CompSci-230: Homework 5
Jitao Zhang

1. Purpose

The purpose of this exercise is to verify experimentally that the load balancing strategy described above works and estimate how long it takes to converge to a balanced workload among processors.

2. Clarify

The definition of "balanced among neighbors" is that the load units in the current processor is smallest in its two neighbors. So the current processor cannot "give" the "marble" to its neighbors.

The definition of "balanced state of the system" is evolved during I simulate the task.

a) First I naively try to make load units in each processor as the same as possible for convergence. For example, if there are 100 units in 3 processors, it must be 34, 33, 33, and cannot be other distribution like 35, 33, 32. But I realize that it totally make no sense, because you might need much more time for the processor to do distribution just for the different 2 units.
b) Then I change my idea. I think I should make the lowest units of all the processors which is L, and the highest units of all the processors which is H. So H - L should be in an appropriate range as convergence. So I give it a constant like 3 or 4. It can convergence when the load units are not so large, but it is not appropriate for the case where the total load units are so large like 10000.
c) Then I think the constant should be relevant with the number of average of the load units(AVG). Like the 5 percentage of AVG, it works better. But when the k is so large, it is still not perfect for k is too large.
d) Finally, I came up with the final rule. The H - L should less than

$$\log_{10} K * 0.07 * AvgLoadUnits$$

3. Code

a) Main function:

```
function simulation(numOfProcessor) {
  let cycle = 0;                    // the current cycles
  let totalUnits = 0;               // the totalUnits in all processors
```

```
  const P = [];                       // the current load unit in each
processors
  const nextCycleForBalance = [];    // the next cycle to balance load units
for each processor

  for (let i = 0; i < numOfProcessor; i++) {
    // init
    P[i] = random(10, 1000);
    nextCycleForBalance[i] = random(100, 1000);
    totalUnits += P[i];
  }

  while (!isBalanced(P, numOfProcessor, totalUnits)) {
    for (let i = 0; i < numOfProcessor; i++) {
      if (nextCycleForBalance[i] == cycle) {
        // traverse each processor, if it in a balance cycle, then balance
        let prev = trans(i - 1, numOfProcessor);
        let next = trans(i + 1, numOfProcessor);
        // the current processor will first balance prev, then if possible,
balance next
        if (P[i] > P[next]) {
          let transLoad = Math.floor((P[i] - P[next]) / 2);
          P[i] -= transLoad;
          P[next] += transLoad;
        }
        if (P[i] > P[prev]) {
          let transLoad = Math.floor((P[i] - P[prev]) / 2);
          P[i] -= transLoad;
          P[prev] += transLoad;
        }
        // generate next balance cycle
        nextCycleForBalance[i] += random(100, 1000);
      }
    }
    cycle++;
  }
  console.log(`${cycle} cycles are used for converge to balance`);
}
```

b) Some helper function:

```
function random(left, right) {
  let c = right - left + 1;
        return Math.floor(Math.random() * c + left);
}

function trans(i, numOfProcessor) {
  // the processors are ring
  if (i < 0) return numOfProcessor - 1;
  if (i >= numOfProcessor) return 0;
  return i;
}

function getBaseLog(x, y) {
  return Math.log(y) / Math.log(x);
}
```

```
function isBalanced(P, numOfProcessor, totalUnits) {
  // init the l and h
  let l = 10000, h = -1;
  for (let i = 0; i < numOfProcessor; i++) {
    if (P[i] < l) l = P[i];
    if (P[i] > h) h = P[i];
    if (h - l > Math.floor(getBaseLog(10, numOfProcessor) * 7 * totalUnits /
numOfProcessor / 100)) {
      return false;
    }
  }
  return true;
}
```

