ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2016, T. 45, Nr. 3

# Combined Load Balancing Algorithm in Distributed Computing Environment

**Luka Filipović**

*Center of Information System, University of Montenegro, Cetinjska 2, 81000 Podgorica, Montenegro, e-mail: lukaf@ac.me*

**Božo Krstajić**

*Faculty of Electrical Engineering, University of Montenegro, Džordža Vašingtona bb, 81000 Podgorica, Montenegro, e-mail: bozok@ac.me*

crossref http://dx.doi.org/10.5755/j01.itc.45.3.13084

**Abstract**. Load balancing algorithms and task scheduling are one of the most important tasks in parallel application design and implementation. Proper task assignment to processor cores can minimize execution time and increase the performance of a parallel application. In this paper, we propose a combined load balancing algorithm based on a mixture of well-known domain decomposition and master-slave algorithms. The proposed algorithm minimizes load imbalance and communication between independent tasks. The proposed algorithm improved parallel efficiency using task rescheduling, which had been confirmed with simulation results.

**Keywords**: Load balancing algorithms; Process scheduling; Parallel programming; Resource utilization.

## 1. Introduction

Many multidisciplinary scientific fields, such as bioinformatics, biochemistry, electrical engineering and physics, use scientific computing and distributed computing resources for simulations of experiments. Distributed computing clusters consist of closely interconnected servers with multi core processors [1]. The primary focus of many researches in the area of distributed computer scheduling is finding a way to distribute tasks among the CPU cores in order to achieve better performance, such as minimizing job execution time, minimizing communication and maximizing resource utilization. In order to determine this, proper assignment of the tasks to the processor and monitoring of their execution is crucial. Achieving parallelism by redistributing the workload of parallel application segments as computation progresses is referred to as load balancing [2].

The main goal of load balancing algorithms is to find an optimal schedule for the tasks which defines a starting time and an execution resource for each task in order to minimize overall computational time [3]. Execution time of parallel program is the time elapsed between the start of the first task and the completion of the slowest process or process on slowest core. Performance optimization of parallel applications can be done using load balancing algorithms by managing tasks execution during application runtime [4].

The theory of the design of load balancing algorithms started more than forty years ago [5]. Load balancing techniques in parallel systems were developed in two ways: job scheduling on infrastructure and task scheduling inside parallel applications. Various scheduling techniques were developed for high performance and grid clusters [6] [7] and for Cloud infrastructure [8] [9], to achieve maximum utilization of resources, optimize application or virtual machine execution, minimizing timespan between jobs. Similar load balancing algorithms were implemented for scheduling into parallel applications. A certain part of the developed algorithms was developed as general purpose algorithms which are using various application and infrastructure load parameters [10]. On the other side, there are a number of load balancing algorithms created only for scheduling inside specific applications.

Algorithms should be accurate, efficient, stable, portable, and maintainable [11]. Efficient parallel algorithm must avoid communication overheads and load imbalance. Load imbalance of parallel application can occur due to uneven load of computing cores which affect low utilization of distributed system [12]. Many load balancing algorithms have been developed in an effort to coordinate execution time on each core separately, reduce computation time and load imbalance. An efficient load balancing algorithm can improve application performance and help avoid unnecessary delays [3]. In a real distributed environment, a workload of resources varies and it's not always possible to get the resources that are completely free or equally burdened. In addition, many modern supercomputer architectures (such as multi-core or SMP clusters) that look homogeneous from the outside actually conceal a heterogeneous and dynamic environment on the inside. For instance, processors located within the same node are actually competing for shared resources, and intra-node communication is typically much faster than inter-node communication [13]. Losses during execution of parallel applications can happen in imbalanced applications as a result of CPU cores idle time once they have finished their work and wait for slowest core or group of core to finish the given tasks. This phenomenon occurs on heterogeneous clusters or dynamic clusters with a variable load (clusters on which multiple users simultaneously execute parallel applications and thus burden the resources) [11] [14] [15].

