Real-Time Scheduling with MLFQ-RT Multilevel Feedback Queue with Starvation Mitigation

Kenneth Hoganson
Department of Computer Science
Kennesaw State University
Kennesaw, USA
khoganso@kennesaw.edu

Joseph Brown
Department of Computer Science
Kennesaw State University
Kennesaw, USA
Jbrow313@students.kennesaw.edu

***Abstract*-- Process scheduling for real-time processes is a critical function of real-time operating systems, which are required to guarantee soft and hard deadlines for completing real-time processes. The behavior of Multi-Level Feedback Queue (MLFQ) scheduling mechanisms intrinsically support a scheduling that favors short CPU bursts to the complete exclusion of all other processes in the ready queues. This MLFQ feature can be extended to support meeting both hard and soft real-time process deadlines. This research proposes a new derivative of MLFQ for real-time scheduling call MLFQ-Real-Time (MLFQ-RT) investigated through simulation.**

The MLFQ-RT real-time extension builds upon research solving a known weakness of MLFQ scheduling: a vulnerability to starvation of processes in the lowest priority queue, so that the operating system is unable to guarantee that all processes will make progress. Simulation research demonstrates that MLFQ-RT can support hard and soft real-time process scheduling while simultaneously mitigating starvation in low priority queues.

Keywords—Real-Time Scheduling; Multi-Level Feedback Queue; MLFQ; Scheduling

I. INTRODUCTION

Multilevel Feedback Queue scheduling was first described in [1] as a multiuser CPU timeshare scheduling system, utilizing a multilevel process queue. It was designed to favor shorter, interactive processes over longer, lower-priority processes (Shortest Job First – SJF) and to minimize operating system overhead through minimal context-switches. A MLFQ scheduler (Figure 1.1) consists of a set of two or more FIFO process queues, and a scheduling policy. Associated with each level in the MLFQ is a maximum time quantum that each process in that queue may consume before it must release the CPU. Highest priority queues have smaller time quantums, which increase with each successive queue. Processes whose CPU bursts are incomplete drop down to lower priority queues.

Processes exit the queues for input/output or when the process is complete. The scheduler may not provide CPU time to any queue unless the preceding (higher priority) queues are empty, thus favoring high priority, interactive, and new processes arriving in Q1. CPU intensive processes will drop down through the queue structure until landing in the final queue (Qn), where successively longer time quantums for each queue minimize overhead due to context switching. Processes in Qn remain in Qn until their CPU burst is complete. Queues store pointers to Process Control Blocks (PCB) containing state data about the associated process and references to memory locations.

A MLFQ system does not differentiate between different types of processes, and processes aren't inherently prioritized; they naturally find their way to the queue appropriate for their behavior [2] by dropping down through the stack of ready queues that compose the Multi-Level Feedback Queue. MLFQ is a non-clairvoyant scheduler, as scheduler decisions are made independent of the characteristics of the schedulable candidates [3]. MFLQ can be modeled as an advanced case of a Tandem Queue with Sequential Service Switching independent inter-arrival and service times, and a single server (CPU) [4].

Research has been conducted to minimize $Q1$ latency and maximize overall throughput in MLFQ. Experiments have been conducted to allow known-interactive threads to borrow against future CPU allocation time in order to be scheduled earlier [6]. Dynamic CPU allocation per process according to system load has been simulated [7], and a variation on MLFQ called IMLFQ was proposed, which changes the number of