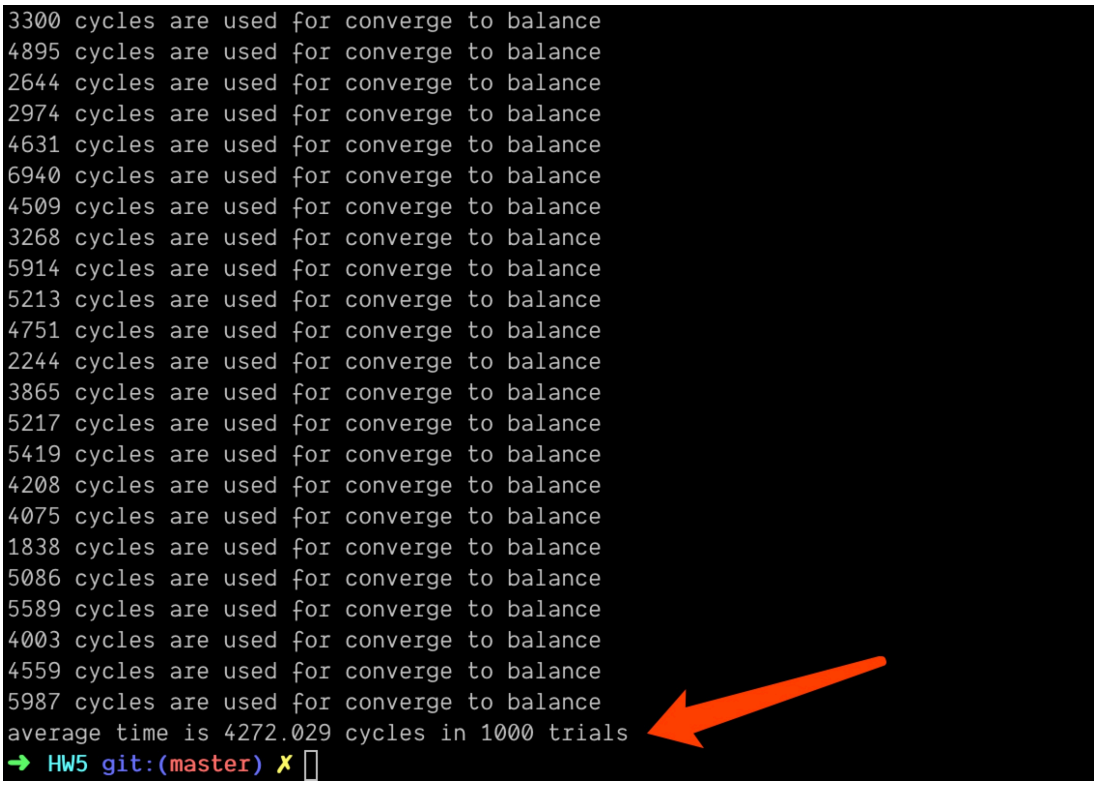
4. Result

I make 1000 trials for k in 5, 10 and 100.

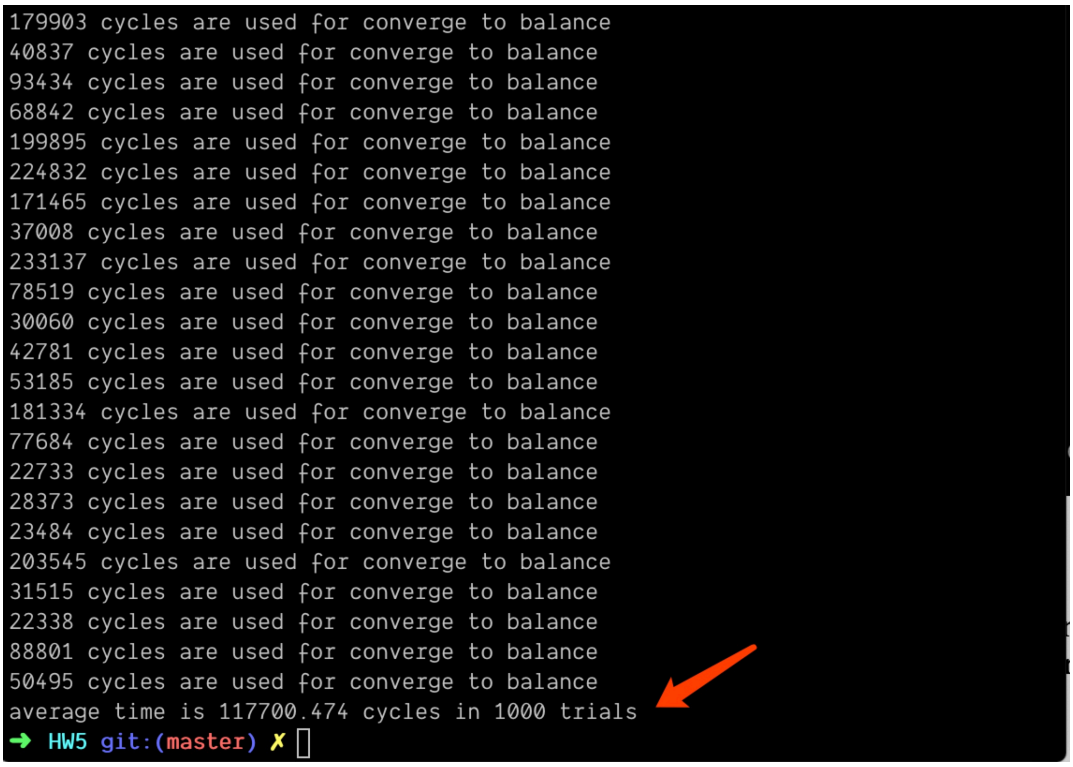
For k = 5:

```
2085 cycles are used for converge to balance
2139 cycles are used for converge to balance
1666 cycles are used for converge to balance
2802 cycles are used for converge to balance
2227 cycles are used for converge to balance
1126 cycles are used for converge to balance
1628 cycles are used for converge to balance
2381 cycles are used for converge to balance
1661 cycles are used for converge to balance
2357 cycles are used for converge to balance
1868 cycles are used for converge to balance
1883 cycles are used for converge to balance
1460 cycles are used for converge to balance
2359 cycles are used for converge to balance
1464 cycles are used for converge to balance
2380 cycles are used for converge to balance
1902 cycles are used for converge to balance
1421 cycles are used for converge to balance
1038 cycles are used for converge to balance
2779 cycles are used for converge to balance
1986 cycles are used for converge to balance
2127 cycles are used for converge to balance
1121 cycles are used for converge to balance
average time is 1803.696 cycles in 1000 trials
➜ HW5 git:(master) ✗
```

For k = 10:



For k = 100:



It is obvious that all the 3 cases are finally in convergence. As you can see, the larger k is, the more cycles needed for convergence when the load units are in the same distribution.