Load balancing algorithms can be classified as static and dynamic. Static load balancing algorithms have good usability on homogeneous clusters while they execute tasks on all cores which have similar duration. Performance of programs using these algorithms is reduced at the end of the runtime without possibility of rescheduling. One of widely used static algorithms is domain decomposition algorithm. On the other side, dynamic algorithms can give better efficiency on heterogeneous resources, but make unnecessary communication during executing time. The master slave algorithm is a typical representative of dynamic algorithms [16] [17] [18].

In this paper, we analyze domain decomposition and master slave algorithms, their strengths and weaknesses. We create a new load balancing algorithm by combining these two in order to minimize useless communication between tasks and idle time. The proposed algorithm improves parallel efficiency using task rescheduling, which has been verified through numerical demanding simulation.

The paper is organized as follows. A brief description of the Domain distribution and Master slave load balancing algorithms is given in Section 2. In Section 3, the proposed algorithm is presented. Finally, the simulation results and concluding remarks are given in Sections 4 and 5.

## 2. Domain distribution and Master-slave load balancing algorithms

Scheduling of parallel tasks using domain decomposition and master slave can be used on a various type of distributed computer resources: homogenous clusters and heterogeneous clusters. Depending on the type of allocated resources, scheduling algorithms can give different efficiency results.

In static load balancing, the assignment of tasks to processors is performed before program execution begins. Scheduled task is always executed on the assigned CPU core. Static scheduling methods minimize the overall execution time of a concurrent program and minimize the communication delays. The advantage of static scheduling methods is that all the overhead of the scheduling process is incurred at compile time, resulting in a more efficient execution time environment compared to dynamic scheduling methods [16] [19] [20].

Domain decomposition (DD) algorithm [17] is one of the most used static algorithms. Many applications in physics, chemistry, mechanics and climate modeling simulations are parallelized using domain decomposition algorithm. With this scheduling policy, tasks are dispatched to all CPU cores with equal probability, according to pre-defined rules or random order. Efficiency of the algorithm is maximal till moment when first cores finishes assigned jobs in $T_{min}$ (Figure 1). From this moment, the fastest core or group of cores are in the idle state, which induces load imbalance and utilization losses, until $T_{max}$ moment when parallel application finishes its work. Load imbalance happens because duration of tasks is not known in advance as well as the impact of external factors which may disrupt performance, which is the main disadvantage of this algorithm. Efficiency of applications using domain distribution algorithm is strongly affected by heterogeneity and variability of distributed computer system. Domain distribution algorithm is the most efficient when the computational problem can be divided into equal parts and computational load is equally distributed among the processor cores [13].

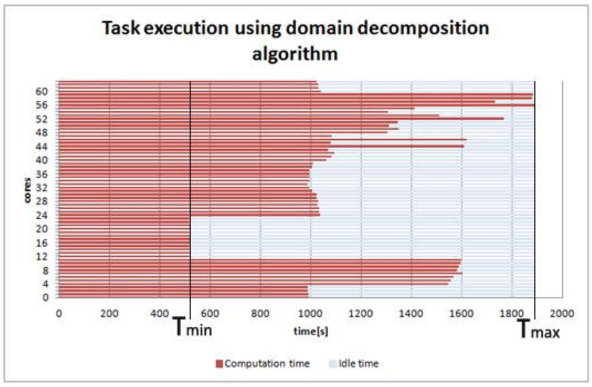


**Figure 1.** Distribution of the tasks using domain decomposition algorithm

Dynamic load balancing is based on the rescheduling of processes among the CPU cores during execution time of parallel program. Rescheduling is performed by transferring tasks from the heavily loaded processors to the lightly loaded cores with the aim of reducing the execution time and minimizing load imbalance. The load balancing operations may be managed by a single core or distributed among all the processing elements that participate in the load balancing process. Each core passes its current load information to its neighbors at the specified time intervals, resulting the redistribution of load among all the processing elements in a short period of time [21]. Main advantages of dynamic algorithms is a fact that system doesn't need to be aware of the tasks run-time behavior before execution and adjustment of task scheduling to the resources. Disadvantage of dynamic algorithms is run-time overhead for transferring load information among processors, decision making for the processes redistribution and communication delays for task relocation [22].

