait for notification of remote writes to local arrays |A|.
$status| =$ mppa_aio_wait$(A\_portal\_aiocb);$
//− − JACOBIANCOMPUTE − −//
$jacobi(A, Anew, NX\_LOCAL, NY\_LOCAL);$
// Contribute back local array Anew into the portal of master array Anew.
$status| =$ mppa_pwrite$(Anew\_portal\_fd, \&Anew[NX\_LOCAL],$
                              $sizeof(Anew) - sizeof(float) * 2 * NX\_LOCAL, slice\_offset);$
$mppa\_exit((status < 0)); return\ 0;$
```

Figure 3.1 graphically illustrates the strategy followed by the *SM* approach. It consists of dividing the matrix into equal blocks which are sent from main memory to local memory. To avoid race condition, each of the blocks includes 2 additional rows (gray and white rows in Figure 3.1) which correspond to the upper and lower adjacent rows of the block. These additional rows work as ghost-cell, which are only used in local memory. The blocks transferred from local memory to global memory (Figure 3.1-right) do not include these additional rows (ghost rows).

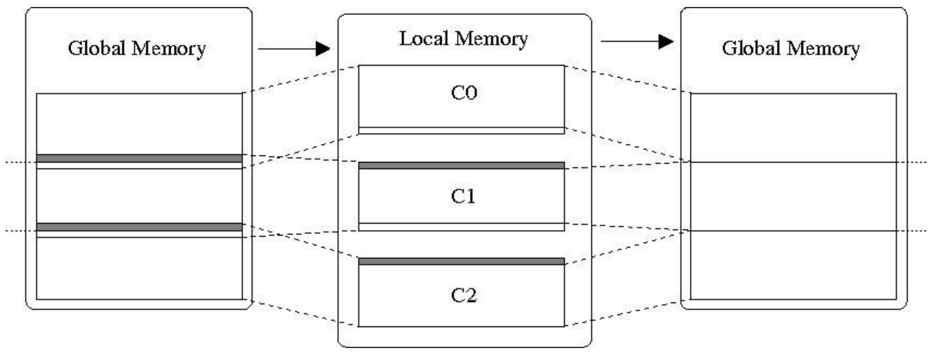

Fig. 3.1. *Master (Global Memory) ↔ Slave (Local Memory) Communication.*

Otherwise the communication among global and local memory is not necessary in the *NoC* approach. The master (IO code) is only used for synchronizing. The synchronization is necessary at the beginning and at the end of each *Master* code. I/O core and the rest of cores in each of the clusters must be also synchronized. In particular the synchronization between IO core and computing cores ($I/O- > C1$ *sync* section in Algorithm 5)

Algorithm 4 *NoC* I/O pseudocode.

```
const char $*global\_sync = "/mppa/sync/128:1"$;
const char $*IO\_to\_C0\_sync = "/mppa/sync/0:2"; \ldots$
const char $*C0\_to\_C1\_channel = "/mppa/channel/1:1/0:1"; \ldots$
static const char $*exe[CLUSTER\_COUNT] = \{"mainCLUSTER0",$
                $"mainCLUSTER1", \ldots\}$;
int mainIO(int argc, const char *argv[])
// Global sync.
int ret, global_sync_fd = mppa_open(global_sync, O_RDONLY);
long long match = $-(1 << CLUSTER\_COUNT)$;
mppa_ioctl(global_sync_fd, MPPA_RX_SET_MATCH, match));
// − −IO_TO_C#_SYNC − −//
int IO_to_C0_sync_fd = mppa_open(IO_to_C0_sync, O_WRONLY);
int IO_to_C1_sync_fd = mppa_open(IO_to_C1_sync, O_WRONLY); ...
// − −LAUNCHING_SLAVES − −//
for int $i = 0 \to CLUSTER\_COUNT$ do
    mppa_spawn(i, NULL, exe[i], argv, 0);
end for
// Wait for other clusters to be ready.
mppa_read(global_sync_fd, NULL, 8);
// Write into I/O− > C# sync to unlock C# cluster.
mask = 1; mppa_write(IO_to_C0_sync_fd, &mask, sizeof(mask));
mppa_write(IO_to_C1_sync_fd, &mask, sizeof(mask)); ...
// − −WAITING TO THE END OF CLUTERS EXECUTION − −//
for int $i = 0 \to CLUSTER\_COUNT$ do
    ret = mppa_waitpid(i, &status, 0); mppa_exit(ret);
end for
```

is necessary to guarantee that there are no cluster reading into channels before the corresponding cluster has opened the channel. After computing the Jacobi method in each of the clusters, some rows of the local blocks must be transferred to/from adjacent clusters. The first row computed (white upper row C1 in Figure 3.2) must be transferred to the upper adjacent cluster (C0) to be stored in the last row. Also, the last row computed (gray lower row C1 in Figure 3.2) must be transfered to the lower adjacent cluster (C2) to be stored in the first row. This pattern must be carried out in all clusters except the first and last clusters where a lower number of data-transfers is necessary.

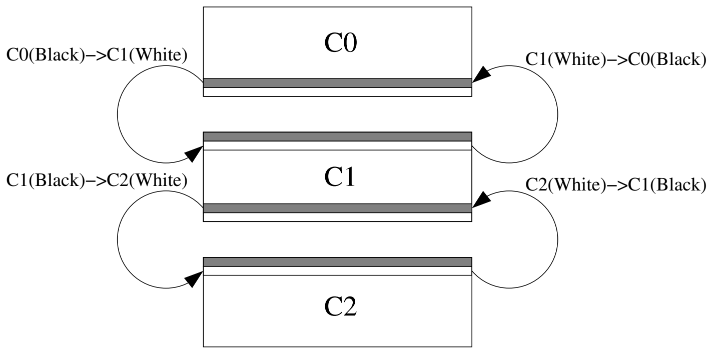

FIG. 3.2. *Pipeline (Bus) Communication.*

4. Performance Study.

4. Performance Study. In this section, we analyze deeply both approaches, *SM* and *NoC*, focusing on communication, synchronization and computing separately. In order to find/focus on the performance of both approaches, we have used a relatively small problem which can be fully stored in local memory.

Algorithm 5 *NoC* CLUSTER pseudocode

```
int mainCLUSTER1(int argc, char *argv[])
float A[SIZE_LOCAL], Anew[SIZE_LOCAL];
// Open all the resources needed for transfers.
//Global sync.
int global_sync_fd = mppa_open(global_sync, O_WRONLY);
// C1-> C2 channel.
int channel0_fd = mppa_open(C1_to_C2_channel, O_WRONLY);
// C2-> C1 channel.
int channel1_fd = mppa_open(C2_to_C1_channel, O_RDONLY);
// C1-> C0 channel.
int channel2_fd = mppa_open(C1_to_C0_channel, O_WRONLY);
// C0-> C1 channel.
int channel3_fd = mppa_open(C0_to_C1_channel, O_RDONLY);
// I/O-> C1 sync.
int IO_to_C1_sync_fd = mppa_open(IO_to_C1_sync, O_RDONLY);
long long match = -(1 << 1/* We sync only with I/O cluster */);
mppa_ioctl(IO_to_C1_sync_fd, MPPA_RX_SET_MATCH, match)
// Write into global sync to unlock I/O cluster.
long long mask = 1 << mppa_getpid();
mppa_write(global_sync_fd, &mask, sizeof(mask))
// --WAIT_FOR_IO_TO_C1_SYNC--//
mppa_read(IO_to_C1_sync_fd, NULL, 8);
// --CLUSTERS COMMUNICATION--//
// Write data for cluster 0.
mppa_write(channel0_fd, &A[NX_LOCAL * (NY_LOCAL - 2)], sizeof(float) * NX_LOCAL);
// Read data from C0.
mppa_read(channel1_fd, A, sizeof(float) * NX_LOCAL);
// Read data from C2.
mppa_write(channel2_fd, &A[NX_LOCAL], sizeof(float) * NX_LOCAL);
// Write data for cluster 2.
mppa_read(channel3_fd, &A[NX_LOCAL * (NY_LOCAL - 1)], sizeof(float) * NX_LOCAL);
mppa_exit(0);
```

Next we present the commands used to compile and launch both approaches: Compiling lines:

```
k1-gcc -O3 -std=c99 -mos=rtems io.c -o io_app -lmppaipc
k1-gcc -O3 -std=c99 -fopenmp -mos=nodeos cluster.c -o cluster -lmppaipc
k1-create-multibinary --cluster cluster --boot=io_app -T multibin
```

Launching line:

```
k1-jtag-runner --multibinary multibin --exec-multibin=IODDR0:io_app
```

The communication among I/O and computing cores in the *NoC* approach is more complex and it is in need of a higher number of synchronizations. This causes a higher execution time with respect to the *SM* approach, being almost 2.5× bigger (Figures 4.1 and 4.2). Note that we use a different vertical scaling in each of the graphics illustrated in Figures 4.1 and 4.2. Despite of the overhead caused by a higher number of synchronizations, the use of the NoC interconnection makes the *NoC* approach (Figure 4.2) about 55× faster than the *SM* approach.

As expected the time consumed for computing the Jacobi method is equivalent in both approaches. The time consumed by synchronization, communication and computing in the *NoC* approach is more balanced than in the *SM* approach. This can be beneficial for future improvements, such as asynchronous communication.

Finally, we analyse the performance in terms of GFLOPS. First, we compute the theoretical FLOPS for the Jacobi computation. The variant used in this study performs six flops per update (Algorithm 1). Therefore, the theoretical FLOPS is equal to the elements of our matrix multiplied by six.

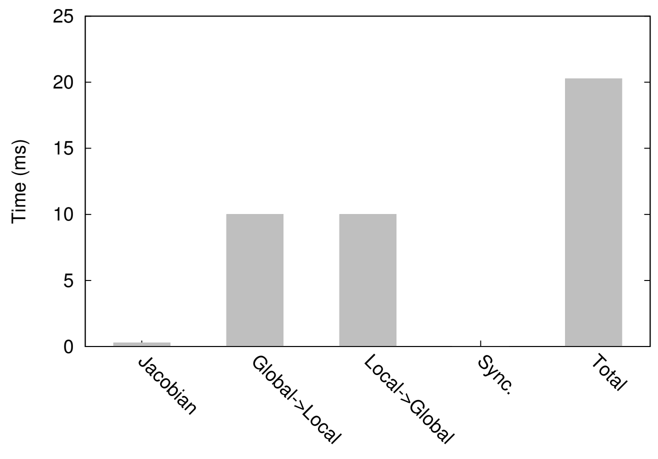

Fig. 4.1. *Time consumption for the* SM *approach.*

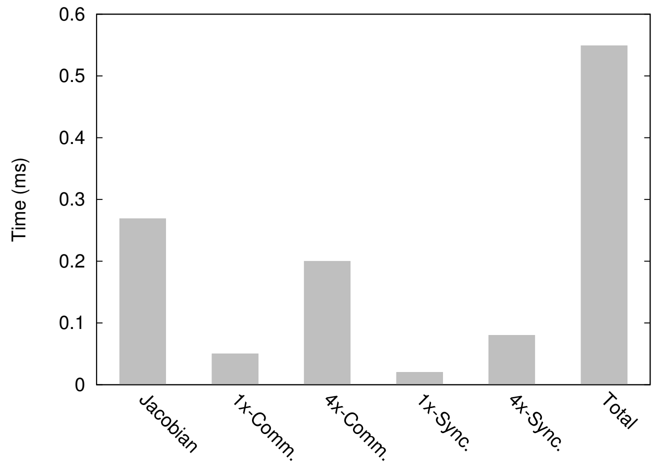

Fig. 4.2. *Time consumption for the* NoC *approach.*

In order to evaluate the overhead of each of the strategies, first, we show the GFLOPS achieved by the Jacobi computation without the influence of the synchronization and communication (see *Jacobian* in Figure 4.3). It achieves almost the peak of performance of our platform (*GFLOPS-Peak* in Figure 4.3). The computation of the Jacobian method is exactly the same in both approaches (*SM* and *NoC*). Next, we include the overhead of the communication. Although both approaches present a fall in performance when taking into account the time consumed by the communication, the fall shown by the *NoC* approach is not so dramatic as the overhead suffered by the *SM* approach (Figure 4.3).

The software development kit provided by Kalray allow us to measure the power consumption of our applications. This is done via this command:

$$k1-power\ --k1-jtag-runner\ --multibinary\ multibin\ --exec-multibin = IODDR0 : io\_app$$

Executing our binary using *k1-power* we obtain the power achieved in terms of Watts. The average power achieved by the *NoC* approach is about $8.508W$, while the *SM* approach achieves an average of $5.778W$ in every execution. This is almost a 50% more power when executing the *NoC* approach. However, the reduction in

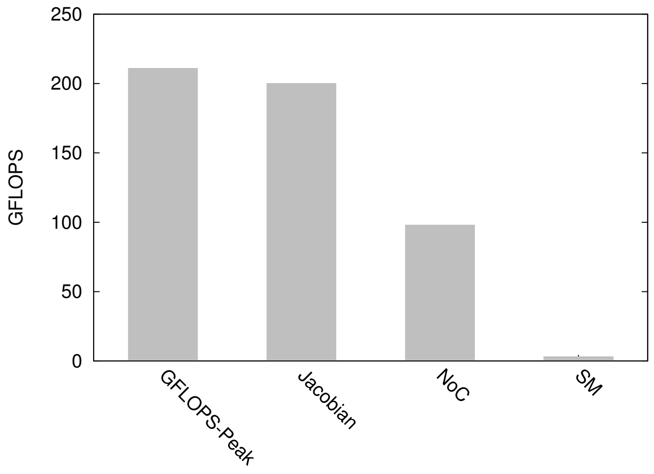

FIG. 4.3. *GFLOPS achieved by both approaches.*

execution time obtained by the NoC approach has an important impact on the overall power consumed. Joules are computed by following the next expression:

$$Joules = Watts \times Time$$

obtaining an overall consumption about $0.0047J$ and $0.16J$ for the NoC and the SM approaches respectively. This is a 96% less of power consumed by the NoC approach.

5. Conclusions and Future Work.

Embedded many-core architectures such as Kalray have emerged as a new HPC platform to deal with the problem of the excessive power consumption.

In this work, we have presented two different approaches to implement the communication among the processing elements of the Kalray architecture. Both approaches implement a ghost-cell strategy to avoid race conditions among the different blocks assigned to each of the processing elements (clusters). This strategy has been adapted to the particular features of our embedded processor and approaches, *SM* and *NoC*, to minimize the number of transfers.

Although the communication via shared memory is more habitual and easier to implement on many-core architectures, the particular features of the Kalray architecture, in particular the communication via Message-Passing on NoC connection, offers a much faster alternative. Although, the use of NoC consumes more power, the reduction in time makes this approach more efficient in terms of power consumption.

We plan to investigate other problems and more efficient strategies for memory management and data distribution, such as the overlapping of communication and computing via asynchronous transfers. In particular, the NoC approach could take advantage of the asynchronous communication as the time consumed by its major steps is balanced.

Acknowledgments. This research has been supported by EU-FET grant EUNISON 308874, the Basque Excellence Research Center (BERC 2014-2017) program by the Basque Government, the projects founded by the Spanish Ministry of Economy and Competitiveness (MINECO): BCAM Severo Ochoa accreditation SEV-2013-0323, MTM2013-40824 and TIN2015-65316-P. We acknowledge Research Center in Real-TIme & Embedded Computing Systes - CISTER for the provided resources.

REFERENCES

[1] S. A. Kalray. *Mppa accesscore posix progamming reference manual.* 2013.
[2] B. Dupont de Dinechin, R. Ayrignac, P.-E. Beaucamps, P. Couvert, B. Ganne, P. Guironnet de Massas, F. Jacquet, S. Jones, N. Morey Chaisemartin, F. Riss, and T. Strudel. *A clustered manycore processor architecture for embedded*

and accelerated applications. In *IEEE High Performance Extreme Computing Conference, HPEC 2013, Waltham, MA, USA, September 10-12, 2013*, pages 1–6, 2013.

[3] B. Dupont de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti. *Guaranteed services of the noc of a manycore processor.* In Proceedings of the 2014 International Workshop on Network on Chip Architectures, NoCArc '14, Cambridge, United Kingdom, December 13-14, 2014, pages 11–16, 2014.

[4] B. Dupont de Dinechin, D. van Amstel, M. Poulhiès, and G. Lager. *Time-critical computing on a single-chip massively parallel processor.* In Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014, pages 1–6, 2014.

[5] M. Dev Gomony, B. Akesson, and K. Goossens. *Coupling tdm noc and dram controller for cost and performance optimization of real-time systems.* In 2014 Design, Automation Test in Europe Conference Exhibition (DATE), pages 1–6, March 2014.

[6] T. Goubier, R. Sirdey, S. Louise, and V. David. Σ*c: A programming model and language for embedded manycores.* In Algorithms and Architectures for Parallel Processing - 11th International Conference, ICA3PP, Melbourne, Australia, October 24-26, 2011, Proceedings, Part I, pages 385–394, 2011.

[7] M. Monchiero, G. Palermo, C. Silvano, and O. Villa. *Exploration of distributed shared memory architectures for noc-based multiprocessors.* Journal of Systems Architecture, 53(10):719 – 732, 2007. Embedded Computer Systems: Architectures, Modeling, and Simulation.

[8] P. Valero, J.L. Sánchez, D. Cazorla, and E. Arias. *A gpu-based implementation of the mrf algorithm in itk package.* The Journal of Supercomputing, 58(3):403–410, 2011.

[9] P. Valero-Lara. *Accelerating solid–fluid interaction based on the immersed boundary method on multicore and gpu architectures.* The Journal of Supercomputing, 70(2):799–815, 2014.

