Real Life Application of Producer Consumer Problem

*Note: Sub-titles are not captured in Xplore and should not be used

1st Ricardo Mangandi
Department of Computer Science
University of Central Florida
Orlando, USA
ricardo.mangandi@gmail.com

2nd Kyle Karacadag
Department of Computer Science
University of Central Florida
Orlando, USA
kylekaracadag1@gmail.com

3rd Nick Thiemann
Department of Computer Science
University of Central Florida
Orlando, USA
nathiemann1@gmail.com

4th Samuel Hearn
Department of Computer Science
University of Central Florida
Orlando, USA
shearn@knights.ucf.edu

5th Courtney Than
Department of Computer Science
University of Central Florida
Orlando, USA
courtthan@gmail.com

***Abstract*—**
***Index Terms*—**

I. Introduction

The problem we are addressing in our research is the impact of nonblocking data structures for real-world applications and benchmarking. An application we are interested in is web scraping different websites using a queue with several producer/consumer threads to search for specific data. Producer threads produce a job for the consumers.

II. State of the Art in this Research Area

A. Maintaining the Integrity of the Specifications

Filler Filler

III. Discussion of Related Work

This paper talks about a model that is based on the producer/consumer paradigm that construct real-time applications that interact with external environment which is one of the most important features of a real-time system. The system is not an independent system, it responses to any external inputs that comes from devices such as sensors,.. then it process the data and sent the responses in order to control the overall system.

The producer-consumer paradigm is a standard paradigm for multitasking. The connection between the producer and consumer is synchronized using a monitor that wait for consumer to consume data and wait for the producer to produce data. This paradigm is considered as a model of interaction between internal and external processes where the producer acts as the external process since it collects data from external devices and the consumer as the internal process which create computations based on the data provided by the producer (external process).

However, to ask if the producer-consumer paradigm could be viable for the real-time system or not. No, it couldn't be and there are two main reasons for this. First of all would be the external process problem; if the producer is an external process which means that it is not under control of the computer and therefore, it would be impossible to control this external process to wait for the consumer to consume the data. For that reason, it's important to make sure that the consumer will consume all the data provided without the wait statement in the Insert function. Secondly, using wait statements in real-time systems is not acceptable.

As a result, the real-time systems producer-consumer model means that all consumer tasks must process data in a time frame imposed by its producer. In other words, in order to apply the producer-consumer paradigm in the real-time system, we need to consider all processes from our system to be periodic and the following characteristics for each computing:

- task identifier IDi that defines a task uniquely [1]
- task i execution or computing time Ci– the execution time of a task can vary within an interval [CBi, Cwi] where CBi and Cwi are the best respectively the worst case execution times for the task i (in most of the cases Worst Case Execution Time is denoted by WCETi); generally, only the best and the worst execution time of each task are known. When modelling the timing behaviour of tasks,

Identify applicable funding agency here. If none, delete this.

this is a natural approach, because exact computation time of a task cannot be establish. Consequently, both times have to be considered when creating a task behaviour model, and results must be compared; another approach could analyze the timing behaviour starting from the idea of variable execution times. In our model we will rely on WCET. [1]
- task deadline Di – deadline is a typical task constraint in real-time systems: is the time before which the task must complete its execution. Usually, the deadline of the task is relative, meaning that, from the moment when a task Ti arrives (from its arrival time), it should finish within Di time units. [1]
- task period Tiper – for periodic tasks, task period Tiper is considered to be equal to task deadline Di. Because in most real-time control and monitoring systems the tasks typically arise from sensor data or control loops are periodic, we will consider in our model that all tasks are periodic. [1]
- task ready (arrival) time Ri – represents the moment of time when the task Ti is ready for execution.[1]

One of the algorithms that is an optimal algorithm for periodic tasks is the Earliest Deadline First algorithm. Earliest Deadline First is a priority driven algorithm where higher priorities are assigned to the requests that have the earliest deadlines.

IV. Contributions of Our Paper to the Problem

Filler Filler

V. Our Application

Filler Filler

VI. Our Experiments

A. Web Scraping

In this program, the website transfermarkt.com was websracped to gather data on soccer games that were played in the past 15 years. The program gathers data for over 5000 games and stores it into a csv file. 29 data points were gathered.

Python was used for this program with the BeautifulSoup and Requests libraries and pandas to store the data into a CSV file.

B. Reading from CSV

The data that is stored into the CSV from the previous program will be stored into an array using parallel computing.

C. Reading Stocks data from CSV

Stocks data on fortune 500 companies are stored in CSV files and will be analyzed using parallel computing.

D. Machine Learning Parallelization

Parallelizing a manual implementation of single variable linear regression. The section we will parallelize is Performing Gradient Descent. Each thread will receive a certain range of epochs and will derive four values of a, b, c, and d that will help us build our linear regression model. Matplotlib will be used to plot the results of the algorithm.

VII. Experiment Results

Filler Filler

VIII. Experiment Results

Filler Filler

IX. Appendix

Being able to coordinate with other thread workers to reduce redundant work. This would include coming up with the logic to split the workload between each thread. Since our project allows the user to decide how many working threads they would like, this brings a challenge to coordinate based on N number of threads. The problem would be much easier if we had a set number of threads and knew what tasks would be run on those threads.

The redis data structure queue has a timer in seconds, when tasks get inserted they get a set timer imposed on them, the default is currently two minutes. If the timer expires before the tasks are able to get enqueued they all get dequeued at the same time. Coordinating the logic with the timer is also a challenge.

Simulating a blocking-queue involves the usage and understanding the thread lifecycle. The thread lifecycle in Redis workers is different compared to real threads. When a task gets accomplished a normal thread would go to terminate itself. When a task gets done with Redis workers they stay up. Coordinating a creative way to kill them once they are done is a challenge.

A. User Stories Completed

- As a user I would like to produce content and have it be queued inside the Redis Queue.
- As a user I would like the ability to specify an N number of threads to accomplish my task with error handling if I exceed a certain amount.
- As a user I would like the ability to begin my threads whenever I need to.

- As a user I would like for my threads to enqueue only when there are items in the queue.
- As a user I would like the ability to visualize my queues with a UI.
- As a user I would like the ability to kill all my running threads.

B. User Stories that Need to be Completed

- As a user I would like the ability to enqueue a task that will web scrap and specify the number of threads to do it and visualize the events occurring with a UI.
- As a user I would like the ability to enqueue a task that will complete long mathematical computations and specify the number of threads to do it and visualize the events occurring with a UI.
- As a user I would like the ability to run a task and have my threads get automatically terminated just like a normal thread would in its lifecycle.
- As a user I would like the ability to analyze large chunks of data that take hours and parallelize it with running workers coordinating with the time a task is allowed in the queue.
- As a user I would like the ability to specify when to start a task. (This is like wait and modify feature)
- As a user I would like the ability to view results of the time taken for a specific thread and understand which thread took more time.
- As a user I would like to have numerous different tasks to run at the same time with them being parallelized at the same time..
- As developers we need to set up a small database that will be able to log tasks in for us and specify the time it took and what sort of tasks were involved.
- As developers we need to set up SQL queries to query certain tasks and get data to document our experiments.
- As developers we need to set up a workflow on the sort of experiments we will be running.

C. Proposed Workflow

- Run a task with two threads, three threads, four threads, n threads.
- Run queries to get data, log data.
- Write a summary for each number of threads run and for each task.
- Repeat for different tasks and continue.

Acknowledgment

Filler Filler

References

[1] D Zmaranda, G. A. Gabor, A. Nicula, "Producer-Consumer Paradigm in Real-Time Applications," Gianina Adela Gabor, 2009.