Master-slave (MS) paradigm [18], as one of the basic and the most used dynamic scheduling algorithms, is often used in computational biology parallel simulations [23] [24]. It involves two types of computing cores. Preprocessing, task allocation and post processing is performed on the master core, while task execution is performed on slave cores. Master core generates a list of tasks that need to be executed and sends one or more instructions to slave cores. Slave core, upon completion of given tasks, signals the end of assigned tasks whereupon master core allocates them the next task or list of tasks. This routine is repeated until all processes are finished. The advantage of the algorithm is reflected in a good management process. One disadvantage of an algorithm is an increased communication between the master and slave cores and potential waiting of slave cores for allocation of new tasks waiting for execution. Tasks cannot be executed on master core, so this is another disadvantage, especially during the execution on the smaller number of cores [25].

## 3. Proposed combined algorithm (CA)

In this section, a new load balancing algorithm, based on combination of DD and MS algorithms, is presented. The motivation was to improve load balancing performance and execution time for parallel applications which consist of many independent tasks. The proposed combined algorithm consists of three phases.

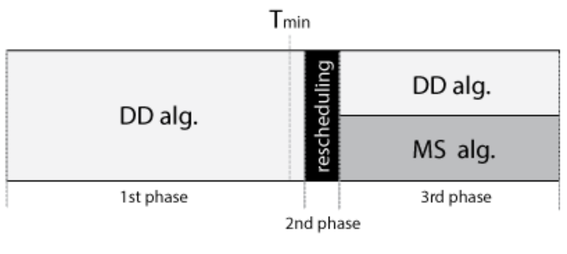


**Figure 2.** The proposed algorithm

*Phase 1.* As seen in Figure 1, application's usability with DD algorithm is 100% till $T_{min}$ moment. Opposed to this algorithm, master-slave makes load imbalance from the start of application because tasks can't be executed on master core. Therefore, DD algorithm is used in the first phase of the proposed CA algorithm (Figure 2). In the first phase CA works as DD until $T_{min}$ when all assigned tasks to the fastest core are finished. The fastest core then sends signal to each other core to finish task which it executes and terminate execution of rest assigned unfinished tasks. This phase is finished when all cores finish executing started tasks.

*Phase 2.* In the second phase, all computing cores send reports to the pre-specified core. Each report includes:

1. status of assigned tasks,
2. execution time of finished tasks,
3. information about resources (core speed and allocated memory).

Pre-specified core analyzes received information, makes a list of unfinished tasks, chooses algorithm (DD or MS) for the third phase, and performs rescheduling. The choice of the algorithm is performed according to the number of computing cores, heterogeneity of a cluster (c) and execution time of each performed task (b).

Domain decomposition algorithm is chosen for the third phase if:

DD1) application runs on homogeneous cluster on less than 32 cores,

DD2) each core from the first phase executes similar number of tasks with similar duration,

DD3) number of unfinished tasks is less or equal to the number of cores where parallel application executes.

Master–slave algorithm is chosen for the third phase if:

MS1) application runs on more than 32 cores, i.e. when master-slave algorithm can't produce a significant loss of utilization due execution.

MS2) application runs on heterogeneous cluster or on clusters where load rapidly changes,

MS3) duration of tasks is significantly different,

MS4) each core from the first phase executed significantly different number of tasks.

Rescheduling algorithm makes a list of unfinished tasks in accordance with selected algorithm. If DD algorithm is selected, each core receives a list of unfinished tasks for execution. The number of assigned tasks to each core is calculated according to the number of tasks finished in the first phase on each core separately. If MS algorithm is selected, then the master core receives a list of all unfinished tasks which will be assigned to the slave nodes for execution in the third phase.

*Phase 3.* Selected algorithm from the second phase is executed in the third phase of the proposed algorithm.

Figure 3 presents operations of combined algorithm per stages. In the first phase of the proposed algorithm, the DD algorithm was used (marked with blue). The second phase selected MS algorithm because parallel application was started on 64 cores (according MS1 rule) and nearly half of tasks were unfinished after termination (according MS4 rule) and made rescheduling (marked with red). Regarding decision from the second phase in this example, MS was executed in the third phase (marked with green).

