Timing Analysis of the Priority based FRP System

Chaitanya Belwal
cbelwal@cs.uh.edu
Dept. of Computer Science, University of Houston, TX

Albert MK Cheng
cheng@cs.uh.edu
Dept. of Computer Science, University of Houston, TX

Walid Taha
taha@rice.edu
Dept. of Computer Science, Rice University, Houston, TX

Angela Zhu
angela.zhu@cs.rice.edu
Dept. of Computer Science, Rice University, Houston, TX

Abstract

Kaiabachev, Taha, Zhu [1] have presented a declarative programming paradigm called Functional Reactive Programming, which is based on behaviors and events. An improved system called P-FRP uses fixed priority scheduling for tasks. The system allows for the currently executing lower priority tasks to be rolled back to restoring the original state and allowing a higher priority task to run. These aborted tasks will restart again when no tasks of higher priority are in the queue. Since P-FRP has many applications in the real time domain it is critical to understand the time bound in which the tasks which have been aborted are guaranteed to run, and if the task set is schedulable. In this paper we provide an analysis of the unique execution paradigm of the P-FRP system and study the timing bounds using different constraint variables.[*]

1. Introduction

Reactive Programming has been found to be ideal in the area of real time systems. Most real time systems are reactive where the host raises events which are acted upon in a certain time frame. Functional programming is a paradigm based on lambda – calculus and offers various advantages over non-Neumann style of programming that is prevalent in standard languages. In [4] and [5] Functional Reactive Programming has been implemented for Real Time applications. Wan, Taha, Hudak [2] have given a statically-typed language called RT-FRP for real time systems which considers and space and time cost of execution. In [3] a compilation strategy to convert RT-FRP semantics into efficient code is given. The code of this new system called E-FRP has been tested on a small microcontroller driven robot.
All events in E-FRP are assumed to have the same priority. Events go into the queue and are executed in order, and the next event can execute only when the one before has completed execution. System interrupts with critical deadlines will have to wait for the execution queue to complete before it can start. This will cause the interrupts to miss its deadline leading to potentially catastrophic results. To overcome come this, a priority based FRP (P-FRP) system has been developed. This system used fixed priority scheduling to assign a priority number to every task before execution. If a task is executing and a higher priority enters the queue then the currently executing task is stopped and using a rollback mechanism the task is aborted and system state is restored. This prevents any side effect from the execution of the lower priority task. The higher priority task then starts execution. Though it may seem that the lower priority task has been 'preempted', when it starts execution it will have to restart. Hence from an execution standpoint the task can be considered non preempt-able, even though significant CPU resources might have gone into executing and then rolling it back. The system also needs to account for asynchronous and aperiodic tasks. These combined with the semantics of rollbacks offer significant challenges in the study of bounds of various task execution parameters. By constraining other variables we can assume that the entire task set is non – pre emptive. However this will give an inaccurate picture of the actual resources used by the system since even though the task has rolled back and has not executed it has still consumed CPU resources. The actual resource bound will not be the same as when the tasks are considered simply non preempt-able. For example if the FRP system runs on a power aware real time host the actual power consumed will be much more than if the tasks are considered to be simply non-preempt-able not have executed. Rollbacks take significant CPU (and disk) resources, and hence should be considered in the timing analysis.

2. E-FRP

The original semantics of E-FRP follow no priority or deadline scheduling. This scheme can be compared to First in First out (FIFO) scheme where tasks that come in first are executed. New tasks are put in queue and wait while other tasks ahead in are completed. As shown in [14] FIFO gives an infeasible schedule when deadlines and priorities are given. It is easy to put a general upper bound on the wait time of the task. Once a task is put in the queue it has to wait for the all the previous tasks to

[*] This work is supported in part by the U.S. National Science Foundation under Award Nos. 0720856 and 0720857.

finish. If there are n tasks and t_i is execution time for task *i*, then the maximum possible wait for task *k* is when it is placed last in the queue. In this case the wait time will be sum of execution times of all tasks before k. Therefore maximum wait time = $\sum_{1}^{n} t_i - t_k$.

