Spring 2020

Analysis of jitter control using real time scheduling

FNU Geetika-Singh

Analysis of Jitter Control using Real Time Scheduling

Geetika Singh
Department of Electrical and Computer Engineering at Iowa State University
geetika@iastate.edu

Abstract

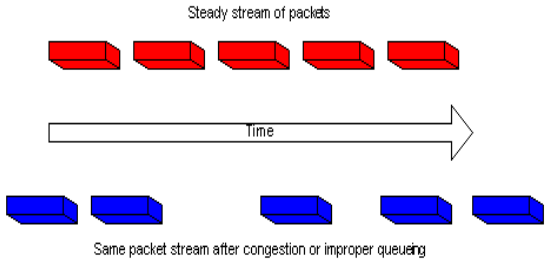
Jitter is defined as variation in the delay of received packets. In other words, jitter can be termed as the varying part of delay. Jitter is the variation in the latency on a packet flow between two systems, when some packets take longer than expected from one system to the other. At the transmitting side, packets are sent in a continuous stream with the packets spaced equally apart in most cases. Due to network congestion, improper queuing, channel properties or configuration errors, this steady stream can become lumpy, or the delay between each packet can vary instead of remaining constant which results in jitter. Jitter reduction is a major problem in mission critical applications which requires stringent timing guarantees with jitter limitation of about 1µs. Extensive research is being carried out to reduce this jitter to a great extent. Various methods have been adopted to reduce jitter and one such method for reduction is usage of optimal scheduling algorithms for packet scheduling. In this project, I have reviewed some of the scheduling algorithms which have been used to reduce jitter and compared their results.

Keywords- *Deadline Scaling Based (DSB), Task Decomposition Based (TDB), Initial Mandatory Final (IMF), Delay Variation Reduction (DVR).*

1. Introduction

Mission critical applications have gained popularity in every real-world scenario. For example, in automotive industry, there has been a development of driverless cars which is further moving to the point where vehicles communicate with each other. This communication needs to happen under hard real-time constraints as packets related to critical data will be sent such as the distance between the vehicles, information about an accident way ahead, deviation from the right lane etc. In augmented and virtual reality applications, there should be timely delivery of messages so that the experience of a real-world environment is not disrupted.

However, not focusing on this real-time communication aspect will result in introduction of delays and jitter at various points. These delays cause the system to behave in a non-periodic manner, causing the real performance to be degraded compared to the expected response. In most cases, this communication happens to have a hard real-time constraint and hence any difference in the delay will have catastrophic effects. For example, if the distance between the vehicles is not communicated accurately, then the distance might continue to decrease to a point where the cars are so close that there is a high chance of an accident. If the cars are obtaining the information of the traffic signal, then this information needs to happen in a timely manner. Any delay in the delivery of this information might lead to cars jumping the red signal which will be catastrophic to human life. Jitter reduction is a concern even in augmented and virtual reality applications. Hence, our main concern for this project is to study methods used at the scheduling level to reduce jitter.



As shown in the above figure, jitter is the variation in the latency on a packet flow between two systems, when some packets take longer than expected from one system to the other. In other words, jitter can be termed as the varying part of delay. Stability and robustness

can be a major concern in the systems discussed above due to the large magnitudes of random delay and jitter. There are various causes due to which jitter might be introduced while packet delivery. If the rate at which the input being sent is more than the service rate that the channel can handle, then this will lead to an excess of packets being sent by the input which leads to network congestion. Network congestion leads to packets being delivered in an untimely manner. Other reasons for jitter include improper queuing, channel properties or configuration errors. Hence, one needs to control the jitter.

