Schedae Informaticae Vol. 24 (2015): 211–220
doi: 10.4467/20838476SI.16.018.4359

An Optimal Strategy of Resource Sharing in a Case of State-toggling Agents

Tomasz Wójtowicz
Institute of Computer Science and Computing Mathematics, Jagiellonian University
ul. Łojasiewicza 6, Kraków 30-348, Poland
e-mail:*tomasz.wojtowicz@ii.uj.edu.pl*

Abstract. This paper presents an optimal scheduling solution for a case of agents sharing a resource. The amount of resource can not satisfy all agents at once and in case of runout there is a penalty. Each agent randomly toggle its state between requiring and not requiring the resource. Using the knowledge of previous state and probability of change, the scheduling algorithm is able to calculate optimal number of concuring agents for one turn, that minimizes possibility of collision yet provides as much throughput as possible. Several different scheduling strategies are tested. The optimal solution adapts automatically to the value of probability of change. Further experiments show that optimality is retained if only the average probability of a set of agents is known. A case of practical application is provided.

Keywords: hard real-time systems, scheduling, shared resource, optimization, round-robin.

1. Introduction

Resource sharing is one of the most popular topic in computer science regarding economical usage of available resources. This topic has been covered in a number of books and resarch reports beginning with E. Dijkstra's famous *Dining philosophers* problem.

The problem presented in this paper regards agents requiring access to a common resource pool. The number of available resource items per one turn is obviously lower than the number of agents. The agents are unable to coordinate their requirements

nor there is no prior knownledge whether a given agent would require a resource item during upcomming turn. A situation where agents' requirement exceeds available resource creates a failure which induces waste in other resources as for instance time or energy. This type of problems is being researched at least since Kendall's paper [9]. The approach presented in that and following works assumes that a resource item is produced/consumed with a frequency derived from a distrubition. It means that at given point an agent consumes with a probabilty of x or abstains with a probabilty of $1 - x$. In this paper we are presenting a different approach. An agent has a state property. It indicates whether in the previous turn the agent was or was not in the state of consuming. The state can toggle in each turn with the probability of p (or remain the same with probability $1 - p$). This approach radically changes the meaning of probability in this class of problems, but also introduces the ability to forecast the behaviour of an agent based on its state from the previous turn.

This class of problems is important for distributed real-time systems [6, 2]. The resource that is essential is the available processing throughput per a segment of time. If the system operates in transactional mode, then either all requests are served or none. In case the requests overload the system causing failure, that segment of time is wasted for all of them. In this paper, a state-aware scheduling algorithm reduces the frequency of failures while keeping the running time at possible minimum.

There are several known papers regarding hard real-time scheduling, such as [7, 4, 3, 5, 2, 1]. In this paper the main concern is not exactly about processor throughput. Memory usage is the resource that responds to the concept of resource tokens in most cases, although occasionally processor time may be that resource as well.

Section 1.1. presents the model of state-toggling set of agents. Theoretical properties of this model are outlined in 2. Section 3. presents results of simulation which is followed by contruction of optimal scheduling solution in 3.3. then extended to chaotically behaving agents in 3.4. Section 4. presents a real-life application of this solution.

1.1. The model

Given is a set of N agents which independently perform their tasks. Each agent must complete T tasks in a possibly shortest time. Each task takes one round to complete. The shared resource is a set of tokens which agents may require to complete a particular task. In the discussed system the number of tokens is $L = \frac{N}{2}$, so only a half of the agents can use a token in each round. If the number L is exceeded, collision occurs and the round is wasted for all agents.

Each agent has a property which keeps information whether the agent was in state of using a token in the previous round. In each round the state can be changed with the probability of p. This specific behaviour is an original approach to the resource sharing problem, because solutions discussed in the common literature usually assume that p is a probability that an agent will be using the resource, whereas here it means that an agent will toggle it state. An algorithm may be introduced to govern agents

in order to minimize the number of collisions and time wasted therein. The algorithm operates by deactivating a certain agent. For a deactivated agent time still is running, only the agent does not operate, therefore a possible request for a token does not happen. This paper investigates several algorithms which allow a certain fraction of agents to operate at once, thereby reducing the number of possible collisions. Then an optimal algorithm is constructed upon gathered experience. The agents which completed their tasks also are deactivated, only the time is no longer running.

