Homework 8: Load Balancing
CS231P
5/26/2025

TEAM: Destin Wong 64848542, Midhuna Mohanraj 39922268, Eric Huang 59504944

Explain the structure of your program, and discuss the results obtained. Also discuss and give answers to the following 6 questions.

1. Structure of the program

We defined each processor as a struct, since we needed each processor to have its own load, time interval, and a flag to determine if the processor was "steady". We generated random values for the load and time interval for each processor, and set the isSteady flag to 0 initially. We assume that the time interval that each processor is assigned at the beginning of our program stays the same throughout the entire simulation.

Main function:
Initializes the array with k processors and assigns each processor a random initial load (Lmin to Lmax) and a random time interval (Dmin to Dmax).

Simulation Loop:
We begin the simulation using a for loop for 1,000,000 cycles, though we always end much earlier than that. Then for each processor, we check if the current cycle number matches their assigned interval using a modulus operation. If it's not time for a processor to balance their load, we do nothing. Otherwise, we run the load balancing activity.

Load Balancing Activity:
To balance a processor's load, we first calculate the average between the current processor and its immediate left and right neighbors, wrapping around the processor array if needed. If the current processor's load is greater than the average, then it proceeds to distribute its load. If the left neighbor's load is less than the average, then the current processor gives it load until the left neighbor load reaches the average. We then check if the right neighbor's load is less than the average, and give it load until it reaches the average. If any distributions were made during a processor's balancing activity phase, we set its steady flag to 0. If no swaps were made, we set it to 1.

Steady state and Balance state check:

During each cycle, the program first looks to see if all the processors have stopped moving load around, this is what we call the steady state. Once every processor is steady, the system then checks if the loads are fairly balanced. By "balanced," we mean that each processor's load is close to the average load across all processors, within a margin of ±k. If both these conditions are met, no more load changes and loads are roughly equal the program concludes that it has reached a stable and fair distribution of work.

2. Results

Each time the program was run, the system eventually reached a steady state as well as a balanced state. For a 5 processor system, it took around 2,000-4,500 cycles for the system to converge to a steady state. For a 10 processor system, it took around 5,000-9,000 cycles. For a 100 processor system, it takes 50,000-250,000 cycles. As expected, the time it takes to reach a steady state increases proportionally as the number of processors increases. Having more processors in the system would lead to more instances of load sharing between neighbors. We also found that the difference in final loads at the steady state between processors would also increase proportional to the number of processors, so we adjusted our definition of "balanced state" to change according to the number of processors that were in the system.

3. What is your definition of "balanced among neighbors"?

Our definition of "balanced among neighbors" was when the load for immediate neighbors (left, current, right) was around the average load between each of the three. Sometimes, there would be cases where the processors on the left and right were average, and the processor in the middle (current) would be slightly larger than the average. In that case, the system would still be balanced, since all processors were at or a little above the average.

4. What is your definition of the system in "steady state"?

Our definition of the system in "steady state" was a state where each processor, during its last load balancing activity, did not give its load to any of its neighbors. We added a "isSteady" flag to each of our processors to signal whether it had given any of its load to neighbors. Each processor's "isSteady" flag would need to be set to 1 (during their load balancing activity) to have the system in a "steady state".

5. What is your definition of the system in "balanced state"?

First, we took the average by adding up the sum of all the loads in the system, and dividing it by the number of processors in the system. Then, we compared that average to the load of each processor. If the difference between the average and the load of any processor was greater than a certain factor (for our setup, the number of processors in the system), then the system would not be balanced.

6. Does the proposed strategy converge to a balanced state?

Our proposed strategy converges to a balanced state.

7. **Can you prove that the strategy will converge to a balanced state for any possible initial load-units distribution?**

Base case: in a situation where the system has three processors, the minimum number of processors where our balancing strategy is applicable, we can use our neighbor balancing strategy to ensure the load is balanced amongst neighbors. For example, if we have 3 processors with loads 1 4 4, we just need to balance each processor's load once to balance the neighbors. After the load is balanced amongst neighbors, since all the processors in the system are immediate neighbors, the entire system is therefore also in a balanced state.

If we add more processors onto the base case, we can split the system into groupings of 3 immediate neighbors. To balance the system, all subsystems of neighboring loads must be balanced. For example, in a processor with loads 1 2 3 4, the neighbors 1 2 3 must be balanced, 2 3 4 must be balanced, 3 4 1 must be balanced, and 4 1 2 must be balanced. The ring configuration of the processor helps prevent the final load from accumulating too much on one side of the processor ring, forming a gradient.

Each processor goes through load balancing more than once during the simulation, ensuring that it and its left and right neighbors are balanced, we can see that the strategy will converge to a balanced state with enough cycle iterations in the cycle. Given enough cycles, each processor will eventually reach the point where it is "balanced among neighbors", ensuring that a balanced state will be reached.

8. **What could be the "worst" possible initial load-units assignment?**

A worst-case scenario for the initial load distribution happens when processors have a very uneven load, like one or a few processors carry a very heavy load while the others have very light loads [1000, 10, 10, 10, 10] In this case, the heavily loaded processor will have to give away a large amount of load to neighbors who have very little. Meanwhile, the processors with smaller loads can't do anything, as they can't take loads from other processors. Due to this factor, the heavy load can only be shared gradually, with the heavily loaded processor as an origin point. This slows down the overall balancing process because the load has to move stepwise across processors, and the heavily loaded processors will have to wait for several balancing cycles before the system evens out.