Trial Testing Efficiency of Algorithms for Task Execution in Multi-Processor Systems

Experimentation System for Priority Scheduling and Open Pool of Tasks

Magdalena Respondek, Leszek Koszalka, Iwona Pozniak-Koszalka, and Andrzej Kasprzak
Department of Systems and Computer Networks
Wroclaw University of Technology,
Wroclaw, Poland
170966@student.pwr.wroc.pl,{leszek.koszalka, iwona.pozniak-koszalka, andrzej.kasprzak}@pwr.wroc.pl

***Abstract*—The need of an increased throughput has led to a new approach in the computer system design. In order to face the growing demands of a potential user, focus on multitasking and maximally enhance the capabilities- multiprocessor systems have been introduced. In such systems, two or more Central Processing Units (CPU) are working in parallel, sharing computer bus and communicating through shared memory. In this paper, the homogenous system, with three identical processors, having a common ready tasks' queue, is considered. It is modeled with the use of the adequate simulator. The challenge is to apply the best possible scheduling algorithms, as to provide an optimal system, which meets all the quality of service requirements. The conducted research on both- open and closed pool of tasks is fully described. The results are presented and thoroughly analyzed in order to choose the best possible algorithms for the discussed cases.**

Keywords-multiprocessor system; tasks scheduling; algorithm; time efficiency; simulation

I. INTRODUCTION AND MOTIVATION

The objective of this paper is to plan and, by simulating the multiprocessing system, to conduct such a research as to maximally optimize it in terms of time efficiency.

According to [1], there are usually considered 5 different criteria of system optimization from efficiency point of view:

- Maximum CPU utilization- each processor should be as busy as possible;
- Maximum throughput- number of processes finished in one CPU cycle- there should be more work done in less time;
- Minimum turnaround time- the time needed for the process to be executed (since its arrival to a system till completion);
- Minimum waiting time- time spent by the process in ready queue;
- Minimum response time- time between CPU request and the first answer.

In order to fulfill all of the above requirements, each of parallel processors has to have scheduling algorithm implemented. With the use of such algorithm, each CPU is able to schedule the execution of system processes and to properly manage the workload.

The research was a continuation of the studies already conducted on this matter – described in technical reports [2] and [3]. Additionally, besides examining, how basic algorithms behave in the controlled, closed pool of tasks, new research was made to analyze the topic under conditions close to real systems. Closer look was taken at the priority scheduling algorithm and the open pool of tasks.

The rest of the paper is organized as follows. Section II focusses on the experimentation system used in the research. In Section III, each of the used scheduling algorithms is fully described, along with its manner of working. Section IV is the main part containing the full studies, with the methods, research descriptions and exemplary results analyzed for two cases: Priority Scheduling and Open Pool of Tasks. The final remarks and plans for further research appear in Section V.

II. EXPERIMENTATION SYSTEM

In order to investigate the properties of the scheduling algorithms for the multiprocessor's system, a proper tool had to be utilized. For this research the created and implemented three-processor experimentation system, called 'Simulator Pro 3', has been used. The basic scheme of the input – output system, is presented in Fig.1.

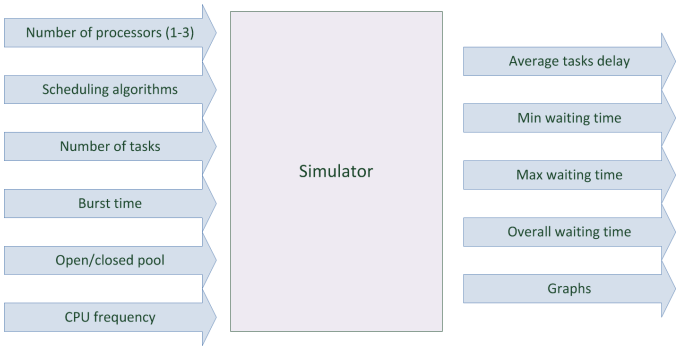

Figure. 1. Scheme of the simulator as input - output system.

The core of the system was the program described in [4]. 'Simulator Pro 3' was designed using C++ and must be launched on the MS Windows operating system. The system provides multiple options to a user, with a reasonable usage of which, the

 ISBN: 978-1-61208-283-7

complex research can be conducted. It can be noticed, that there are many options that can be modified in order to model the environment as specifically as it is possible. Hence, the constraints of the problem are those of a real-life system, including the number of processors along with their frequencies and pool of tasks, which system has to deal with (number of tasks, burst time, open/closed pool).