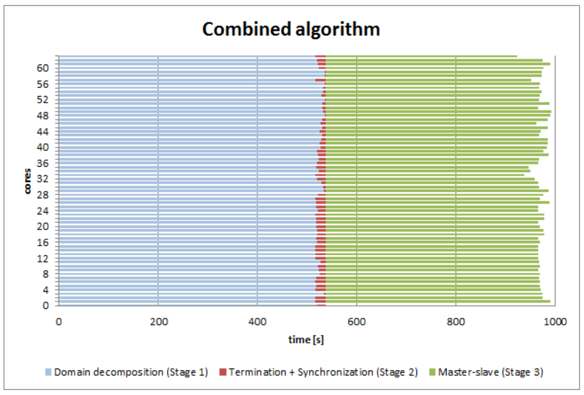


**Figure 3.** Task scheduling using combined algorithm

According to analysis and rescheduling in the second phase, the proposed algorithm increases the efficiency and reduces execution time for parallel applications in the third phase. The execution time of the proposed algorithm is shorter (Figure 3) than execution time of the standard DD algorithm measured in same conditions (Figure 1). Moreover, the efficiency of the proposed algorithm is increased due to minimized idle time and improved resource usage.

Disadvantages of the combined algorithm are termination of assigned tasks at the end of the first phase and duration of the second phase. Duration of the second phase is insignificantly low and can't affect the efficiency of parallel application. Termination of assigned tasks in the first phase can increase duration of this phase only if there is one task whose duration is enormously higher than duration of others. This increase of first phase duration can affect the performance of the whole algorithm. In that case, there is no improvement in efficiency compared with DD and MS.

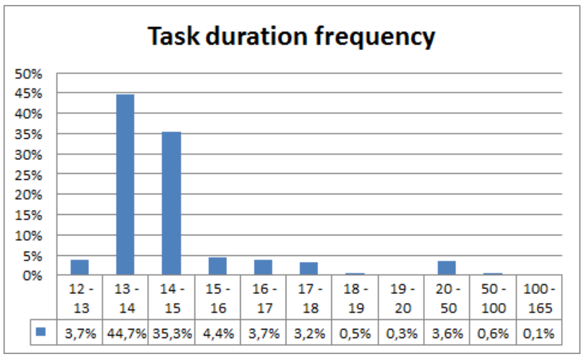
The combined algorithm works as DD algorithm during their maximal efficiency and interrupts work when its effectiveness starts to weaken. It has a similar performance as DD in the case when DD has a high efficiency. The proposed algorithm has better performance than DD when DD has low efficiency due to interruption and rescheduling.

CA has better performance than MS because MS doesn't execute tasks on master core whole time and has less communication loses during execution time. The MS algorithm produce less efficiency than the proposed algorithm which starts as DD and makes rescheduling to achieve better resource usage.

## 4. The results of simulation and analysis

Performance of the combined algorithm is verified through numerical demanding application Cross-Point queued switch (CQ) simulator for performance analysis [26]. Simulator is parallelized using MPI [27]. It executes simulations of eight different switching algorithms (LQF, RR, ERR, FBRR, EELQF, ELQF, FBLQF and RAND) with 12 different buffer sizes on 32 input files of generated traffic. Simulation was performed for matrix of 16x16 and 1.000.000 time slots. During the preprocessing, simulator prepares 3072 independent tasks. Simulation was performed on Paradox [28] HPC cluster during HP-SEE project [29].

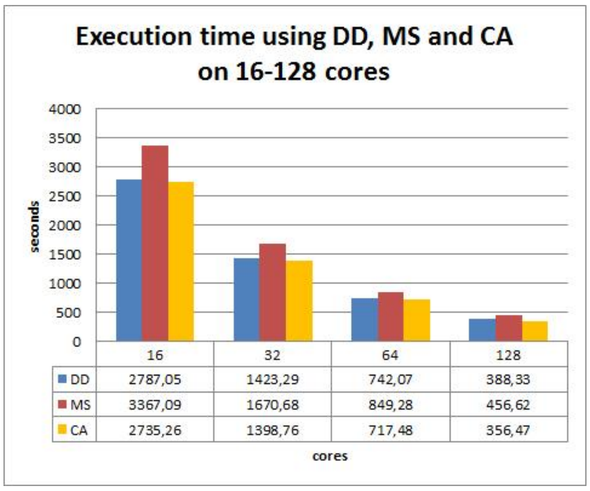
Figure 4 presents frequency of computational time of CQ tasks. 95% of tasks finished assigned work between 12 and 18 seconds. The average execution time was 15.10 seconds, while the longest task was executed in 165 seconds. The statistics is based on the pattern of 800.000 executed tasks.



