CS 231P Homework 8 Report

Amay Kadre | 40909550 | akadre
Parth Shah | 54809514 | parths4

Consider a system with k physical processors defined by an array processor[0] to processor[k-1]. For a processor "i",
Load on i is defined as **processor[i]**.
Neighbors of i are **(i+1)mod k** and **(i-1) mod k**.
Load on neighbors = **processor[(i+1)mod k]** and **processor[(i-1)mod k]** respectively.

"Balanced among neighbors" is defined as:

A processor "i" is said to be balanced among neighbors if the difference in loads between processor i and its neighbors processor [(i+1)mod k] and processor [(i-1)mod k] is +1, 0 or -1. In other words,

abs(processor[i] - processor[(i+1)mod k]) <= 1, and
abs(processor[i] - processor[(i-1)mod k]) <= 1, and
abs(processor[(i+1)mod k] - processor[(i-1)mod k]) <= 1

"Balanced state of the system" is defined as:

The current state of the system is said to be balanced if the difference between the minimum load and maximum load at any processor is at the most 1.

Current Strategy:

- In the current strategy, a processor can only **give** its load to its neighbors in case of an unbalanced state.
- To avoid an infinite execution, a time limit of **100** seconds is imposed on each run of k processors.
- Scheduling load activity and distributing load among processors follows a uniform distribution and is achieved using a random number generator.
- The current strategy **does not** converge to a balanced system or converges with a very small probability. This random unlikely convergence might be because of the randomly assigned load on processors, producing a state that is easy to balance.
- This approach does not converge because, the randomly selected processor cannot receive load from its neighbors, the load will not be distributed when the load on that processor is less than or equal to the average load of the three processors. The same situation can be repeated when the neighbor is selected and hence the system will never reach convergence.

Alternate variation which works:

In our current strategy, in place of a processor **giving** the load to its neighbors, if we devise a strategy in which the current processor can even receive the load from its neighbor processors and we can **distribute the load between the current processor and its neighbors**, it would lead to convergence to a balanced state. This is because, when a processor has unbalanced neighbors and this system of three processors (current processor, left neighbor and right neighbor) distributes the load such that the current processor can give as well as take load from its neighbors, it would lead to a stable state.

For example, if we encounter the following structure of load among the processors:

p[i-1] = 20
p[i] = 30
p[i+1] = 40

=> ..20-**20-30-40**-30..

According to the suggested strategy, 30 is unbalanced amongst its neighbors but it can not balance this configuration since processor i has no extra nodes to give to p[i-1].

If we implement the alternate variation to balance load, 20-30-40 would become 30-30-30 since current nodes and its neighbors can distribute the load amongst each other. Hence, such configuration can be balanced using the alternate approach.

Following are the observations of the alternate variation:

Number of processors (k) (processor[i] or load lies between [1,1000])	**Number of cycles required for balanced state**
5	1600 - 1900
10	6000 - 8000
25	12000 -15000
50	45000 - 80000
100	700000 - 900000

We can observe that this new approach converges to a balanced system. This is because the new strategy now allows the processor to share the load of neighbors instead of just distributing excess load among neighbors.