A Review on Game Theoretic Scheduling

Shahzaib Waseem
B.Eng. Electronics Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
shahzaib.waseem@stud.hshl.de

***Abstract*—This document is a model and instructions for LaTeX.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Task scheduling is an essential aspect of operating systems, real-time systems, cloud computing, and many other areas in computer science. It is a means by which computer processes and their threads are allowed to utilize computer system resources such as CPU time, memory, I/O data lines, network connections, etc [3]. For a multi-core system, task scheduling is done to perform load balancing, which reduces the piled up workload on one single core by dividing said workload among the multiple cores in the system [3]. The need for task scheduling, especially for real-time systems, stems from the fact that more than one task or operation may need to be executed by a system, where all these operations are critical and necessary for proper functionality of said system. In such a scenario, all these operations must be performed not only accurately, but also timely [3]. This means that a set of tasks that a processor needs to execute must be scheduled in a way such that all the tasks are finished within their corresponding deadlines. In soft-real-time systems such as general-purpose computers where there isn't really a strict deadline, task scheduling is still beneficial for multitasking. Modern laptops and desktop computers allow users to have several internet browser tabs at once, on top of having a handful of other programs running in the background - task scheduling is powerful and computers use it all around us. Although task scheduling in computers is a topic in Computer Science, perhaps there may be a discipline of Mathematics that could help.

A field of Mathematics known as Game Theory is the study of strategic situations [1]. A strategic situation is one in which at least two decision-making elements or players equip themselves with multiple plans of participating in said situation. Such a strategic situation is called a 'game', and the plans of these players are akin to strategies such that the result of the strategic situation depends on said plans and actions of these participating players [1]. A game always specifies instructions or rules of the game that all players need to be aware of and follow to play - players formulate their strategies centered around these rules [1]. Any given game is composed of the following elements [1]:

- A finite set of players $P = \{1, 2, 3, ..., n\}$
- A finite set of 'reward functions' $R = \{r_1, r_2, r_3, ..., r_n\} \mid \forall\ p_n \in P\ \exists\ r_n \in R$
- A finite set of 'strategies' $S = \{s_1, s_2, s_3, ..., s_n\} \mid \forall\ p_n \in P\ \exists\ s_n \in S$
- A finite set of 'outcomes' or result of the game that is dependent on the strategies chosen by the participating players. The outcome determines the rewards for each player involved in the game

Games can also be categorized according to the following criteria: [1]:

- **Rationality:** the essence of playing the game for players is to maximize their gains and minimize their losses. To do this, players take actions that lead them to obtain rewards. Strategies in a game are then defined as these actions corresponding to and made by each player in the game. Players in the game develop strategies revolving around making rational decisions keeping in mind the rules of the game, such that the outcome of the game results in their win.
- **Cooperation:** players involved in a given game may or may not act in the interest of one another. When two or more players work together to increase their rewards compared to that of their individual efforts, a joint strategy is developed. A motivating factor for cooperation may be rewards for all cooperating players.
- **Zero-summation of rewards:** the outcome of a game like football is such that one player or team wins while the other player or team loses. If the rewards of the players in the game are added together, the total sum is zero - the individual rewards of players are either positive or negative depending on their win or loss. This is called a zero-sum game. Interestingly, majority of 'games' in the real world are non-zero-sum games. This means that it is possible for all players to win or lose simultaneously. Examples of such cases include political situations, a boxing match, etc.

