FINDING TIME QUANTUM OF ROUND ROBIN CPU SCHEDULING ALGORITHM IN GENERAL COMPUTING SYSTEMS USING INTEGER PROGRAMMING

Samih M. Mostafa[1], S. Z. Rida[2] & Safwat H. Hamad [3]

[1, 2] Faculty of Science, Mathematics Department, South Valley University, Qena, Egypt.
[3] Faculty of Computer & Information Sciences, Ain Shams University, Abbassia, Cairo, Egypt.

ABSTRACT

In Round Robin Scheduling, the time quantum is fixed and then processes are scheduled such that no process get CPU time more than one time quantum in one go. If time quantum is too large, the response time of the processes is too much which may not be tolerated in interactive environment. If time quantum is too small, it causes unnecessarily frequent context switch leading to more overheads resulting in less throughput. In this paper, a method using integer programming has been proposed to solve equations that decide a value that is neither too large nor too small such that every process has reasonable response time and the throughput of the system is not decreased due to unnecessarily context switches.

Keywords: *CPU Scheduling, Residual Time, Cyclic queue, Survived tasks*

1. INTRODUCTION

Round Robin (RR) is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in circular order ([13], [2], [3], [4], [5]), handling all processes without priority ([22], [23], [24]), arguably, the major issue in RR is the time slice ([19], [16], [4], [14]). Round Robin scheduling is both simple and easy to implement, and starvation-free. Different variants of Round Robin scheduling can be applied to other scheduling problems ([6], [11], [12], [15], [17], [21], [25], [20]), such as data packet scheduling in computer networks ([7], [28]). Effectiveness and efficiency of RR are arising from its low scheduling overhead of $O(1)$, which means scheduling the next task takes a constant time ([8], [7], [1]). In computer science, a scheduling algorithm is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance a system effectively or achieve a target quality of service. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously). Scheduling is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. Scheduling algorithms have been found to be NP-complete in general form (i.e., it is believed that there is no optimal polynomial-time algorithm for them [27]). Software known as a scheduler and dispatcher carry out this assignment.

The **scheduler** is concerned mainly with:

- *CPU utilization* - to keep the CPU as busy as possible.
- *Throughput* - number of processes that complete their execution per time unit.
- Turnaround - total time between submission of a process and its completion.
- *Waiting time* - amount of time a process has been waiting in the ready queue.
- *Response time* - amount of time it takes from when a request was submitted until the first response is produced.
- *Fairness* - Equal CPU time to each thread.

There are two types of general purpose computing systems [9], batch processing and interactive systems. Batch processing is execution of a series of programs ("jobs") on a computer without manual intervention. Batch jobs are

set up so they can be run to completion without manual intervention, so all input data is preselected through scripts or command-line parameters. This is in contrast to "online" or interactive programs, which prompt the user for such input. A program takes a set of data files as input, processes the data, and produces a set of output data files. This operating environment is termed as "batch processing" because the input data are collected into batches on files and are processed in batches by the program.

Despite their long history, batch applications are still critical in most organizations. While online systems are now used when manual intervention is not desired, they are not well suited to the high-volume, repetitive tasks. Therefore, even new systems usually contain a batch application for cases such as updating information at the end of the day, generating reports, and printing documents.

Modern batch applications make use of modern batch frameworks such as Spring Batch, which is written for Java, to provide the fault tolerance and scalability required for high-volume processing. In order to ensure high-speed processing, batch applications are often integrated with grid computing solutions to partition a batch job over a large number of processors. Either preemptive or non-preemptive can be used in batch environment.

As known, that **Integer Programming** (**IP**) problem is any mathematical optimization or feasibility program in which some or all of the variables are restricted to be integral. In many settings, the term *integer program* is used as short-hand for integer linear programming. The general mixed *integer programming problem* is to:

$$Minimize\ z = cx$$

$$\text{Subject to}$$

$$(1) \quad x_j \text{ integer } j \in J \subseteq \{1,\ldots,n\}, \quad J \neq \phi,$$

$$(2) \quad x_j \geq 0, \qquad j = 1,\ldots,n,,$$

$$(3) \quad Ax = b.$$

In this paper, a method using integer programming problem has been proposed that decides a value that is neither too large nor too small such that every process has got reasonable response time and the throughput of the system is not decreased due to unnecessarily context switches. This method depends on changing time quantum in each round over the cyclic queue; we call this method as CTQ technique. In the following sections, we will define the CTQ terminology and explain its implementation.

