华 中 科 技 大 学

研 究 生 课 程 考 试 答 题 本

考 生 姓 名 周鑫宜

考 生 学 号 M202173632

系 、 年 级 计算机应用技术、2021 级

类 别 学硕

考 试 科 目 计 算 机 系 统 分 析 与 性 能 评 价

考 试 日 期 2021 年 11 月 15 日

Implementation and Verification of Supermarket Model

Abstract—Supermarket Model refers to a dynamic system with n servers, and customers arrive at the system as a Poisson stream of rate λn(λ<1). Each customer has d choices, and the customer waits for service at the server from these d choices currently containing the fewest customers (ties being broken arbitrarily). Supermarket Model (or Supermarket System) is an open dynamic system, and this concept was first proposed in 2002 by Michael Mitzenmacher in [1]. Supermarket Model shows an excellent rule of thumb in the design of distributed load balancing systems: Systems where items have two (or few) choices can perform almost as well as a perfect load balancing system with global load knowledge. However, with the improvement of computer system performance in the past 20 years, I wonder if this strategy still works in my system. So I decide to write my own system to test how well Supermarket Model can perform in a distributed system compared to a perfect load balancing system with global load knowledge and to a load balancing system with only one choice. After several simulation tests on my computer, I find out that compared with a perfect load balancing system, the performance of the Supermarket Model with d = 2 is significantly worse, which is inconsistent with the conclusion given in [1]. On the other hand, in the case where n = 8, the performance of a supermarket system with d = 4 is roughly the same as that of a perfect load balancing system, which means that the system with a few choices can perform nearly as well as a perfect load balancing system.

Index Terms—Queueing theory, Supermarket System, C++ implementation

1 Introduction and Analysis

In the abstract, I introduced the basic definition of Supermarket Model. However, in order to analyze it with more precise, I should introduce it more properly. Consider

the following natural dynamic system: Customers arrive as a Poisson Stream of rate $\lambda n(\lambda<1)$, at a collection of n servers. Each customer chooses d servers independently and uniformly at random. The customer waits for service The customer waits for service at the server from these d choices currently containing the fewest customers (ties being broken arbitrarily). Customers are served according to the first-in-first-out (FIFO) protocol and the service time for a customer is exponentially distributed with mean 1. We call this model the supermarket model, or the supermarket system. Figure 1 is an example of Supermarket Model where d = 2.

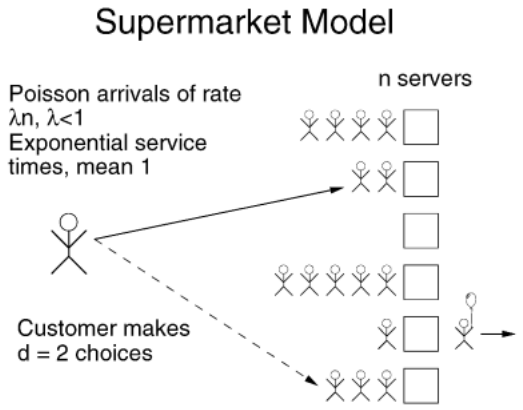

Figure 1 an example of Supermarket Model

The following theorem is representative of the results the author obtains in [1].

Theorem:

For any fixed T and $d\geq 2$, the expected time a customer spends in an initially empty supermarket system over the first T units of time is bounded above by:

$$\sum_{i=1}^{\infty} \lambda^{\frac{d^i-1}{d-1}} + o(1)$$

where the o(1) term is understood as $n\rightarrow\infty$(and may depend on T and λ)

In this article, I don't want to give a lot of mathematical details to proof of this theorem. I just make an outline of how the author proves this theorem. Firstly, He introduces a

deterministic limiting system related to the finite Supermarket Model. The time evolution of the limiting system is specified by the following set of differential equations:

$$\begin{cases} \frac{ds_i}{dt} = \lambda(s_{i-1}^d - s_i^d) - (s_i - s_{i+1}) & \text{for } i \geq 1 \ , \\ s_0 = 1. \end{cases}$$

In the equations s_i denotes the fraction of queues with length at least n.