There exists a popular non-zero-sum, cooperative game known as 'The Prisoner's Dilemma'. In this game, two prisoner's are charged with the same crime, and are confined in different cells. The police officers in charge of their investigation expect each of the two prisoners to either confess to the crime, or to not confess, resulting in a total of 4 different outcomes. If both of them do not confess to the crime, they both receive a reduced sentence of 1 month. If both of them

confess to the crime, they both get sentenced to 5 years in prison. If only one of them confesses to the crime, and the other prisoner chooses not to confess, then the one who confessed is let go to be free while the other is arrested and sentenced to 10 years in prison. Given these circumstances, game theory seeks to study whether it is rational to confess or not to confess. [1]

The reward matrix below illustrates the possible outcomes and corresponding rewards for each player in the Prisoner's Dilemma:

	C	D
C	(-0.083, -0.083)	(-10, 0)
D	(0, -10)	(-5, -5)

The matrix above denotes C for cooperating, meaning that a given prisoner does not confess. Defecting is denoted as D, meaning that a given prisoner confesses. As shown in the matrix, we can see that the best strategy for both of the prisoner's to maximize both their rewards is for both of them to cooperate. However, the two prisoners do not know whether the other prisoner will cooperate, or defect. If one prisoner cooperates, there is a risk that the other prisoner defects, leading to the worst case scenario for the cooperator. This is what makes the Prisoner's Dilemma interesting and tricky. Any player in any game does not know for sure, with absolute 100% certainty what is going to be the next move of any other player.

II. Model-based design of Game Theoretic Scheduling

This paper aims to describe two-player game theoretic scheduling approach in a cloud computing environment, as discussed in [2]. In such an environment, there are three main elements: virtual machines, schedulers, and tasks [2]. Described below is a description of each sub-model:

- **Task model**

  Let T be the set of independent tasks in the game such that $T = \{t_1, t_2, t_3, ..., t_n\}$. Each task in T is described in a way that $t_i = \{a_i, e_i, d_i\} \mid 1 \leq i \leq n$ where a_i, e_i, d_i are the arrival times, execution times, and deadlines of the task, respectively. The time at which a task enters the ready-queue of a computer is known as its arrival time [2]. The time at which a task gets assigned to a virtual machine is denoted as h_i. The waiting time w_i of a task is the difference of its assignment time and its arrival time, described by $w_i = h_i - a_i$ [2].

- **Virtual machine model**

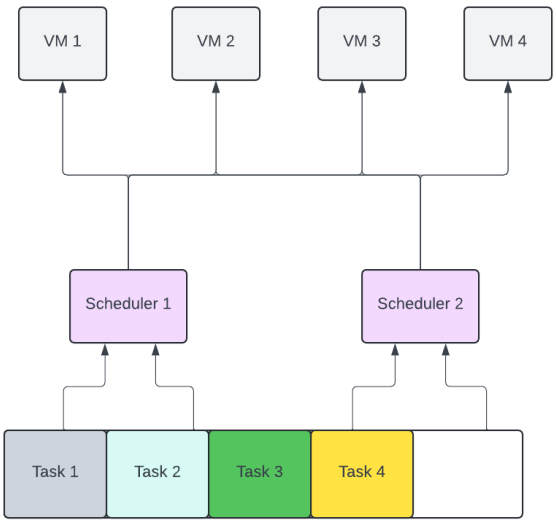
  Cloud environments function by using virtual machines to execute tasks remotely. For this reason, in our model, let V be the set of virtual machines described by $V = \{v_1, v_2, v_3, ..., t_m\}$ such that $v \in V$ provides computing capabilities and resources for a task to be executed [2]. The total number of virtual machines in the model described in this paper have to be at least equal to the number of tasks. If n is the total number of tasks, and m is the total number of virtual machines, then the following must hold true for the system: $m \geq n$, where $m \in \mathbb{N}^*$ [2].

- **Scheduling model**

  The purpose of a scheduler in our model is to receive two tasks as input, and generate two outputs, each corresponding to an assignment of a task to any one virtual machine in the model [2]. The scheduler first sorts the incoming tasks in an ascending order with respect to their deadlines, and then selects two tasks at a time for assignment to virtual machines. Since two tasks will be taken at a time, the total number of tasks in our model must be even. For this reason $n = 2 \cdot k \mid k \in \mathbb{N}^*$. The total number of schedulers h is described as $1 \leq h \leq \frac{n}{2}$, so that all schedulers in the system are always utilized and busy [2].

An example of the overall model used for game theoretic scheduling with four tasks, two schedulers, and four virtual machines is shown in Figure 1 [2].