The **oblivious** algorithm allows all agents to operate simultaineously, therefore providing no protection against resource collision. Because no agent is put to inactive state, then for all agents equally the round will end as a success or failure. This is the unique property of the oblivious algorithm, that the average execution time is equal to maximal execution time. This so just solution is however slow – the average of 1000 runs of simulation shows that execution time is as high as $1.85T$, which means that almost every secound round ends in failure. On the other hand this algorithm can be noted for its efectiveness – in rounds that end successfully the average usage of tokens is 91.6%.

The opposite approach is called the **play-safe** algorithm. Because the number of available tokens is $L = \frac{N}{2}$, then the algorithm first disactivates half of the agents, allowing the active half to finish its job in T time, then it activates the other half what results in maximum execution time of $2T$. By diving the agents into priviliged and non-priviliged class, the execution can be done without any collisions, at the cost of twice larger execution time for the half of all. In such approach the argument that the average execution time has been lowered to $1.5T$ is about as usefull as conclusion that a human and a horse have on average three legs.

Studying the properties of the play-safe algorithm we learn that the average usage of tokens is 49.8%. This gives a clue of possible optimalisation, allowing a certain fraction of non-priviledged class to run along the priviledged class. Because the toggling process is stochastic, we can not guarantee that even just one supernumerary agent would not cause resource collision. We can though make that the collisions would not happen too often. If we denote by K the number of priviledged agents which requested token in the previous round, we can activate following number of non-priviledged agents:

$$A = r \cdot (L - K) \tag{1}$$

Note the importance of the coefficient r. If $r = 0$ then we have the ordinary play-safe algorithm. Because roughly half of tokens is used by the priviledged class, then $r \approx 3$ should provide an oblivious-like mode. The search for the most optimal value of r is a topic of Section 3. in this paper.

It is however important to note, that all non-priviledged agents must be run in a round-robin scheduling strategy. Otherwise, if just one agent is inactive during the first T rounds, then maximum execution time will be at least $2T$, invalidating any effort. Round-robin is the simpliest strategy that provides each agent with an equal opportunity to advance. The specific constraint $L = \frac{N}{2}$ is based on real-world

application of the model (see Section 4. for details). Proportion like $L = \frac{N}{3}$ would duplicate the problem (dividing agents into first priviledged class, second priviledged class and the rest), complicating the problem in a way that is actually irrelevant to the main matter.

2. Theoretical considerations

The state of the system can be expressed as a number of agents which require a token during given round. The system has therefore $N + 1$ states, which we will denote as $a_0 \dots a_N$. It is desired that states a_{L+1} or higher are entered as rarely as possible. If $x < y$ then we can calculate probability of the system changing state from a_x to a_y as:

$$\begin{aligned} P_{y|x} &= \sum_{i=0}^{x} \binom{x}{i} p^i (1-p)^{x-i} \binom{N-x}{y-x+i} p^{y-x+i} (1-p)^{N-y-i} \\ &= \sum_{i=0}^{x} \binom{x}{i} \binom{N-x}{y-x+i} p^{y-x+2i} (1-p)^{N+x-y-2i}. \end{aligned} \tag{2}$$

The formula is very complicated, besides it only provides a probability of state change. Also it is assumed that all agents have the same toggle probability of p. Would p be different for each agent, the complication becomes unsupportable. The point of this chapter is to show that this system has no usefull analytical solution. Simulation is the only practical way to determine the system's properties.

3. Simulation and results

3.1. Setup

The simulation was written in C language. Mersenne Twister library was used as the random number generator, because stdlib is untrustworthy in this matter. Each agent is represented as a structure containing: activation flag, state flag (whether was using token in the previous round), progress counter, all the necessary activity measurement variables and individual state-change probability if used in the given simulation.

In each round, at first it was decided which agents are requiring the token. The decision for an agent was based on the input of its state flag, global or individual

probability and a random number. The obtained number of agents which require the token was the decision point whether the round will end in success or failure. Then for each agent its variables were updated according to the occured set of events. The simulation ended when all agents have switched to the inactive state due to reaching the preset value of progress counter.

The main experiment was run with the following properties: agents $N = 100$, tokens $L = 50$, tasks to be completed $T = 100$. Each simulation was run 1000 times to ensure repetitiveness and the average was taken as the result.

3.2. Common toggle probability

In the first experiment, the toggle probability p was set the same for all agents and tested on values $[0.1, 0.9]$ step 0.1. The goal was to find the optimal value of r in the formula $A = r(L - K)$. First, the algorithms oblivious and play-safe were simulated, then a devised algorithm, which runs half the agents as the priviledged class and supplements the number of active agents from unpriviledged class in amount of formula, running them in a round-robin strategy. The value of r tested from 0.25 to 3.00 with the densiest increment of 0.05 between 0.9 and 1.75 where the most interesting results belong.