1.1. Methods to control jitter

Below are few mentioned techniques that have been widely adopted till now to reduce jitter:

1. Usage of Jitter Buffers: In network-controlled systems, where jitter is caused due to difference in the timing of delivery of consecutive packets at the destination, a buffer can be utilized to reduce the jitter. A jitter buffer is a shared data area where voice packets can be collected, stored and sent to the voice processor in evenly spaced intervals. Variations in packet arrival time, called jitter, can occur because of network congestion, timing drift, or route changes. The jitter buffer, which is located at the receiving end, intentionally delays the arriving packets so that the end user experiences a clear connection with very little distortion.
2. Usage of traffic shapers: As discussed before, network congestion leads to jitter and this is mainly due to excessive input being sent on the channel. To restrict the amount of input being sent in the network, one needs to make use of traffic shapers. Traffic shapers like token bucket and leaky bucket ensure that only a certain amount of traffic is sent through the channel thereby regulating jitter.
3. Usage of optimized scheduling algorithms while packet scheduling.

In most applications, these methods are used in combination to reduce jitter. The EDF algorithm is a basic periodic real time algorithm, but this algorithm is not good in terms of jitter, this is because when the algorithm was devised, jitter was not a big concern. Now, there are new ways that are implemented to prevent jitter. We will study about the various algorithms devised to reduce jitter.

2. Algorithms Devised

In computation, various algorithms have been devised and these have been used in combination for different Quality of Service requirements depending on the application. In my project, I have primarily focused on algorithms used for jitter reduction. Following are few such algorithms:

2.1. Deadline Scaling Based Technique

This algorithm mentioned in [8] reduces the deadline of the tasks by a critical scaling factor beyond which the tasks set becomes infeasible. The following steps were followed to reduce the deadlines:

1. Finding the minimum deadline of a periodic task. This analysis was made off-line. The maximum reduction implies that a lower deadline makes the system not feasible.
2. Finding the minimum deadline of any task job. This algorithm was applied on-line, whenever it was convenient to reduce some task jobs during the execution of the system.
3. Finding the maximum reduction for all task deadlines, that is, the critical scaling factor. The critical scaling factor is the lowest number by which the deadline of all tasks in a set can be multiplied without rendering the task set infeasible.
4. When all the tasks have the same importance, deadlines of all tasks will be reduced with the same ratio.
5. When there are some tasks more important than others, these tasks are selected for reducing their deadlines.

The method consists of calculating the point with the minimum slack to calculate the critical scaling factor. This slack is the maximum amount that a task deadline can be reduced. However, the same amount for all task deadlines will lead to different ratios. Hence, as a first approach, critical scaling factor was chosen as the minimum ratio when new task deadlines are $D_i = D_i - slack$. These steps are repeated until there is no slack available. In this way, new deadlines are obtained.

2.2. Task Decomposition Based Technique

In the paper [22], the tasks are divided into Initial, Mandatory and Final subtasks and hence this model is known as IMF model. Each of the deadlines is brought

close to its worst case execution time such that the final tasks set obtained is schedulable.

2.3. Deadline Variation Reduction Algorithm

In this paper [6], an on-line adaptive approach which directly minimizes delay variations for both decomposable and non-decomposable control tasks simultaneously was proposed. The difference between DSB and DVR technique is that this considers the weights of the tasks. This approach was carried out in three steps:

1. The general IMF model as given in the previous section is used for both decomposable and non-decomposable tasks and hence each task is divided into Initial, Mandatory and Final subtasks.
2. To minimize the delay variations of all the final subtasks, the delay variation reduction problem was formulated as an optimization problem. The mathematical equation is given by:

   Minimize $\sum_{i=1}(wi\ ((D_{if} - C_{if})/ P_i)^2)$

   Where D_{if}= Deadline of the final subtask

   C_{if}= Computation time of the final subtask

   P_i= Period of the task

   n= Number of tasks

   The use of wi allows one to capture the relative importance of control tasks in the objective function. Hence, wi gives the relative weights of the task.
3. An efficient heuristic was developed to solve the optimization problem. The efficiency of the algorithm readily supports an adaptive framework which can adjust deadlines of control tasks on-line in response to dynamic changes in workloads. The heuristic is then incorporated into an adaptive framework.