Fig. 1. Overall model example of game theoretic scheduling



In this paper, cooperation is explored completely; a cooperative game is analyzed along with a non-cooperative game [2]. All the necessary aspects of the game related to game theory are mapped to the elements in the model as follows:

- **Players:** Any given task in the model acts as a player. Referring back to the set of all players in the game P, and the set of all tasks in the model T, the following holds true for all players and tasks:

$$P = \{p_1, p_2, p_3, ..., p_n\}$$
$$T = \{t_1, t_2, t_3, ..., t_n\}$$
$$p_i \in P \mapsto t_i \in T \mid 1 \leq i \leq n$$

- **Strategies:** All virtual machines in the model act as strategies [2]. The reasoning behind this is that assignment to a virtual machine can be thought of as an action corresponding to a task. This action of being assigned to a specific virtual machine may have varying waiting times associated to that task. Referring back to the set of all strategies S in the game, and the set of all virtual machines in the model V, the following holds true for all strategies and virtual machines:

$$S = \{s_1, s_2, s_3, ..., s_m\}$$
$$V = \{v_1, v_2, v_3, ..., v_m\}$$
$$s_i \in S \mapsto v_i \in V \mid 1 \leq i \leq m$$

- **Rewards:** Since every task in our model is equivalent to a player in a game, it follows that the task must be rational. One possible quantifiable metric that a task could use in order to be rational is its waiting time [2]. If W is the set of all waiting times associated to all tasks in the model, and the set R contains all rewards in a game, then the following holds true for all waiting times, and rewards:

$$R = \{r_1, r_2, r_3, ..., r_n\}$$
$$W = \{w_1, w_2, w_3, ..., w_n\}$$
$$r_i \in R \mapsto w_i \in W \mid 1 \leq i \leq n$$

- **Rewards matrix:** A rewards matrix describes all the possible rewards a player can get for all possible combinations of each their actions [2]. If the difference between two rewards of two competing players p_1 and p_2 is described as

$$r_j - r_{j+1}$$

where $j \in \mathbb{N}$, r_j corresponds to the reward of p_1 and r_{j+1} corresponds to the reward of p_2, then the expression being positive would imply that the player p_1 won, and the other way around for the expression being negative with the winning player being p_2 having the reward r_{j+1} [2].

In a two-player game such as one that is described and used in this paper, the two players are rational, or play the game rationally using their corresponding rules as described below [2]:

- The first player, p_1, is the so-called maximizer. This means that p_1 will always choose a strategy that results in the maximum possible reward for it. This means that, given all its possible strategies, p_1 will choose the best worst-case scenario by selecting the strategy with the maximum reward, in a set of minimum rewards. This strategy would the best move for p_1 in the game, given a rewards matrix, to ensure that it receives the least-hindering or least-damaging reward among a collection of such possible unwanted rewards [2].
- The second player, p_2, is the so-called minimizer. This means that p_2 will always choose a strategy that results in the minimum possible reward for it. This means that, given all its possible strategies, p_2 will choose the best worst-case scenario by selecting the strategy with the minimum reward, in a set of maximum rewards. This strategy would the best move for p_2 in the game, given a rewards matrix, to ensure that it receives the least-hindering or least-damaging reward among a collection of such possible unwanted rewards [2].

In this section, we will analyze a game with the following constraints for both cooperative, and a non-cooperative setting:

- **Number of players:** 2
- **Number of strategies:** 4

First, we will consider an example of a non-cooperative game. In this non-cooperative game, the following rewards matrix is considered using random values. In the real-world scenario where a computer would actually implement this game theoretic scheduling approach, the actual waiting times would be calculated and used as the rewards instead, using the tasks' arrival times, execution times, etc.

p_1 \ p_2	v_1	v_2	v_3	v_4
v_1	8	10	-4	2
v_2	-9	-6	7	1
v_3	-10	-6	-2	-3
v_4	9	6	-8	4

Each row in the matrix above describes the strategies that p_1 would use in the game, where each reward in this row is a result of p_2 using every possible strategy, given p_1's strategy, or in response to this strategy of p_1. The same applies the other way around for p_2 where instead of every row, every column describes p_2's strategy, and every reward in this column is a result of p_1 making every possible strategy in response to this strategy of p_2. For example, if p_1 uses the strategy v_1, and p_2 uses the strategy v_2, then the normalized or net reward would be a 10, resulting in a huge win for p_1 [2].