[10] P. Valero-Lara, F.D. Igual, M. Prieto-Matías, A. Pinelli, and J. Favier. *Accelerating fluid–solid simulations (lattice-boltzmann & immersed-boundary) on heterogeneous architectures.* Journal of Computational Science, 10:249–261, 2015.

[11] P. Valero-Lara and J. Jansson. *LBM-HPC an open-source tool for fluid simulations. case study: Unified parallel C (UPC-PGAS).* In Cluster Computing (CLUSTER), 2015 IEEE International Conference on, pages 318–321. IEEE, 2015.

[12] P. Valero-Lara, P. Nookala, F.L. Pelayo, J. Jansson, S. Dimitropoulos, and I. Raicu. *Many-task computing on many-core architectures.* Scalable Computing: Practice and Experience, 17(1):32–46, 2016.

[13] P. Valero-Lara and F.L. Pelayo. *Full-overlapped concurrent kernels.* In Architecture of Computing Systems. Proceedings, ARCS 2015-The 28th International Conference on, pages 1–8. VDE, 2015.

[14] P. Valero-Lara, A. Pinelli, J. Favier, and M. Prieto-Matias. *Block tridiagonal solvers on heterogeneous architectures.* In Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, ISPA '12, pages 609–616, Washington, DC, USA, 2012. IEEE Computer Society.

[15] P. Valero-Lara, A. Pinelli, and M. Prieto-Matias. *Fast finite difference poisson solvers on heterogeneous architectures.* Computer Physics Communications, 185(4):1265 – 1272, 2014.

[16] D. M. Young. *Iterative solution of large linear systems.* 2003. Unabridged republication of the 1971 edition [Academic Press, New York-London, MR 305568].

[17] H. Zhao, O. Jang, W. Ding, Y. Zhang, M. Kandemir, and M.J. Irwin. *A hybrid noc design for cache coherence optimization for chip multiprocessors.* In Proceedings of the 49th Annual Design Automation Conference, DAC '12, pages 834–842, New York, NY, USA, 2012. ACM.

Edited by: Dana Petcu
Received: Dec 16, 2016
Accepted: Mar 26, 2017

DOI 10.12694/scpe.v18i2.1288
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 161–175. http://www.scpe.org © 2017 SCPE

IMPACT OF PROCESS ALLOCATION STRATEGIES IN HIGH PERFORMANCE CLOUD COMPUTING ON AZURE PLATFORM

HANAN A. HASSAN[†] AYA I. MAIYZA[‡] AND WALAA M. SHETA[§][¶]

Abstract. Nowadays, there is an increasing demand in the High-Performance Computing (HPC) community to make use of different public cloud service provider. The question of which cloud provider is superior for a certain application and usage configuration is very important for the successful deployment of HPC application on the cloud. In this paper, we evaluate the performance of HPC applications on Microsoft Azure cloud platform using the well-known NAS parallel benchmarks. These benchmarks are considered as examples of general scientific HPC applications to test the communication performance. Different process allocation strategies are performed in terms of MOPS and Speedup. Our results show that allocating one process per instance achieves higher scalability at the expense of the cost. The results compared with the same results with the same experiments in Amazon platform. We found that Azure platform has better shared-memory communication performance than Amazon platform. In contrast, Amazon is superior to Azure platform in terms of Ethernet bandwidth.

Key words: Cloud Computing, High Performance Computing, Message Passing Interface, Nas Parallel Benchmark

AMS subject classifications. 68M14, 68M20

1. Introduction.

Cloud computing is a revolutionary technology based on sharing resources to provide diverse types of e-services to end users. Cloud computing provisioning is basically based on virtualization techniques to obtain an abstract view of physical resources with the same interfaces. It offers several benefits such as the possibility of using same physical resources for different users and runtime environments simultaneously [1]. Additionally, dealing with Virtual Machine (VM) is much easier in management, maintenance setup, and administration. Moreover, its cost is optimized as payment is based on pay-as-you-go (PAYG) model. In PAYG model, the charge is only calculated for the actual usage of the physical resources.

The cloud service models are categorized as (i) Software as a Service (SaaS), which completely deals with applications such as iCloud. (ii) Platform as a Service (PaaS), which provides an environment for application development framework such as operating Systems. (iii) Infrastructure as a Service (IaaS), which provides the computing hardware virtual Infrastructure and virtual storage. Recently HPC as a Service (HPCaaS) is considered as one of the promising services on the cloud. The cloud computing offers advantages to HPC applications users including virtualization benefits, resources scalability, abstraction of cluster setup cost and time, and energy consumption [1, 2].

Nevertheless, there are challenges for running HPC application on cloud because of the performance diversity and poor network performance [2]. There are several IaaS public cloud Providers such as Amazon Elastic Cloud Compute (EC2), Microsoft Windows Azure, and Rackspace. These clouds are suitable for running HPC application on them [3]. Microsoft's Azure cloud provides an amiable development environment (.NET, SQL Server, and Visual Studio) with a wide group of capabilities for developers to construct robust applications over those [4, 5]. It provides powerful storage and resources through hardware level virtualization. In addition, it is possible to build virtual parallel clusters easily. This motivated us to study the performance of HPC on the Microsoft Azure Cloud.

In our previous work [6], the Ethernet and shared memory communication were measured in terms of bandwidth and latency on Azure platform. Furthermore, the scalability impact of running HPC applications on Azure platform was assessed using up to 128 cores [6]. As an extension, the impact of different process allocation strategies is studied in terms of performance and cost. The scalability is evaluated and assessed using up to 512 cores as well.

Furthermore, this paper is aimed to assess the performance of running HPC applications on different types of cloud platform, identify challenges that affect the performance, and presenting a set of possible solutions for

[†]Informatics Research Institute, The City of Scientific Research and Technological Applications (hali@srtacity.sci.eg).
[‡]Informatics Research Institute, The City of Scientific Research and Technological Applications (amaiyza@srtacity.sci.eg).
[§]Informatics Research Institute, The City of Scientific Research and Technological Applications (wsheta@srtacity.sci.eg).
[¶]CECS Department, University of Louisville, USA.

performance improvement. Scalability is evaluated by NAS Parallel Benchmarks (NPB) kernels [7] in terms of MOPS (millions operation per second) and Speedup, which is calculated by dividing serial over parallel execution time. The experiments were performed on one cluster of virtual machines on Microsoft's Azure cloud (A10 size). A10 size has specifications close to one of Amazon Elastic Compute Cloud (EC2) specifications. Then, the results are compared with the study performed on Amazon EC2 [8].

Intuitively, involving all the VM resources (virtual cores) to run a certain HPC process should result into the best cost/performance scenario (value verses money). This is called normal process allocation strategy (NPAS). However, this selection is not going to be constantly the best in terms of performance. So, this paper reveals that the strategy of allocating one process per instance (1ppi) is significantly much more expensive. However, it is able to achieve higher performance in several applications [8].

The NAS offers a set of HPC kernels, which varies in their types of being either communication or computation intensive. They also differ in several data types such as integer and floating point performance [3].

The rest of the paper is organized as follows: Section 2 shows the related work. Section 3 represents the evaluation methodology, which contains experiments configurations. Section 4 analyzes the experimental result, and compares between two process allocation strategies optimized for performance and cost. In addition, the section discusses the obtained result. Finally, section 5 recapitulates the conclusion and represents future work.

2. Related Work. This section is oriented toward two dominant issues: (1) Performance evaluation of running HPC on the public cloud providers using well-known benchmarks (2) The possibility of running HPC applications using a virtualized cluster on cloud platforms and evaluating its performance compared with traditional HPC platform.

Previous research of running HPC application on cloud platform focused on the performance of VM compared to physical HPC cluster. The result was not useful because of poor network performance, multi-tenancy, and resources disproportionate [2, 9, 10, 11]. Nevertheless, the performance improvement of running HPC applications on cloud is still a research target to get accepted performance compared to traditional HPC. In spite of its performance, it is suitable for some applications, which required scale up/down resources and can easily drop them out when the task is done at the end [12, 13, 14].

In general, several well-known benchmarks have been used to analyze the performance of parallel computing. But in particular, they use their own applications to compare the performance between different techniques or running in different infrastructures. NAS parallel benchmarks (NPB) was used in [2, 3, 6, 8, 15, 16] and High-Performance Linpack benchmark (HPL) was used in [5, 16, 17] to compare between the performance of using traditional HPC and of using HPC hosted on public or private cloud.

Akioka and Muraoka [15] used Amazon EC2 as an alternative to HPC environment using NPB benchmarks and HPL benchmark. The performance and cost efficiency were evaluated. Then, a performance comparison was performed between Amazon cloud instance and a physical machine with NPB benchmarks using the serial version (NPB-SER). They found that running HPC on cloud is not suitable for some kinds of applications, such as performance critical applications.

Gupta et al. [2] presented the performance analysis and the tradeoffs of cost for HPC applications with/without virtualization using NPB benchmarks. They found that running HPC on the cloud is considered more cost-effective for non-communication-intensive applications such as embarrassingly parallel and tree-structured computations over high processor count and for communication-intensive applications over low processor count and HPC-optimized clusters are outstanding for the rest.

Roloff et al. [3] provided an inclusive analysis of three important phases of HPC on the cloud: deployment, performance, and cost-efficiency. This study was performed on three public clouds, namely Amazon Elastic Compute Cloud, Rackspace and Microsoft Azure, as well as a traditional cluster using NPB benchmark. They showed that virtualized HPC had a better performance and cost efficiency than the cluster, up to 27% and 41%, for several benchmarks.

Strazdins et al. [16] presented performance results for two benchmarks and two large scientific applications running in private VM cluster, an Amazon HPC EC2 cluster and traditional HPC environment. They used the MPI micro-benchmark, the NAS Parallel macro-benchmarks, the UK Met Office's MetUM global climate model and the chaste multi-scale computational biology code. They succeeded to build application codes in a pure HPC environment and replicate these into VMs which ran on private VM cluster and on public HPC cluster running on Amazon's EC2.

Expósito et al. [8] analyzed the major performance bottlenecks in HPC application running on Amazon EC2 Cluster Computing platform. They compared between two flavors of instances CC1 and CC2. First, they evaluated the communication performance on ten gigabits Ethernet network and shared memory using Intel MPI Benchmarks suite (IMB) on three HPC message-passing middleware, namely MPICH2, OpenMPI, and FastMPJ. They found that CC2 instances are somewhat better point-to-point communication performance and provide more computational power. Second, they appraised the scalability of representative message-passing codes using up to 512 cores using NAS Parallel Benchmarks (NPB) kernels. They found that CC2 instances have poorer scalability than CC1 instances for communication-intensive applications and CC1 instances are more effective than CC2 instances in terms of cost. Third, they achieved higher scalability using one process per instance only. Finally, several levels of parallelism have been used to achieve most scalable and cost-effective using hybrid technique between message-passing with multithreading.

Hassani et al. [12] implemented the MPI version for parallel Radix sort and evaluated its performance on Amazon cloud infrastructure. Then, they compared the result with traditional HPC platform. Parallel HPC applications use considerably Message Passing Interface (MPI) [18, 19, 20]. Previously, they implemented parallel Radix sort programming using MPI, Pthreads, and OpenMP and evaluated the performance using benchmarking test. Their experimental results proved that MPI is better than others at the time of parallel sorting, so they implemented the parallel Radix sort on Cloud. They spotted considerable improvement in speed up and scaled up for up to 8 virtual nodes. Cloud response rate was more 20 percent parallel efficiency than the traditional HPC cluster.

Zhang et al. [5] implemented a virtual HPC environment on Azure cloud for hydrological applications. They presented a case study on groundwater uncertainty analysis in Heihe River Basin. They proved that the Azure cloud can outperform traditional HPC infrastructure and can be useful for hydrological researchers to improve computing efficiency of the model.

Belgacem and Chopard [17] had successfully connected Amazon EC2 based cloud cluster situated in USA, to a private HPC cluster (Scylla) located at their university in Switzerland. They ran a large distributed multiscale application on this hybrid HPC infrastructure. They ran a distributed multiscale application using the software developed in their MAPPER project [21]. They found that the distributed computing performance is less than the monolithic one. Their analysis showed that this low performance because of the long-distance between the two continents thus resulting in very poor network performance.

Hassan et al. [6] evaluated the performance of the HPC applications on Azure cloud in terms of MOPS and Speedup. The performance was tested under several configurations of cluster sizes. In addition, the performance of point-to-point communication between processes was assessed in terms of bandwidth and latency. They found that the best performance was achieved using only a single VM (shared memory communication model), especially with IS and FT NPB-kernels.

Cala et al. [13] presented their experience in porting a genomics data processing pipeline (Next-Generation Sequencing Genetic test) from an existing scripted implementation deployed on a traditional HPC, to a workflow-based design deployed on the Microsoft Azure public cloud. Most of HPC systems used sophisticated MPI-based algorithms, but the current NGS tools do not require it. Rather an effective data splitting techniques are used to convert the Big Data to an embarrassingly parallel problem. They found that using public cloud provided several benefits such as speed, scalability, flexibility and cost-effectiveness for NGS Genetic test.

Mohrehkesh et al. [14] presented a feasibility study using cloud resources for Image Guided Neurosurgery (IGNS). They computed the deformable registration or non-rigid registration (NRR) of brain MR images using their local private cloud (Turing cluster) at Old Dominion University, as well as Microsoft Azure (Microsoft HPC pack) [22]. Using these clusters, they analyzed more than 6TB of images. They evaluated the accuracy of registration by speculative execution, the overhead time of running jobs on the Azure cloud and cost comparison of running jobs on a private versus public cloud. Their results indicated that the public cloud provides practical and cost-effective means for a hospital that supports IGNS solutions. Moreover, the accuracy of NRR could be improved up to 57% using cloud resources.

This diversity of research efforts to run HPC systems on cloud computing using various types of applications, benchmarks, and technologies. Table 2.1 summarizes the previous work of running HPC cluster on public cloud

Table 2.1
The relevant related work.

Authors Name	Benchmarks and HPC Applications	Libraries	Platform	Evaluated Metric	Recommendation for Running HPC on Cloud
S. Akioka, Y. Muraoka [15]	- NPB-MPI 3.3 (Class C) - HPL 2.0 Benchmark	- MPICH2 1.2	- Amazon EC2 - Physical machine (NPB-SER)	- Mops/second - Gflops - Cost	Not suitable for performance critical application
A. Gupta, M. Dejan [2]	- NPB-MPI (Class B) - NAMD - NQueens	- MPI	- Open Cirrus (Private cloud) - Eucalyptus (Private cloud) - Taub (Private HPC)	- Speedup - Cost	Cost-effective for non-communication-intensive application up to high processor count and for communication-intensive application up to low processor count
E. Roloff, M. Diener, A. Carissimi, P.O.A.Navaux[3]	- NPB-MPI 3.3.1 (Class B)	- MPI - OpenMPI	- Amazon EC2 - Rackspace - Microsoft Azure - Traditional HPC cluster	- Normalized average execution time - Cost efficiency	Higher performance and cost efficiency than the cluster up to 27% and 41% respectively for several benchmarks
P.E. Strazdins, J. Cai, M. Atif, J. Antony [16]	- MPI microbenchmark - NPB-MPI 3.3 (Class B) - memory intensive simulationapplications (Chaste 2.1 & UM 7.8)	- MPI	- Private VM cluster - Amazon EC2 - Traditional HPC cluster	- Bandwidth (MB/s) - Latency (s) - Speedup	Ability to successfully build large scale HPC applications on Cloud
R.R. Expósito, G.L. Taboada, S. Ramos, J. Tourio, R. Doallo [8]	- IMB - NPB-MPI 3.3 (Class C) - NPB-MZ 3.3.1 (Class C)	- MPICH2 1.4.1 - OpenMPI 1.4.4	- Amazon EC2	- Bandwidth (Gbps) - Latency (μs) - MOPS - Speedup - Cost efficiency (USD/GOPS)	Achieving higher scalability using 1ppi and using hybrid technique between message-passing with multithreading
R. Hassani, M. Aiatullah, P. Luksch [12]	- parallel Radix sort programming	- MPI	- Amazon EC2 - Dedicated HPC platform	- Execution times (sec)	MPI is better than others at the time of parallel sorting. Cloud response rate was more 20 percent parallel efficiency than the pure HPC cluster

Table 2.1 (continued)

Authors Name	Benchmarks and HPC Applications	Libraries	Platform	Evaluated Metric	Recommendation for Running HPC on Cloud
M.B. Belgacem, B. Chopard [17]	- large distributed multiscale application	- OpenMPI 1.4.5	- Hyprid Amazon EC2 cluster with private HPC cluster (Scylla)	- Execution times (sec)	The distributed computing performance is less than the monolithic one
H.A. Hassan, S.A. Mohamed, W.M. Sheta [6]	- IMB NPB-MPI 3.3 (Class C)	- MPICH2 1.4.1 - OpenMPI 1.4.4	- Microsoft Azure	- Bandwidth (Gbps) - Latency (s) - MOPS - Speedup - Cost efficiency (USD/GOPS)	Achieving higher performance when running entirely on a single VM (shared memory communication model), especially IS kernel and FT kernel.
J. Caa, E. Marei, Y. Xu, K. Takeda, P. Missier [13]	- Workflow-based application (Next-Generation Sequencing)		- local HPC cluster - Microsoft Azure	- Response time (Hours) - Throughput (samples/day) - Relative processing effectiveness (%) - Cost per sample	Public cloud could provide benefits such as speed, flexibility, scalability and cost-effectiveness for NGS.
S. Mohrehkesh, A. Fedorov, A. B. Vishwanatha, F. Drakopoulos, R. Kikinis, N. Chrisochoides [14]	- Image Guided Neurosurgery (IGNS)	- MPI - Microsoft MPI (built-in)	- private cloud (Turing cluster) - Microsoft Azure (Microsoft HPC pack)	- Registration accuracy (%) - Azure overhead time (s) - Monthly Cost (1000 US$)	Public cloud provides practical and cost-effective solution more than private cluster.