2.4. Pfair Algorithm

Proportional fair is a compromise-based scheduling algorithm. It is based upon trying to maximize total throughput while at the same time allowing all users to have at least a minimal level of service. This is done by assigning each data flow a data rate or a scheduling priority that is inversely proportional to its anticipated resource consumption. [10]

The main idea of PFair algorithm is based on proportionate fairness i.e. if there is a task A with computation time Ci and period Pi,, then Ci/Pi part of the task must be scheduled for every time slot. The lag defined in this case is given by:

Lag (A,t)= (t* Ci/Pi) – allotted (A,t)

= the part of the task that should have executed – actual part of the task executed.
Pfair ensures that the lag lies in the bounds -1<lag<1

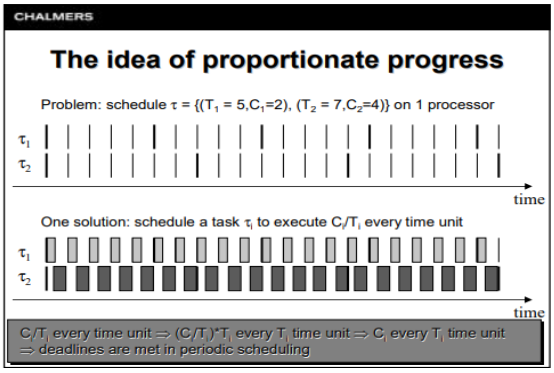
The schedulability test for PFair is given as:

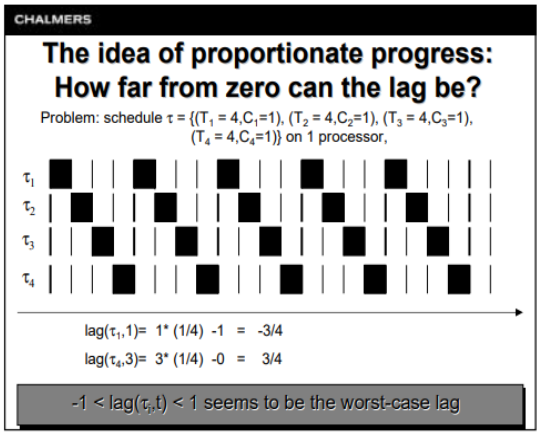
$\sum i=\{1,2,\ldots,n\}\ (Ci/Pi) \leq m$

Where m = number of processors.

The above condition is a necessary and sufficient condition for schedulability. Below figures show an illustrative example of implementation of Pfair algorithm on a simple task set. The below figure shows the expected division of tasks based on the ratio of computation time and period for each task. In the first figure, we notice that the tasks are expected to preempt and complete every portion of computation time/period in non-integer time periods so that the lag for each task at every time instance is 0. However, this is not possible. Hence, the preemption occurs at integer time intervals and the condition for lag is relaxed such that the lag should remain in the absolute value of 1 as shown in second figure.

The way this algorithm works is that at every time instance the lag for each task is calculated. It is ensured that by default all tasks will have lags in the limit of -1 to 1 due to preemptions. The task which has its lag approaching 1 will be scheduled. This ensures that the lag for the current task is reduced in the next time instance.





This PF scheduling algorithm is used in LTE. It provides a good tradeoff between system throughput and fairness by selecting the user with highest instantaneous data rate relative to its average data rate. However, in every block, PF scheduler informs the user equipments about their allotted slot positions of radio resources thus increasing scheduler complexity and overhead. [16]

The proportional fair algorithm also has been proposed as a technique to improve the throughput of multiple packet-data users sharing a wireless downlink channel while preserving fairness. Proportional fair algorithm is part of the HDR and HSDPA standard.

I have discussed Pfair algorithm to understand R-Pfair and Jfair algorithm which will be discussed in the below sections.

2.5. R-Pfair Algorithm

In this paper [25], a novel scheduling policy was proposed known as Regularity-Aware Proportional Fair (R-PF) scheduling, for cellular networks with fading wireless channels. In R-PF, each client may specify a single parameter that represents its preference between service regularity and throughput to the base station. Clients that do not support R-PF are assigned a default value for the parameter. The base station then provides services tailored to the preferences of clients. With the design of R-PF, clients that prefer higher service regularity are guaranteed to be scheduled more regularly, but may receive lower long-term average throughput. On the other hand, clients that prefer higher throughput indeed obtain higher throughput at the cost of less regularity. Therefore, R-PF allows clients to do fine grained tradeoff between service regularity and throughput by themselves. In this project, I haven't focused on this algorithm in detail as it deals more with throughout rather than jitter reduction.