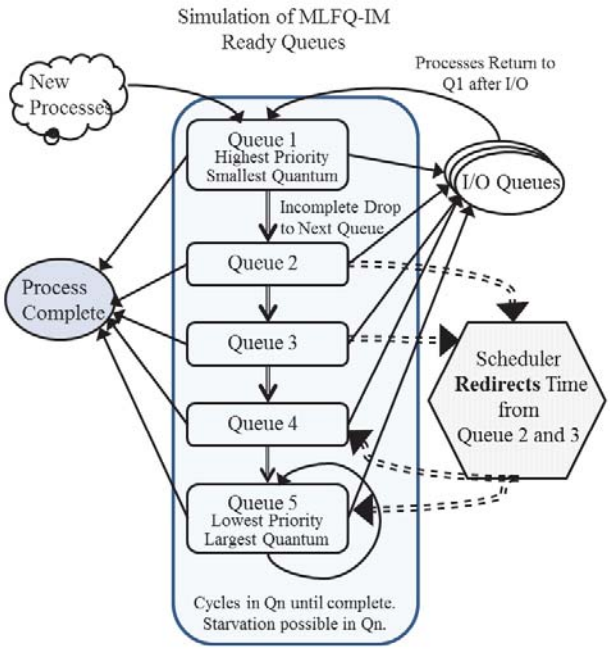

Fig .1 A Multi-Level Feedback Queue System

queues and associated quanta in MLFQ as the system load varies [8]. Simulation experiments explored replacing the FCFS scheduling within each queue with SJF, in order to increase throughput of the processes with the shortest runtime [9]. An advanced starvation mitigating scheduler variation called Multi-Level Feedback Queue with Intelligent Mitigation (MLFQ-IM) has been investigated through simulation research [10] and is the foundation for the proposed MLFQ-RT variation of this research.

A. MLFQ Starvation Mitigation

Under heavy load, the MLFQ will prioritize CPU bursts of short duration in Q1, disfavoring processes enqueued in the lowest priority queue Qn. Processes in any queue may not receive CPU time unless the higher priority queues have been service and are cleared of processes. Under extreme circumstances of very heavy processing load, CPU-intensive processes in the lowest queues can be temporarily starved receiving no CPU time, for a short or potentially long-term period, as the higher priority queues must always be serviced first. Thus when under heavy load, processes in the lowest queues may be unable to complete within a reasonable amount of time. Ideally, while favoring shortest CPU burst processes first (SJF), an operating system process scheduler must also ensure that all processes may make progress. Starvation can be identified by a build-up of processes in one or more of the queues below Q1, which receive little or no CPU time quanta. Usually starvation will occur in Qn, as all other queues must be empty before processes in Qn may be allocated CPU time.

B. MLFQ Starvation Mitigation

MLFQ-NS and this research into MLFQ-IM address the starvation problem by safely redirecting a portion of the CPU time available for Q2 and successive queues, to service processes in Qn. No CPU time is redirected from Q1, which is prioritized for interactive and newly arrived processes. MLFQ-IM also safely redirects CPU time to a set of the lowest priority queues when starvation is detected. In this simulation research, when starvation is occurring, CPU time is redirected to the two lowest priority queues, Qn-1 and Qn.

In this research, Q1 performance is monitored via mean process wait-time, which is the intervening period of time before receiving service in Q1 and the first CPU burst given to the process. Mean Q1 wait-times must not increase significantly through redirection, or the mitigation will be at the price of failing to follow SJF and servicing interactive and new processes.

II. MLFQ For Real-Time (MLFQ-RT)

Utilizing the characteristic of MLFQ scheduling that executes processes in Queue 1 to the exclusion of processes in lower-priority queues, Multi-Level Feedback Queues for Real-Time (MLFQ-RT) can guarantee hard and soft real-time deadlines requiring only the differentiation of real-time processes which execute in Queue1, and all other processes enter the MLFQ-RT structure with Queue2. Processes in

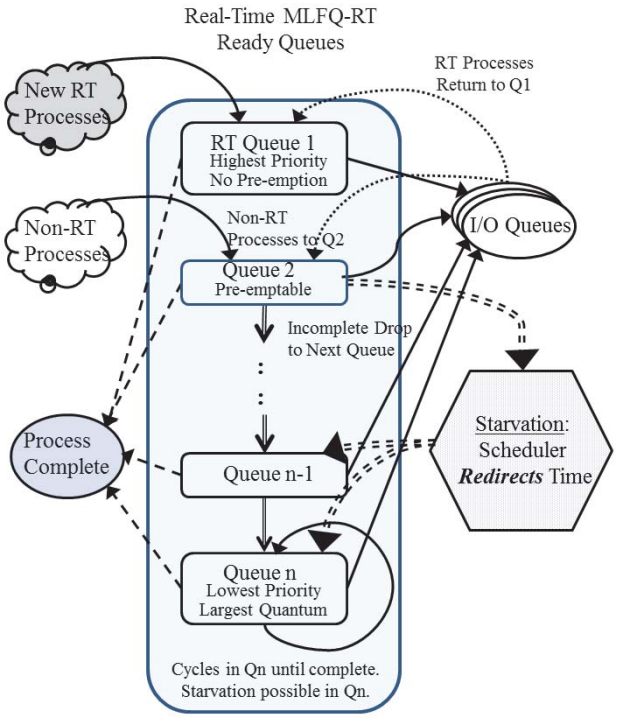

Fig. 2 Multi-Level Feedback Queue for Real-Time.