Table 3.1
Cloud instances Specifications [8, 23].

Cloud Platform	Instance Size	Cores	CPU Type	RAM	RAM Type
Azure	A10	8	Intel Xeon E5-2670 @ 2.6 GHz	56 GB	DDR3-1600 MHz
Amazon	cc1	8	Intel Xeon X5570 Nehalem @ 2.93 GHz	23 GB	DDR3

providers. These publications are classified according to the benchmarks types, evaluated metrics, used libraries, platform, and their main result. Concludes that the best cloud provider depends on the type and behavior of the application, in addition to the intended usage scenario. In our experiments, HPC system is assessed on Microsoft Azure Cloud using NAS Parallel Benchmarks (NPB) under different process allocation scenarios.

3. Evaluation Methodology. This section presents the used configurations and benchmarks for evaluating the performance of using HPC on Azure platform.

3.1. Experimental Configuration. Two experiments are conducted to evaluate the performance of a cluster of 64 VMs (512 cores) on Microsoft Windows Azure [23]. Table 3.1 shows the used Azure A10 instance specifications. MPICH and OpenMPI are used as a standard implementations of the Message Passing Interface (MPI) with GNU compiler. These two well-known implementations are used in our experiment with releases MPICH2 1.4.1 [24] and OpenMPI 1.4.4 [25].

The performance is evaluated using the most communication-intensive kernels of the NPB benchmarks on Azure platform (using 16 'A10' instances). The performance metrics are Million Operation Per Second (MOPS) and Speedups.

There are two process allocation strategies in our experiments. The first one is called normal process allocation strategy (NPAS). All cores in one instance must be used before extending the cluster with another instance. In this case, we used 8 process per instance (8ppi). The number of used A10 instances is the total number of cores divided by 8 cores per instance. For example, for 64 cores, eight instances were used.

On the other hand, the second process allocation is called expensive process allocation strategy (EPAS), which is used to improve the performance of the previous configuration. The performance is evaluated using the same kernels of NPB using only one process per instances (1ppi) until 64 cores (64 instances), then posteriorly 2, 4, and 8 processes per instance to reach 128, 256, and 512 cores (64 instances). For examples, 32 instances are used for 32 cores using 1ppi and 64 instances are used for 128 cores using 2ppi. This because, there is a limitation on the maximum used number of cores (512 cores).

For the two configurations, the cost is calculated to compare between them. The performance evaluated using the two implementations of message passing interface MPI and OpenMPI. It is evaluated using the result of NPB kernels as productivity in terms of USD per Giga Operations Per Second (USD/GOPS). In this paper, the results are the average of 6 measurements for NPB kernels on the explained experiment configuration. Finally, a comparison between our study and the previous study on Amazon platform [8] is performed taking in account a comparable instances specifications as shown in Table 3.1.

3.2. NAS Parallel Benchmarks (NPB). NPB benchmarks are a well-accepted and well-known HPC benchmark suites. In these experiments, the NPB-MPI version 3.3 [26] benchmarks are used to evaluate our two process allocation strategies on Azure platform. Most communication-intensive kernels have been chosen, namely CG, FT, IS and MG, with class C. Table 3.2 shows an overview of each used benchmark kernel [7, 26].

4. Experimental Results and Discussion. This section presents an analysis of the performance and scalability of HPC NPB kernels on Azure platform under two different process allocation strategies, using the selected benchmarks and their representative kernels, which were described before in sect. 3.2. In addition, there is a subsection for comparing the results with Amazon platform.

4.1. Normal process allocation strategy (NPAS). Fig. 4.1 (dotted line) shows the performance of CG, FT, IS and MG using up to 512 cores on Azure platform (hence, using 64 A10 Azure VMs). The used performance metrics are MOPS (left graphs), and Speedup (right graphs). The used number of VMs is the total number of cores divided by 8 cores (number of cores for A10 VM).

TABLE 3.2
NAS Parallel Benchmarks used Kernels [26].

NPB Kernel	Properties	Problem Size (Class C)	Message Size (Class C)
CG (conjugate gradient)	irregular memory access and communication	150000	146.5 Kbyte
FT (Fourier Transform)*	discrete 3D fast Fourier Transform, all-to-all communication	512^3	128 Mbyte
IS (Integer Sort)	random memory access	2^{27}	128 Mbyte
MG (Multi-Grid)**	Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive	512^3	128 Mbyte

*denote the most communication-intensive
**denote the least communication-intensive

The execution of CG kernel on different cluster configurations show that the maximum performance is obtained by using four VMs (32 cores). The performance degrades significantly with the higher number of cores due to the communication overhead in the virtual network. The performance can be improved significantly by using a faster network such as InfiniBand and hardware based virtualization technology such as single-root I/O virtualization SR-IOV [27, 28, 29].

An optimum performance of IS kernels is achieved with 8 cores (1 VM). FT kernels show similar maximum performance with 8 cores (1 VM) and 128 cores (16 VMs). MG kernels have a maximum performance at a cluster of 256 cores (32 VM) using OpenMPI. NPB kernels performance shows that the evaluated applications obtain better results when running completely on only one VM (intra-communication) using up to eight cores, because of the higher scalability and performance of shared memory communications. However, when using greater than single VM, the kernels poorly perform, because of the network virtualization overhead. IS kernels obtain the poorest scalability.

CG kernel is characterized by multiple point-to-point data movements. The best speedup value is 7.3, which is achieved by using 4 VMs (32 cores). The performance is significantly dropped from that value on as it has to depend on ten gigabits Ethernet communications, due to the network virtualization bottleneck. FT kernel achieves a limited scalability on A10 VM. The best speedup value is approximately 5.6, which is achieved by using 8 and 128 cores. IS kernel is a communication-intensive code whose scalability is greatly affected by point-to-point communication start-up latency. Thus, this kernel obtains its highest results when using only one VM due to the high performance of shared memory transfers. It suffers a critical slowdown when using a cluster of VMs. Finally, MG kernel is a limited scalability. The speedup values are close to each other. The best speedup value is approximately 10, which is achieved by using 256 cores for MPICH2.

4.2. Impact of expensive process allocation strategy (EPAS). This set of experiments aims to analyze the impact of expensive process allocation strategy (1ppi) on HPC kernel performance. In the previous experiments, 8 processes per instance strategy are used to maximize the CPU utilization as Azure type A10 instance has 8 cores. This intuitive assumption is not necessarily valid in the case of virtualization, as there is a significant amount of CPU resources consumed to deal with the hypervisor and manage virtualized CPUs according to the type of the hypervisor used. Therefore, we investigate how the number of allocated processes will affect the overall performance in a virtualized HPC environment.

In this experiment, the performance of the NPB kernels using only one process per instance strategy (1ppi) is compared with the normal process allocation strategy (8ppi). This layout can be expensive as we use only one core per instance for each allocated process. For example, we use just 8 instances to allocate 8 processes on the cluster, with each process running on one instance (1ppi). Hence, 32 VMs are used to provide 32 cores with 32 processes on the cluster instead of 256 processes in the normal setting of experiments as the case of sect. 3. Two, four, and eight processes per instance are used to reach 128, 256, and 512 cores; respectively, because of limited resources (64 instances).

Fig. 4.1 (solid line) shows a significant improvement of both MOPS and speedup for all the four kernels. CG kernel's speedup has increased to 8 times its corresponding value of NPAS in MPICH2 environment. Moreover,

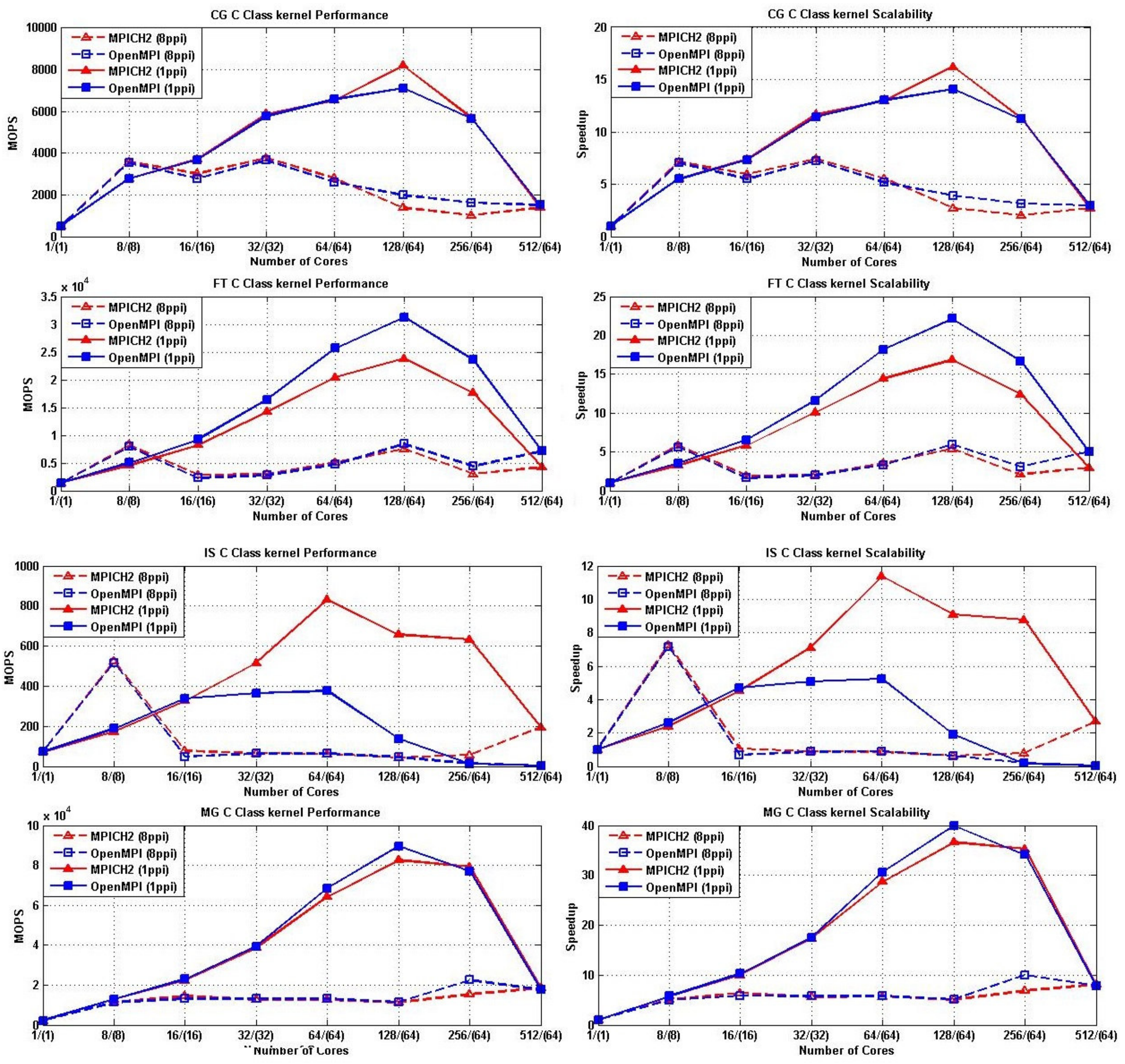

FIG. 4.1. *NPB kernels performance and scalability on A10 Azure instances. For 1ppi: horizontal axis presents x/(z), where x is number of cores and z is number of instances. For 8ppi: horizontal axis presents x, where x is number of cores (number of instances=(number of cores/8))*

the optimum configuration is achieved at 128 cores, i.e. a cluster of 64 instances running a total of 128 processes, instead of (32 cores) a cluster of 4 instances running 32 processes. FT kernel's speedup has improved to more than 4 times compared with NPAS in OpenMPI environment with 128 processes. IS kernel has progressed in MPICH2 more than OpenMPI environment. Its best speedup is achieved at 64 cores. MG kernel achieves a considerable betterment using EPAS. Its best speedup value obtained 40 by using 64 VMs (128 cores).

Fig. 4.2 compares the performance of NPB kernels using 8 cores on Azure platform using NPAS (hence, using one A10 VM) and EPAS (hence, using 8 VMs). It's clear to show that NPAS always gives better performance than EPAS at 8 cores for CG, FT and IS kernels. That is because intra-VM communication (shared memory)

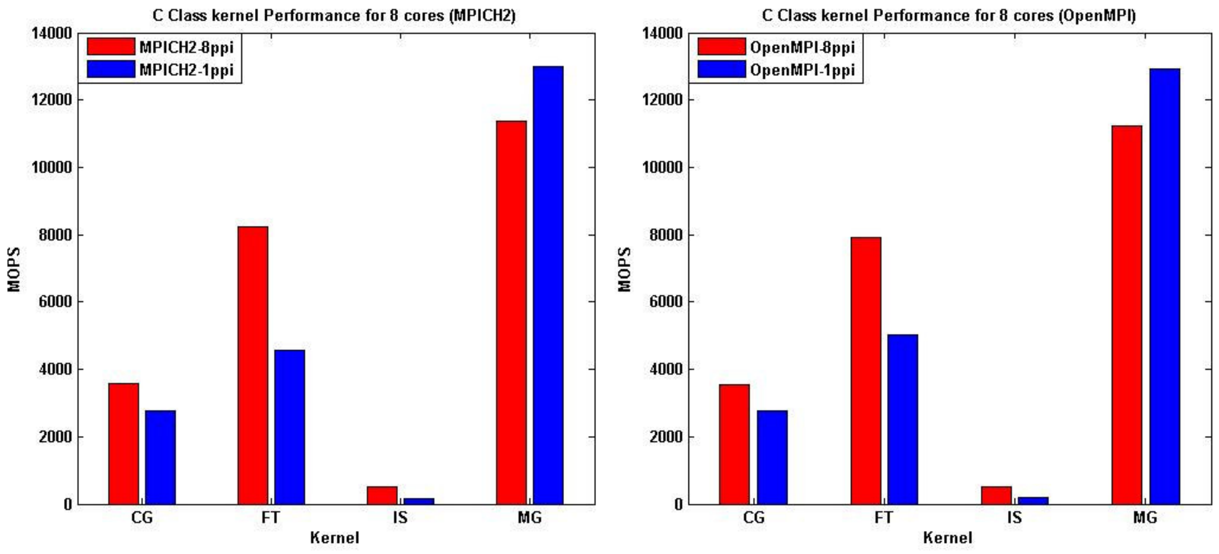

FIG. 4.2. *NPB kernels performance on A10 Azure instances for 8 Cores*

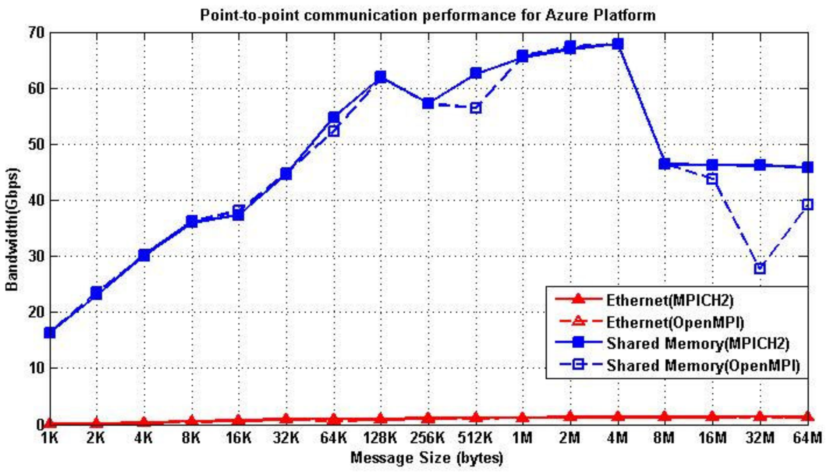

FIG. 4.3. *Point-to-point communication performance on Azure platform [6]*

is better than inter-VM communication (Ethernet) on that level of the cluster as shown in Fig. 4.3. But as the number of cores increases in the cluster using EPAS, its performance supersedes its corresponding NPAS. In contrast, MG kernel achieves higher performance using EPAS, because MG is least communication-intensive as mention in Table 3.2.