| | 12 - 13 | 13 - 14 | 14 - 15 | 15 - 16 | 16 - 17 | 17 - 18 | 18 - 19 | 19 - 20 | 20 - 50 | 50 - 100 | 100 - 165 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 🟦 | 3,7% | 44,7% | 35,3% | 4,4% | 3,7% | 3,2% | 0,5% | 0,3% | 3,6% | 0,6% | 0,1% |

**Figure 4.** Frequency of task computational time

Total execution time depends on the duration of each task and their scheduling. Simulations using DD, MS and CA are performed on 16-128 cores. Input files were copied on nodes in the preprocessing phase of the application. Average results of twenty executions at different clusters loads are presented at Figure 5.



| | 16 | 32 | 64 | 128 |
|---|---|---|---|---|
| 🟦 DD | 2787,05 | 1423,29 | 742,07 | 388,33 |
| 🟥 MS | 3367,09 | 1670,68 | 849,28 | 456,62 |
| 🟨 CA | 2735,26 | 1398,76 | 717,48 | 356,47 |

**Figure 5.** Average execution time using three scheduling algorithms on 16-128 cores

CA gave better results than DD and MS algorithms in all conditions. The impact of rescheduling and reduced runtime was more noticeable as the number of cores increased. In few cases, CA required up to 0.5% time more than DD. In the best case, CA finished execution 21.6% faster than DD due task rescheduling at the end of application. MS algorithm required more time than DD and CA, especially during execution on 16 cores due to described disadvantages.

Figure 6 presents execution time comparison between CA and other algorithms. The difference between CA and DD ranges from 1.7% to 8.2%. DD required more time to execute than CA due to static scheduling process. The difference between CA and DD was higher when the application was started on larger number of cores.

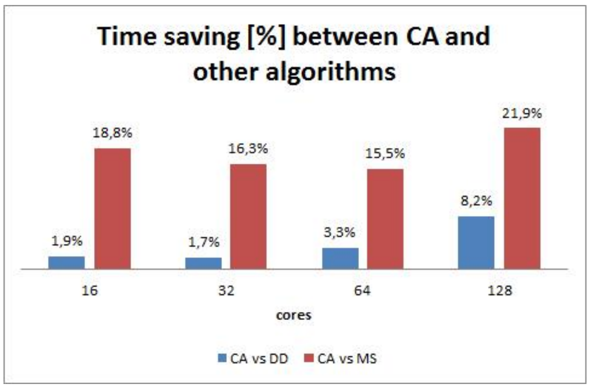


**Figure 6.** Execution time comparison between CA, DD and MS

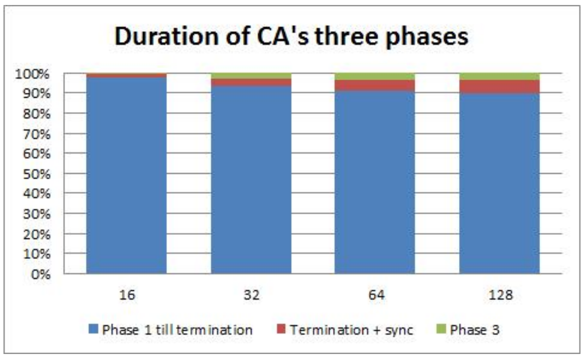


**Figure 7.** Time distribution of CA’s phases

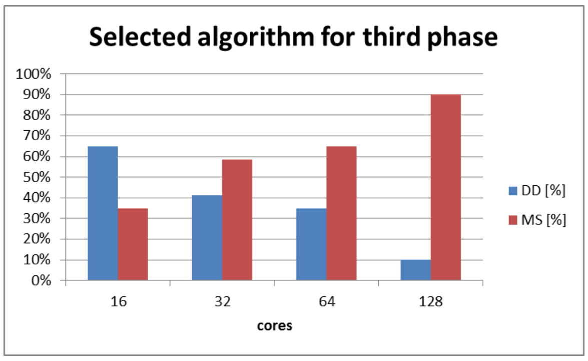


**Figure 8.** Algorithm choice in CA’s third phase

The difference between CA and MS was bigger, since MS algorithm had specified weaknesses. Inability to execute tasks on master core produced losses during execution on lower cores. Communication between cores during the entire process of execution caused lower efficiency on 128 cores.