Then he proves that the equation has a unique fixed point, with $\sum_{i=1}^{\infty} s_i < \infty$ given by

$$s_i = \lambda^{\frac{d^i - 1}{d-1}}$$

And he uses the notion of potential function to show that every trajectory of the supermarket system converges to the fixed point in an appropriate metric.

At last, he concludes the expected time in the limiting system and he apply Kurtz's Theorem to the supermarket model with finite servers to obtain bounds on the expected time a customer spends in the system and the maximum queue length.

2 Design and Implementation

First of all, in my design, there doesn't have to actually be n servers because in many modern computing systems (such as modern operating system), threads or processes can be viewed as a "server". Thus I choose C++ language and use the standard thread provided by C++ standard library to implement the simulation of Supermarket System.

I choose C++ language to implement my design of Supermarket System mainly for the following reasons: (1) Compared to other high-level language like Java or Python, C++ runs faster and is closer to hardware. Thus whenever a bug occurs or something goes wrong, I can focus on debugging my own code. (2) Compared to C, C++ language provides standard thread library to fully utilize CPU and I can get more

insight of how my program goes.

In my design, the whole system of my program is a Producer-Consumer Model. A Producer thread is created to produce tasks and n threads are created to consume tasks. Consumer threads are the threads that simulate the servers of the Supermarket Model.

First of all, the Supermarket Model needs customers. Customers refer to many computing tasks that has random computing time. In my design, a computing task refers to a classical problem: N-Queens. N-Queens Problem is described like this: N queens are placed on the 8-grid chess so that they can't attack each other, that is, any two queens can't be put in the same row, column or slash. Ask how many kinds of placing methods there are.

The solution to N-Queens Problem is a typical backtracking algorithm, which time complexity is O(N!). The specific algorithm of N-Queens Problem can be consulted online. With random N, the execution time of a N-Queens Problem conforms to exponential distribution. In my code, I design a class named EightQueens to solve the N-Queens Problem.

Secondly, I need to solve the problem of how to simulate a waiting queue of a server. Similarly, I choose to use queue in C++ stl (standard library for short). However, queue in C++ stl is not thread safe, which means when I create multiple threads to simulate a Supermarket Model, it may go wrong somehow. So I create another class named ThreadSafeQueue to encapsulate queue to make sure thread safety.

Finally, in my main.cpp, I design multiple functions to simulate the Supermarket Model:

(1) Function simulation(int n,int d,int total) is the main function to simulate the Supermarket Model, n denotes the number of servers, d denotes the number of

choices, and total denotes the number of tasks in the system.

(2) Function taskCreation(int n,int d,int total) is the function for task generating thread to execute. It creates tasks continuously for task consuming threads to execute and calculate.

(3) Function thread_execute(int threadId) is the function for task consuming thread to execute. Each thread simulates a server, and it continuously call task_execute function to solve a N-Queens Problem.

(4) Function task_execute(int id, int n) is the actual function to complete a task. id denotes the ID of current task (which is important because customers choose d servers and go to one currently containing least customer and this tie can be broken arbitrarily). In this function, if a task (customer) is being calculated in a thread, then other threads will not calculate it again. n denotes the N in N-Queens Problem.

And after each simulation, I find the slowest task processing thread and I record its execution time so as to find the maximum time a customer spends in the Supermarket Model. In addition, I calculate the ratio of the average execution time of the task processing thread to the execution time of the task production thread in order to find the average service time of a customer in a Supermarket Model.

3 Experiment Results

Before showing results of my simulation, I should introduce the hardware and software systems of my PC:

Hardware: CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz

RAM:8.00 GB(5.95 GB available)

System: 64 bit operating system, x64 based processor

Software: Operating System: windows 10 home edition

Language chosen: C++ 14

IDE and Compiler: Visual Studio 2019

I choose n = 5 and 8, and for each n, I choose d = 1,2,5 and d = 1,2,4,8, in which d == n gives us a perfect load balancing system. For fixed n and d, the number of tasks increase from 500 to 5000 in steps of 500.

After multiple times of simulation test, I record the slowest task executing threads and the results are visualized as shown as follows.

Firstly, with fixed n and d, increasing the number of total tasks shows the maximum execution time for the fixed n and d based on current task number as shown in Figure 2.