2. CTQ DEFINITIONS

To provide a more in depth description of CTQ, we first define more precisely the state CTQ associates with each round, and then describe in detail how CTQ uses that state to schedule tasks. We define the terminology list we use in table 1.

Table 1. CTQ Terminology.

T_i	Task i.
$NTQ[T_i] = NTQ_i$	The number of times the task T_i exploits the time quantum TQ.
$BT[T_i] = BT_i$	The burst time of the task T_i.
TQ	The time quantum.
N	The number of the tasks.
$SLTQ[T_i]$	The starting of the last time quantum of T_i.
$WT[T_i]$	The waiting time of task T_i.
TWT	The total waiting time of all tasks.
$AVGWT$	The average waiting time of the tasks in the run queue.
$RST[T_i]$	The residual time of T_i.

The following equations determine the time quantum TQ that gives the smallest average waiting time in each round.

$$NTQ[T_i]=\begin{cases}\left\lfloor \frac{BT[T_i]}{TQ} \right\rfloor^{1} & if \quad BT[T_i] \neq l*TQ \\ & l=1,2,3,\ldots \\ \frac{BT[T_i]}{TQ}-1 & if \quad BT[T_i]=l*TQ \\ & l=1,2,3,\ldots \end{cases} \tag{1}$$

Table 2 exhibits an example, in which each task with its burst time:

Table 2. Example 1.

TASK ID	BURST TIME
T1	24
T2	3
T3	3

If we use a time quantum of 4 *ms*, therefore, the Gantt Chart is:

	T1	T2	T3	T1	T1	T1	T1	T1
0	4	7	10	14	18	22	26	30

So, the $NTQ[T_1]$ is 5, the $NTQ[T_2]$ is 0, and the $NTQ[T_3]$ is 0, although the number of context switches of T_1 is 1, the number of context switches of T_2 is 0, and the number of context switches of T_3 is 0.

$$SLTQ\ [T_i]=\begin{cases} 0+\sum_{k=1}^{i-1}\begin{cases} TQ & if \quad NTQ_k>0 \\ BT_k & if \quad NTQ_k=0 \end{cases} & if \quad NTQ_i=0 \\ NTQ_i*TQ+ & \\ \quad \sum_{k=1,\ k\neq i}^{n}\begin{cases} BT_k & if \quad NTQ_k<NTQ_i \quad and \quad k\neq i \\ BT_k & if \quad NTQ_k=NTQ_i \quad and \quad k<i \\ (NTQ_i*TQ) & if \quad NTQ_k\geq NTQ_i \quad and \quad k>i \\ (NTQ_i+1)*TQ & if \quad NTQ_k>NTQ_i \quad and \quad k<i \end{cases} & if\ NTQ_k>0 \end{cases} \tag{2}$$

In the above example, the $SLTQ[T_1]$ is 26, the $SLTQ[T_2]$ is 4, and the $SLTQ[T_3]$ is 7.

$$WT[T_i]=SLTQ[T_i]-NTQ[T_i]*TQ \tag{3}$$

$$TWT=\sum_{i=1}^{n}WT[T_i] \tag{4}$$

$$AVGWT=TWT/n \tag{5}$$

In this paper, an efficient algorithm has been formulated to find an *optimal integer solution* of the given system of equations with respect to a given linear objective function. Therefore, the problem can be represented as:

$$\begin{aligned} &Min \\ &\quad Equation\ (5) \\ &s.t. \\ &\quad Equations\ (1),(2),(3),and\ (4) \end{aligned} \tag{6}$$

[1] $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x.

3. THE CHANGEABLE CONSIDERATION

CTQ combines the benefit of low overhead round-robin scheduling with low average response time and low average waiting time, this depends on the size of the preselected time quantum. If we have n tasks in a round $r1$ and m tasks that have burst times equal to or less than the time quantum used in $r1$, then there are $n-m$ tasks in the next round, where $n \geq m$. The residual time of the task T_i in the round number q is determined from the equation:

$$RST[T_i] = BT[T_i] - \sum_{k=1}^{q-1} TQ[k] \qquad (7)$$

where $TQ[k]$ is the time quantum in round number k. In each successive round we solve the equations with respect to the residual times of the survived tasks.