4.3. Cost Analysis. Using public cloud infrastructure like Azure, the cost has to be taken into account. Fig. 4.4 and Fig. 4.5 present the productivity in terms of USD per GOPS (Giga Operations per Second) of the already evaluated NPB kernels. This metric depends on the total number of used cores. The cost of each instance is 1.16 $/hour for each A10 instance [23]. When the cost behavior of kernels is inversely proportional to the cluster size, it means that it is better to use a large cluster rather than a small one from both perspectives of performance and cost. On the other hand, if the cost behavior is directly proportional to the cluster size, this indicates that the solution is getting expensive as the cluster is enlarged and therefore there is no need to use a large cluster. An efficient operation cost is obtained at a minimal value of the productivity curve which

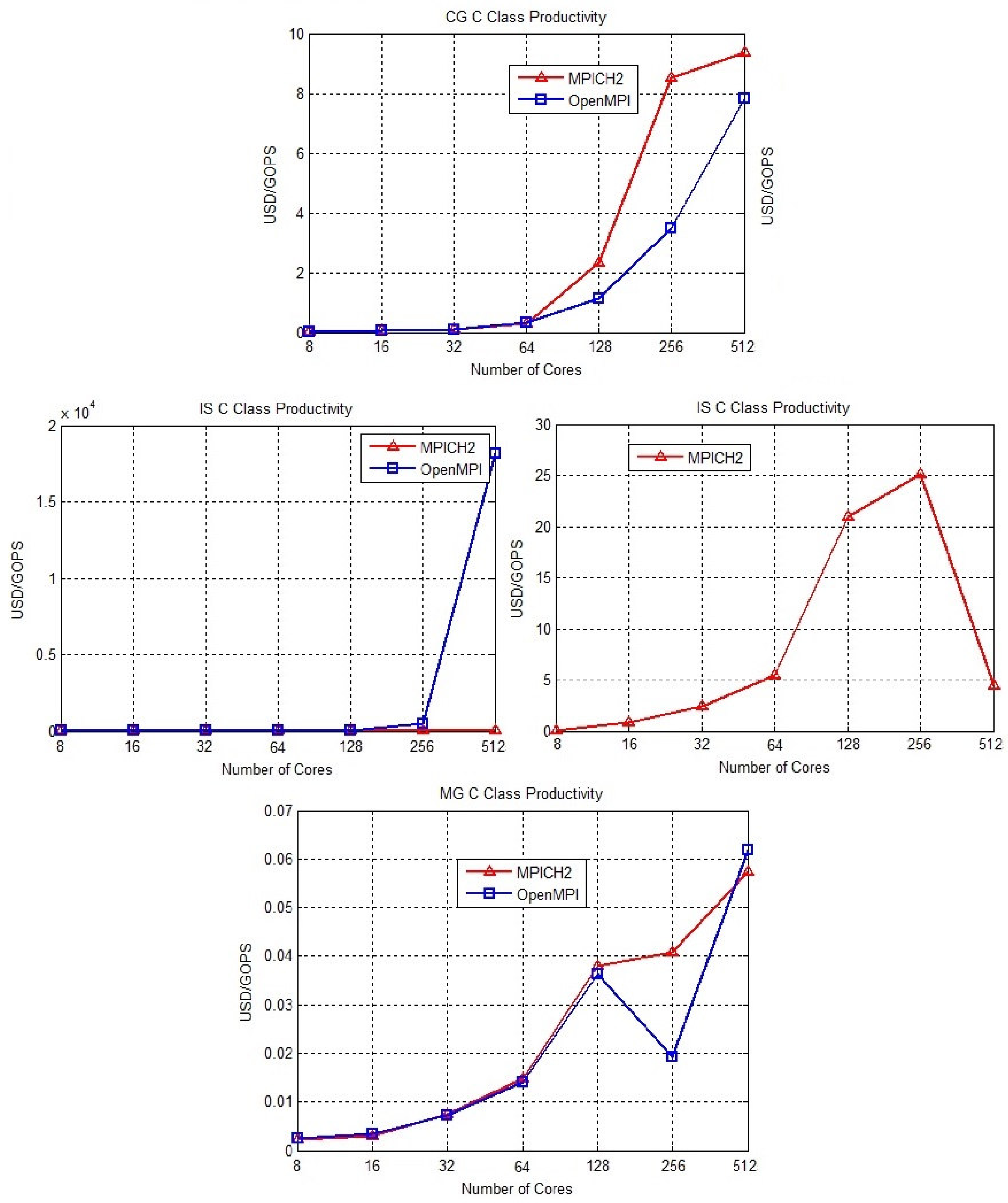

FIG. 4.4. *NPB kernels productivity on Azure instances (Normal process allocation strategy).*

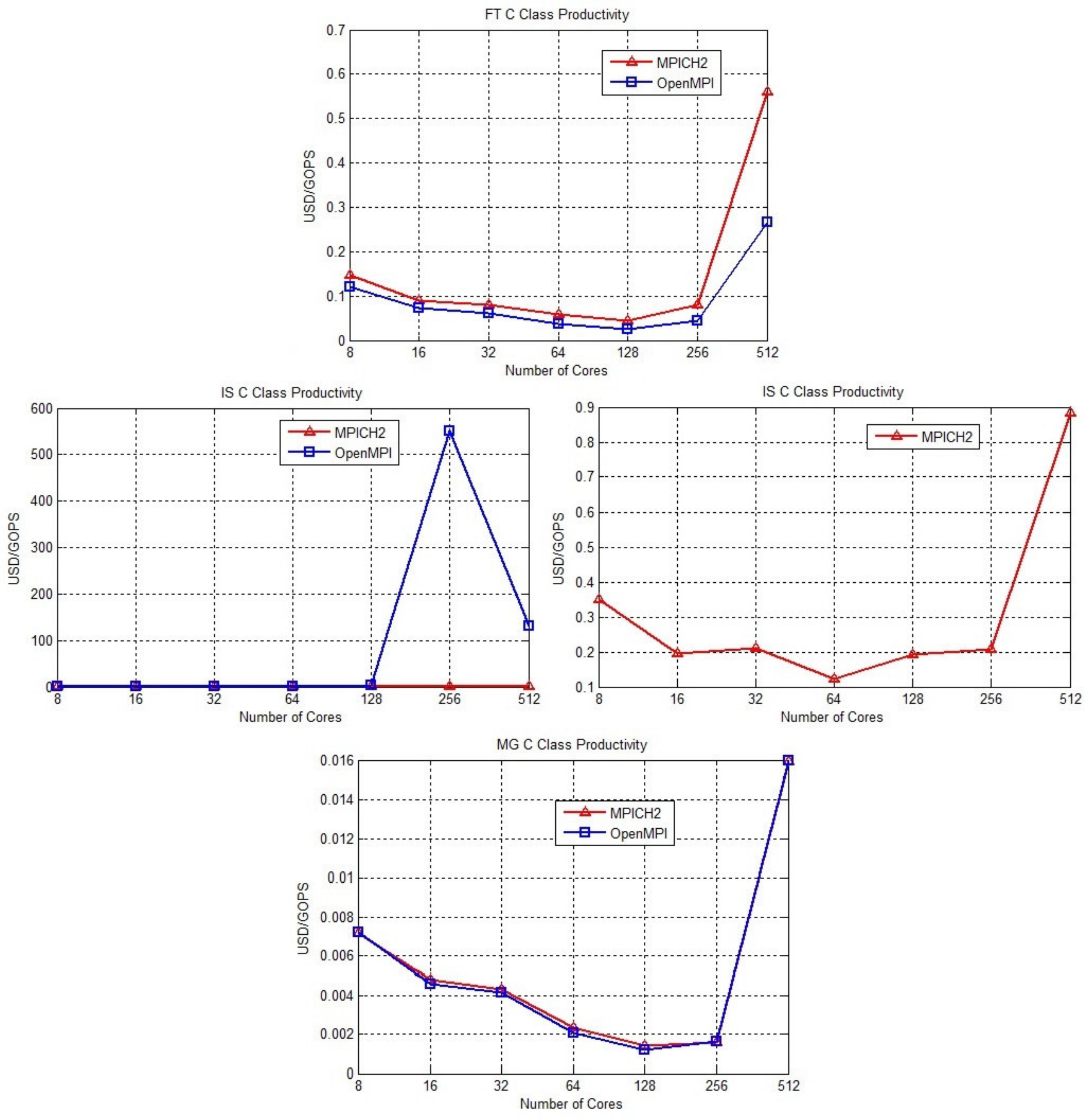

FIG. 4.5. *NPB kernels productivity on Azure instances (Expensive process allocation strategy).*

indicates a low cost per operation versus expensive operations at the maximum value of the curve.

Regarding the first strategy (NPAS), the cost behaviors of CG, IS, and MG kernels are directly proportional to the cluster size as shown in Fig. 4.4. The only exception is valid for IS Kernel using 512 cores (MPICH2 Environment) and MG Kernel using 256 cores (OpenMPI Environment). The reason for these exceptions is the performances of these two configurations, in terms of MOPS and Speedup, increase as shown in Fig. 4.1.

Fig. 4.5 shows the cost behavior in USD/GOPS for EPAS. An optimum value of the productivity curve of FT and MG kernels are achieved at 128 cores which are compatible with the results obtained from Fig. 4.1.

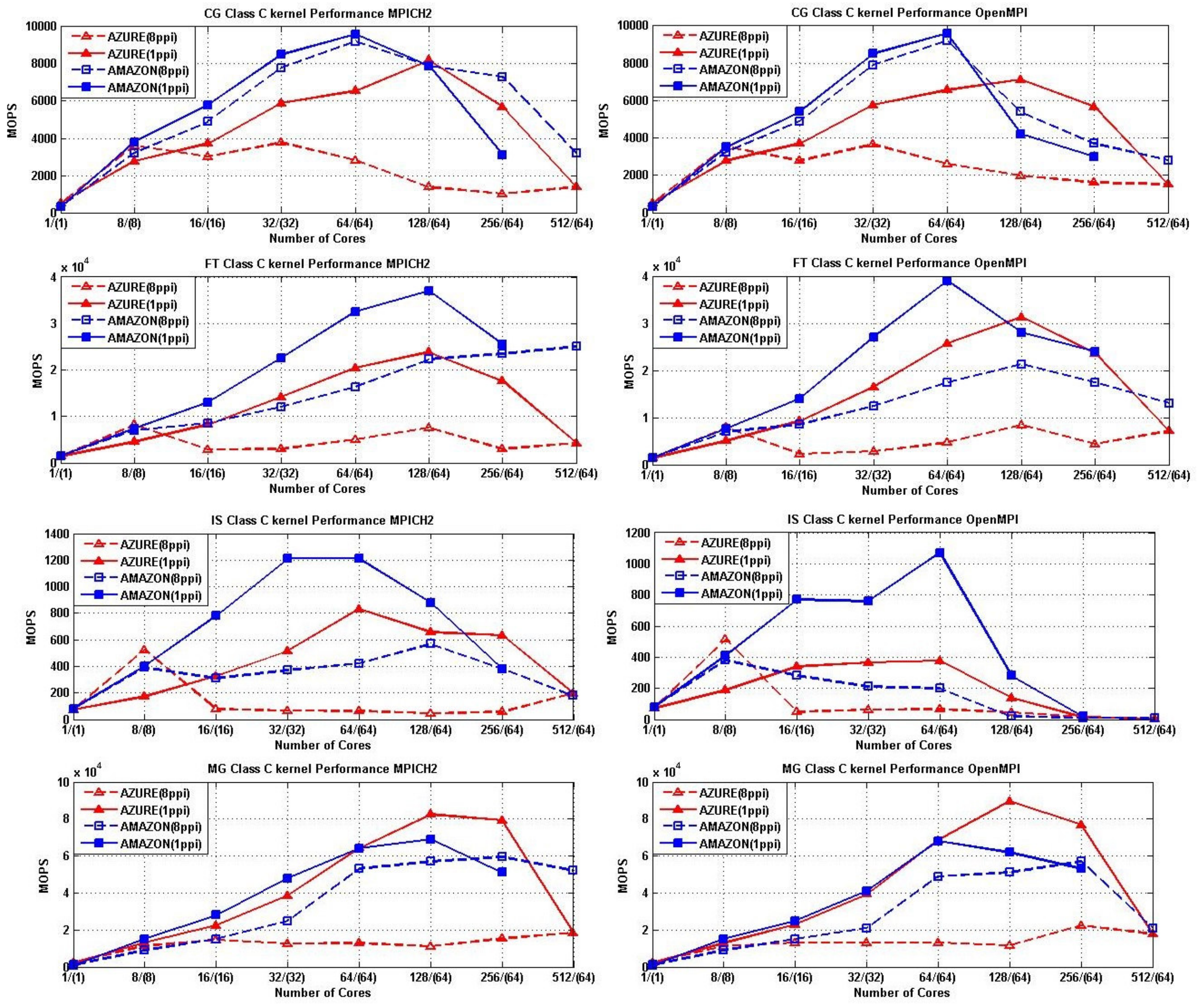

FIG. 4.6. *NPB kernels performance and scalability on A10 Azure instances and cc1 Amazon instances [8]. For 1ppi: horizontal axis presents x/(z), where x is number of cores and z is number of instances. For 8ppi: horizontal axis presents x, where x is number of cores (number of instances=(number of cores/8))*

Then, they dramatically increase using 512 cores, because MOPS and Speedup decrease with the same cluster size (64 instances). For IS kernel, the optimum solution can be achieved at 64 cores with MPICH2, because the cost increases and the performance decreases when the number of cores increases more than 64 cores as shown in Fig. 4.5.

4.4. Comparison between Azure and Amazon Performance. This section compares between the performance of our evaluated experiments on Azure platform and Amazon platform [8] using two different libraries, namely MPICH2 and OpenMPI.

Fig. 4.6 shows the performance of NPB kernels using up to 512 cores on Azure and Amazon platform (hence, using 64 of A10 instances in Azure and using 64 of cc1 instances on Amazon). The performance metric is MOPS only, which is available in the compared paper. The left and right graphs figure out the performance with MPICH2 and OpenMPI libraries, respectively. Dotted and solid lines show the improvements of process allocation strategy from NPAS (8ppi) to EPAS (1ppi), respectively.

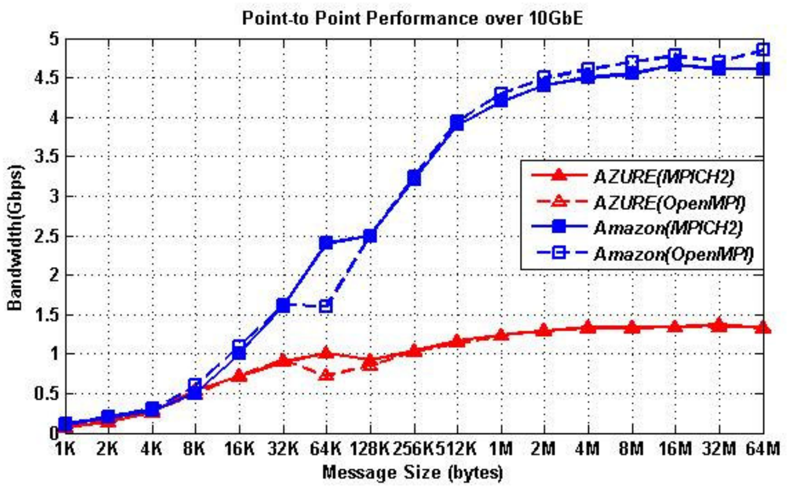

FIG. 4.7. *Point-to-point communication performance on Azure [6] and Amazon instances [8].*

4.4.1. Normal process allocation performance comparison (NPAS). For 8ppi, the overall performance is better in Amazon than Azure platform for all evaluated kernels except using 8 cores. This variance could be due to the different hypervisors which used in the two platforms. Amazon platform uses Xen paravirtualization guests for improving network and disk performance [8], whereas Azure platform uses Hyper-V virtual environments [30].

Fig. 4.7 compares between the bandwidth of Amazon and Azure platform. The bandwidth results of Azure [6] and Amazon [8] platform are performed with the same instance configurations using the same HPC messaging middleware such as MPICH2 1.4.1 [24] and OpenMPI 1.4.4 [25]. A communication is performed through an Ethernet network link using a ping-pong test. It is clear to see that the bandwidth of Amazon platform is better than Azure platform whether MPICH2 or OpenMPI. This observation considers another reason explaining why the overall performance of Amazon platform is better than Azure platform.

In contrast, the only configuration that gives better performance in Azure than Amazon platform is achieved using 8 cores allocated in one instance for all kernels.

Fig.4.8 compares the point-to-point performance of message-passing transfers in the intra-communication between Azure [6] and Amazon [8] platform, where data transfers are implemented on shared memory (hence, without accessing the network hardware). These results obtained with the Intel MPI Benchmarks suite (IMB). Table 3.2 shows the message size of the four evaluated kernels. For CG kernel, the shared memory message-passing communication performance is better using Azure platform over Amazon platform by 1 and 7 Gbps using MPICH2 and OpenMPI, respectively, where the message size for CG kernel is 146.5 Kbytes approximately for C class as mentioned in Table 3. For the three other kernels, the message size is 128 Mbytes. Unfortunately, the measured shared memory performance was performed in [8] and [6] up to 64 Mbytes message size. So, extrapolation is calculated to expect a value for intra-communication performance for this message size. The approximated shared memory communication performance is better using Azure platform over Amazon platform by 14 and 21 Gbps using MPICH2 and OpenMPI, respectively. These results explain why the performance is better using one Azure instance than Amazon instance for the four evaluated kernels with 8 core configuration.

4.4.2. Expensive process allocation performance comparison (EPAS). For EPAS, the most configurations using Amazon give better performance than Azure platform, nevertheless the overall improvement (from using NPAS to EPAS) is greater in Azure for CG and MG kernels than Amazon platform. That is because these two kernels are the least communication-intensive kernels. Amazon has the best performance, except for CG and MG kernels using 128 and 256 cores, FT kernels using 128 cores (OpenMPI), and IS kernels using 256 cores (MPICH2).

There is another important observation appears in Fig. 4.6, all the performance results drop as the number

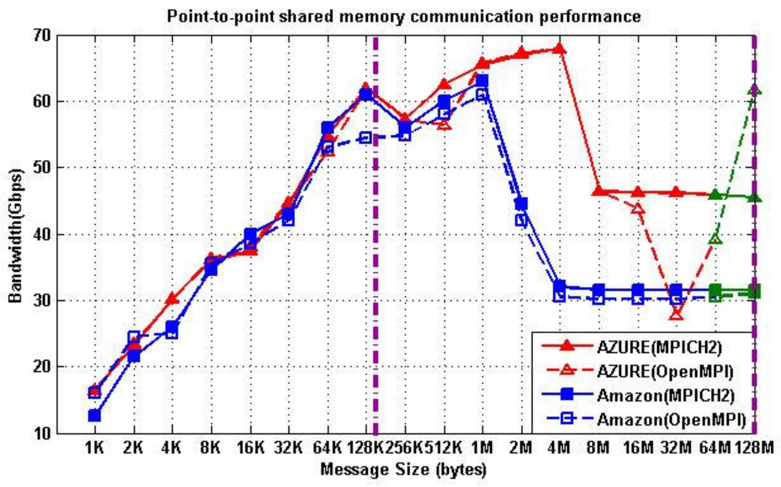

FIG. 4.8. *Point-to-point shared memory communication performance on Azure [6] and Amazon instances [8].*

of cores increase after using 64 or 128 cores. That is because 64 instances are used for 64, 128, 256 and 512 cores using 1, 2, 4, 8 processes per instances, respectively. These configurations are performed whether using Azure or Amazon platform. The performance degradation using Amazon platform is more than using Azure platform, because the shared memory bandwidth is better for Azure as previously explained in Fig. 4.6.