Queue2 and below receive zero computation time as long as there are real-time processes in Queue1.

Figure 2 shows a MLFQ-RT scheduling system that includes Intelligent Mitigation to redirect a percentage of available CPU processing time that is not required in Queue2 (which is time not needed to service real-time processes in Queue1, and thus available to Queue2 and below), to the final queues in the MLFQ structure, Queu(n) and Queue(n-1) in order to mitigate starvation in those queues.

In order for the MLFQ-RT scheduler to guarantee hard real-time deadlines, it is merely sufficient that the worst case real-time process analysis be met by the full attention of CPU processing performance. That is, the CPU processing speed and capacity must be able to meet the worst-case real-time workload, as all CPU time will be devoted to real-time processes in Queue1. Only when there is CPU time not needed for real-time processes, will processes in Queue2 and below be scheduled. More precisely, the worst-case analysis for MLFQ-RT real-time computation analysis is the sum of the worst-case real-time processing plus context switching time, plus the time to pre-empt a non-real-time process currently executing.

III. MLFQ with Intelligent Mitigation (MLFQ-IM)

The performance of MLFQs under heavy load has been explored in [5], and specifically enhanced with the redirection of some CPU time from Queue 2 (and below) to the lowest queue in order to combat starvation of CPU attention for those processes. Simulation research showed that a portion of CPU time could be safely redirected without jeopardizing

performance in Queue 1, through a conservative metric and plan called MLFQ-NS [5]. This redirection is done by taking a portion of the available time not needed in Q1 which would otherwise fall down to Q2 and below, and reallocating that time to the lowest priority queue (Qn). Time available in Q1 is monitored, and a specified fraction of available time not needed in Q1 is redirected to Qn. Simulation results confirmed that redirection could be done safely by observing the wait time for processes in Q1, which were not adversely affected. MLFQs can be further enhanced with the another level of CPU time redirection investigated in this research called "Intelligent Mitigation" (MLFQ-IM), which redirects CPU time to both Qn-1 and Qn shown in Figure 2, in order to mitigate starvation which may occur in Qn-1 or Qn. Overall system throughput is shown to be enhanced without affecting the performance of Q1.

Simulation showed that under heavy load, processes experienced starvation in Qn-1 as well as Qn, calling for redirection of CPU time to both of these queues in order to mitigate starvation and enhance throughput. CPU time not required for Q1 was available for diversion even at high system load.

The MLFQ-IM CPU and process scheduling policy contains these elements:

- CPU time is never diverted from Q_1
- The scheduler may not service processes in Qi, unless Q1 through Qi-1 are empty
- MLFQ-IM defines a set of final queues, $Q_{[M,\ldots,N]}$, where CPU time is allocated to mitigate starvation
- MLFQ-IM defines a set of intermediate queues from which CPU time is diverted, numbered $Q_{[2,\ldots,M-1]}$,
- MLFQ-IM uses the same mathematical functions–equations 1.2,1.3,1.4–to determine the amount of time to divert to $Q_{[M,\ldots,N]}$, check what these are in thesis
- To redirect CPU time, the scheduler will check Q_N to Q_M, until it finds a waiting process. If none are found, CPU time is not diverted from $Q_2, \ldots, Q_{M-1}$.