3.1. Illustrative Counter Examples

To demonstrate the previous consideration we will cosider this example, in which each task with its execution time and arrival time as shown in table 3. Tables 4 and 5 give the results of RR policy and CTQ policy, where the time quantm used in RR is 50 *ms.*

Table3. Example 2.

TASK ID	BURST TIME	ARRIVAL TIME
T1	23	0
T2	75	20
T3	93	22
T4	48	50
T5	2	55

Table 4. Round-Robin policy of Example 2.

Job ID	Service Time	Arrival Time	Start Time	Finish Time	Preemption	Turnaround Time	Waiting Time
T1	**23**	0	0	23		23	0
T2	**75** 25	20	23 173	73 198	*end of quantum;* T3 *starts*	178	103
T3	**93** 43	22	73 198	123 241	*end of quantum;* T4 *starts*	219	126
T4	**48**	50	123	171		121	73
T5	**2**	55	171	173		118	116
Mean						131.8	83.6

Table 5. CTQ policy of Example 2.

Job ID	Service Time	Arrival Time	Start Time	Finish Time	TQ			Preemption	Turnaround Time	Waiting Time
					R1	R2	R3			
T1	**23**	0	0	23	23				23	0
T2	**75** 37	20	23 149	61 186		38	50	*end of quantum;* T3 *starts*	166	91
T3	**93** 55	22	61 186	99 241				*end of quantum;* T4 *starts*	219	126
T4	**48**	50	99	147					97	49
T5	**2**	55	147	149					94	92
Mean									119.8	71.6

In what follows, the number in parentheses in the comment field is the remaining service time for the process. In order of execution:

Time	Ready Queue	Time Quantum	Comments
0	T1	TQ = 23	T1(23) arrives, run
20	T1, T2		T2(75) arrives and is appended to the queue, T1(3) continues to run
22	T1, T2, T3		T3(93) arrives and is appended to the queue, T1(1) continues to run
23	T2, T3	TQ = 38	T1(0) finished, so T2(75) runs
50	T2, T3, T4		T4(48) arrives and is appended to the queue, T2(48) continues to run
55	T2, T3, T4, T5		T5(2) arrives and is appended to the queue, T2(43) continues to run
61	T3, T4, T5, T2		The quantum expires, so T2(37) moves to the end of the queue and T3(93) runs
99	T4, T5, T2, T3	TQ = 50	The quantum expires, so T3(55) moves to the end of the queue and T4(48) runs
147	T5, T2, T3		T4(0) finished, so T5(2) runs
149	T2, T3		T5(0) finished, so T2(37) runs
186	T3		T2(0) finished, so T3(55) runs

4. SIMULATION STUDIES

In this section, we are interseted in studying three performance metrics: average waiting time, average turnaround time, and number of context switches. We simulated RR and CTQ to observe the those metrics. The simulation environment is characterized by a balanced mix of CPU-bound and I/O-bound processes that is, half of the processes are CPU-bound and the rest half are I/O-bound processes based on the processor-sharing model [10]. We assume that CPU-bound processes perform no I/O operations whereas in I/O-bound processes, the I/O occurs exponentially within the CPU time. We built a task generator routine to generate the task sets. Task set contains set of tasks and each task in the task set is a tuple: <*task_id*, *arrival_time*, *CPU_time*, (*I/O_occurence*, *I/O_wait*)*,…,* (*I/O_occurence*, *I/O_wait*)>. The task arrival was modeled as a Poisson random process. Hence, the inter-arrival times are exponentially distributed. An task arrival generator was developed to take care of the process of random arrival of different tasks to the system. The generator produces the inter-arrival times utilizing some specific mean (arrival intensity) of the distribution function.