5. Conclusion and Future Work. The research and industry community would highly benefit from using HPC on the cloud. There is a high potential of running HPC on the cloud due to the capability of running large applications on powerful, scalable hardware without the need to physically have or maintain this hardware. Actually, there are several types of public cloud providers, allowing users to select the best provider for their needs. The best configuration for a certain application depends on its type and behavior, in addition to the usage scenario. In this paper, the performance of using HPC on Microsoft Azure cloud service is evaluated through NAS Parallel benchmarks. The scalability is assessed for representative message-passing kernels (NPB) using up to 512 cores for communication-intensive HPC application. Allocating only one process per instance helps HPC applications that are hosted on the cloud to get higher performance at the expense of cost.

This study helps HPC users to determine their own optimum configuration. HPC user could classify his application according to the presented NPB kernels, and then select the cluster size according to his budget and the nature of his application. In addition, it will be easy to select between Azure Cloud and Amazon EC2 platform depending on the deployed application.

For the future, it's intended to execute real HPC applications on Azure platform. The performance of another cloud platform could be compared with Azure and Amazon cloud too. Different process allocation strategies (2ppi and 4ppi) could be tested. Moreover, cost analysis could be performed using expensive process allocation strategy (EPAS) with Amazon platform. As a result, a comprehensive study could be presented in terms of performance and cost analysis between Azure and Amazon platform.

Acknowledgments. The authors are grateful for the partial support received by Microsoft to access Azure cloud computing facilities. We also would like to thank Mona Kashkoush and Shimaa Abdellatif Mohamed, from the Informatics Research Institute for their collaboration in conducting the experiments.

REFERENCES

[1] V. Mauch, M. Kunze and M. Hillenbrand, *High performance cloud computing*, Future Generation Computer Systems, 29, 2013, pp. 1408–1416.
[2] A. Gupta and D. Milojicic, *Evaluation of hpc applications on cloud*, in Open Cirrus Summit (OCS), 2011 Sixth, IEEE, 2011, pp. 22–26.

[3] E. Roloff, M. Diener, A. Carissimi, and P. O. A. Navaux, *High performance computing in the cloud: Deployment, performance and cost effciency*, in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on, IEEE, 2012, pp. 371–378.
[4] R. J. Dudley and N. A. Duchene, *Microsoft Azure: Enterprise Application Development*, Packt Publishing Ltd, 2010.
[5] G. Zhang, Y. Yao, and C. Zheng, *HPC Environment on Azure Cloud for Hydrological Parameter Estimation*, in Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on, IEEE, 2014, pp. 299–304.
[6] H. A. Hassan, S. A. Mohamed, and W. M. Sheta, *Scalability and communication performance of HPC on Azure Cloud*, Egyptian Informatics Journal, 2015.
[7] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, et al., *The NAS parallel benchmarks*, International Journal of High Performance Computing Applications, 5, 1991, pp. 63–73.
[8] R. R. Expósito, G. L. Taboada, S. Ramos, J. Touriño and R. Doallo, *Performance analysis of HPC applications in the cloud*, Future Generation Computer Systems, 29, 2013, pp. 218–229.
[9] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer and D. H. Epema, *Performance analysis of cloud computing services for many-tasks scientific computing*, Parallel and Distributed Systems, IEEE Transactions on, 22, 2011, pp. 931–945.
[10] R. Hassani, A. Fazely and P. Luksch, *Optimizing Bandwidth by Employing MPLS AToM with QoS Support*, in Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on, IEEE, 2012, pp. 104–108.
[11] R. Hassani and P. Luksch, *DMT: A new Approach of DiffServ QoS Methodology*, 2012.
[12] R. Hassani, M. Aiatullah, and P. Luksch, *Improving hpc application performance in public cloud*, in Networking, IERI Procedia, 10, 2014, pp. 169–176.
[13] J. Cała, E. Marei, Y. Xu, K. Takeda and P. Missier, *Scalable and efficient whole-exome data processing using workflows on the cloud*, Future Generation Computer Systems, 2016.
[14] S. Mohrehkesh, A. Fedorov, A. B. Vishwanatha, F. Drakopoulos, R. Kikinis and N. Chrisochoides, *Large Scale Cloud-Based Deformable Registration for Image Guided Therapy*, in Connected Health: Applications, Systems and Engineering Technologies (CHASE), 2016 IEEE First International Conference on, IEEE, 2016, pp. 67–72.
[15] S. Akioka and Y. Muraoka, *HPC benchmarks on Amazon EC2*, in Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, IEEE, 2010, pp. 1029–1034.
[16] P. E. Strazdins, J. Cai, M. Atif, and J. Antony, *Scientific Application Performance on HPC, Private and Public Cloud Resources: A Case Study Using Climate, Cardiac Model Codes and the NPB Benchmark Suite*, in Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, IEEE, 2012, pp. 1416–1424.
[17] M. B. Belgacem and B. Chopard, *A hybrid HPC/cloud distributed infrastructure: Coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications*, Future Generation Computer Systems, 42, 2015, pp. 11–21.
[18] R. Hassani, A. Malekpour, A. Fazely, and P. Luksch, *High performance concurrent multi-path communication for MPI*, Springer, 2012.
[19] R. Hassani and P. Luksch, *Scalable high performance computing in wide area network*, in High Performance Computing and Simulation (HPCS), 2012 International Conference on, IEEE, 2012, pp. 684–686.
[20] R. Hassani, G. Chavan and P. Luksch, *Optimization of Communication in MPI-Based Clusters*, in Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2014 International Conference on, IEEE, 2014, pp. 143–149.
[21] "Mapper project,", `http://www.mapper-project.eu/`
[22] "Microsoft HPC pack (windows HPC server),", `https://technet.microsoft.com/en-us/enus/library/cc514029.aspx`
[23] `http://azure.microsoft.com/blog/2015/03/05/new-a10a11-azure-compute-sizes/`
[24] "MPICH2 Website:", `http://www.mcs.anl.gov/research/projects/mpich2.org/`
[25] "OpenMPI Website:", `http://www.open-mpi.org/`
[26] `https://www.nas.nasa.gov/publications/npb.html`
[27] T. Pais Pitta de Lacerda Ruivo, G. B. Altayo, G. Garzoglio, S. Timm, H.-W. Kim, S.-Y. Noh, and I. Raicu, *Exploring Infiniband Hardware Virtualization in OpenNebula towards Efficient High-Performance Computing*, tech. report, Fermi National Accelerator Laboratory (FNAL), Batavia, IL (United States), 2014.
[28] N. Regola and J.-C. Ducom, *Recommendations for virtualization technologies in high performance computing*, in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 409–416.
[29] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, *High performance network virtualization with SR-IOV*, Journal of Parallel and Distributed Computing, 72, 2012, pp. 1471–1480.
[30] V. A. Gandhi and C. Kumbharana, *Comparative study of Amazon EC2 and Microsoft Azure cloud architecture*, International Journal of Advanced Networking & Applications, 2014.

Edited by: Dana Petcu
Received: Dec 26, 2016
Accepted: May 1, 2017

DOI 10.12694/scpe.v18i2.1289
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 2, pp. 177–193. http://www.scpe.org © 2017 SCPE

DR-SWDF: A DYNAMICALLY RECONFIGURABLE FRAMEWORK FOR SCIENTIFIC WORKFLOWS DEPLOYMENT IN THE CLOUD

KHADIJA BOUSSELMI[*], ZAKI BRAHMI[‡], AND MOHAMED MOHSEN GAMMOUDI[§]

Abstract. Workflows management systems (WfMS) are aimed for designing, scheduling, executing, reusing, and sharing workflows in distributed environments like the Cloud computing. With the emergence of e-science workflows, which are used in different domains like astronomy, life science, and physics, to model and execute vast series of dependents functionalities and a large amount of manipulated data, the workflow management systems are required to provide customizable programming environments to ease the programming effort required by scientists to orchestrate a computational science experiment. A key issue for e-science WfMS is how to deal with the change of the execution environment constraints and the variability and confliction of end users and cloud providers objectives for the execution of the same workflow or sub-workflow. They have to customize their management processes to insure the adaptability of the execution environment to the scientific workflows specificities, especially when dealing with large-scale (data, computing, I/O)-intensive workflows. In this paper, we propose a dynamically re-configurable framework for the deployment of scientific workflows in the Cloud (called DR-SWDF) that allows customizing the workflow deployment process according to a set of objectives and constraints of end users or cloud providers defined differently for the tasks or partitions of the same workflow. The DR-SWDF framework offers a K-means based algorithm that allows dynamically clustering the input workflows or sub-workflows in order to identify the most convenient techniques or algorithms to be applied for their scheduling and deployment in the cloud. The simulations results run on three examples large-scale scientific workflows show that our proposed framework can achieve better results than the use of a generic purpose approach.

Key words: workflow management system, scheduling, partitioning, provisioning, configuration, cloud computing, scientific workflows, K-means.

1. Introduction.

The Cloud Computing environment offers multiple advantages for hosting and executing complex applications such as scientific workflows. The elasticity asset of the Cloud Computing resources helps such workflows to dynamically provision their compute and storage resources. Scientific workflows (called also e-Science workflows) are used in different domains such as astronomy, life science, physics, to model and execute vast series of dependents functionalities and a large amount of manipulated data [6]. Scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the steps from dataset selection and integration, computation and analysis, to final data product presentation and visualization [16].

According to [17], the e-science workflow lifecycle, like traditional workflows, is defined by four phases, namely: the workflow composition and representation, the mapping of tasks to resources, the execution of the workflow and finally the recoding of metadata and provenance. These phases are usually included in a Scientific Workflow Management System (SWfMS) like Triana [24], Pegasus [25], and Taverna [23]. The goal of e-science workflow management systems is to provide a specialized programming environment to simplify the programming effort required by scientists to orchestrate a computational science experiment [17]. The SWfMS system supports the specification, modification, run, re-run, and monitoring of a scientific workflow using the workflow logic to control the order of executing workflow tasks [16].

A critical challenge of SWfMS for executing Scientific Workflows on the Cloud is how to efficiently allocate Cloud resources to the workflow tasks in order to optimize a predefined set of objectives. To do this, application schedulers employ different policies that vary according to the objective function: minimize the execution time, minimize cost, minimize the energy consumption, balance the load on resources used while meeting a fixed deadline or user budget constraints of the application, etc. In the Cloud Computing environment, the overall objective of task scheduling strategy is to guarantee the service-level agreements (SLAs) of allocated resources to make cost-efficient decisions for workload placement. Since task scheduling problem on Cloud Computing environments is NP-complete [19], several heuristics have been proposed to solve using Genetic Algorithms [38], Particle Swarm Optimization [39], Ant Colony Optimization [40], and Cat Swarm Optimization [5][6]. However, the problem is still challenging due to the dynamic nature of the Cloud Computing environment and to the variability and confliction of the defined scheduling objectives.

[*]Faculty of Sciences of Tunis, University of Tunis El Manar (khadija.bousselmi@gmail.com)
[‡]ISITCOM, University of Sousse (zakibrahmi@yahoo.fr)
[§]ISAMM, University of Mannouba, RIADI-GDL Laboratory, Tunisia (gammoudimomo@gmail.com)

The scheduling is particularly more challenging for large-scale scientific workflows as they may have thousands of tasks and data dependencies. Among the solutions, workflow partitioning is an approach to divide a workflow into several sub-workflows and then submit these sub-workflows to different execution sites (virtual clusters) [44]. Indeed, partitioning the original workflow into several fine-grained sub-workflows can ease the complexity of the workflow and increase its performances according to the partitioning objectives. The partitioning objectives can be varied and conflicting such as: optimizing the overall execution time of the workflow, minimizing the dependencies between the workflow partitions, minimizing the number of allocated resources, etc. For this reason, the problem of partitioning scientific workflows in the cloud computing environment is still an open research issue. Several approaches were proposed in the literature to partition scientific workflows according to one or multiple conflicting objectives [6][10][44].

Therefore, a major concern for a SWfMS is to be able to adapt his proposed features such as: the definition, scheduling, and resource provisioning to the scientific workflow's characteristics (computing, data or I/O intensiveness, number of tasks, input and output data size, etc.) and to the end users requirements and constraints (predefined deadline or budget, quality of service preferences, data placement constraints, etc.). The concept of adaptability has been introduced in the late 1990s and with the emergence of Adaptive Process Management (APM) systems. APM constitutes an evolutionary extension of the production workflow paradigm which intends to remedy the deficiencies related to dynamic process change [18]. According to [20], the adaptability of SWfMS can be defined at different layers, namely: the business context layer (defines the adaptability to a changing business context), the process layer (deals with changes to workflow models and their constituent workflow tasks), the resource layer (data model and organization changing) and the infrastructure layer (in case of system modifications or new technical settings).

In this work, we are interested in adaptable WfMS to the context modification as the scientific workflow can be considered as specific very complex workflows that are more difficult to be managed than traditional ones. Integrating a general-purpose WfMS in a specific business and organization context needs adaptation and reconciliation from both sides [20]. Dealing with context-different workflows with different requirements can impact the application performances of a workflow system. Thus, workflow systems should be prepared to adapt themselves to a range of different business and organization settings and also to a changing context [20]. Several SWfMS systems proposed in literature [23][24][25][26] offer adaptive management processes for the management and execution of scientific workflows in the Cloud by integrating the possibility of configuring these processes according to the user's requirements or the workflow's properties. However, local adjustments concerning a single task or a workflow partition are often neglected. Such local adjustments are very useful for a user to situate his/her work environment, including making decision in response to a special situation and on the basis of a variety of choices, reporting exceptional cases that are out of his/her responsibility, and so on [20]. More precisely, our research issue in this paper aims to dynamically configuring the workflow deployment process according to specific requirements of end users or cloud providers. We propose a dynamically configurable framework for the deployment of scientific workflows in the Cloud Computing environment inspired from the architecture of DR-FPGA (Dynamic Reconfigurable Field Programmable Gate Arrays) [8]. In our framework, we propose a K-means based algorithm that allows dynamically clustering the input scientific workflows in order to identify the most convenient techniques to apply for its deployment in the cloud. The proposed framework allows personalizing the deployment of the workflow at a fine-grained level (for specific tasks or partitions of the workflow) using different techniques at runtime and according to the input workflow's parameters, the user's requirements and the cloud provider's objectives. For instance, we consider only workflow partitioning, scheduling and resource provisioning as the main features of the SWfMS that can be dynamically configured; however, our framework is general purpose and can be extended to include more features.

This paper is an extension of the works originally reported in [5] and [6]. We use our proposed partitioning algorithm in [6] for the partitioning of scientific workflows and our proposed scheduling algorithm in [5] for their scheduling in the cloud.

The reminder of this paper is organized as follows. In Section 2, we present our mathematical model for scientific workflows deployment and, we introduce briefly the Dynamic Reconfigurable FPGA architecture. In Section 3, we present our proposed dynamic reconfigurable framework for scientific workflows deployment in the cloud. In Section 4, we exhibit our experimental setting and results. In Section 5, we present related works and finally in section 6. we conclude.

2. Background. In this section, we will present the problem formulation for scientific workflows scheduling and deployment in the cloud. Then, we will present briefly the Dynamic Reconfigurable FPGAs as the architecture of our proposed framework in this paper is inspired from it.

2.1. Problem statement. In this section we will focus on modeling the scientific workflows in the Cloud computing environment guided. We will start by modeling the execution environment of the Cloud, modeling scientific workflows, characterizing QoS metrics and their evaluation techniques for workflows, defining and assessing the energy consumption of the Workflow. Finally, we will formulate our configurable optimization problem of workflows scheduling in the Cloud that we will use when scheduling the workflow in our proposed approach.

2.1.1. Modeling the execution environment. For the execution environment, we use the same modeling as proposed in our previous paper [6]. Indeed, the execution environment of scientific workflows can be organized as following: We dispose of a set of t data servers $S = \{S_i | i = 1, ..., t\}$ from the Cloud Computing environment. Each data server S_i is composed of m computing resources (Virtual machines or VM) such as $S_i = \{VM_i^k | k = 1, ..., m\}$. The Cloud providers offer different kinds of VM instances, adapted to different types of tasks (I/O, intensive computing, memory). We define a virtual machine VM_i by the quadruplet (R_i, C_i, Q_i, P_i) where:

- $R_i = \{R_i^k | k = 1, ..., d\}$ is the vector of types of resources of a virtual machine VM_i such as the processor, the internal memory SSD, the network device ... with d the maximum number of resource types of VM_i .
- $C_i = \{C_i^k | k = 1, ..., d\}$ is the vector of capacities of the resources of the virtual machine VM_i.
- $Q_i = (T_{exe}, C, A, R...)$ is the vector of the metrics of quality of service used to evaluate the performance of VM_i, such as $T_{exe} = \sum_{i=1}^{k} Dt_{in_i} * T_u$, where T_u is the time needed to process an input data of a unit task. $C = T_{exe} * C_u$ where C_u represents the cost per time unit incurred by using the virtual machine VM_i . A is the availability rate of VM_i and R its reliability value.
- $P_i = \sum_{k=1}^{d} P_i^k$ represents the energy consumption of the virtual machine (in Watts), defined according to its resource types where P_i^k defines the energy consumed by a resource type R_i^k of the VM for the execution of a unit task T per unit of time per unit data.