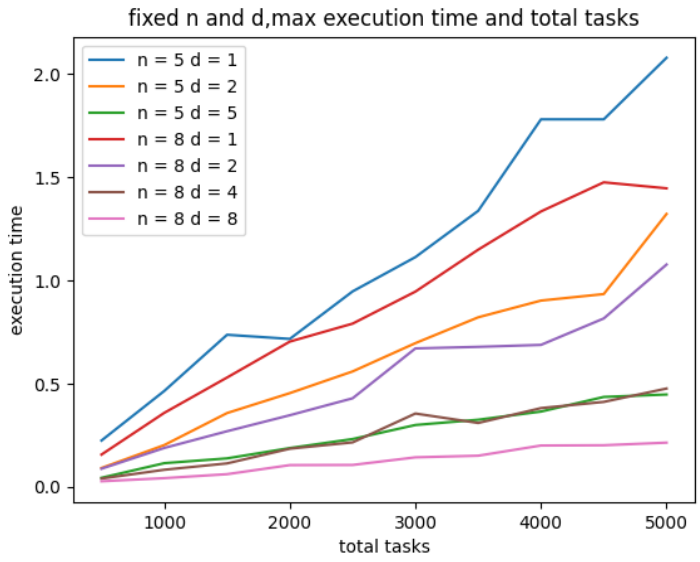

Figure 2 for fixed n and d, relationship between maximum time and task number

As is shown in Figure 2, with number of total tasks growing, maximum execution time grows as well. And the maximum execution time for n = 8 is less than n = 5. This result shows that the maximum service time will decrease with the increase of servers, which is also consistent with the experience of daily life.

Now we focus on the result of n = 8, as is shown in Figure 3.

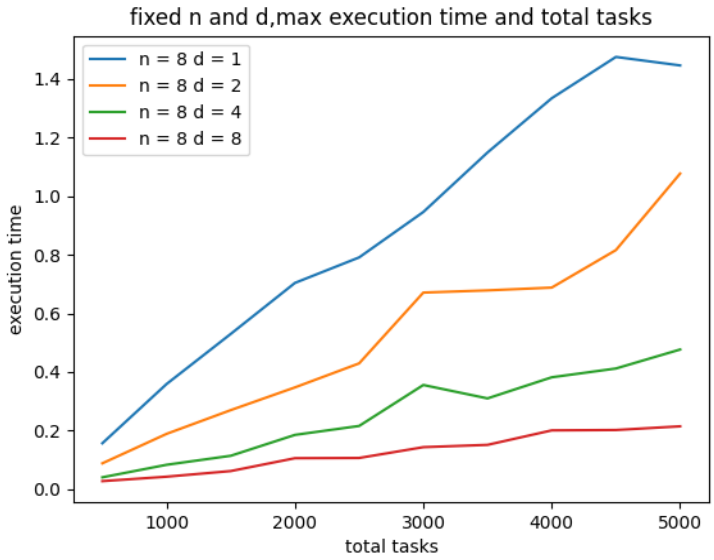

Figure 3 for n = 8, relationship between maximum time and d

It is obvious from the figure that the maximum execution time decreases with the increase of d. However, there is still a big gap between the maximum execution time of d = 2 and the perfect load balancing system of d = 8. On the contrary, the difference between the maximum execution time of d = 4 and the perfect load balancing system of d = 8 is not so great. This result may come from short execution time of each task and small number of tasks. However, due to the shortage of equipment and time, large-scale experiments can not be carried out to verify the correctness of the Supermarket Model.

Moreover, I record the average execution time for fixed n and d, and the results are visualized. Similarly, we take a look at result containing both n = 5 and n = 8 at first as shown in Figure 4.