Simulation time is tracked in 1000ms periods, with simulations being timed over successive periods Ti,Ti+1,...,Ti+n. For each period Ti, cpu time spent servicing processess in Q1 is tracked as TQ1 . When T advances from Ti to Tj , CPU time spent servicing processes in intermediate queues, Q2;...,Q4, is calculated by subtracting TQ1 from |T|. This is time Tavail, from which a small percentage T% may be redirected to service processes in Q5. This time is combined with time diverted in a previous period Ti after multiplication with a weight-factor α using exponential averaging, shown in Equation 1:

$$TnQn = T\% * ([TAve(t-1) * \propto] + [(1-\propto) * TQi(1)]) \quad (1)$$

Specific time quanta are associated with each level in the queue, shown in Table 1.

Table II CPU Time quanta for queue levels

Q1	Q2	Q3	Q4	Q5
16ms	32ms	64ms	128ms	256ms

Table I Job Distribution Percentiles

55%	44%	1%
1-16ms	16-256ms	256-1256ms

Newly arriving processes into Queue 1 had CPU time requirements in milliseconds, which were uniformly distributed within percentiles as shown in Table 2.

IV. MLFQ Simulation Performance

MLFQ-IM was simulated with a system of five queues to maintain direct comparability with [5]. Burst performance in Q_5 and Q_4 is analyzed, and and mean wait-times in Q_1 monitored to maintain performance of Q1 where wait-times must not be significantly impacted. The impact of MLFQ-IM on scheduling in intermediate queues (Q2 and Q3) is also analyzed.

Simulation modeled processes arriving into the MLFQ in Q1 using a normal distribution with. System load was modeled up to 300% of maximum CPU time to explore system behavior under extreme circumstances with runs of 10,000 CPU bursts allocated to processes in the queues. Both MLFQ-NS and MLFQ-IM were modeled, to validate the system performance compared to MLFQ in [5]

A. Starvation Mitigation in Q5

Figure 3 shows the CPU bursts granted to processes in Q_5. Queue MLFQ CPU bursts decline and drop quickly to zero for MLFQ without redirection. Both MLFQ-NS and MLFQ-IM show CPU bursts in Q5 continuing which cease altogether at approximately 150% of CPU capacity with both plateauing at about 140% of CPU capacity. MLFQ-IM performs significantly better than MLFQ and MLFQ-NS beyond 100% of CPU capacity.

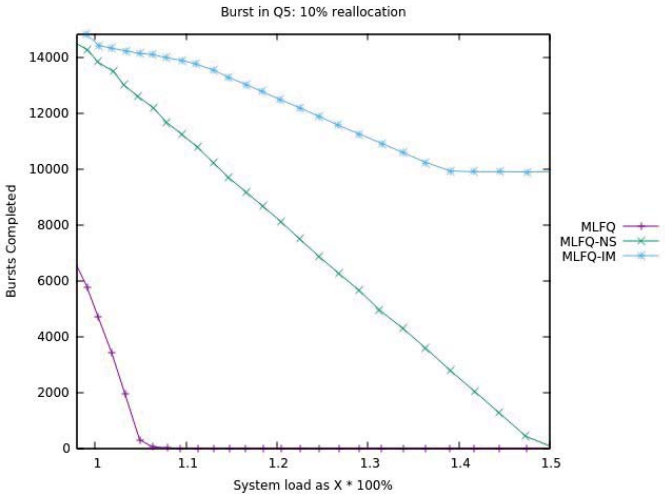

Fig. 3 MLFQ-IM Burts in Q5

Processes in Q_5 continued to receive some CPU time until approximately 300% of compute capacity. However, at 300% of capacity the very concept of mitigation becomes irrelevant.

B. Starvation Mitigation in Q4

Figure 4 shows the CPU bursts granted to processes in Q_4. For Q_4 , MFLQ outperforms MLFQ-NS and MLFQ-IM until approximately 140% of compute capacity when starvation begins to occur. MLFQ-NS and IM are comparable till between 95% and 140% capacity. Similarly to bursts in Q_5, Q_4 bursts plateau at 140%. S tarvation in Q_5 caused by starvation in Q_4 begins at 140% of compute capacity.