Figure 7 presents time distribution of CA phases. Three segments are indicated: the first phase till termination, termination and synchronization phase and the third phase for simulations performed on 16-128 cores. Duration of termination and third phase increased as simulation was performed on bigger number of cores. Termination phase time, marked by red color, increased on 128 cores because the fastest core waited for more cores to finish tasks which were executed in time when termination signal was sent. We noticed high efficiency of CA and DD algorithm on 16 cores and higher number of tasks which were rescheduled on 64-128 cores.

Figure 8 shows which algorithm performed scheduling in the third phase. Domain distribution was selected in the most cases when simulation was executed on 16 cores, because the program detected Paradox as a homogenous cluster with a number of allowed cores less than 32. On the other hand, master-slave had priority in other cases because the algorithm from the second phase detected tasks with different duration and selected this dynamic algorithm for the third phase.

## 5. Conclusion

In this paper, the combined load balancing algorithm for parallel applications which consist of many independent tasks has been presented. The algorithm is created on the strengths of the domain decomposition and master slave algorithms and task rescheduling. Using mixture of these, standard static and dynamic, algorithms we reduced execution time, minimized load imbalance and improved performance of parallel application in various distributed environments. This paper also identifies situations when the proposed algorithm doesn’t provide improvements, but it still maintains performance comparable to constituent algorithms.

The main contribution of this paper can be summarized as follows:

- the combined load balancing algorithm based on domain decomposition and master slave algorithms has been proposed,
- heuristic approach for the selection of load balancing algorithm after domain decomposition in the first phase,
- new algorithm improves the performance of parallel application which consist of many independent tasks,
- simulation results which confirmed better performance of the combined algorithm when compared with domain decomposition and master slave algorithms.

Further research will be focused on evaluation of the combined algorithm on various heterogeneous clusters as well as its implementation for practical parallel applications.

## Acknowledgments

This work makes use of results produced by the High-Performance Computing Infrastructure for South East Europe's Research Communities (HP-SEE), a project co-funded by the European Commission (under contract number 261499) through the Seventh Framework Programme. HP-SEE involves and addresses specific needs of a number of new multi-disciplinary international scientific communities (computational physics, computational chemistry, life sciences, etc.) and thus stimulates the use and expansion of the emerging new regional HPC infrastructure and its services. Full information is available at: http://www.hp-see.eu/.

## References

[1] **H. D. Karatza, R. C. Hilzer**. Parallel Job Scheduling in Homogeneous Distributed Systems. *Simulation,* 2003, Vol. 79, No. 5-6, 287-298.

[2] **J. Dinan, S. Olivier, G. Sabin, J. Prins, P. Sadayappan, C.-W. Tseng**. Dynamic Load Balancing of Unbalanced Computations Using Message Passing. In: *Parallel and Distributed Processing Symposium, 2007, IPDPS 2007, IEEE International*, Long Beach, CA, USA, 2007.

[3] **T. Rauber, G. Rünger.** Parallel Programming: for Multicore and Cluster Systems. *Springer*, 2010.

[4] **D. M. Abdelkader, F. Omara.** Dynamic task scheduling algorithm with load balancing for heterogeneous computing system. *Egyptian Informatics Journal,* 2012, Vol. 13, No. 2, 135-145.

[5] **M. J. Atallah, M. Blanton.** Algorithms and Theory of Computation Handbook. Second Edition, *Chapman & Hall/CRC*, 2009.

[6] **S. Patil, A. Gopal.** Cluster performance evaluation using load balancing algorithm. In: *International Conference on Information Communication and Embedded Systems (ICICES) 2013,* pp. 104-108.

[7] **D. G. Feitelson, L. Rudolph**. Job Scheduling Strategies for Parallel Processing. *Springer*, 1995.