After p_1 and p_2 maximize and minimize their rewards, respectively, the following rewards matrix is obtained from each player's best worst-case rewards selections [2]:

p_2 / p_1	v_1	v_2	v_3	v_4	Worst reward p_1
v_1	8	10	-4	***2***	**-4**
v_2	-9	-6	7	1	-9
v_3	-10	-6	-2	-3	-10
v_4	9	6	-8	4	-8
Worst reward p_2	9	10	7	**4**	

The matrix above has been updated to allow for p_1 and p_2 to find the worst reward for each of their strategies corresponding to all possible strategies their opponent can take. These worst rewards are selected and included in an additional column and row in the matrix with bold text-formatting. For example, in the scenario that p_1 chooses v_2, there are different rewards depending on which strategy p_2 chooses. p_2 choosing v_1 results in a net reward of 8, 10 for v_2, -4 for v_3, and 2 for v_4. Out of all these rewards, the worst-case reward for p_1 is -4, since p_1 is a maximizer. p_1 carries out this procedure for all its possible strategies, and chooses the greatest value in the "Worst-reward p_1", since this is the best option for p_1 as a maximizer. p_2 carries out this procedure accordingly as the minimizer. In the matrix above, the best-worst reward of p_1 is -4, and the best-worst reward of p_2 is 4. The intersection cell of both these values is a bold-italic text-formatted value of 2, which describes that the scheduler should assign t_1 to v_1, and t_2 to v_4. This is how a non-cooperative game theoretic approach can be applied for assigning tasks to virtual machines or any core [2].

A cooperative game on the other hand handles computations of reward values differently. A cooperative game implies that the two players are working together to find the combination of strategies that benefits both of them equally. This means that a rewards matrix for a cooperative game must contain cells with non-negative numbers by using the absolute values of the differences of the two players' individual waiting times [2]. Finally, since both players in a game are rational, this cooperative game setting makes it so that they both seek an equal reward by refusing to allow the other player to receive more reward than themselves - the difference between their rewards should be as low as possible [2]. An example of a rewards matrix for a cooperative game is shown below:

p_2 / p_1	v_1	v_2	v_3	v_4
v_1	6	10	13	3
v_2	14	7	8	12
v_3	6	**2**	9	10
v_4	12	5	11	4

Referencing the rewards matrix above, the least value of 2 corresponds to the strategy v_3 for p_1, and v_2 for p_2. Hence, the scheduler performs these assignments accordingly for the two tasks.

III. CDFG of Game Theoretic algorithm

A. Algorithm

```
struct Task {

    int id;
    int arrivalTime;

};

struct VM {

    int currentLoad;
    int inputTaskID;

};


int rewardsMatrix[NUM_OF_VMS+1][NUM_OF_VMS+1];

struct VM virtualMachines[NUM_OF_VMS];

void initializeRewardsMatrix(struct Task* t_1,
struct Task* t_2){

    int task_1_waitingTime = 0;
    int task_2_waitingTime = 0;

    for(int i = 0; i < NUM_OF_VMS; i++){
        for(int j = 0; j < NUM_OF_VMS; j++){

            task_1_waitingTime =
            virtualMachines[i]
            .currentLoad - t_1->arrivalTime;
            task_2_waitingTime =
            virtualMachines[j]
            .currentLoad - t_2->arrivalTime;

            rewardsMatrix[i][j] =
                abs(task_1_waitingTime
                - task_2_waitingTime);
        }
    }

}


void assignVMsToTasks(struct Task* t_1,
struct Task* t_2){

    int min = 100;
    int task1_VM_index;
    int task2_VM_index;

    for(i = 0; i < NUM_OF_VMS; i++){
```

```
        for(int j = 0; j < NUM_OF_VMS; j++){

            if(rewardsMatrix[i][j] < min){
                min = rewardsMatrix[i][j];
                task1_VM_index = i;
                task2_VM_index = j;
            }

        }
    }


    virtualMachines[task1_VM_index].
    inputTaskID = t_1->id;
    virtualMachines[task2_VM_index].
    inputTaskID = t_2->id;

}
```