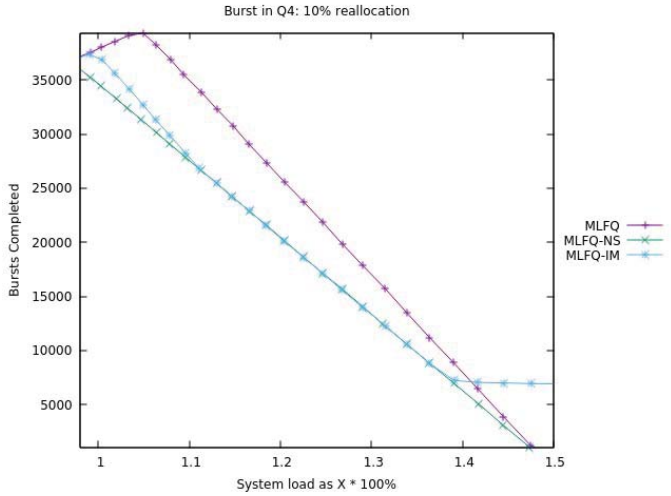

Fig. 4 MLFQ-IM Bursts in Q4

With Q_5 and Q_4 burst plateaus comes the implication of constant performance in those two queues. This then implies that performance in other queues would likely suffer. At *System-load* $\gg$ 95% there is more work in some period of time T than there is compute capacity to process demand in T. This will be addressed in a subsequent section.

C. Effect on CPU Redirection in Q1

Figure 5 shows the mean and maximum wait-times in Q_1 for MLFQ, MLFQ-NS and MLFQ-IM. For maximum wait times in Q1, the mitigating schedulers perform better than MLFQ from 100% $\leq$ *System-Load* $\leq$ 107%, and MLFQ without redirection outperforms MLFQ-NS and MLFQ-IM beyond 107%. This is because with no pre-emption, newly arrived processes and those returning to Q1 do not have to wait while a longer CPU burst redirected to Q4 or Q5 completes.

MLFQ-IM outperforms MLFQ-NS with shorter wait-times. This is due to with MLFQ-IM, redirecting time to Q4 instead of Q5 is allocated in shorter time quantums, resulting in a newly arrived process and processes returning to Q1 having to wait less time while the CPU burst allocated in Q4 completes (there is no pre-emption).

Most importantly, mean wait-times are unaffected by either mitigation techniques, a requirement for a successful scheduling strategy. Figure 5 shows that Q_1 mean wait times are not adversely affected by mitigation strategies, and the scheduler continues to follow SJF and meet the needs of interactive processes.

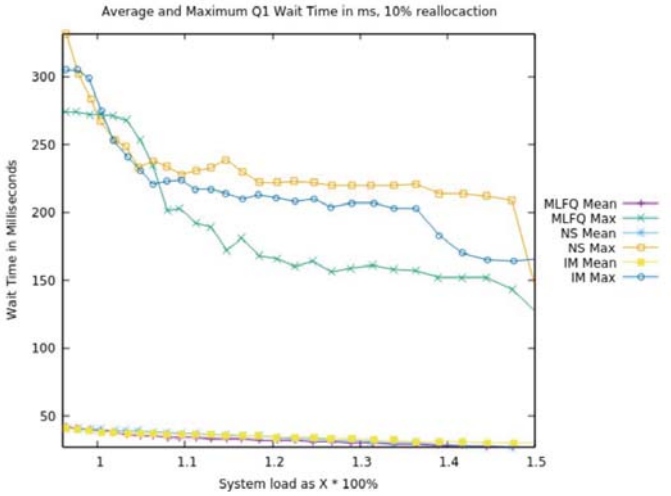

Fig. 5 Q1 Wait Times

D. Throughput Effects with MLFQ-IM

Figures 6, 7, and 8 show the quantity of processes left in Q_3 *thru* Q_5, as a check on the throughput of processes as the migrate through the queues from higher to lower priority. Processes migrate through the higher-priority queues, receiving less CPU time as they "fall down" to the lower priority queues.

Figure 6 shows that at most 1 process remains in Q_3 at the end of a simulation period of 10,000 CPU quantum at 100% CPU utilization, showing that in this simulation, processes are migrating through service in Q3 and do not stall or bunch up until Q4 and Q5. At approximately 140% of CPU capacity, the number of processes remaining in Q_3 rises sharply, to 10,000 just before system load reaches 150% as the system if overloaded and unable to complete CPU bursts in Q3.