2.1.2. Modeling scientific workflows. As described in [6], we model a scientific workflow as a directed acyclic graph DAG which can be defined by the couple G = (V, E), where V=$\{T_1, ..., T_n\}$ is the vector representing the tasks of the workflow, with n the total number of tasks. E represents the functional links between tasks, and more specifically, the data dependencies between them, such as $(T_i, T_j) \in E$ if the output data of the task i are required for the execution of the task j. A workflow task is defined by the triplet T= (D, Dt_{in}, Dt_{out}) with:

- $Dt_{in} = \{Dt_{in} | i = 1, ..., k\}$ represents the amounts of data to be processed by each task of the workflow with k is the number of input datasets of the task, each dataset Dt_{in_i} need a minimum time T_i to be processed.
- $D = \{D_i | i = 1, ..., d\}$ is the vector of demands of the task in terms of resource capacities with D_i is the minimum capacity required of the resource type i to process the task and d the number of resource types required.
- $Dt_{out} = \{Dt_{out} | i = 1, ..., s\}$ represents the amounts of data generated after the completion of the workflow task.

2.1.3. Modeling the energy consumption of a workflow. Workflows execution in the Cloud involves allocating computing, network and storage resources used respectively for the execution of tasks, the transfer of data between dependent tasks and the storage of input or generated data [6]. The energy consumed by these allocated Cloud resources could then be divided into three components, namely: the processing energy, the storage energy and the data communication energy [6]. The processing energy is the energy consumed by the virtual machine to execute a specific task. This energy depends essentially on the virtual machine's configuration (processors number and capacities, memory, etc.), the time T_{exe} needed to execute a unit task per unit data, the amount of data to be treated [6]. In this paper, we use the formula (2.1) proposed in [6] for the assessment of the processing energy of a virtual machine VM_i for the execution of a unit task T_j:

$$P_{exe_{ij}} = T_{exe_i} * P_i * Dt_{in_j} \tag{2.1}$$

with P_i is the VM's power consumed for the execution of a unit task per unit time per unit input data and Dt_{in} is the data amount to be treated by the VM for the execution of the task T_j. The storage energy represents the energy needed to store data on a permanent storage device (disk memory) [6]. This type of energy is usually negligible compared to the processor's energy [7].

The communication energy is the energy rate needed to transfer data from a virtual machine to another one using a network bandwidth [6]. The communication energy includes the data access energy (read/write) from DIMM memory (dual in-line memory modules) in source and destination VMs and the network power needed to transfer the data from one network to another E_{ij} [6]. Thus, we consider the formula (2.2) proposed in [6] to assess the energy consumed to transfer an amount of data Dt_{out_i} from a VM_i to a VM_j.

$$P_{transfer} = (Dt_{out_i} * E_{ij})/d_{ij} \tag{2.2}$$

where d_{ij} is the bandwidth rate between the source and destination machines.

In summary, the energy consumed for the execution of a workflow P_w is the sum of its total processing and data communication energy. This energy is described by the formula (2.3) used in our previous work [6]:

$$P_w = \sum_{(i=1)}^{n} \sum_{(j=1)}^{m} x_{ij} * (P_{ij} + \sum_{(k=i+1)}^{n} y_{ij} * Dt_{out_i} * E_{ik}/d_{ik}) \tag{2.3}$$

where: n is the total number of tasks of the workflow, m is the total number of VMs and x_{ij} is a boolean variable that is equal to 1 if the task i is assigned to the VM j and to 0 otherwise. y_{ij} is a boolean variable that is equal to 1 if there is a dependency between tasks i and j and to 0 if there is not.

2.1.4. Modeling and assessing the quality of service of a workflow. For the the modeling and the assessment of the quality of service of a workflow, we refer to our proposed formalism in our previous paper [5]. Indeed, we consider four quality metrics that can represent to our understanding the overall performances of the workflows, namely: the execution time, cost, resources availability and reliability. In the rest of this section we will detail the used equations for the assessment of the quality metrics for the workflow as well as its overall quality of service as described previously in [5]. As a workflow is composed of a set of tasks and data interdependencies between them, we consider that its execution time can be assessed by summing up the execution time of tasks and the data transfer time between them using the formula (2.4).

$$T_w = T_{execution} + T_{dataTransfer} \tag{2.4}$$

Likewise, as the Cloud providers (as Amazon or Google for example) fix a price for each of their basic services like executing a single operation on a specific computing resource or transferring a certain amount of data over network devices, we consider that the economic cost of executing a workflow can be assessed as the sum of the execution cost of the workflow tasks and the data transfer cost between dependent tasks using communication networks, as illustrated by the equation (2.5).

$$C_w = C_{execution} + C_{dataTransfer} \tag{2.5}$$

To assess the global values of the quality metrics of the workflow during its execution, we use aggregation functions for the different considered quality metrics. The formulas used for these aggregation functions are described in our previous work [5].

Once the quality metrics are assessed, we use the Simple Additive Weighting (SAW) normalization method, as in [5], to determine the value of the overall quality of service of the workflow. This latter Q_w can be evaluated via the equation (2.6).

$$Q_w = \sum_{i=1}^{K} w_i * q_i \tag{2.6}$$

where K is the number of considered quality metrics, q_i is one quality metric value assessed for the overall workflow (i.e. the execution time of the workflow) and w_i is the importance weight accorded to each quality metric with the sum of all importance weights is equal to one.

2.1.5. Configurable Scheduling model. Our ultimate objective in this work is to offer a personalized deployment process for each task or set of tasks of the workflow. This includes configuring the scheduling model of workflow to be adapted to the personalized scheduling objectives of its tasks, which implies defining multiple scheduling models for the same workflow. In this case we define a parameterized scheduling model that includes multiple scheduling objectives of both end users and cloud providers and to which we can add specific parameters to adapt it to a specific task or workflow.

The considered objectives in our configurable scheduling problem are: optimizing the quality of services described via the metrics of time, availability, and reliability, minimizing the cost (of execution and data transfer), and minimizing the energy consumption of the workflow, meeting a certain deadline, respecting a specific user budget. These metrics are conflicted for the following reasons:

- To execute quickly a workflow task, we should better use high performance Cloud resources (i.e. virtual machines) which are more expensive than others.
- High performance virtual machines use more resources (CPU, memory) which means that they consume more energy.
- Highly available and reliable resources are usually more expansive than others.

Thus, we modeled the workflows scheduling problem in the Cloud as a multi-objective optimization problem using the following linear representation:

$$\begin{cases} Energy : \text{minimize } w_1 P_w \\ Quality : \text{maximize } w_2 Q_w \\ Cost : \text{minimize } w_3 C_w \end{cases} \tag{2.7}$$

where: w_1 , w_2 and w_3 define respectively the importance weights of the energy consumption rate, the overall quality of service of the workflow and its overall cost to be defined when configuring the workflow scheduling. The quality of the workflow is, in turn defined to be configurable, by using a set of importance weights as illustrated by the equation (2.6). We can thus consider only the quality metrics that we want optimize.

2.2. Dynamic Reconfigurable FPGA (DR-FPGA). Field Programmable Gate Arrays (FPGAs) are one of the fastest growing parts of the digital integrated circuit market in recent times. They can be configured to implement complex hardware architectures. FPGA reconfiguration typically requires the whole chip to be reprogrammed even for the slightest circuit change [7]. The basic architecture of FPGA is based on an array of logic blocks connected through programmable interconnections as illustrated by the figure 2.1.

Dynamic reconfiguration means modifying the system when it is under operation. Reconfigurable computing utilizes hardware that can be adapted at run-time to facilitate greater flexibility without compromising performance. Reconfigurable architectures can exploit fine grain and coarse grain parallelism available in the application because of the adaptability [8].

Dynamically reconfigurable FPGA (DRFPGAs) systems can adapt to various computational tasks through hardware reuse [7]. Currently there are very large capacity DRFPGAs which contain many highly parallel fine grain parallel processing power, and the ability to define high bandwidth custom memory hierarchies offers a compelling combination of flexibility and performance. In addition FPGAs are able to adapt to a real-time processing. We can integrate functional improvements without wasting time receiving hardware or modifying the layout of the circuit. The exploitation of parallelism provides performance advantage over conventional microprocessors [8]. As illustrated by the figure 2.1, the red and the green solid lines represent an example of two kinds of connection ways respectively that allow processing differently the same input data to generate different outputs. The definition of the connection path to be performed can be done in real time according to a certain number of parameters (input data, output data, performance preferences, etc).

The granularity of the reconfigurable logic of FPGA is defined as the size of the smallest functional unit (configurable logic block, CLB) that is addressed by the mapping tools [8]. High granularity, which can also be known as fine-grained, often implies a greater flexibility when implementing algorithms into the hardware.

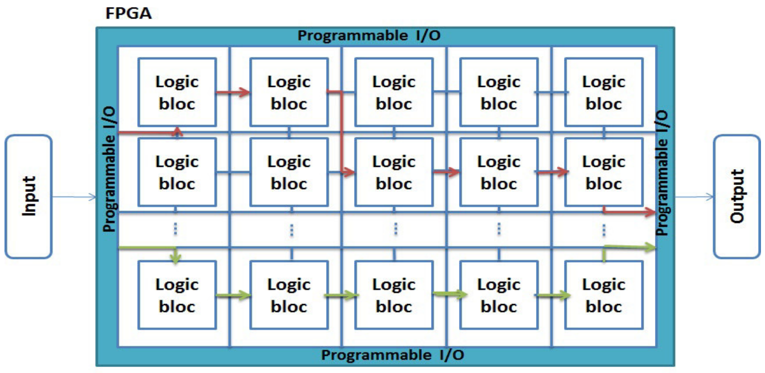

Fig. 2.1. *Architecture of DR-FPGA*

However, there is a penalty associated with this in terms of increased power, area and delay due to greater quantity of routing required per computation [8]. In the contrary, small granularity on coarse-grained architectures are intended for the implementation of algorithms needing word-width data paths. As their functional blocks are optimized for large computations and typically comprise worldwide arithmetic logic units, they will perform these computations more quickly and with more power efficiency than a set of interconnected smaller functional units.

3. DR-SWDF: Dynamic Reconfigurable Scientific Workflows Deployment Framework. In this section, we describe our proposed dynamic reconfigurable framework for scientific workflows deployment that allows dynamically configuring and executing these workflows in the Cloud environment. The main advantage of our proposal is that, for the same input workflow, we can apply different scheduling, partitioning or resource provisioning techniques to each sub-group of interdependent tasks of the workflow at design time and at runtime. We consider that each portion of tasks the workflow can have different characteristics (such as the tasks types, input and output datasets, etc), different scheduling objectives (prefixed budget, deadline, performance constraints, energy consumption rate, etc) and other constraints (such as the placement of input/output data). These characteristics can be defined during the configuration of the workflow before its deployment. Our framework allows also dynamically clustering the input workflow or sub-workflow according to its description by using a K-means clustering algorithm in order to identify the most convenient techniques to be applied for its deployment in the cloud.

The main components of our workflows deployment framework are, namely: the configuration, partitioning, scheduling, provisioning, and the deployment component. Each of the offered components can implement different algorithms and/or techniques. Each algorithm is better suited to the requirements of a specific category of scientific workflows. The figure 3.1 illustrates the main functional components of our proposed DR-SWDF. We inspired the architecture of our framework from that of the DR-FPGA presented in the previous section by replacing each logic bloc by a software component (algorithm, technique, tool, etc). Like DR-FPGAs, the execution process of a scientific workflow using our proposal follows a directed path relying different components starting from the configuration step to the deployment of the workflow. More specifically, the DR-SWDF framework allow to redefine the processing path for a specific partition of the workflow at design time or at runtime in order to use more suitable algorithms to the sub-workflow's characteristics (tasks type, scheduling constraints, input/output data size, etc) for its partitioning, scheduling or provisioning. As an example, the paths illustrated by red and green solid lines in the figure 3.2 represent two different processes to be applied when executing a scientific workflow for two of its partitions. The dynamic reconfiguration step can be performed dynamically before the execution of each partition of the workflow by examining its characteristics and considering personalized techniques for its processing which allow improving the performances of the overall workflow. We consider that the reconfiguration step can be performed without affecting the performances of the overall workflow execution as it can be planned well in advance.

As illustrated by the figure 3.2, the functional process of our proposed framework follows a directed path. For

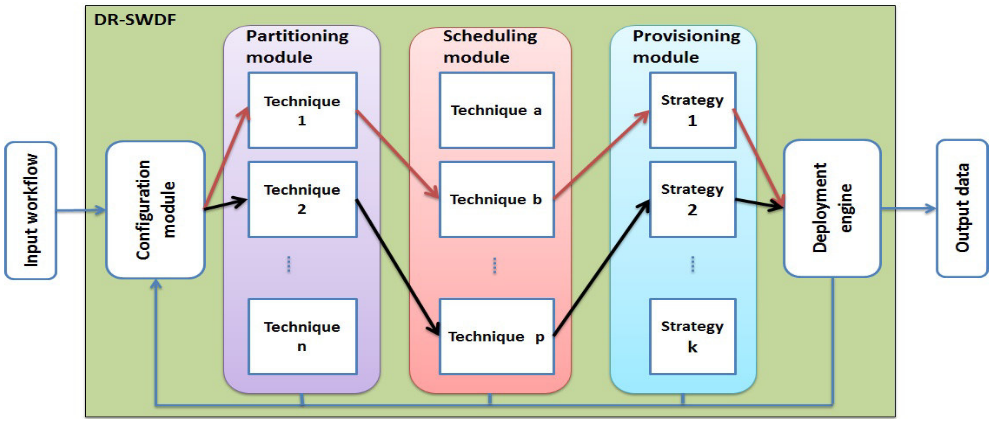

FIG. 3.1. *Overview of our proposed DR-SWDF framework*

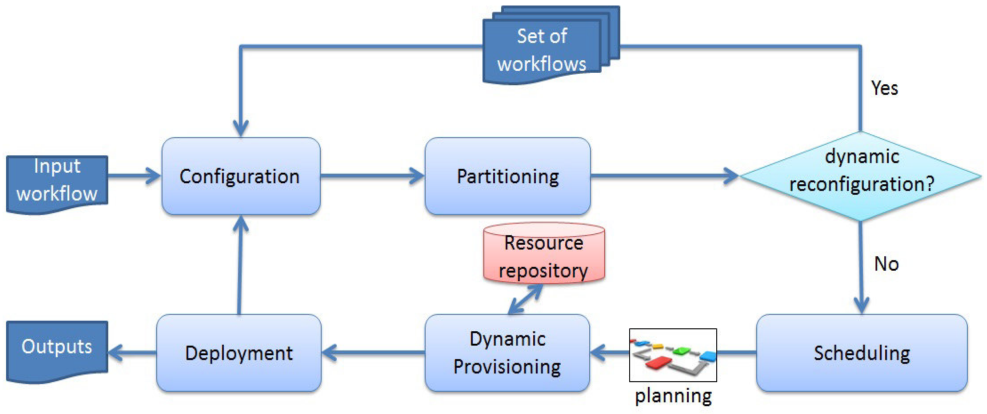

FIG. 3.2. *Functional overview of DR-SWDF framework*

a specific input workflow, the DR-SWDF precedes to the configuration of the workflow deployment by defining, firstly, the workflow type (as computing, I/O or memory intensive) using a K-means clustering algorithm. Based on the workflow type, number of tasks and the number of available resources, the configuration algorithm determines the most convenient partitioning, scheduling and provisioning techniques or algorithms to apply if it's possible. After the partitioning step, if the dynamic reconfiguration option is active (which is active by default), the set of generated workflow partitions are returned to the configuration step to be dynamically reconfigured as distinct workflows with their own characteristics and input/output data. If the dynamic reconfiguration option is not active (which means that the workflow is sufficiently fine grained to be processed at once), the deployment process moves to the scheduling and provisioning of the workflow using the selected techniques during the configuration step.

The framework design is generic and can be extended by adding others components and/or algorithms for the deployment of workflows in the Cloud environment like data reuse, data cleanup and fault tolerance optimization. For instance, we consider only partitioning, scheduling and provisioning of scientific workflows with using two different algorithms for each function to illustrate the advantages of our approach. In the following, we will detail the role of each component and the used algorithms or techniques for each one.

3.1. Workflow Configuration and Dynamic Reconfiguration. The configuration is a key feature of our approach. It allows defining the techniques or strategies to be used for the deployment of the input workflows. As in some cases sub-parts of the workflow might present different characteristics from the overall workflow [41], it is critical then reconsider these parts as separate workflows that requisite different techniques to be deployed in the cloud, and here where the dynamic reconfiguration is needed.

The dynamic reconfiguration can be performed for the overall scientific workflow at once if all its parts have the same characteristics or for a specific partition of the workflow. Its main objective is to reconsider the characteristics of the input sub-workflow for the choice of the most suitable algorithms or strategies to be used for its deployment in the cloud. These characteristics include the following metrics: the workflow type (computing intensive, memory intensive, I/O intensive or a combination of these types), the number of tasks, the estimated makespan and energy consumption of each one, the quality constraints of end users (budget, makespan, etc), the cloud provider's constraints (limited number of VMs, storage memory, etc), and the input and the output data sizes.

Based on the workflow size, model and structure, we defined a set of metrics to initially classify the input workflow or sub-workflow of our framework as computing intensive or data intensive or I/O intensive, etc. Then, we used a clustering algorithm to be able to automatically classify the input workflows, especially those which include different types of sub-workflows.

For instance, we consider the equation (3.1) to identify whether the input workflow is data intensive or not.

$$\sum_{i=1}^{n} \frac{(D_{in}^{i} + D_{out}^{i})}{n} \geq Limit_{dataSize} \tag{3.1}$$

where: n is the overall number of tasks, D_{in}^{i} is the input data size of the task i and D_{out}^{i} is its output data size which is not an input data for another task. $Limit_{dataSize}$ is the threshold of data size beyond it we can consider the workflow as data intensive.

We can identify also a computing intensive workflow by the fact that its tasks need a long time to be processed. Hence, we defined the equation (3.2) to automatically define a computing intensive workflow.