Figure 1 presents the average running time for the entire simulation. As the simulation was repeated 1000 times, the numbers presented here and in the following plots are the average of 1000 runs. There are 9 plots for $p \in [0.1, 0.9]$, only 3 are shown for better readability. As seen in this figure, the plots have a minimum point between 1.0 and 1.5 depending on the value of p. These points are the target points for the optimalisation discussed in this paper. Note that target points are well below the oblivious and play-safe results, therefore there is a profit out of this effort.

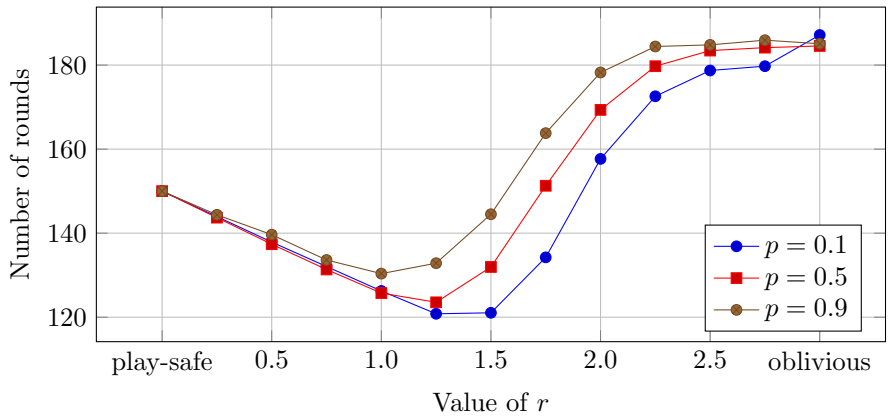

Figure 1. Average running time in relation to p and r.

Figure 2 presents the maximum running time, i.e. the time at which the last agent completed its tasks. The minimum points are shifted a little to the right in

comparison to the average time plots in Figure 1. Luckily they are not too far, so there is no need for trade-off. Again the minimum points are well below the results of the oblivious and play-safe algorithms, proving usefullness of this optimalization.

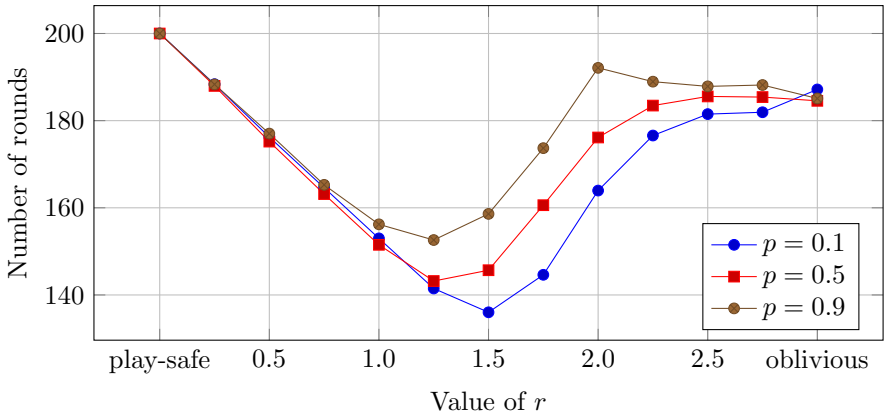

Figure 2. Maximum running time in relation to p and r.

Figure 3 presents the averaged number of collisions per simulation. All plots exhibit similar properties. At first they rise slow, in the middle the rise fast, perhaps exponentially, then again rise slow. The optimal points should remain in the first sector, before the exponential rise part. Interestingly, the target points for each value of p lay just at the end of the first sector.

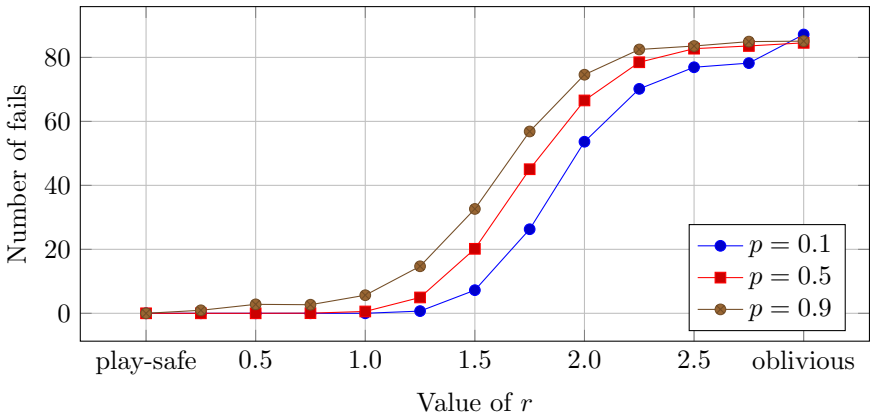

Figure 3. Number of fails for entire simulation in relation to p and r.