2.6. Jfair Algorithm

This algorithm [23] was designed as an extension to Pfair algorithm. As opposed to Pfair, in the proposed approach, the constraint on the lag limits for each application could be individually and arbitrarily selected. The fairness concept is defined in the sense that tasks progress roughly proportional to their allocated resource share. However, as discussed before, Pfair algorithm can only guarantee equal lag limits. As opposed to Pfair, this algorithm provides guarantees on the independent amount of jitter an application can tolerate and therefore the algorithm is referred to as Jfair. The fairness does not necessarily translate into equality as applications may tolerate different amount of variations in the response time in order to remain stable.

For a given task schedule, the lag δi(t) of task τi is defined as the difference between the amount of time that should have been allocated to task τi until time t according to its utilization Ui and the amount that is actually allocated. The lag limit is defined as the maximum allowed deviation in the absolute value of the lag for task τi, and is denoted by δi. A schedule is feasible if the lag limits are satisfied.

That is, a feasible schedule guarantees that,

$|\delta i(t)| \leq \delta i, \forall t \geq 0, i= 1, \ldots, n.$

Where n = number of tasks

2.7. Jitter Tolerance Factor Algorithm

In this paper [24], a transformation on a system of periodic tasks is done such that the schedule generated by EDF on the transformed system has better jitter performance than the schedule generated by EDF on the original system, while continuing to be a correct schedule for the original system.

It is assumed that each periodic task is characterized by a jitter tolerance factor with the interpretation that tasks with large values are more tolerant to jitter, in addition to the execution requirement ei and the period pi.

The weighted jitter of r is defined to be:

$$\mathrm{WtdJitter}(\Gamma) \stackrel{\mathrm{def}}{=} \max_{T_i \in \Gamma} \left\{ \frac{\mathrm{AbsJitter}(T_i)}{\phi_i} \right\}$$

The goal of this paper was to devise scheduling algorithms that accept input periodic task systems and generate schedules for that, meet all deadlines and attempt to minimize WtdJitter(r).

The general model described above can represent several interesting forms of jitter which may be of relevance to the application system designer. For instance, if ϕi = 1 for all i, then WtdJitter is the same as AbsJitter. Setting di = pi for all i in the above equation, the notion of relative jitter i.e. the jitter of a task measured as a fraction of its period is obtained. Similarly, task systems can be modeled in which only a subset of the tasks are jitter-sensitive by setting the jitter-tolerance factor for the other (non jitter-sensitive) tasks to infinity.

2.8 Genetic Algorithm

This paper [26] proposes the use of genetic algorithms for maintenance of jitter. Jitter has been defined by the equation as below:

$$j_{i,k} = ts_{i,k} - (k * T_i + O_i)$$

where $ts_{i,k}$ **is** the start time of the transmission of the instance k of message *I* parameter i,T_i is the period and O_i is the initial phasing at the start-up of the system. $j_{i,k}$ is the jitter for each message. Overall system jitter is given by:

$$OSJ = \sum_{i=1}^{M} \sum_{k=1}^{\frac{I}{T_i}} j_{i,k}$$

Genetic Algorithms (GA) are stochastic search procedures inspired by the biological principles of natural selection and genetics. The GA gets started with a set of candidate solutions called as population which is usually defined in a random manner. An element of the initial population is called an individual which is then evaluated using a fitness function that gives a measure of the quality of that individual. The individual is an aggregate of smaller elements or units called genes. Each gene can have different values or alleles. The algorithm then, enters a cycle in order to generate a new population. It gets started by probabilistically selecting the fittest individuals and then undergoing a modification process, using genetically inspired operators like crossover or mutation that will eventually alter the alleles of some genes. Finally, the old and new populations are amalgamated, and the obtained result is the next generation that will in tum be evaluated. The cycle stops when a certain pre-defined condition is achieved (for instance, a pre-defined number of generations, in this case minimum overall system jitter).