3. Priority based FRP

In P-FRP a fixed priority is assigned to every task before compile time. Each event in the system is mapped to its fixed priority, numbers for which are selected from a fixed range of integer values. All events are executed atomically since task preemption is a rollback action. This way P-FRP retains the execution semantics of P-FRP. A bound on the waiting time for low priority tasks has been analyzed as follows.

There are n events, event *i* is represented by $I_{i,}$ each having an arrival rate of r_i which is the number of occurrences of the event per second. Task I_i has a priority of i. The maximum wait for an event *k* has been deduced to be $(n - k)\ G_k$, where

$G_k = 1 / \max(r_{k+1}, r_{k+2},\ldots, r_n\ , (n - k).\ \min(r_{k+1},\ldots\ r_n)$

Tasks k+1, k+2 ... n are of higher priority than k.

However this time bound is restricted if certain conditions are true. These are :

1. $t_k >> t_{k+1}$
2. $G_k >= t_k$
3. Same event will not occur if prior occurrence has not handled.

Where t_k is the execution time of task k. G_k is the maximum gap guaranteed to exist. Gap is the time period that exists between occurrences of task I_j. and task I_m where $j \approx m$ and *m, j > k*. Any task whose priority is greater than k, cannot execute in the gap. The gap is available exclusively to run task k.

The first condition says that tasks with lower priority have an extremely low execution time relative to higher priority tasks. This is valid in some execution scenarios, for example a normal operating system where higher priority tasks can be system interrupts and low priority tasks are normal applications. Most interrupt handlers have small and fast executing code whereas application tasks are large in both time and space. Though no deadline is specified this can be compared to a soft real time system, since interrupts have to be handled fast as other application behavior might depends on them. As an example some application which is waiting for a mouse click event will have to idle till the mouse / keyboard interrupt is handled. Clearly the mouse interrupt has a higher priority and also a soft deadline.

However in general for real time systems both hard and soft this assumption can lead to incorrect results. In such systems execution time of tasks is not indirectly (or directly) proportional to their priorities, and no relation can be formed between the two. It is possible for tasks with a large execution time to have a higher priority than tasks with relatively less execution time. Execution time of tasks is an important consideration in analyzing any real time system. Worse Case Execution Time (WCET) of tasks is used to get the upper bound on wait times and is used when any useful scheduling policy for the system has to be defined.

The second assumption says that the maximum gap available to task *k* should be larger than the execution time of the task. This is important since if the gap is less than the execution time then the task will never be able to complete within the observed time period. In such a case the task will start execution in an available gap, then a higher priority event will enter the queue forcing the executing task to stop and rollback. The aborted task will restart in the second available gap only to be aborted again. This will be repeated many times though the task will still not complete since it has to start re start execution in any available gap. This means the task set is not schedulable and is therefore not suited for study of time bounds. Schedulability of the task set is an inherent assumption with the second condition.

The third condition again implies schedulability of the task set, In E-FRP the length of the task queue is bounded by $\sum_{1}^{n} t_i$. If an event comes and the queue is full then the event will not be run at all. When the first instance runs the length of the queue becomes $\sum_{1}^{n} t_i - 1$. Hence this condition deals with the resource bound ness of the system. Some real time systems can have an event generated before the first one is handled. Hence those systems will not have this time / gap bound, though the queue size can be increased by adding empty task sets. It is clear from the wait time equation that a task of highest priority I_n will require no wait since $(n - k)\ G_k = 0$. Further study is required to find out the tightness of this bound. A new method also needs to be derived by relaxing some of the conditions which should be a more practical representation of existing real time systems. Our work aims to derive an upper bound which accounts

for task execution time and where the WCET is related to priorities of a task.