Let, X be an exponentially distributed random variable. So, using the Inverse Transformation method:
the pdf :

$$f(x) = \lambda e^{-\lambda x} \tag{8}$$

and the CDF:

$$F(x) = 1 - e^{-\lambda x} = u \tag{9}$$

or,

$$x = -\frac{1}{\lambda}\ln(1-u) \tag{10}$$

where $1/\lambda$ is the mean of the exponential distribution, *CPU_time* is generated from uniform distributed for a given mean, and *IO*_occurrence and *IO_wait* times are generated from exponential distribution for a given mean and standard deviation. So, the inputs for the process generator are:

- number of prosesses, N,
- mean values for Poisson and uniform distributions, and
- mean and standard deviation for exponential distribution.

To profit from knocking out short jobs relatively faster in a hope to increase the throughput and reduce the average waiting time [14, 25], we modified the proposed algorithm by sorting the tasks in each round in ascending order.

4.1. Experimental Setup and Results

The parameters for simulation are set as follows.

- The total number of processes in the system is denoted by *N* and varies from 10 to 500.
- The CPU times of processes is denoted by *BT* and is uniformly distributed between 1 *ms* and 1200 *ms.*
- The I/O occurs exponentially within the *BT* of each process. The mean of I/O occurrences is given by $BT \times \frac{1}{n}$, where $0 < n < BT$ for each process. We set the value of *n* to be 10.
- The I/O wait time for each process is given by $BT \times x$, where $x > 0$ and is uniformly distributed (*x* can be real). We choose the value of *x* to be 1.5 that is; I/O wait time is one and half the *BT* of each process.
- The arrival time of processes are Poisson distributed with mean as 1.2 *ms.*
- The value of the fixed time quantum, which will be used in RR, is 100 *ms.*

The following figures explain our improvements over the RR algorithm.

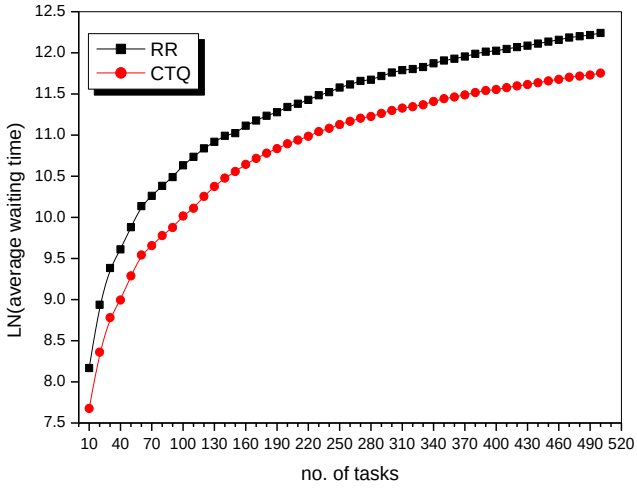

Figure 1: Average waiting time .