$$\max_{1\leq i\leq j} \frac{t_{exe_i}}{k} \geq T_{limit} \tag{3.2}$$

where: k is the number of the workflow tasks that should be performed sequentially, j is the number of tasks that can be executed in parallel, t_{exe_i} is the estimated execution time of the task i and T_{limit} is the execution time threshold beyond it a workflow can be considered as computing intensive. As it is one of the most popular clustering algorithms, we opted for the K-means clustering algorithm [42] to classify the input workflows or sub-workflows and map them to the most relevant partitioning, scheduling and provisioning techniques that fit their characteristics. K-means aims to partition an input set of N points into K clusters by finding a partition such that the squared error between the empirical mean of a cluster and the points in the cluster is minimized. The squared error metric and more details about the K-means algorithm could be found in [42].

The main steps of K-means algorithm are as follows [42]:

1. Select an initial partition with K clusters; repeat steps 2 and 3 until cluster membership stabilizes.
2. Generate a new partition by assigning each pattern to its closest cluster center.
3. Compute new cluster centers.

The K-means algorithm requires three user-specified parameters: number of clusters K, cluster initialization, and distance metric. The number of clusters in this work is limited to three defining the type of the workflow intensity (I/O, memory, computing). The definition of initial clusters is accomplished using the equations (3.1) and (3.2) determining a first classification of the workflows. Then, we use the graph distance measure defined in [55] to evaluate the similarity degree between the DAG graphs of the input workflows. In fact, the distance of two non-empty graphs G_1 and G_2 can be defined by the equation (3.3) [55]. Intuitively, the larger distance of two graphs is, the more similar the two graphs are. The use of this similarity metric for the clustering of scientific workflows is more convenient as it is able to capture the (structural) difference between the workflows more than other vector space based distances such as the Euclidian distance.

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{max(|G_1|, |G_2|)} \tag{3.3}$$

where $mcs(G_1, G_2)$ denotes the maximal common subgraph of two graphs G_1 and G_2 and $|G_i|$ is the number of arcs in the graph G_i. The algorithm 1 describes the steps of the dynamic re-configuration algorithm proposed for our DR-SWDF framework.

Algorithm 1: Workflow Dynamic Re-configuration Algorithm

```
input  : Var $w$;                                               ▷ the input workflow
         Var $budget$;                                          ▷ The end user's budget constraint
         Var $makespan$;                                        ▷ The workflow completion time constraint
         Var $vm - number$;                                     ▷ number of available VMs
         Var $memory - limit$;                                  ▷ The cloud provider's constraint for maximum storage memory
         Var $cpu - limit$;                                     ▷ The cloud provider's constraint for cpu capacity limit
         Var $dynamic - reconfiguration - option$;              ▷ (0: not active, 1: active)
output:  Var $P$;                                               ▷ The index of the partitioning algorithm 'P'
         Var $S$;                                               ▷ The index of the scheduling algorithm 'S'
         Var $Pr$;                                              ▷ The index of the provisioning algorithm 'Pr'
begin
    $dynamic - reconfiguration - option = 1$;
        ▷ set the dynamic-reconfiguration-option to active by default
    $intworkflow - type = K - means - cluster(W)$;
        ▷ define the type of the workflow by applying the K-means algorithm
    $intp - tasks = count - parallel - tasks(W)$;
        ▷ determine the number of tasks that may run in parallel
    if $p - tasks \leq vm - number$ then
        $dynamic - reconfiguration - option = 0$;
            ▷ disable the dynamic reconfiguration option
    else
        $P = partitioning - technique - choice(workflow - type)$;
            ▷ set the partitioning technique corresponding to the workflow type
    if $dynamic - reconfiguration - option == 1$ then
        $partitions[] = workflow - partition(P, W)$;
            ▷ apply the partitioning algorithm and get partitions
        forall the $workflow - partition \in partitions[]$ do
            $Workflow - Dynamic - Re - configuration - Algorithm(workflow - partition)$;
                ▷ for each workflow partition, re-apply the algorithm from the beginning
    else
        $S = scheduling - technique - choice(workflow - type)$;
            ▷ set the convenant scheduling technique
        $set - objective - function(S, String[]parameters)$;
            ▷ Set the objective function parameters such as the Budget and Makespan
        $set - provisioning - constaints(vm - number, memory - limit, cpu - limit)$;
            ▷ set the provisioning constraints
        $Pr = scheduling - technique - choice(workflow - type)$;
            ▷ set the convenant provisioning technique
```

The configuration component should, also, insure the planning of the deployment order of the workflow partitions and their input/output data sets, and handle errors or execution problems. It should be able to decide if the execution process should be reprogrammed or canceled in case of error.

3.2. Workflow Partitioning. The partitioning function aims to split the workflow composed of a large number of tasks into several sub-workflows in order to achieve a high level of parallelism for the execution of the workflow. Many algorithms were proposed in literature for workflows partitioning such as [6][9][10]. Each algorithm can have a different partitioning objective such as minimizing the overall cost of the workflow (by minimizing the data transfer rate between distant resources) or minimizing its makespan (by maximizing the number of tasks that can be performed in parallel). These two last objectives are conflicting because minimizing the data transfer rate implies fostering the use of the same cloud resources or those situated in the same server, while optimizing the makespan implies using powerful and highly available cloud resources regardless of their location. To determine which algorithm is more suitable to a specific workflow or sub-workflow, we consider the following characteristics: the workflow type, its tasks number and their estimated execution time, with the input and output data sizes. Our DR-SWDF framework analyses these data and chooses the most convenient technique to be considered at the configuration step.

For example, for the Montage workflow [2], which is a computing and data intensive workflow designed to manage science-grade mosaics of the sky, we can foster a partitioning technique that allow to minimize the data movement between partitions for the overall workflow at the configuration step such as in [6]. But at the dynamic reconfiguration of the workflow partitions we can opt for the optimization of its processing time by using a partitioning algorithm that insures a high level of parallelism such as in [9]. Hence, we can optimize the makespan of the workflow and minimize its cost (through minimizing the data transfer rate over communication networks) thanks to the use of two different partitioning techniques for the same input workflow.

In our previous work [6], we proposed a Workflow Partitioning for Energy minimization (WPEM) algorithm that allows reducing the network energy consumption of the workflow and the total amount of data communication while achieving a high degree of parallelism. The main idea of the WPEM algorithm is to apply the technique of max flow/min-cut to recursively partition the workflow into two sub-workflows such as the data communication amount between them is minimal. The experimental results show that WPEM allows reducing remarkably the network energy consumption of both the memory and data intensive workflows, however it doesn't perform better than the [9] for computing intensive workflows [6]. In [9], the authors propose a new algorithm WPRC (Workflow Partition Resource Clusters) for scheduling Scientific Workflows in the Cloud environment. The WPRC algorithm partitions the workflow in order to achieve the highest level of parallelism and thus, optimizing the cost and time of execution of the workflow and the tasks of each partition are, then, assigned to the cluster selected according to their execution priority.

Hence to cover computing, data and memory intensive workflows, we will consider only the partitioning algorithms in [6] and [9] as they offer promoting results, each for a specific kind of workflows. However, our framework is flexible and can integrate other partitioning algorithms.

3.3. Workflow Scheduling. The third function of our proposed framework is the scheduling of the input workflow or sub-workflow according to predefined objectives (such as optimizing the cost and/or the makespan of the workflow, optimizing the QoS of the involved Cloud resources or minimizing their energy consumption rate, etc) as illustrated in the section 2.5. The scheduling of a workflow tasks consists to affect every generated sub-workflow (or workflow partition) from the partitioning step to a Cloud data server and schedule the partition tasks using its virtual machines. The workflow scheduling problem is to carry out a mapping of these tasks to VMs that optimizes its scheduling objectives and constraints. These objectives can be defined at the configuration step for the overall workflow or personalized at the dynamic reconfiguration of the workflow partitions. The utility of defining different scheduling objectives for some partitions of the workflow can be explained by the fact that the tasks of these partitions may be more important than others or manipulate critical data that should be placed within certain constraints.

Being an NP-complete problem, several optimization algorithms were proposed in literature to deal with the issue of multi-objective scheduling of workflows like [5][11][12]. In this context, we implement among others our proposed optimization algorithm (EPCSO) in a previous work [5] which is an enhancement of the Parallel Cat Swarm Optimization algorithm [21]. This algorithm allows resolving the multi-objective optimization problem corresponding to the scheduling objectives of the workflow's user and offers betters results compared to others optimization heuristics such as [11][12]. EPCSO allows optimizing the overall QoS of the workflow by considering one or more metrics like the execution time, the data transmission time, and the cost of the workflow, the cloud resources availability, reliability and the data placement constraints. A second optimization algorithm considered by our framework is that proposed in [12], the authors propose a new PSO-based algorithm to solve the scheduling problem of workflows in Cloud Computing environments. The objective of the scheduling problem is to optimize the makespan and the cost of the workflow. The results of the proposed algorithm show its ability to reduce the overall cost of the workflow and maintain an even distribution of work packages among the allocated resources. We consider for this moment only these two optimization algorithms as they cover the most conflicting scheduling objectives that a user of a scientific workflow can express but it remains possible to add other heuristics in the future.

3.4. Resources Provisioning. Resource provisioning is a key step of the workflow deployment process. It aims to insure efficient use of the Cloud resources which is a common goal for both workflow users and cloud providers. The efficiency in virtual resource provisioning has direct influence on the Quality of Service (QoS) of IaaS clouds [13]. For end users, using the right resources for the right tasks will allow saving their time and money by applying the 'pay-what-you-go' model of the Cloud Computing. For Cloud providers, resource provisioning is finality in itself and an important way to save the energy consumption of their cloud resources. The strategy to be used for resources provisioning depends on several factors such as the type of the workflow (computing or data or I/O intensive, etc), its tasks number and their characteristics (the required software configuration, the estimated execution time, the input/output data, etc), and the needed or generated data size and placement constraints. These factors can be different for some tasks of same workflow, thus, our proposed DR-SWDF offers the possibility to consider a different provisioning strategies for each workflow partition at the dynamic reconfiguration step.

In this scope, we implemented two different provisioning techniques from literature: the first one is designed to the provisioning of VMs [14] and the second one is focalized on the provisioning of storage resources [15]. We have chosen these two techniques as they insure the provisioning of both types of involved cloud resources in the process of scientific workflow execution in the cloud. The goal of VM provisioning is to provide sufficient resources to meet the level of QoS expected by end-users. For this purpose, most cloud provides deliver a set of general-purpose VM instances with different resource configurations. For example, Amazon EC2 provides a variety of VM instance types with different amounts of resources. In [14], Jing et al. modified the genetic algorithm with a fuzzy multi-objective evaluation in order to search for a large solution space by considering the conflicting objectives such as power consumption minimization, total resource wastage, and thermal dissipation costs. The authors designed, also, a genetic algorithm for VM placement. The simulation results show that the algorithm outperforms existing traditional greedy approaches in terms of the number of utilized hosts and performance degradations.

Otherwise, as scientific workflows can be very complex, one task might require many datasets for execution; furthermore, one dataset might also be required by many tasks. If some datasets are always used together by many tasks, we say that these datasets are dependents on each other [15]. The goal of storage resources provisioning is to insure an efficient use of the available storage resources in terms of cost and energy consumption by avoiding excessive data access. To do this, we used our proposed algorithm in [5], in which, we modeled the problem of storage resources selection in the cloud as a multi-objective optimization problem based on the criteria of: availability, cost and their approximation to the virtual machines on which the tasks will be executed.

3.5. Workflow deployment. The workflow deployment is not the final step of the workflow management process according to our proposal. It becomes a central component to which the workflow partitions or tasks ready to be executed are submitted. The execution process is started once the process of dynamic reconfiguration is achieved. To do this, after the partitioning of each workflow or sub-workflow that should respect the execution order of the workflow tasks, every partition follows its own process including task scheduling, resources provisioning, before its submission to the workflow deployment engine as an autonomous workflow with its own input datasets, allocated resources, start time and deadline if defined. If the deployment of a specific partition fails due to a specific reason such as unavailable resource or input dataset access failure, a notification should be sent to the configuration component. In this case, the configuration component should decide according to the failure reason whether it will reprogram the execution of the workflow partition or cancel the overall workflow execution. The monitoring of the whole execution process of the workflow is insured by the configuration component which centralizes all information about the workflow (partitions, input/output data, execution result, resources available, etc).

4. Simulations results. We use the WorkflowSim [35] simulation environment, on which we implemented our proposed framework. WorkflowSim extends the CloudSim [36] simulation toolkit by introducing the support of workflow preparation and execution with an implementation of a stack of workflow parser, workflow engine and job scheduler. It supports a multi-layered model of failures and delays occurring in the various levels of the workflow management systems.

We consider three workflows that we configured to be computing and data intensive workflows, notably the Montage [2], Cybershake [3] and Epigenomics [4] workflows. These workflows were chosen because they represent a wide range of application domains and a variety of resource requirements [37]. The Montage project is an astronomy application that delivers science-grade mosaics of the sky [2]. The Montage workflow is I/O intensive and the figure 4.1 illustrates a simplified structure of it. We used the Montage Workflow with 4006 tasks as in [6]. The CyberShake workflow is used to calculate Probabilistic Seismic Hazard curves for several

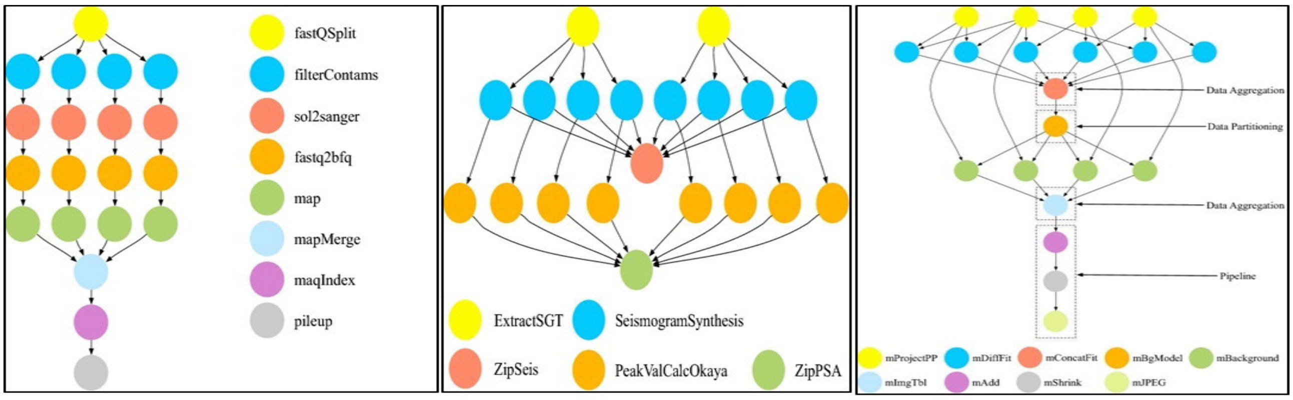

FIG. 4.1. *A simplified representation of the Epigenomics, Cybershake and Montage Wokflows respectively*

geographic sites in the Southern California area [3]. The CyberShake workflow is memory intensive. A portion of this workflow is illustrated by the figure 4.1 containing 2252 tasks. Finally, the Epigenomics project maps short DNA segments collected with high-throughput gene sequencing machines to a previously structured reference genome [4]. The Epigenomics workflow is CPU intensive. A simplified structure of this workflow is described by the figure 4.1. The used Epigenomics Workflow contains 4000 tasks. We used the execution traces of the considered workflows from [1] to generate the values of unitary task's execution time and data communication amounts between dependents tasks of these workflows.

As we don't have enough informations about the different techniques offered by the studied SWfMS like the partitioning algorithm type and objective, we compared our proposed approach to a "generic" one. The "generic" approach corresponds to the use of one technique of partitioning, scheduling and provisioning for the overall workflow among those implemented in our framework and that for all the considered workflows. We use the same simulation environment for the "generic" approach as in our proposed framework. We run four scenarios with different conflicting scheduling objectives:

1. Scenario 1: optimizing both the makespan and the energy consumption
2. Scenario 2: optimizing both the energy consumption and the cost,
3. Scenario 3: optimizing both the cost and the makespan,
4. Scenario 4: optimizing the energy consumption and the overall quality of service of the workflow.

In each scenario, we compare the use of our approach to the generic approach for which we use the most appropriate techniques among those implemented in our framework to apply for the overall workflow. The results of our simulations are detailed below.

For the scenario 1, the overall objective is optimizing the makespan and the energy consumption of the workflow. To satisfy this objective using the generic approach, we used our proposed partitioning algorithm in [6], the scheduling algorithm in [5] and the provisioning strategy of [14]. For our proposed approach, we used for each partition different techniques among those implemented in our framework that corresponds more to the partition type.

The simulation results in figure 4.2 show that our proposed approach allows reducing both the makespan and the energy consumption of all the tested workflows compared to the generic approach. This reduction is due to the use of techniques that are more suited to the workflow tasks/partition rather than the overall workflow.

In the same way, our proposed approach offers better results than the generic one for the second scenario by allowing reducing both the cost and the energy consumption of the tested workflows as illustrated by the figure 4.3.

For the scenario 3, our objective is to minimize both the cost and the makespan of the workflow which depicts the most important objectives of end users. By using our proposed approach, we can achieve better values than using the generic approach designed for all workflow without considering their type and tasks particularities (see figure 4.4).