Further progress is made in the paper by introducing a variant of the Genetic algorithm which reduces the number of messages analyzed simultaneously.

2.9 Jitter Margin Approach

Reducing jitter using appropriate scheduling algorithms falls under the category of optimizing the system at the time of implementation. However, this paper [27] has focused on reducing the jitter during the design of the system. This paper[27] adopts *jitter margin* approach to obtain the acceptable jitter range that ensures reasonably good system performance and develops an integrated real-time design and scheduling method that keeps the system performance within required range, without violating the deadlines of other hard real-time tasks. This is done by selecting a sampling interval for the control tasks such that the minimum control performance is satisfied, while providing for coexistence with other tasks in the system with minimum jitter of the control tasks.

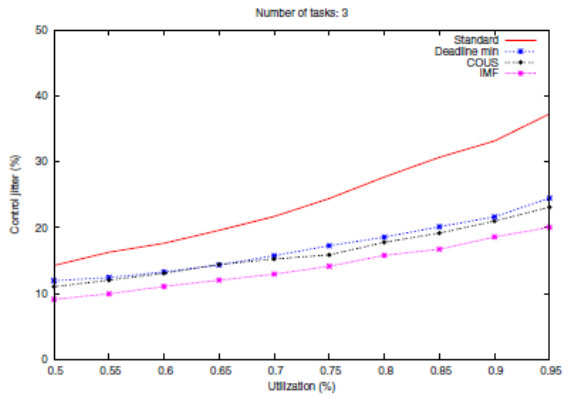
The *minimum time delay*, L, and the *jitter*, J, are then given by $L = BCRT$ and $J = WCRT - BCRT$ where WCRT= worst care response time and BCRT = Best case response time. *Jitter margin* approach [7] can be defined as the largest number $J_m(L)$ for which closed-loop stability is guaranteed for any time varying delay in the interval $[L, L + J_m(L)]$. The largest value of L that guarantee system stability is called *delay margin* and denoted L_m.

3. Results of Each Algorithm

In this section, I will give a brief overview of the results obtained for each algorithm and to what extent each algorithm has been successful to reduce jitter.

3.1. Deadline Scaling Based Technique

Several tests were run, specifically, 1000 task sets were generated for each utilization. Each task was generated by randomly choosing the task period as an integer between 5 and 100, the task utilization between 0 and 1, and then selecting the task computation in such a way that the total system utilization is approximately equal to the desired load.



Jitter produced in DSB Technique is compared with other techniques such as IMF model i.e. Task Decomposition based model, COUS model which is similar to IMF model and standard model. In COUS model, the tasks are split into two subtasks: Calculate Output subtask (CO) and Update State subtask (US) and new deadlines are calculated for each subtask. It is observed that IMF model gives better jitter control whereas DSB technique and COUS have similar jitter control patterns. This was expected, because the more partitioning is made, the lesser is the computation time for final tasks, and hence, the lower is the jitter. However, some important remarks were made in the paper:

1. IMF and COUS models make a task partitioning which implies more context switches than in a non-partitioning task model. As DSB technique and COUS have similar jitter, it is better to choose DSB due to less preemptions.
2. IMF applies a fixed delay to the final task. This means that the action gets delayed to achieve a very small jitter. This may cause a bad performance depending on the system. For this reason, it is sometimes better to have a slightly bigger jitter with no fixed delay. In these situations, DSB technique can be chosen.

To improve jitter reduction, it was proposed that our DSB technique can be used jointly with COUS and IMF models.

3.2. Task Decomposition Based Technique

The following tasks set is taken for testing the algorithm. It is obtained from a robotic application which has the control tasks such as speed control task, position control task, strength control task and sense control task.

	C	D	P
T_1	5	27	27
T_2	8	30	32
T_3	10	45	50
T_4	13	60	70

It was seen that the IMF model removed the negative effects of being the lowest priority task i.e. having higher jitter. This is done by reducing the preemptions of lower priority tasks due to higher priority tasks which in turn reduces jitter. However, the method neglects the subtask dependencies in the IMF task model and tends to generate infeasible solutions for high utilization task sets.