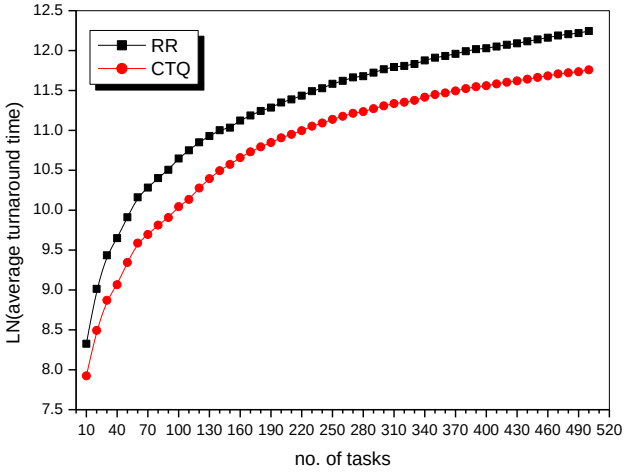

Figure 2: Average turnaround time

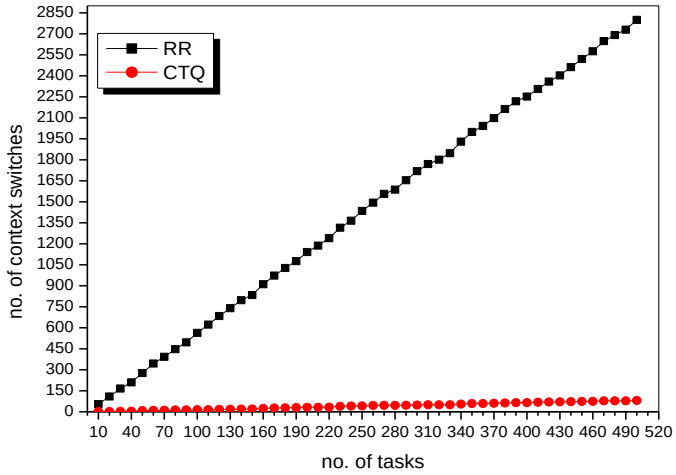

Figure 3: No. of context switches

We termed the previous 50 simulated different combinations of n and *BT* as Data_1. The following chart shows the improvement of the proposed algorithm over RR, where we considered the average of each criterion in Data_1.

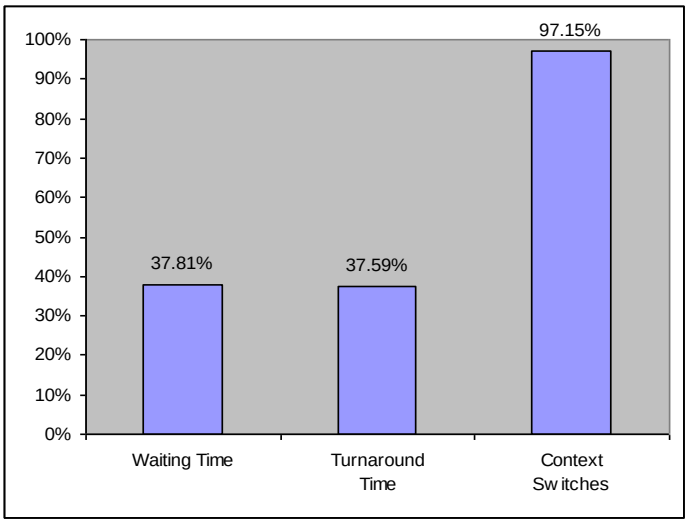

Figure 4: CTQ improvement over RR in Data_1.

5. CONCLUSION

In light of the effectiveness and the efficiency of the RR algorithm, we developed a new CPU local scheduling based on RR named Changeable Time Quantum (CTQ), and we described this technique and its improvement over the RR. From the simulation study, we get an important conclusion; that the performance of CTQ policy is higher than that of RR in general purpose system.

6. FUTURE WORK

The work presented in this paper can be expanded in many directions. Some of the directions are:

- Employing different performance criteria for comparison such as the makespan. The **makespan** is defined as maximum time needed to complete the execution of all the tasks arriving to the system [18].
- Applying scheduling technique on tasks that have dependencies among each other.
- Studying performance in real time applications where tasks have priorities and deadline constraints.
- Applying scheduling technique on distributed system. **A distributed system** is defined as a collection of independent computers that appear to the users of the system as a single computer [26].

7. REFERENCES

[1] L. Abeni, G. Lipari, and G. Buttazzo, "Constant bandwidth vs. proportional share resource allocation", *In Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Florence, Italy, June 1999.

[2] A., Bashir, M.N. Doja and R., Biswas, "Improving the Performance of Round Robin Scheduling Using Fuzzy Logic," *In proceedings of the International Conference on Advanced Computing and Communication Technologies for High Performance Applications sponsored by IEEE & CSI(Cochin, India,* September 24-26, 2008).

[3] A., Bashir, M.N. Doja and R., Biswas, "Conceptual Improvement of Priority Based CPU Scheduling Algorithm Using Fuzzy Logic," *International Journal of Fuzzy Systems and Rough Systems (IJFSRS)* (Vol. 1, No. 1, Janu.-June 2008)

[4] A., Bashir, M.N. Doja and R., Biswas, "Finding Time Quantum of Round Robin CPU Scheduling Algorithm Using Fuzzy Logic," *The International Conference on Computer and Electrical Engineering*, *ICCEE* 2008.

[5] A., Bashir, M.N. Doja and R., Biswas, "Improving the Performance of fair Share Scheduling Algorithm Using Fuzzy Logic," *Mumbai, Maharashtra, India.ACM 978-1-60558-351-8*. January 23–24, 2009.

[6] J. Bennett and H. Zhang, "WFQ: Worst-case Fair Weighted Fair Queueing," *in Proceedings of INFOCOM '96*, San Francisco, CA, Mar. 1996.