A. Input parameters

- *Number of processor*s: 1-3;
- *Scheduling algorithms:* 6 different to choose from;
- *Number of tasks:* randomly or arbitrary chosen;
- *Burst time:* randomly chosen from 1-X range, where X is determined by the user;
- *Open/closed pool of tasks***:** for the open pool additional number of processes can be entered into a queue;
- *Processes' priorities:* Importance of tasks represented by integer values;
- *CPU clock rate:*1 - 4000MHz.

B. Output parameters

- *Average tasks delay:* Total waiting time divided by the number of tasks;
- *Min waiting time:* numbers of cycles representing the waiting time of the earliest executed task;
- *Max waiting time:* the longest time that the task had to wait for to be executed;
- *Overall waiting time:* waiting time obtained by all tasks;
- *Graphs:* bar charts representing waiting times of consecutive processes.

III. IMPLEMENTED ALGORITHMS

In this research, the following requirements are taken into consideration. The access to CPUs is solved with a load sharing methodology, i.e., tasks are waiting in a ready queue, which is common for all three processors. Each CPU chooses the process from the queue to execute it according to a pattern, called scheduling algorithm.

In the modeled system, maximum of three tasks can be executed at the time. Each processor can have different scheduling algorithm implemented.

All the algorithms are described, e.g., in [5] and [6] and are depicted below:

A. First Come First Served

It is later referred to as FCFS. In this algorithm the tasks are executed in an order they request a CPU. That means that importance of a process is measured only by the time of its arrival. It can be managed by a FIFO (First in First Out) queue.

This type of algorithm is associated with the risk of convoy effect, where small tasks have to wait for the bigger one to be executed and get off of the processor.

B. Shortest Job First

Shortest Job First, later referred to as SJF, is the algorithm, in which the importance of the process is meant by the length of its next CPU burst. Hence, the processor chooses the task with the shortest duration time to be executed. When there are two processes with the same CPU burst length in a queue, FCFS scheduling is used.

Shortest Job First scheduling is hard to be implemented, because it is not possible to know the length of the next CPU burst, it can be only predicted.

There are two types of SJF algorithm: preemptive and non-preemptive. They are distinguishable only on the open pool of processes where the new task arrives to a queue while the other is being executed. If the duration of a new task is shorter than what is left of the currently executed one, the preemptive algorithm preempts the "old" process. That is why it can be also called Shortest Remaining Time First. In the same situation non-preemptive algorithm finishes the "old" task's execution and then can start the new one.

C. Round Robin

In Round-Robin (RR) type of scheduling there is a constant time quantum, after which the process is being preempted. Round-Robin works like FCFS scheduling, except that, after a time slice, CPU interrupts the execution of the process and takes on the new task from the queue. The "old" process is added to a queue's tail. Therefore in this algorithm, processes are not waiting long to be started, but because they are switched by the CPU, their delay time is long.

D. Priority Scheduling

In the Priority Scheduling, each process has a priority associated with it. Task with the biggest importance is the first one to be executed. Processes with equal priorities are scheduled according to a FCFS algorithm. SJF can be treated as a special case of Priority Scheduling, where priorities are corresponding with the tasks' lengths.

Priority algorithm can be either preemptive or non-preemptive. The main problem associated with this kind of scheduling is the starvation. On the open pool of task, when the new high-priority tasks are entering the system, the process with the lower significance may never be executed. The solution for this issue is aging where, after defined numbers of cycles, the priorities of long waiting tasks are increased.

IV. INVESTIGATION

The research was mainly focused on two areas: (i) finding the best Priority Scheduling algorithm's parameters, and (ii) finding the best combination of algorithms in the open pool of tasks. After doing preliminary research, two complex experiments were conducted.

A Experiment #1 - Priority Scheduling

The first area of research was the Priority Scheduling algorithm. In this experiment not only the length or a number of

 ISBN: 978-1-61208-283-7

processes can be changed, but also the range of priorities and the aging step for algorithm.

1) Experiment Design

In this research there were taken three basic sets of data, each with 20 tasks. These processes' length was ranging from 1 to 10. First two sets had randomly chosen burst time. The last one's halves were sorted in ascending order.

Research was conducted on one processor only, for the clearer picture of Priority Scheduling manner of working. Once the best parameters were found, they could be applied for the further examination.