3.3. Deadline Variation Reduction Algorithm

Below are the tasks taken to perform simulation and the results obtained from the simulation for this technique.

Task Name	Computation Exec. Time	Deadline	Period	Delay Variations (%) Control Performance Original / DSB / TDB / DVR
Hard Real-time Task	570	3810	3810	33.7 / 33.7 / Fail / 37.82 NA / NA / NA / NA
Control Task τ_1	1570	10000	10000	18.54 / 18.54 / Fail / 12.84 153.3 / 153.3 / Fail / 112.5
Control Task τ_2	855	1500	6860	6.25 / 6.25 / Fail / 1.16 82.2 / 82.2 / Fail / 71.9
Control Task τ_3	429	1500	8570	9.98 / 9.98 / Fail / 3.83 95.3 / 95.3 / Fail / 86.6
Average Delay Variation				11.59 / 11.59 / Fail / 5.94
Total Performance Cost				330.8 / 330.8 / Fail / 271

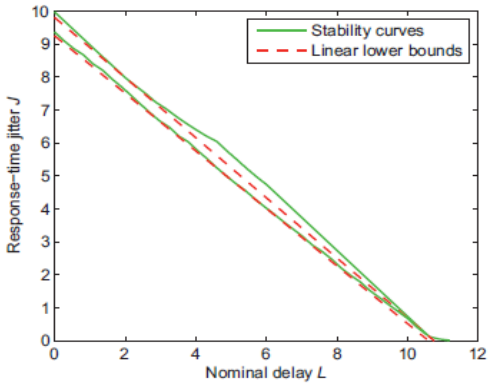
It shows comparison of standard model, DSB technique, TDB i.e. IMF model and DVR technique. We can see that DVR gives the optimal performance and the data clearly show that DVR reduces the average delay variation from 11.59% to 5.94%.

3.4. R-Pfair Algorithm

R-Pfair algorithm is used to obtain a tradeoff between service regularity and throughout depending on the requirements of the clients. It works in a similar manner as that of Pfair and brings out similar jitter reduction. Hence, the results for this algorithm pertain to throughout calculations and not jitter calculations and therefore, I will not be discussing the results of this algorithm in detail.

3.5. Jfair Algorithm

To evaluate the proposed algorithm experimentally, 500 benchmarks with a number of control applications from 2 to 10 were generated. This data considered were chosen from a database consisting of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators. Simulation is done using Jitter Margin toolbox and the variation of jitter w.r.t delay for different utilization levels is shown below. It can be seen that the jitter reduces with increase in delay.



It was noticed that although there was a good amount of reduction in jitter, the number of preemptions by the proposed scheduling policy is at most three times the number of preemptions by any feasible schedule under the given lag limits.

3.6 Jitter Tolerance Factor Algorithm

For each value of the system load, task sets were generated. Each task set was generated by randomly choosing the tasks' computation times as integers between 1 and 10, and then randomly choosing the periods such that the total system load be approximately equal to the desired load. It was deduced from experiments that if the relative jitters of all tasks is to be minimized, this method is not very effective. It was inferred that in general, it is easier to minimize relative jitter of tasks with low utilization.

3.7 Genetic Algorithm

Condition / Resol.	1µs	10µs	100µs
Lowest OSJ	12640	12640	22640
Average OSJ	13947	133645	14619
Worst OSJ	16256	15272	16304
Exec. Time Average	160	171	124
N° of Experiences	20	20	20

Table 2. Summary of optimization results for PSA benchmark at 125 Kbit/s.

Condition / Resol.	1µs	10µs	100µs
Lowest OSJ	0	0	0
Average OSJ	68	178	545
Worst OSJ	230	1374	1964
Exec. Time Average	540	739	928
N° of Experiences	20	20	20

Table 7. Summary of optimization results for SAE benchmark at 250 Kbit/s.