B. CDFG

Fig. 2. CDFG

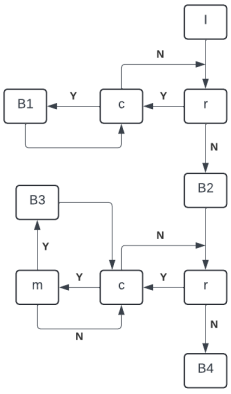


Fig. 3. B1

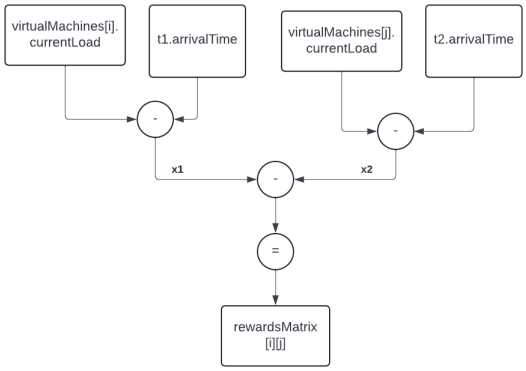
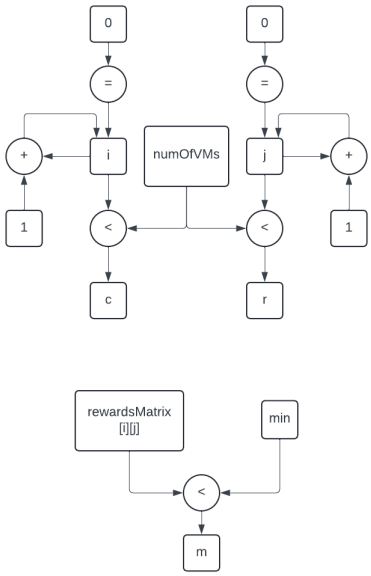


Fig. 4. Loop variables r, c, m



IV. Game theoretic scheduling

A. Algorithm

The algorithm to be used for game theoretic scheduling is shown in Listing 1 []. It is slightly modified in order to facilitate one possible approach to game theoretic scheduling shown in this paper.

```
diffeq {
read(x, y, u, dx, a, n);
do {
        xl = x + dx;
        ul = u - (3*u*x*dx) - (3*y*dx);
        yl = y + u*dx;
        nl = n + 1
        c = xl < a
        x = xl;
        u = ul;
        y = yl;
        n = nl;
} while(c)
write(y, n)
}
```

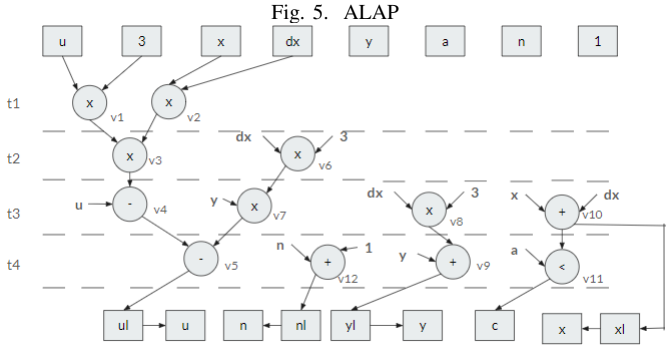
Listing 1. Algorithm used for Game Theoretic Scheduling

B. CDFG and Scheduling

Figure 5 shows ALAP scheduling for the algorithm shown in Listing 1. The resource constraints for this scheduling are as follows:

- Multipliers: 4

Fig. 5. ALAP



- Adders: 2

Fig. 6. ASAP

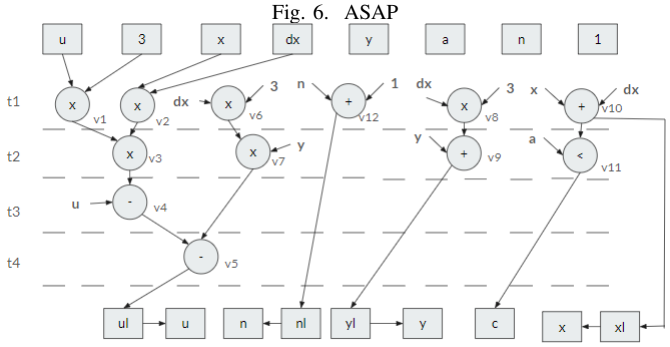


Figure 6 shows ASAP scheduling for the algorithm shown in Listing 1. The resource constraints for this scheduling are as follows:

- Multipliers: 2
- Adders: 4

The mobility values of operations that are more than or equal to 1 are as follows:

- V6 = 1
- V7 = 1
- V8 = 2
- V9 = 2
- V10 = 2
- V11 = 2
- V12 = 3

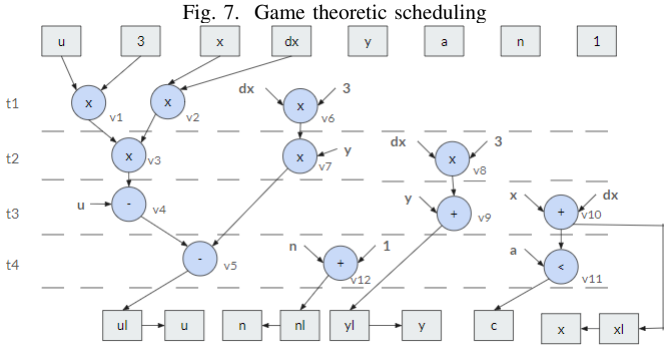
In game theory, as described in the previous sections, a cooperative game is one in which players of the game work together on a joint strategy that benefits all of them equally. Considering such a cooperative game, the table below describes the mapping of concepts in Game Theory to that of HLS Scheduling:

Game Theory	**HLS Scheduling**
Players	Each clock cycle
Payoff	Each operation
Strategy	Total operations per clock cycle

As the table describes the strategy to be the total operations per clock cycle, and the payoffs as each operation, the players or clock cycles themselves would prefer to divide the workload of performing these operations equally among themselves. The players seek to minimize the payoffs in this game, since the payoff is essentially punishment in the form of workload. Hence, they seek to receive as little workload as possible, while at the same time, cooperating with each other as to not be a push-over by receiving more workload than another player.

The HLS scheduling according to game theory, or simply put, game theoretic scheduling of the algorithm described in Listing 1 is shown in Figure 7. It can be seen that there are in total 12 operations, and 4 clock cycles. These operations can be seen in the figure as equally divided as 3 operations per clock cycle.

Fig. 7. Game theoretic scheduling



The resource constraints for this game theoretic scheduling are as follows:

- Multipliers: 3
- Adders: 3

By comparing the resource constraints of for game theoretic scheduling with that of ALAP and ASAP, it can be seen that the maximum number of multipliers and adders has lowered from 4 to 3, and hence, optimization has been achieved.

REFERENCES

[1] Bellal Ahmed Bhuiyan. An overview of game theory and some applications. *Philosophy and Progress*, 59(1-2):111–128, 2018.

[2] Manoj Kumar Patra, Sampa Sahoo, Bibhudatta Sahoo, and Ashok Kumar Turuk. Game theoretic approach for real-time task scheduling in cloud computing environment. In *2019 International Conference on Information Technology (ICIT)*, pages 454–459. IEEE, 2019.

[3] Abraham Silberschatz. Operating system concepts. *The Journal of Supercomputing*, 2008.