Figure 4 presents an averaged percentage of tokens used in each round. These plots do not take part in the optimization. Although we would like the utilization rate to be as high as possible, we are constrained with time as the prime factor. Interestingly these plots show that the more efficient usage of resources, the higher risk of failure, punished by the longer running time. The plot for $p = 0.9$ exhibits an unobvious behaviour around $r = 2.0$. Similar aberration is observed in plots for

$p = 0.8$ and $p = 0.7$ (not shown here). This otherwise interesting behaviour was not explained because it is not in scope of this paper.

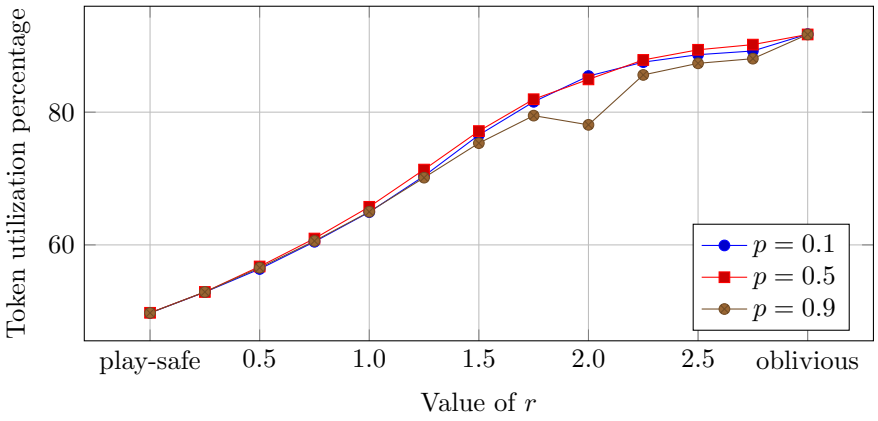

Figure 4. Percentage of token utilization in relation to p and r.

3.3. Optimal solution

The goals for optimization are: minimum average time, minimum maximal time, minimum number of fails, maximal utilization. As seen in the Fig. 1–4; the first three synergize while the last one does not. For this reason, utilization does not take part in optimization. The minimum number of fails is zero for play-safe algorithm and this not a satisfactory solution. Because of that, number of fails plays secondary role in optimization, only to ensure that the number is not too high. The intrinsic decision is whether average or maximum time should be the leading factor in optimization. Either of these can be used depending on needs. According to paper [6], in real-time systems the maximum time should be the leading factor. However, because in this problem maximum and average time correlate at over 85% (play-safe and oblivious excluded), the average time has been chosen as optimization factor. Additional benefit of this is that it lowers the number of fails, because its minimum points in the Fig. 1 correspond with smaller values in the Fig. 3.

As seen in the Fig. 1, the minimum average time depends on the value of p. The minimum points of all 9 plots were found. A linear function $r = 1.4375 - 0.375 * p$ has been found as the one which passes the minimum points with sufficient proximity, giving the ultimate version of Eq. (1):

$$A = (1.4375 - 0.375 * p)(L - K) \tag{3}$$

The above equation was simulated in the same conditions as previous simulation. The experiment has shown that it matches the best value of fixed r algorithms by $\pm 0.3\%$. It is assumed that this small discrepancy is a result that the minimum

points does not lay on a straight line but on a curve of some sort. It would require to interpolate a 9th deegree polynomial to follow that curve, yet it is absolutely unnecessary with such small slippage, therefore the Eq. (3) provides best practical solution to the problem where all agents share the same toggle probability.

3.4. Independent toggle probability

The previous simulations assumed that the value of p is identical for all agents throughout the entire simulation. That approach provided an important insight into the behaviour of the system and allowed to devise a optimal equation (3), which adapts to different values of p. However in real life cases each agent would have its own independent value of p which also would change in time. It is a hypothesis that the algorithm would retain its optimality if the average of all agents' property p is assigned as p in the Eq. (3).