The message sets used were the ones known as the SAE (SAE - Society of Automotive Engineering) and PSA (PSA - Peugeot Society Automobile) benchmarks. These sets correspond to applications in automatically guided vehicles and in automobiles.

It was observed that the solution given by proGA provided a considerable reduction of jitter when compared with other systems that were not optimized. Good results were obtained on all tests, showing that a complete absence of jitter is possible on the SAE benchmark set at 250kbps, with a coarse timer resolution of 100 µs. Additional experiments resulted in proving that the proGA is more efficient than the simple GA. It can be seen from the tables that a timer resolution of 100µs seems to be good enough to get fairly reasonable results out of this jitter reduction technique.

3.8 Jitter Margin Approach

Here, I will be discussing a part of the results, as the results largely pertain to control systems. The experiment was conducted on three continuous systems. The design objective was to find sampling intervals *h* for each system that can fulfill the minimum required performance.

h	Min. delay	Deadline	Jitter
0.45	0.29	0.30	0.01
0.50	0.20	0.22	0.02
0.60	0.45	0.46	0.01

The scheduling policy used here is simulated annealing. The above table is for three systems used and we can see that for certain values of h, the jitter is obtained to very less i.e. in the range of 0.01 to 0.02s.

4. Conclusions

I started my analysis with DSB technique and moved to jitter margin technique. In DSB technique, the algorithm repeatedly reduces the deadlines of all the tasks by the same scaling factor, until the task set becomes non schedulable. The drawback of the method is that the blind deadline reduction may increase the delay variations, which opposes the goal of the algorithm. In TDB technique, the algorithm replaces the deadlines of final subtasks by their respective worst-case response times in the main loop efficiently until the algorithm converges. However, the method neglects the subtask dependencies in the IMF task model and tends to generate infeasible solutions in high utilization task sets. Although Jfair algorithm reduces jitter to a good amount, the number of preemptions increase by almost 3 times. However, this might not be a concern in scheduling from a communication perspective. In jitter tolerance factor method, the method is not very effective if the jitter of all the tasks need to be minimized. In case of genetic algorithms, experimental results showed that the algorithm has significant optimization capabilities for on-line or offline jitter reduction. However, execution times are comparatively high for obtaining fully optimized solutions and hence, there is high computational overhead. Jitter Margin approach seems to provide good results by maintaining the jitter to be around 0.01 to 0.02ms. However, this approach falls under optimizing at the design level which might be a challenge when it comes to real time scheduling from a communication perspective.

5. Findings

The methods studied above have their advantages and disadvantages and can be used depending on the type of application. For applications in the field of communication, it appears that JFair seems to be fairly good fit in terms of jitter reduction. However, one drawback would be the computational overhead resulting from obtaining and sharing the real time schedule of packets for every timeslot. Hence, enough resources need to be used for computation and tradeoff needs to be obtained between jitter reduction and overhead.

6. References

[1] “What is jitter buffer? - Definition from WhatIs.com,” SearchUnifiedCommunications. [Online]. Available: https://searchunifiedcommunications.techtarget.com/definition/jitter-buffer. [Accessed: 15-Dec-2018].

[2] IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS 1 ... (n.d.). Retrieved from http://www.ece.iastate.edu/~hongwei/group/publications/pktRT-TII.pdf

[3] What is jitter? - Definition from WhatIs.com. (n.d.). Retrieved from https://searchunifiedcommunications.techtarget.com/definition/jitter

[4] Di Natale, Marco, and John A. Stankovic. "Scheduling distributed real-time tasks with minimum jitter." *IEEE Transactions on Computers* 49.4 (2000): 303-316.

[5] Reducing Delays in RT Control: The Control Action Interval. (2017, July 07). Retrieved from https://www.sciencedirect.com/science/article/pii/S1474667017574546

[6] Hong, Shengyan, Xiaobo Sharon Hu, and Michael D. Lemmon. "Reducing delay jitter of real-time control tasks through adaptive deadline adjustments." *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*. IEEE, 2010.