In the final scenario, our objective is to optimize the overall quality of service of the workflow and its energy

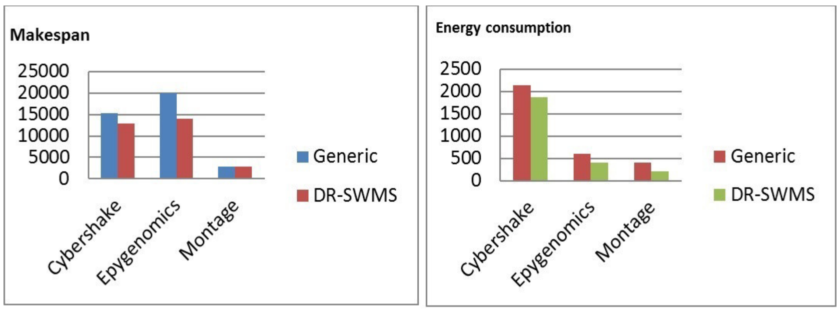

Fig. 4.2. *Simulation results for the scenario 1*

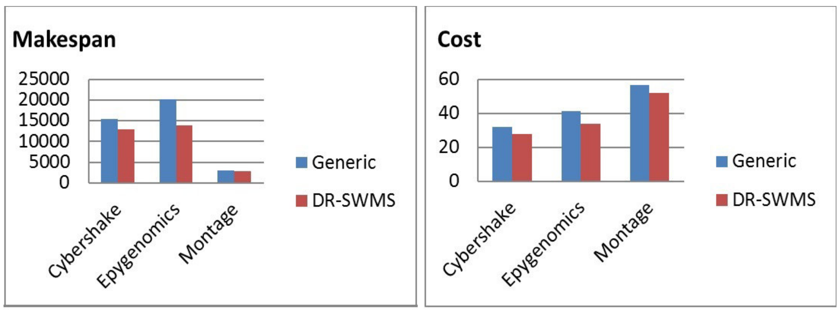

Fig. 4.3. *Simulation results for the scenario 2*

consumption. We assessed the overall quality of service of the workflow by using the metrics of time, availability, and reliability of all the resources involved in the workflow execution. Thus, the considered objectives are conflicting as a high highly available resource is a resource that consumes more energy. The figure 4.5 illustrates the simulation results of our proposed approach compared to the generic approach and confirms that our approach offers better quality of service rate while using less energy than the generic one for all the tested workflows and thus due to the use of more adaptable scheduling objectives for each partition of the workflow, in addition to the use of multiple provisioning and partitioning algorithms for the same workflow. In summary, our proposed framework offers to the end users and cloud providers the occasion to personalize the process of workflow deployment in the cloud by dynamically reconfiguring the partitions or tasks of the workflow as independent workflow and using different techniques for their deployment and conserving the data dependency between them. Our simulation results show that this approach can achieve better results than the use of a generic one within the considered scheduling as in the tested scenarios.

5. Related works. Many SWfMS were proposed in literature, such as Kepler [22], Taverna [23], Triana [24], Pegasus [25], ASKALON [26], SWIFT [27] have demonstrated their ability to help domain scientists on scientific computing problems by synthesizing data, application and computing resources [28]. Other research works that dealt with the scientific workflows deployment in the Cloud proposed their systems as "Software as a service" such as in [31][30][29]. The challenges dealt with in the e-science workflows systems/works can be categorized into three aspects: composition, execution and the workflow composition and learn. The workflow composition aspect involves the features of workflow definition, medialization, usability, etc. The execution aspect includes workflow scheduling, partitioning, resource provisioning, optimization, and workflow adaptability. Finally, the learn aspect includes analyzing the data provenance and results of the workflow execution in order to enhance knowledge learn and the workflow performances.

In this paper, we focus on the execution aspect of the SWfMS, and more precisely on the problems related to adaptability of the execution process to the change of the business context of the scientific workflows. The execution process of such workflows includes the following aspects: the partitioning, scheduling, deployment, resource provisioning, and data storage management especially for data intensive workflows.

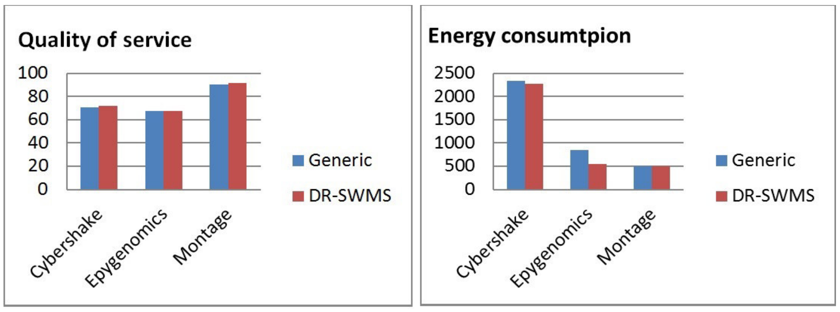

Fig. 4.4. *Simulation results for the scenario 3*

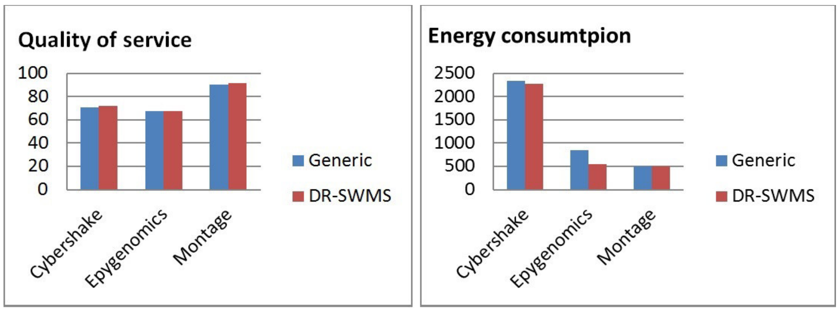

Fig. 4.5. *Simulation results for the scenario 4*

Most of the proposed SWfMS include the data driven adaptability (they deal with the workflow's data changing during the execution process). For example, Triana uses Aspect Oriented Programming (AOP) techniques to perform the workflow rewriting, effectively creating sub-workflows that execute and feed back into the main workflow in order to allow dealing with newly integrated or modified datasets. Pegasus, also, allows, redefining the workflow during the execution by changing the input data or the resources to be used. Likewise, Kepler workflows can modify themselves during execution. Another aspect of adaptability of workflow systems is computation unfolding, one task or sub-workflow could actually need to be iterated or parallelized during the execution based on different conditions. To enable this adaptation aspect, some workflow systems like Pegasus and Swift support abstract workflows [17].

For data intensive workflows, the overall objective of SWfMS is to optimize the data manipulation, transfer and storage within the minimum cost and time. For example, Pegasus technique consists to move data to local or remote locations where scientific applications employed in the workflows will be deployed. If one workflow contains applications deployed on different sites, there have to be multiple data movements during the workflow execution [28]. This technique allows optimizing the overall makespan of the workflow but when the data sizes are very large, times for data movements will be significant and result in very inefficient workflow execution time and cost since data transfer in the cloud is paying. In Triana, in the case of task-based workflow, the user can designate portions of the workflow as compute intensive and Triana will send the tasks to the available distributed resources for execution.

The capability of a SWfMS to allow defining different scheduling objectives and applying multiple techniques for the tasks of the same workflow according to the importance of these tasks or to other parameters (like task type, input/output data size, etc.) is another important feature that SWfMS should consider especially in the era of Big Data emergence. In this context, Pegasus performs a mapping of the entire workflow, portions of the workflow, or individual tasks onto the available resources executed and provides an interface to a user defined scheduler limited to some feature such as the makespan and the execution cost and includes four basic scheduling algorithms, namely: HEFT [33], min-min, round-robin, and random. The Askalon system, designed to support task-level workflows, has a rich environment for mapping workflows onto resources. However, its scheduler makes full-graph scheduling of scientific workflows, using one of the implemented scheduling algorithms such as HEFT.

Resource provisioning adaption is another crucial requirement for SWfMS. Provisioning is slightly more complex than queuing in that it requires users to make more sophisticated resource allocation decisions [34]. Existing provisioning strategies could be classified into two kinds: static and dynamic and are mainly cost or deadline-constrained. In static provisioning the application allocates all resources required for the computation before any jobs are submitted, and releases the resources after all the jobs have finished [34]. In dynamic provisioning resources are allocated by the system at runtime. This allows the pool of available resources to grow and shrink according to the changing needs of the application [34]. Pegasus implements both static and dynamic provisioning to allow cost and deadline-constrained scientific workflows deployment in the Cloud [32]. Other SWfMS like Taverna and Triana include dynamic provisioning algorithms that are applicable for the overall resources implied for the workflow execution. None of the existing SWfMS does offer customizable resource provisioning techniques that correspond to different types of workflows.

6. Conclusions. In this paper, our research objective is to dynamically cluster scientific workflows and ensure their scheduling and deployment according to the specific requirements of end users or cloud providers. To do this, we consider each single partition or task of the workflow as a separate workflow and define a specific execution process for it. We propose DR-SWDF, a dynamically configurable framework for the deployment of scientific workflows in the Cloud that allows using different techniques at runtime according to the input workflow's parameters, the user's requirements and the cloud provider's objectives. The simulations results run on three examples of data and computing intensive workflows show that our proposed framework can achieve better results than the use of a generic one within different conflicting scheduling objectives such as optimizing the energy consumption, the cost or the makespan of the workflow.

In the future, we expect to extend our framework in order to implement more techniques for the considered features and include new features like the data provenance management, data reuse, data cleanup and fault tolerance optimization.

REFERENCES

[1] https://pegasus.isi.edu/workflow_gallery/

[2] http://montage.ipac.caltech.edu/

[3] https://scec.usc.edu/scecpedia/CyberShake

[4] http://epigenome.usc.edu.

[5] Bousselmi, K., Brahmi, Z., & Gammoudi, M. M., *QoS-Aware Scheduling of Workflows in Cloud Computing Environments*, In 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA) (pp. 737–745). IEEE.

[6] Bousselmi, K., Brahmi, Z., & Gammoudi, M. M., *Energy efficient partitioning and scheduling approach for Scientific Workflows in the Cloud*, In Services Computing (SCC), 2016 IEEE International Conference on (pp. 146–154). IEEE.

[7] Kwiat, K. A., *Dynamically reconfigurable fpga apparatus and method for multiprocessing and fault tolerance*, U.S. Patent No 5, 931–959, 3 Aug 1999.

[8] Bondalapati, K. and Prasanna, V., *Reconfigurable computing systems*, Proceedings of the IEEE, 2002, vol. 90, no 7, p. 1201–1217.

[9] Bagheri, R., & Jahanshahi, M., *Scheduling Workflow Applications on the Heterogeneous Cloud Resources*, Indian Journal of Science and Technology, 8(12).

[10] Tanaka, M., and Osamu T., *Workflow scheduling to minimize data movement using multi-constraint graph partitioning*, Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012). IEEE Computer Society.

[11] Pandey, S., Wu, L., Guru, S. M., & Buyya, R., *A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments*, In 2010 24th IEEE international conference on advanced information networking and applications (pp. 400–407). IEEE.

[12] Bilgaiyan, S., Sagnika, S., & Das, M., *A multi-objective cat swarm optimization algorithm for workflow scheduling in cloud computing environment*, Intelligent Computing, Communication and Devices, Advances in Intelligent Systems and Computing, 308, Springer India 2015.

[13] Lu X, Wang H, Wang J, Xu J, Li D., *Internet-based virtual computing environment: beyond the data center as a computer*, Future Generation Computer Systems, 2013, 29(1): 309–322

[14] Xu, J., & Fortes, J. A., *Multi-objective virtual machine placement in virtualized data center environments*, In Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom) (pp. 179–188). IEEE.

[15] Yuan, D., Yang, Y., Liu, X., & Chen, J., *A data placement strategy in scientific cloud workflows*, Future Generation Computer Systems, 26(8), 1200–1214.

[16] Lin, C., Lu, S., Lai, Z., Chebotko, A., Fei, X., Hua, J., & Fotouhi, F., *Service-oriented architecture for VIEW: a visual scientific workflow management system*, In Services Computing, 2008. SCC'08. IEEE International Conference on (Vol. 1, pp. 335–342). IEEE.

[17] Deelman, E., Gannon, D., Shields, M., & Taylor, I., *Workflows and e-Science: An overview of workflow system features and capabilities*, Future Generation Computer Systems, 25(5), 528–540.

[18] Guenther, C. W., Reichert, M., & van der Aalst, W. M., *Supporting flexible processes with adaptive workflow and case handling*, In Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2008. WETICE'08. IEEE 17th (pp. 229–234). IEEE.

[19] L. Guo, S. Zhao, S. Shen, and C. Jiang., *Task scheduling optimization in cloud computing based on heuristic algorithm*, Journal of Networks, 7(3):547–553, 2012.

[20] Han, Y., Sheth, A., & Bussler, C., *A taxonomy of adaptive workflow management*, In Workshop of the 1998 ACM Conference on Computer Supported Cooperative Work.

[21] Tsai, P. W., Pan, J. S., Chen, S. M., Liao, B. Y., & Hao, S. P., *Parallel cat swarm optimization*, In Proceedings of the seventh international conference on machine learning and cybernetics, Kunming, China, 1:3328–3333,2008.

[22] Ludscher B., Altintas I., Berkley C., Higgins D., Jaeger E., Jones M., Lee E., Tao J., & Zhao Y., *Scientific workflow management and the Kepler system*, Concurr Comput Pract Exp 18 (10): 1039–1065.

[23] Oinn T., Addis M., Ferris J., Marvin D., Senger M., Greenwood M., Carver T., Glover K., Pocock M.R., Wipat A., & Li P., *Taverna: a tool for the composition and enactment of bioinformatics workflows*, Bioinformatics 20 (17): 3045–3054, Oxford University Press, London.

[24] Taylor I., Shields M., Wang I., & Harrison A., *The Triana workflow environment: architecture and applications*, In: Taylor I, Deelman E, Gannon D, ShieldsM(eds)Workflows for e-Science. Springer, New York, pp 320–339.

[25] Deelman E, Mehta G, Singh G, Su M, Vahi K, *Pegasus: mapping large-scale workflows to distributed resources*, In: Taylor I, Deelman E, Gannon D, Shields M (eds)Workflows for e-Science. Springer, New York, pp 376–394.

[26] Fahringer T, Jugravu A, Pllana S, Prodan R, Seragiotto Jr, C, Truong H, *ASKALON: a tool set for cluster and Grid computing*, Concurr Comput Pract Exp 17 (2–4): 143–169, Wiley InterScience.

[27] Zhao Y., Hategan M.,Clifford B., Foster I., von Laszewski G., NefedovaV., Raicu I., Stef-Praun T.,Wilde M., *Swift: fast, reliable, loosely coupled parallel computation*, Proceedings of 2007 IEEE congress on services (Services 2007), pp 199–206.

[28] Yang, X., Wallom, D., Waddington, S., Wang, J., Shaon, A., Matthews, B., ... & Vasilakos, A. V., *Cloud computing in e-Science: research challenges and opportunities*, The Journal of Supercomputing 70 (1), 408–464.

[29] Hosni, E., & Brahmi, Z., *OaaS Based on Temporal Partitioning with Minimum Energy Consumption*, Procedia Computer Science 96, 540–549.

[30] Esteves, S., & Veiga, L., *WaaS: Workflow-as-a-Service for the Cloud with Scheduling of Continuous and Data-intensive Workflows*, The Computer Journal 59 (3), 371–383.

[31] Wei, Y., & Blake, M. B., *Adaptive service workflow configuration and agent-based virtual resource management in the cloud*, In Cloud Engineering (IC2E), 2013 IEEE International Conference on (pp. 279–284). IEEE.

[32] Malawski, M., Juve, G., Deelman, E., & Nabrzyski, J., *Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds*, In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (p. 22). IEEE Computer Society Press.

[33] Topcuoglu, H., Hariri, S., & Wu, M. Y., *Performance-effective and low-complexity task scheduling for heterogeneous computing*, IEEE transactions on parallel and distributed systems 13 (3), 260–274.

[34] Juve, G., & Deelman, E., *Resource provisioning options for large-scale scientific workflows*, In eScience, 2008. eScience'08. IEEE Fourth International Conference on (pp. 608–613). IEEE.

[35] Chen, W., & Deelman, E., *Workflowsim: A toolkit for simulating scientific workflows in distributed environments*, In E-Science (e-Science), 2012 IEEE 8th International Conference on (pp. 1–8). IEEE.

[36] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R., *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms*, Software: Practice and Experience 41 (1), 23–50.

[37] Chen, Weiwei, and Ewa Deelman., *Partitioning and scheduling workflows across multiple sites with storage constraints*, Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2011, 11–20.

[38] M. Rahman, X. Li, and H. N. Palit., *Hybrid heuristic for scheduling data analytics workflow applications in hybrid cloud environment*, In Proceedings of the 25th IEEE International Symposium on Parallel and Distributed, ser. IPDPS Workshops. Anchorage (Alaska) USA, Anchorage (Alaska) USA, pp. 966–974, May 2011.

[39] Rodriguez, M. A., & Buyya, R., *Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds*, IEEE transactions on Cloud Computing, 2014.

[40] R. Achary, V. Vityanathan, P. Raj, and S. Nagarajan, *Dynamic job scheduling using ant colony optimization for mobile cloud computing*, Advances in Intelligent Systems and Computing, 321, Springer International Publishing Switzerland 2015.

[41] Ramakrishnan, L., & Plale, B., *A multi-dimensional classification model for scientific workflow characteristics*, In Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science (p. 4). ACM.

[42] Jain, A. K., *Data clustering: 50 years beyond K-means*, Pattern recognition letters 31 (8), 651–666.

[43] Bunke, H., & Shearer, K., *A graph distance metric based on the maximal common subgraph*, Pattern Recognition Letters, 19 (3-4): 255–259, 1998.

[44] Chen, W., & Deelman, E., *Integration of workflow partitioning and resource provisioning*, In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012) (pp. 764–768). IEEE Computer Society.

Edited by: Dana Petcu
Received: Jan 13, 2016
Accepted: May 31, 2017

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:
- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:
- programming environments,
- debugging tools,
- software libraries.

Performance:
- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:
- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:
- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in $\LaTeX 2_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.