The experiments were conducted on every combination of sets and priorities range. For each composition of input parameters aging step had been changed gradually to observe the improvement.

The features of experiment design were taken as follows:

- 1 active processor;
- Non-preemptive Priority Scheduling applied;
- Three basic sets of tasks: SET 1 = 107, SET 2 = 134 tasks, and SET 3 = 110 tasks (see Table 1);
- Three ranges of priorities: 1-5,1-10 and 1-20;
- Changing aging step (max value depending on the set).

TABLE 1. BASIC SET OF TASKS

No.	SET 1	SET 2	SET 3
1	9	6	1
2	2	10	2
3	6	3	3
4	8	8	4
5	2	5	5
6	4	8	6
7	7	10	7
8	9	7	8
9	9	5	9
10	4	8	10
11	5	5	1
12	6	5	2
13	4	9	3
14	9	3	4
15	8	8	5
16	3	10	6
17	7	5	7
18	2	2	8
19	1	10	9
20	2	7	10

2) Results

As the research was extensive, the results for two sets are presented in details: SET 1 of 107 tasks and SET 2 of 134 tasks - both for priority range of 1-5. Two cases are analysed in details (the average delay, the delay for consecutive tasks).

- **Average delay**

In Fig. 2 and Fig. 3, the averaged delay of all tasks in relation to aging step parameter is shown. Graphs are generalized and do not present the most essential data- for how long does the important tasks (with the high priority) have to wait for the execution.

What is more, when analyzing the graphs one can observe that the bigger number of tasks determines the longer average delay in the system. However, there is no clear consistency between the results for different length of processes. While the results are quite static for SET 1, for the second set of data delay time clearly rises gradually along with the aging step.

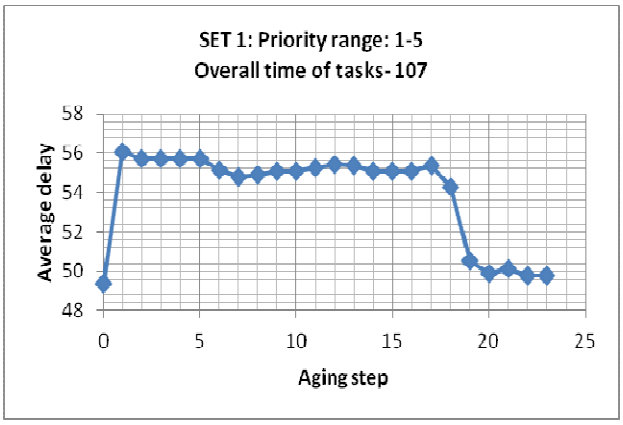

Figure 2. Average delay, SET 1, priorities 1-5.

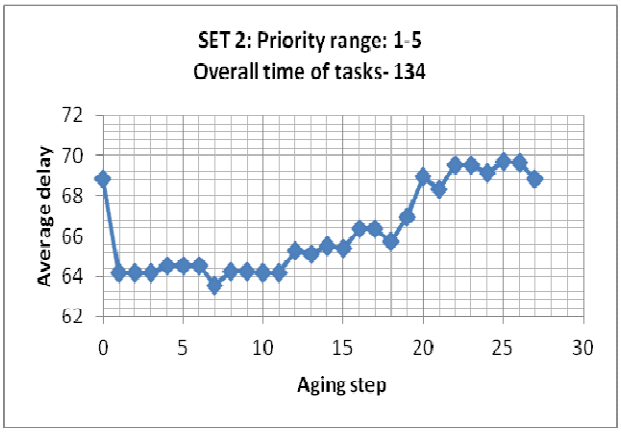

Figure 3. Average delay, SET 2, priorities 1-5.

Observing results presented in Fig.2 it can be noticed that the aging step of 20 has the same average delay that at the step of 0. It can be explained by reminding that no aging applied actually means that the value of aging step equals to a number of tasks in the system.

At the same time the best results are achieved by step 17 and equal to 14.1 cycles.

 ISBN: 978-1-61208-283-7

The shortest delay is observed for Fig.3 is the aging step of 7 and equals to approximately 64 cycles.

As no further conclusions can be drawn, more data is needed to analyse the priority problem. The outcome of this test can be fully reliable only when juxtaposed with the graph of delays for single tasks.