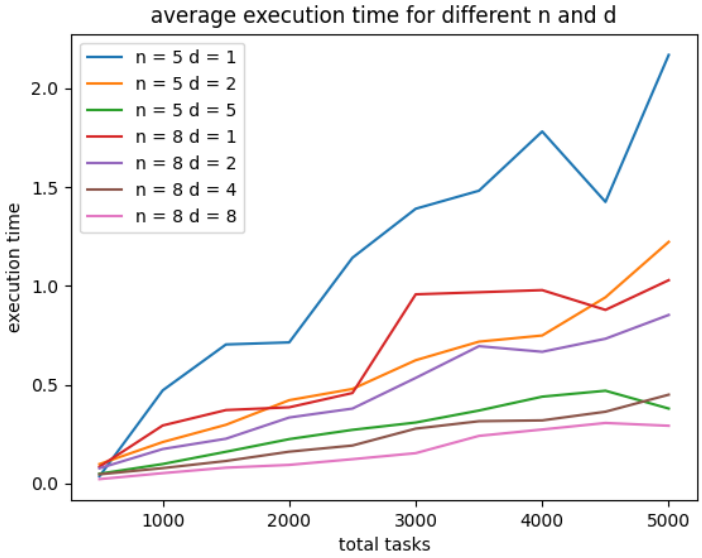

Figure 4 Average Execution time for different n and d

It's obvious in Figure 4 that average execution time for n = 8 is less than that for n = 5. This result shows that the average service time decreases with the increase of the number of servers, which is also consistent with the experience in daily life.

And we focus on the result of n = 8, as is shown in Figure 5. This result shows that in the case of n = 8, the average service time of a supermarket system with d = 4 is roughly the same as that of a perfect load balancing system, which is consistent with results given by [1].

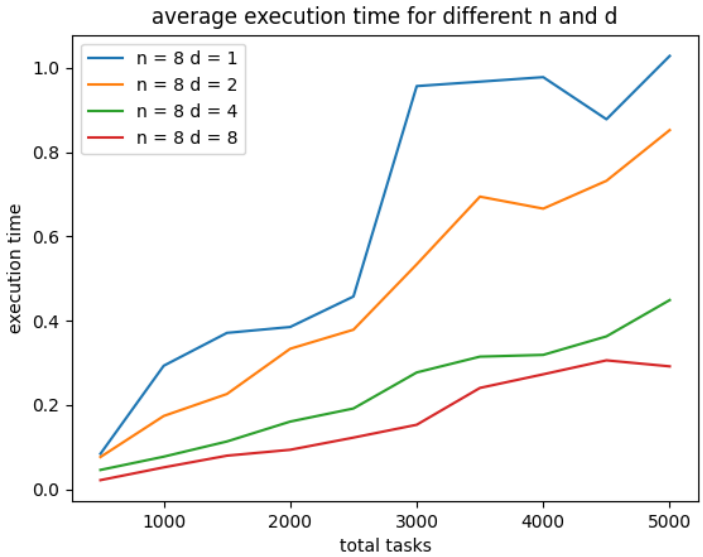

Figure 5 for n = 8, relationship between average execution time and d

4 Summation and Introspection

My complete code for this program can be found on my github:

https://github.com/warthecatalyst/GraduateCourse/tree/main/ComputerSystemAnalysis

If anyone is interested in my "research" or has any problem in reading my program, feel free to contact me through my Email: 18317011442@163.com

In my work, I reproduced Supermarket Model introduced in [1] and ran multiple simulation tests on my computer. The result of my simulation shows that there is still a big gap between the maximum execution time of a Supermarket Model where d = 2 and the perfect load balancing system where d = 8, which is not so consistent with the conclusion given by [1]. However, in the case of n = 8, the average service time of a supermarket system with d = 4 is roughly the same as that of a perfect load balancing system, which means that the system with a few choices can work nearly as well as the perfect load balancing system. This result is consistent with results given by [1].

In addition, I would like to stress some vulnerabilities in my code implementation. According to [1], the customer's choice of server can be changed arbitrarily. I simply use a bool array to record if a task has been executed, and I put the copy of the task in all servers' queues that has been chosen by the customer. For example, if a customer chooses server 1 and 4, then the task will be put in queues of both server 1 and 4. This strategy can be slowing down the actual process time because a thread needs to constantly access the bool array to judge whether a task has been executed. Moreover, this strategy is using more space than needed, because a task has multiple copies in d servers' queues. These defects may have fatal performance impact on a formal distributed service system.

References

[1] Michael Mitzenmacher, “The Power of Two Choices in Randomized Load Balancing”, IEEE 2002

[2] Reference - C++ Reference (cplusplus.com)