The timing analysis in [1] also does not consider the start time of tasks. Higher priority tasks are sporadic though a minimum period of separation is not specified. They also do not have any explicit deadline. It is assumed that a high priority the task starts execution immediately on entering the queue. When deadline and task execution time is considered the time taken for rollback will also have to be accounted for. If roll back time is too much a higher priority task may miss its deadline. We have to find a relation between size of the task and the time taken to abort it, do get a real picture on the schedulability of the task. We will also try to find out the cost in term of CPU time incurred during rollbacks. The total time can be accounted as context switches time, though in this case it is more prominent and cannot be ignored. An upper bound on context switch will have to be derived while finding the maximum wait time. It will also impact the bound ness of CPU resources, and can be used to find out the power consumed by the system in a more accurate way.

4. Example

Consider the following set of 3 tasks T_1, T_2, and T_3 . t_i is the execution speed of task *i* in seconds, and r_i is the arrival rate (number of occurrences / second)

T_1: $r_1 = 1$, $t_1 = .7$
T_2: $r_2 = 2$, $t_2 = .1$
T_3: $r_3 = 3$, $t_3 = .05$

In E-FRP the maximum wait time for task T_2 will be:

$$\sum_{1}^{3} t_i - t_2 = 0.75$$

Now we assign a static priority order to this task set. p_i is the priority of task *i*, and $p_3 > p_2 > p_1$. The execution times for this task set satisfy the necessary condition for the gap bound given in [1] to be used. Hence the maximum wait time for $T_2 = (3 - 2)\ G_2$.

$G_2 = 1 / \max(r_3 , (3 - 2).\ \min(r_3))$
$= 1 / \max(3 , (3 - 2).\ \min(3))$
$= 1 / \max(3 , 3)$
$= 1 / 3$

$\therefore$ Maximum wait time = 1 * 1/ 3 = 0.33

Hence we can see that with P-FRP, higher priority tasks will have a lesser wait time.

5. Real – time Databases

The P-FRP system has asynchronous release of tasks, the intervals between them are aperiodic and executed tasks can be rolled back without completion. This makes the task set non-preempt-able though it implements preemption semantics. Studying the time bound of such a system is challenging. Research has been done where the task set running on the CPU is non-Preemptive with variable execution time [6], is asynchronous where the start time is unknown[7] and where task set is non preemptive and sporadic [8]. In [9] algorithms have been given to find multiple feasible intervals (gaps) for a non-preempt-able task run. However no study has been done where these variables exist alongside with the consideration of an executing task set aborting and restarting again. We have looked at systems which have real time behavior but support task aborts. The rollback and abort mechanisms are implemented by databases and if we add time constraints the subset is real time databases.

To allow for data consistency every database transaction is atomic with respect to each other. Hence all databases implement a system for concurrency control to guarantee atomicity of the transactions. Concurrency control strategies in databases are generally of two types pessimistic and optimistic. Pessimistic strategies block the execution of a transaction that will lead to data conflicts. An optimistic strategy continues with the operation till the end and then rollback the transaction that will lead to conflicts. In our study we will look at optimistic strategies that have been implemented with timing constraints. This models the priority based FRP closely.

According to Shu [10] abort – oriented protocols were mainly developed to cope up with situations where the blocking property provided by pure locking protocols such a priority ceiling were not capable of scheduling tasks due to excessive blocking. A transaction is aborted if it prevents the completion of other high priority tasks. Though this allows the transaction set to be scheduled, it incurs additional costs in terms of aborting and re-execution. This cost has been studied in the Shu's work. Aborting a task also leads to priority inversion where a low priority task can run before a higher priority one. Method like the Priority Ceiling Protocol [12] prevents this from occurring. Byun, Burns, Wellings [9] do a response time analysis of hard real time transactions. For concurrency control they use priority abort where a lower priority transaction is aborted to allow transaction of a higher priority to run. However transactions that are waiting for a commit are not aborted to save time. Liang, Kuo and Shu [11] provide a class of abort oriented protocols for real time databases. The motivation for

developing these protocols is to avoid excessive blocking. This paper analyzes which standard scheduling algorithms like Earliest Deadline First (EDF) or Least Laxity First (LLF) can be used with transactions without affecting the validity of the data. Compatibility between the two is important, and this study will be important for P-FRP when new scheduling algorithms like Rate Monotonic, or dynamic Algorithms like EDF / LLF will replace the current priority assignment of tasks. A Basic Aborting Protocol (BAP) and its various derivations have been given. Tasks in BAP are classified as abortable or non-abortable which is determined by an offline schedulability analysis. In our study we have to consider all tasks as abortable because P-FRP does not distinguish tasks which can be aborted or not. Cheng [15] and Cheng, Chang [16] have developed schedulability tests for transactions in real-time systems.