- **Delay for consecutive tasks**

The delay of consecutive tasks is a more complex and accurate way to present the results. The execution of a particular task can be observed and juxtaposed with the delay of other processes.

When analysing results (Fig.4) it appears that the bigger the aging step, the more rapid and uneven the graph. The maximum delay time is obviously the biggest for no aging applied, when the aging step is equal to a number of tasks (approx. 130 cycles for the task no. 11). The lower the aging step, the more the same task's delay decreases (by about 60 cycles in the minimum point).

When the aging step is big, the tasks are executed in an order of their priorities, no matter the size. It can cause the starving of the remaining processes (with lower importance).

On the other hand, small value of aging parameters cause that tasks are executed in the order of parameters but also causes a uniform distribution of processor's time. This solution may not be accepted, due to significant lowering the importance of the tasks.

3) Conclusion

To sum up the problem of choosing the optimal values of aging parameters, it can be said that the system must be well known, to model it properly. As it was observed on the above examination, both - set parameters and the length of tasks in the queue - have a high influence on the average time of tasks execution.

It appeared that the average delay of tasks for different aging parameters may not be accurate enough. On such representation, the data about the most important tasks is lost, so that the user may not have the good overall view of the system. It can help, however, when the large system is modelled and it is not possible to display characteristics of every task.

The more accurate and the more complicated at the same time is the graph representation of delay for every single task. It showed the remarkable influence of aging parameter on the system's time of processes execution. Due to this representation a connection between aging parameter and the accuracy of result (meant by the emphasis placed on priorities) was visible.

All in all, the means of choosing the best algorithm parameter is to find the best ratio between the aging parameter and pool of tasks' size, best fitted for the given requirements.

B Experiment #2 - Open pool of tasks

The second experiment was focused on open pool of tasks. Usually, in the real-life system, the new tasks are constantly coming to the processor, requiring its time. Therefore, the closed pool system is not accurate enough for modelling the tasks execution. It can only show the overall view of the problem. The cases considered in this research are the rough approximation of a real-life dynamic system, which is hard to be predicted and requires an on-line tasks management [9].

One point has to be remembered when it comes to simulation of an open loop system - the data randomness. Due to this "issue" each experiment has to be repeated and the data must be averaged for more reliability of results.

1) Experiment Design:

The research was very exhaustive. As there were 6 different algorithms and 216 unique combinations for three processors should be considered. For each experimentation point (basic tasks plus new tasks introduced to the system) all those combinations had to be computed three times. At the end the mean and the standard deviation had been calculated for the average delay of the system.

The overall results for every combination had been compared and 20 - best and worst (in terms of tasks' delay) -schemes of algorithms had been found.

The features of experiment design were taken as follows:

- 6 algorithms- along with preemptive ones
- 2 sets: 10 and 20 basic tasks;
- Open pool of 10 or 20 tasks;
- Basic parameters of a system;
- 3 repetitions of each experimentation point;
- Clock rates of CPUs- 2000Hz.

2) Results

The outcome is displayed on the bar graphs representing the average value of processes' delay in the modeled system along with the standard deviation of results. The considered algorithms are symbolized by:

- FC - concerns FCFS;
- PN - non-preemptive priority scheduling;
- PW - preemptive priority scheduling;
- SN - non-preemptive SJF;
- SW - preemptive SJF;
- RR - Round-Robin.

The exemplary results will be shown for the two cases analyzed in details:

Case 1 : 10 basic tasks plus 10 new tasks,

Case 2 : 10 basic tasks plus 20 new tasks.