[7] B. Caprita, W.C. Chan, and J. Nieh, "Group Round-Robin: Improving the Fairness and Complexity of Packet Scheduling", *Technical Report CUCS-018-03, Columbia University*, June 2003.

[8] B. Caprita, W.C. Chan, J. Nieth, C. Stein, and H. Zheng, "Group ratio round-robin: O(1) proportional share scheduling for uni-processor and multiprocessor systems," *In USENIX Annual Technical Conference*, 2005.

[9] J., Chelladurai, "Fair And Efficient CPU Scheduling Algorithms," *The university of North British Columbia*, December, 2006.

[10] E., G., Comffman, R., R., Muntz and H., Trotter, "Waiting Time Distributions for Processor-Sharing Systems," *Journal of the Association for Computing Machinery*, PP. 123-130, 1978.

[11] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *in Proceedings of ACM SIGCOMM '89*, Austin, TX, Sept. 1989, pp. 1–12.

[12] R. Essick, "An Event-Based Fair Share Scheduler," *in Proceedings of the Winter 1990 USENIX Conference*, USENIX Berkeley, CA, USA, Jan. 1990, pp. 147–162.

[13] Q., GUO and Y., Liu, "THE EFFECT OF SCHEDULING DISCIPLINE ON CPU-MEM LOAD SHARING SYSTEM," *4th International Conference on Wireless Communications, Networking and Mobile Computing. WiCOM '08.* 2008.

[14] T. Helmy and A. Dekdouk, "Burst round robin as a proportional-share scheduling algorithm," *In Proceedings of The fourth IEEE-GCC Conference on Towards Techno-Industrial Innovations*, pp. 424-428, 11-14, at the Gulf International Convention Center, Bahrain. 2007. http://eprints.kfupm.edu.sa/1462/

[15] G. Henry, "The Fair Share Scheduler," *AT&T Bell Laboratories Technical Journal*, 63(8), Oct. 1984, pp. 1845–1857.

[16] A. Harwood and H. Shen, "Using fundamental electrical theory for varying time quantum uni-processor scheduling," *Journal of Systems Architecture: the EUROMICRO Journal*, volume 47, issue 2, Feb. 2001.

[17] J. Kay and P. Lauder, "A Fair Share Scheduler," *Communications of the ACM*, 31(1), pp. 44–55, Jan. 1988.

[18] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen and Richard F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing,* volume 59, Pages: 107-131, 1999.

[19] Rami J Matarneh, "Self-Adjustment Time Quantum in Round Robin Algorithm Depending on Burst Time of the Now Running Processes," *American Journal of Applied Sciences*, 6(10): 1831-1837 , 2009.

[20] J. Nieh, C. Vaill and, H. Zhong, "Virtual-Time Round-Robin: An O(1) Proportional Share Scheduler," *In Proceedings of the USENIX Annual Technical Conference*, June 2001.

[21] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case," *IEEE/ACM Transactions on Networking*, 1(3), PP. 344–357, June 1993.

[22] A. Silberschatz, P. B. Galvin, G. Gange, "Operating Systems Concepts with Java," *John Wiley and Sons.* 6Ed 2004.

[23] A. Silberschatz, P.B. Galvin, and G. Gagne, "Operating System Concepts," *John Wiley and Sons*. 6Ed 2005.

[24] A., Silberschatz, J. L., Peterson, and P.B., Galvin, "Operating System Concepts," *Addison Wesley*, 7ED 2006.

[25] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round-Robin," *in Proceedings of ACM SIGCOMM '95*, 4(3), PP. 231-242. Sept. 1995.

[26] A. S. Tanenbaum and M. Van Steen, "Distributed Systems: Principles and Paradigms," *Prentice Hall*, 2002.

[27] J. D. Ullman, "Polynomial complete scheduling problems," *In Proc. of the fourth ACM symposium on Operating system principles*, PP. 96 – 101, 1973.

[28] X., Yuan and Z., Duan, "Fair Round-Robin: A Low-Complexity Packet Scheduler with Proportional and Worst-Case Fairness," *The IEEE TRANSACTIONS ON COMPUTERS*, PP. 365-379, VOL. 58, NO. 3, March 2009.