[8] **M. Katyal, A. Mishra**. A Comparative Study of Load Balancing Algorithms in Cloud Computing Environment. *International Journal of Distributed and Cloud Computing*, 2013, Vol. 1, Issue 2, 5-14.

[9] **N. Shahapure, P. Jayarekha**. Time sliced and priority based load balancer. *Advance Computing Conference (IACC), 2015 IEEE International,* pp. 154-159.

[10] **A. Chhabra, G. Singh, S. S. Waraich, B. Sidhu, G. Kumar**. Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment. In: *Proceedings of World Academy of Science, Engineering and Technology (PWASET)*, 2008, Vol. 2, No. 4, pp. 1292-1295.

[11] **Y. Deng**, Applied Parallel computing. World Scientific Publishing Company, 2012.

[12] **D. Thiébaut.** Parallel Programming in C for the Transputer, 1995.

[13] **C. Banino-Rokkones**. Domain Decomposition vs. Master-Slave in Apparently Homogeneous Systems. In: *Parallel and Distributed Processing Symposium, IPDPS 2007, IEEE International*, Long Beach, CA, USA, 2007.

[14] **B. Blaise**. Introduction to Parallel Computing. *Lawrence Livermore National*, 2012.

[15] **L. Filipović, B. Krstajić**. Modified master-slave algorithm for load balancing in parallel applications. *ETF Journal of Electrical Engineering*, 2014, Vol. 20, No. 1, 74-83.

[16] **V. Sarkar**. Partitioning and Scheduling Parallel Programs for Multiprocessors. *MIT Press*, 1989.

[17] **W. D. Gropp**. Parallel Computing and Domain Decomposition. In: *Fifth Conference on Domain Decomposition Methods for Partial Differential Equations*, 1990, pp. 249-361.

[18] **S. Sahni**. Scheduling Master-Slave Multiprocessor Systems. *IEEE Transactions on Computers,* 1996, Vol. 45, No. 10, 1195-1199.

[19] **B. Shirazi, M. Wang, G. Pathak**. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributed Computing,* 1990, Vol. 10, No. 3, 222-232.

[20] **B. A. Shirazi, A. R. Hurson, K. M. Kavi**. Scheduling and Load Balancing in Parallel and Distributed Systems. *Wiley-IEEE Computer Society Press*, 1995.

[21] **D. L. Eager, E. D. Lazowska, J. Zahorjan**. Adaptive Load Sharing in Homogeneous Distributed Systems. *IEEE Transactions on Software Engineering,* 1986, Vol. 12, No. 5, 662-675.

[22] **Chhabra, G. Singh**. Qualitative Parametric Comparison of Load Balancing Algorithms in Distributed Computing Environment. In: *International Conference on Advanced Computing and Communications, ADCOM 2006,* 2006, pp. 58-61.

[23] **M. Depolli, R. Trobec, B. Filipič**. Asynchronous master-slave parallelization of differential evolution for multi-objective optimization. *Evolutionary computation,* 2013, Vol. 21, No. 2, 261-291.

[24] **D. Hadka, K. Madduri, P. Reed**. Scalability Analysis of the Asynchronous, Master-Slave Borg Multiobjective Evolutionary Algorithm. In: *IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum*, 2013, pp. 425-434.

[25] **S. Sahni, G. Vairaktarakis**. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization,* 1996, Vol. 9, No. 3-4, 357-377.

[26] **M. Radonjic, I. Radusinovic**. Impact of scheduling algorithms on performance of crosspoint-queued switch. *Annals of Telecommunications,* 2011, Vol. 66, No. 5-6, 363-376.

[27] MPI Forum, MPI: A Message-Passing Interface Standard. University of Tennessee, Knoxville, Tennessee, 1994.

[28] **A. Balaz, O. Prnjat, D. Vudragovic, V. Slavnic, I. Liabotis, E. Atanassov, B. Jakimovski and M. Savic**. Development of Grid E-Infrastructure in South-Eastern Europe. *Journal of Grid Computing,* 2011, Vol. 9, No. 2, 135-154.

[29] High-Performance Computing Infrastructure for South East Europe's Research Communities (HP SEE) [Online]. Available: https://www.hp-see.eu/.

Received September 2015.