 ISBN: 978-1-61208-283-7

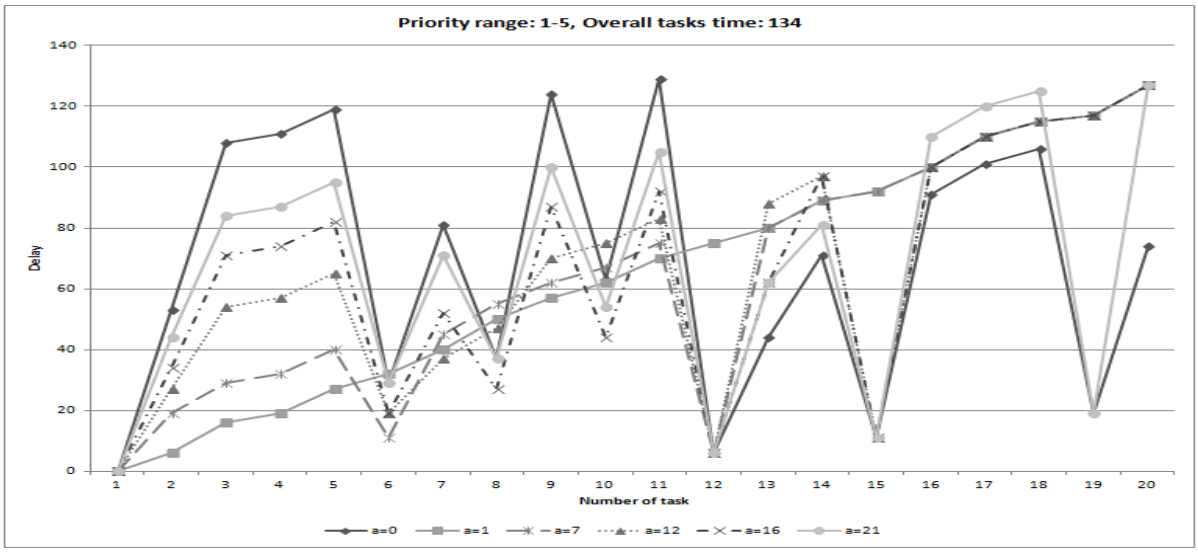

Figure 4. Delay for consecutive tasks; SET 2, priorities: 1-5.

- **Case 1: 10 basic tasks + 10 new tasks**

The outcome is shown in Fig.5. It can be observed that the best results for pool of 10 new tasks are achieved by mixed scheme of non-preemptive and preemptive SJF algorithm and equals to 2.65 cycles. The part of each best combination is the best SJF algorithm (either preemptive or non-preemptive). What can be surprising it is usually combined with the considered as not-so-good Priority and FCFS algorithms ([2] and [8]). Preemptive algorithm seems to be slightly better than non-preemptive one.

What more can be noticed, is that there is a very high variation of results, even for such a small pool of tasks. The deviation for best results shows that they are not so stable. So, it would be wiser to choose one of the 10 best combinations of algorithms, which have the lowest standard deviation.

The worst for this experimentation point is the combination of 2 Round- Robin algorithms with a preemptive priority scheduling and equals to 9.75 cycles. The result is more than three times worse that the best result. The least efficient of the bunch seems to be the Round-Robin algorithm, being a part of each combination with the biggest delays.

The variation of the bad result is not as high as for the good ones. It can be explained by the fact that the worst solutions usually give constant average delays while at the same time "good" combinations are similarly efficient.

- **Case 2: 10 basic tasks +20 new tasks**

The bigger the number of new tasks introduced to the system, the more consistent the outcome. It can be established by observing the standard deviation of results. There is also a bigger difference between the outcomes. The more efficient combination's average delay equals to 1.77 cycles. The worst outcome reaches 7.53 cycles, which is more than 4 times longer. When looking at the results for the best algorithm combination (Fig.6.), same tendencies as before can be observed. The lowest delays are obtained by the same algorithms. There are slight differences for the worst outcome. The main part of worst combination is the FCFS algorithm along with the Priority Scheduling. It is not possible to differentiate which is worse-preemptive or non-preemptive version of algorithm.

3) Conclusion

Because of the randomness of data introduced to the system, it is hard to find the one optimal solution for the given problem. The best or the worst outcome can only be approximated to a smaller group of possible algorithms. Because of the exhaustive research, it was hard to truly examine the problem. This part of a research could be an introduction for the further study. It gives the overall view of the best and worst algorithms.

V. CONCLUSION

The research presented in this paper allowed for only 'local' conclusions. At this stage of investigation, we can conclude about the remarkable influence of aging parameter on the system's time of processes execution and initially recommend combinations with SW for open pool of tasks, and not recommend combinations with RR from the minimizing average delay point of view.

The created 'Simulator Pro 3' is being used as teaching and research tool in Electronics Faculty, Wroclaw University of Technology, Poland. It still gives an opportunity for further investigations. In the nearer future, the various scenarios with Priority Scheduling is planned to be applied. What is more, as it was described in Section IV, the problem of open pool of tasks is very complex and needs far more investigation to fully explore it.

The authors of this paper are planning to perform studies with the several other scheduling algorithms, e.g., based on evolutionary ideas, which were proven their efficiency in [9] and [10], and in own works [11] and [12].