The following experiment tested the proposed hypothesis. The value of p was randomized for every agent before each round. To make even more difficult and life-like, the random numbers were taken from uniform, not normal distribution. This way, due to $\sigma = \infty$ the entropy was maximalized. The experiment showed that the hypothesis is correct. The obtained results indicated that the average running time is only 0.22% worse than the previous experiment with fixed $p = 0.5$. The maximum execution time was only 0.67% worse, the number of fails increased by 2.9% and average utilization of tokens increased by 1.4%. With the uniform distribution the average value remained around 0.5. To ensure correctness over greater range of p, subsequent experiments tested distributions averaging on other values between 0.25 and 0.75 and they all came with coherent results.

This experiment proved that the proposed solution, only with knowing the average of p, can impose an optimal scheduling for agents behaving accordingly to their independent toggle probability.

4. Discussion

The presented algorithm has been successfully implemented in one of scientific laboratories in Poland. The laboratory is using a unique data aggregation engine, which probably conforms with the relational model of database, yet it works unlike any popular SQL-based database engine. The engine is able to provide a certain number of orthogonal locks both for reading and writing operations which guarantee a correct concurrency management. However if the number of locks is exceeded, all following operations are done in non-locking mode, what with 99% chance means data corruption. In this state all pending operations are cancelled, and the database is rolled back to the last coherent state, thus wasting all recent effort. It doesn't mean data

loss, because unsaved data remains with the client applications, but it does mean a substantial time waste.

At first, the database engine was operating like the oblivious algorithm. Once the growing number of clients exceeded the number of available orthogonal locks, the users begun to experience annoying data corruption breaks.The number of available locks could not have been easily increased because it would require more memory and faster processor to merge that more input in the same time. Because the database engine was specifically written for that operating system and hardware, nothing could have been replaced without enormous investment in a total recomputerisation including new client software. For this reason the management pursued the option of solving the problem in software. The number of clients never exceeded twice the number of locks ($L < N < 2L$), hence the assumption in the model that $L = \frac{N}{2}$.

The first attempt made by company's administrator implemented the play-safe algorithm. While it remedied the data corruption problem, the user's annoyance even grew for those whose client program was unluckly assigned to the unpriviledged group. The author has then devised an optimal scheduling algorithm using the same methodology as presented in this paper. As in the simulation, the average and maximum running time are decreased in comparison to both oblivious and play-safe strategies, while the rate of data corruption breaks is being kept at manageable level. With the achieved optimalisation, the system's performance apparently is within the tollerance threshold of a human user.

The author was not authorized to publish the name of the company nor any actual information regarding the real system for which the optimalisation algorithm was developed.

5. References

[1] Andersson B., Raravi G., *Real-time scheduling with resource sharing on heterogeneous multiprocessors,* Real-Time Systems, 2014, 50(2), pp. 270–314.

[2] Buttazzo G.C., Bertogna M., Yao G., *Limited Preemptive Scheduling for Real-Time Systems. A Survey,* IEEE Transactions on Industrial Informatics, 2013, 9(1), pp. 3–15, doi:10.1109/TII.2012.2188805.

[3] Saifullah A., Li J., Agrawal K., Lu C., Gill C., *Multi-core real-time scheduling for generalized parallel task models,* Real-Time Systems, 2013, 49, pp. 404–435.

[4] Shekhar M., Sarkar A., Ramaprasad H., Mueller F., *Semi-Partitioned Hard-Real-Time Scheduling under Locked Cache Migration in Multicore Systems,* 24th Euromicro Conference on Real-Time Systems (ECRTS), 2012, pp. 331–340, doi:10.1109/ECRTS.2012.27.

[5] Davis R.I., Burns A., *A survey of hard real-time scheduling for multiprocessor systems,* ACM Computing Surveys (CSUR), 2011, 43(4), doi:10.1145/1978802.1978814.

[6] Martyna J., *Distributed Hard Real-Time Systems: Notions and Performance Measures,* Zeszyty Naukowe UJ, Prace Informatyczne, 1998, 8, pp. 29–44.

[7] Tindell K., Clark J., *Holistic schedulability analysis for distributed hard real-time systems,* Microprocessing and Microprogramming, 1994, 40(2–3), pp. 117–134, doi:10.1016/0165-6074(94)90080-9.

[8] Silberschatz A., Peterson J.L., *Operating Systems Concepts,* Addison-Wesley, 1988, ISBN 0-201-18760-4.

[9] Kendall D.G., *Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain,* The Annals of Mathematical Statistics, 1953, 24(3), pp. 338–354.