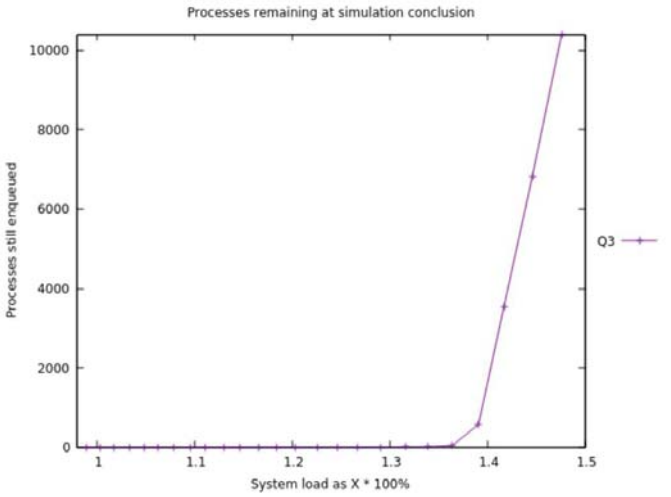

Fig. 6 Processes Remaining in Q5

Figure 7 shows that the number of processes not completed and remaining at the end of the simulation in Q4

increases under heavy load. While not completely starved for CPU time, the time consumed in the higher-priority queues reduces the throughput in Q4. The number of processes remaining in $Q4$ steadily rises from around 0 at approximately 95% system load to 45,000 remaining at 140% system load. The number decreases after 140% system load as the effect of CPU diversion of MLFQ-IM mitigates the starvation of Q4.

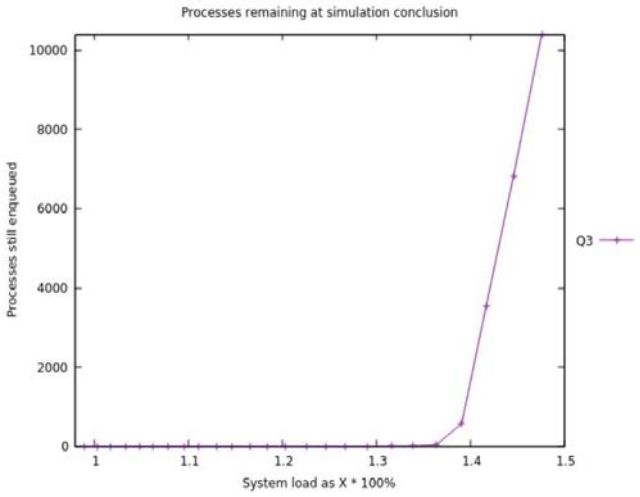

Fig. 7 Processes Remaining in Q4

V. Conclusion

This research investigates a new method and algorithm for mitigating the starvation problem of Multi-Level Feedback Queues, called Intelligent Mitigation (MLFQ-IM). Intelligent Mitigation reduces starvation in Multi-Level Feedback Queue CPU time allocation strategies, and is an extention of previous findings for MLFQ-NS redirecting time from Q1 to Qn mitigating starvation safely. This MLFQ-IM extension allows the redirection of CPU time from Q2 … Qn-2 to both Qn and Qn-1 in order to further mitigate starvation in the low-priority queues. The simulation studies have show this to be effective in reducing starvation when the system is under heavy load, without jeopardizing the quick response times required in Q1 to support the ideal allocation strategy of Shortest-Job-First. Starvation is show to be mitigated with throughput achieved in the lower-priority queues. This research shows that MLFQ as an operating system CPU time allocation strategy is effective and efficient, and can be made to mitigate starvation under heavy load with MLFQ-NS and MILF-IM. MLFQ-IM (and MLFQ-NS) are candidate CPU effective and efficient scheduling algorithms for computing systems.

The concept of MLFQ-IM can be extended for systems of many queues, to monitor and mitigate starvation in the lower queues including Qn and Qn-1, but also for a larger set of defined lower priority queues. Additional simulation research can validate that for this larger set of queues, the MLFQ-IM algorithm for redirecting CPU time can balance the allocation of CPU-time across a system of MLFQs and obtain higher system throughput even under heavy load, without adverse impacts on Q1.