 ISBN: 978-1-61208-283-7

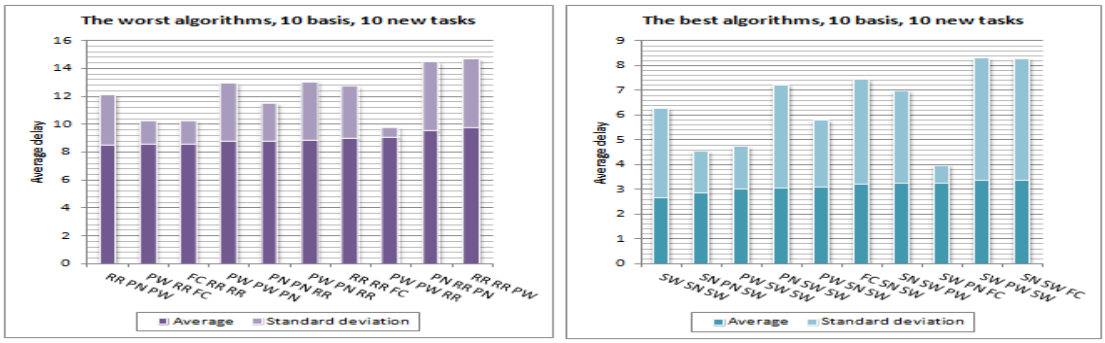

Figure 5. The worst and the best outcome for 10 basic and 10 new tasks set.

Figure 6. The worst and the best outcome for 10 basic and 20 new tasks set.

ACKNOWLEDGEMENT

This work was supported by the statutory funds of the Department of Systems and Computer Networks, Faculty of Electronics, Wroclaw University of Technology, No S20010W4.

REFERENCES

[1] A. Silberschatz, J. L. Peterson, and P. B. Galvin, Operating System Concepts, WNT, Warsaw, 2006.

[2] M. Respondek, "Trial testing efficiency of tasks execution in three-processor system", Technical Report, Faculty of Electronics W4K2, Wroclaw University of Technology, 2011.

[3] M. Respondek, "Evaluation of algorithms for tasks execution in three-processor system with priority scheduling.", Research Report, Faculty of Electronics W4K2, Wroclaw University of Technology, 2012.

[4] B. Czajka, S. Zagorski, and L. Koszalka, "Three-processor computer simulation. The shortest tasks execution optimization', Research Report, Faculty of Electronics W4K2, Wroclaw University of Technology, 2006.

[5] B. Czajka and I. Pozniak-Koszalka,: "Evaluation of tasks scheduling algorithms in multi-core and multi-queuing environments using system MESMS2", Proceedings to IARIA ICONS, IEEE CPS,2009, pp. 17-22.

[6] Process management, Multiprocessor systems. Available on: http://siber.cankaya.edu.tr [retrieved: November, 2011].

[7] F. Ramming, "Real time operating systems", University Paderborn, Heinz Nixdorf Institute, SBCCI'01. Available on http://www.inf.ufrgs.br/~flavio/ensino/cmp502/TutorialRammig [retrieved: August, 2012].

[8] E .O. Oyetunji and A. E. Oluleye, "Performance assessment of some CPU scheduling algorithms", Research Journal of Information Technology, Vol. 1, August 2009, pp. 22-26.

[9] V. Gaba and A. Prasha,, "Comparison of processor scheduling algorithms using genetic approach", International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 2, Issue 8, August 2012, pp. 37-45.

[10] D. M. Zydek and H. Selvaraj, "Fast and efficient processors allocation algorithm for torus-based chip multiprocessors," Journal of Computers & Electrical Engineering, Vol. 37, Issue 1, January 2011, pp. 91-105.

[11] D. Krol, D. Zydek, and L. Koszalka, "Problem independent approach to multiprocessor dependent task scheduling", Journal of Electronics and Telecommunications, Vol. 58, Issue 4, 2012, pp. 369-381.

[12] I. Poźniak-Koszałka, W. Proma, L. Koszałka, M. Pol, and A. Kasprzak, "Task allocation in mesh structure: 2Side leapFrog Algorithm and Q-learning based Algorithm", Lecture Notes in Computer Science, Springer, Vol. 7336, 2012, pp. 576-587.

 ISBN: 978-1-61208-283-7