6. Conclusion

We are looking to determine the timing bounds of the priority FRP system which allows for time bound tasks to run in the system and allows task pre emption by aborting the tasks. The task abortion finds an analogy in databases. Real time databases allow for both task aborting and timing constraints to be present in the system. Hence a study of system in real time database is important to understand the timing requirements of the P-FRP system. We also have to account for asynchronous release of tasks which are aperiodic in nature and study the Worse Case Response Time of the system. The original paper has studied this response time which is subject to lot of constraints. Our task is to come out with an improved timing analysis which closely models real time systems in practice today.

References

1. R. Kaiabachev, W. Taha, A. Zhu, 'E-FRP with priorities', In the Proceedings of the 7th ACM & IEEE international conference on Embedded software**,** Pages: 221 - 230 , 2007
2. Z.Wan, W. Taha, and P. Hudak. Real –time FRP, In ICFP'01, Pages: 146-156, ACM Press ,2001
3. Z. Wan, W. Taha, and P. Hudak, Event driven FRP, In PADL'02, Lecture Notes on Computer Science. Springer, 2002
4. J. Peterson, G.D. Hager and P. Hudak, A Language for Declarative Robotic Programming, ICRA'99. IEEE, May 1999
5. R. Kieburtz, Real-Time Reactive Programming for Embedded Controllers. Available from author's home page, March 2001
6. I. Alzeer, P. Molinaro, Y. Trinquet, Response Time Calculation for non-Preemptive Tasks with Variable Execution time. In Proceedings of ETFA '03. Pages: 131 - 136
7. G. Bernat, Response Time Analysis of Asynchronous Real-Time system, In Real-Time System, Pages 131-156 , Springer, 2004
8. K. Jeffay, D.F. Stanat, C.U. Martel, On Non-preemptive scheduling of Periodic and Sporadic tasks, In Proceedings of the 12th IEEE Symposium on Real-Time Systems, Pages: 129-139 ,December. IEEE, 1991
9. J.J. Chen, J. Wu, C.S. Shih, T.W. Kuo, Approximation algorithms for Scheduling Multiple Feasible Interval Jobs, In Proceedings of RTCSA'05 , Pages: 11 - 16 ,2005
10. J. Byun, A. Burns, A. Wellings, A Worst-Case Behavior Analysis for Hard Real-Time transactions, Workshop on Real-Time Databases, 1996
11. L. Shu, A Characterization of Re-execution Costs for Real-Time Abort-Oriented Protocols, Proceedings of RTSCA 1998 , Pages: 286 - 292 Issue, Oct 1998
12. M.C. Liang, T.W. Kuo, L. Shu, BAP: A Class of Abort-Oriented Protocols Based on the Notion of Compatibility, Proceedings of RTCSA '1996,118 - 127, Oct- Nov 1996
13. L. Sha, R. Rajkumar, J.P.Lehoczky, Priority Inheritance Protocols: An approach to Real Time Synchronization, Transactions on Computers Volume 39, Issue 9, Sep 1990 Page(s):1175 - 1185
14. A.M.K. Cheng, "Real Time Systems –Scheduling, Analysis and Verification", Wiley, 2002
15. A.M.K. Cheng, Scheduling Transactions in Real-Time Database Systems, Proc. IEEE-CS Computer Conf., San Francisco, CA, pages 222-231,Feb. 1993
16. A. M. K. Cheng, L. Zhang, An Efficient On-Line Scheduler for Real-Time Main Memory Database Systems, Proc. IEEE Intl. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Hong Kong, pages 680-685, May 1994.