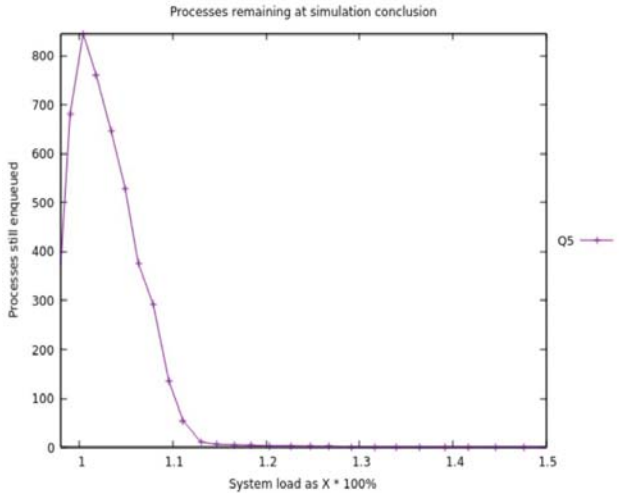

Fig. 8 Processes Remaining in Q5

A further extension of MLFQ-IM is to redirect the time redirected from middle queues to Qn and Qn-2 by varying the time quanta for Qn and Qn-1, but also allowing pre-emption in Qn and Qn-1. This is expected to have the effect of reducing context switching overhead in Qn and Qn-1 with longer time quanta for those queues. Increasing the time quanta for Qn and Qn-2 would otherwise also increase the maximum wait times in Q1, but with the addition of allowing pre-emption of executing processes in Qn and Qn-1 with now extended time quanta, pre-emption will reduce the time before servicing processes in Q1, with the effect of reducing the average and maximum wait times , predicting shorter wait times in Q1 than either MLFQ-NS and the new MLFQ-IM.

References

[1] 1 Writer's Handbook. Mill Valley, CA: University Science, 1989.

[1] Fernando J. Corbató, Marjorie Merwin-Daggett, and Robert C. Daley. "An Experimental Time-sharing System". In: *Proceedings of the May 1-3, 1962, Spring Joint Computer Conference*. AIEE-IRE '62 (Spring). San Francisco, California: ACM, 1962, pp. 335–344. DOI: 10.1145/1460833.1460871. URL: http://doi.acm.org.proxy.kennesaw.edu/10.1145/1460833.1460871.

[2] Lisa A Torrey, Joyce Coleman, and Barton P Miller. "A comparison of interactivity in the Linux 2.6 scheduler and an MLFQ scheduler". In: *Software: Practice and Experience* 37.4 (2007), pp. 347–364.

[3] Guido Schafer et al. "Average case and smoothed competitive analysis of the multi-level feedback algorithm". In: *IEEE FOCS03* (2003), p. 462.

[4] Tsuyoshi Katayama. "Mean sojourn times in a multi-stage tandem queue served by a single server." In: *J. OPER. RES. SOC. JAPAN*. 31.2 (1988), pp. 233–247.

[5] Kenneth Hoganson. "Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging". In: *J. Comput. Sci. Coll.* 25.2 (Dec. 2009), pp. 196–202. ISSN: 1937-4771. URL: http://dl.acm.org.proxy.kennesaw.edu/citation.cfm?id=1629036.

[6] Kenneth J. Duda and David R. Cheriton. "Borrowed-virtual-time (BVT) Scheduling: Supporting Latency-sensitive Threads in a General-purpose Scheduler". In: *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*. SOSP '99.

[7] Set Problem". In: *Commun. ACM* 31.10 (Oct. 1988), pp. 1220–1227. ISSN: 0001-0782.

[8] M. EffatParvar et al. "An Intelligent MLFQ Scheduling Algorithm (IMLFQ) with Fault Tolerant Mechanism". In: *Sixth International*

Conference on Intelligent Systems Design and Applications. Vol. 3. 2006, pp. 80–85.

[9] M. Thombare et al. "Efficient implementation of Multilevel Feedback Queue Scheduling". In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. 2016, pp. 1950–1954.

[10] K. Hoganson, J. Brown. "Intelligent Mitigation in Multi-Level Feedback Queues", ACM Southeast 2017. April 2017, Kennesaw State University.