[7]https://www.researchgate.net/publication/268405040_Scheduling_algorithm_for_delay_and_jitter_reduction_of_periodic_tasks_in_real-time_systems

[8] Balbastre, Patricia, Ismael Ripoll, and Alfons Crespo. "Minimum deadline calculation for periodic real-time tasks in dynamic priority systems." *IEEE Transactions on computers*57.1 (2008): 96-109.

[9] Aminifar, Amir, Petru Eles, and Zebo Peng. "Jfair: a scheduling algorithm to stabilize control applications." *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE*. IEEE, 2015.

[10] Proportionally fair. (2018, November 16). Retrieved from https://en.wikipedia.org/wiki/Proportionally_fair

[11] Earliest deadline first scheduling. (2018, December 13). Retrieved from https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling

[12] Buttazzo, Giorgio C. "Rate monotonic vs. EDF: judgment day." *Real-Time Systems* 29.1 (2005): 5-26.

[13] Avidor, Dan, et al. "On some properties of the proportional fair scheduling policy." *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*. Vol. 2. IEEE, 2004.

[14] Burns, Alan, et al. "Partitioned EDF scheduling for multiprocessors using a C= D task splitting scheme." *Real-Time Systems* 48.1 (2012): 3-33.

[15] Mohankumar, N. M., and J. T. Devaraju. "Performance Study of Proportional Fair Scheduling Algorithm with Transmit Diversity Multi-Antenna Technique for Lte Network." *International Journal of Electrical, Electronics and Data Communication* 1.9 (2013): 67-69.

[16] Swetha, Mohankumar NM, and J. T. Devaraju. "Performance evaluation of round robin and proportional fair scheduling algorithms for constant bit rate traffic in LTE." *International Journal of Computer Networks and Wireless Communications (IJCNWC)* 3.1 (2013): 41-45.

[17]http://www.cse.chalmers.se/edu/year/2015/course/EDA421/Documents/Literature/Pfair.pdf

[18] García, José Javier Gutiérrez, and Michael González Harbour. "Minimizing the effects of jitter in distributed hard real-time systems." *Journal of systems architecture* 42.6-7 (1996): 431-447.

[19] CSC 714 Real Time Computer Systems Project Report …(n.d.). Retrieved from http://moss.csc.ncsu.edu/~mueller/rt/rt05/readings/g1/report4.pdf

[20] S. Hong, X. S. Hu and M. D. Lemmon, "Reducing Delay Jitter of Real-Time Control Tasks through Adaptive Deadline Adjustments," 2010 22nd Euromicro Conference on Real-Time Systems, Brussels, 2010, pp. 229-238.

[21] Chen, Yu, et al. "Probabilistic Per-Packet Real-Time Guarantees for Wireless Networked Sensing and Control." *IEEE Transactions on Industrial Informatics* 14.5 (2018): 2133-2145.

[22] Balbastre, P., Ripoll, I., Vidal, J. et al. Real-Time Systems (2004) 27: 215. https://doi.org/10.1023/B:TIME.0000029049.50766.fa

[23] A. Aminifar, P. Eles and Z. Peng, "Jfair: a scheduling algorithm to stabilize control applications," 21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, 2015, pp. 63-72.

[24] S. Baruah, G. Buttazzo, S. Gorinsky and G. Lipari, "Scheduling periodic task systems to minimize output jitter," Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306), Hong Kong, China, 1999, pp. 62-69.

[25] A. Jain and I. Hou, "R-PF: Enhancing Service Regularity for Legacy Scheduling Policy," in IEEE Transactions on Wireless Communications, vol. 15, no. 1, pp. 258-266, Jan. 2016.

[26] F. Coutinho, J. Barreiros, J. A. Fonseca and E. Costa, "Jitter minimization with genetic algorithms," 2000 IEEE International Workshop on Factory Communication Systems. Proceedings (Cat. No.00TH8531), Porto, Portugal, 2000, pp. 267-273.

[27] M. Behnam and D. Isovic, "Real-Time Control and Scheduling Co-Design for Efficient Jitter Handling," 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007), Daegu, 2007, pp. 516-524.