cs230-hw5

chiyu cheng

March 2019

For this homework, The definition of "balanced among neighbors": If the load units of a processor p is not higher or lower than its two neighbor by 2.

The definition of "balanced state of the system": If all of the processors are qualified with the definition "balanced among neighbors," then the systems reaches the balaced state.

1 Program Description

The logic of this program is first initialing a random load unit number for each processor, then scheduling the load activity randomly for each processor in range [10,1000] in a hashmap, then keep repeat this schedule over and over until the system reaches the balanced state or time limits. Each loop works as one time unit, the system will monitor the time and select the right processor to do its load activity with its neighbors. After that, the system will check if the system reaches the balanced state. Return the time counter if it reaches the state, otherwise return -1

2 Results and analysis

For the original balancing strategy:
Took 6833 circles to reach the balance for 5 processors
Took 32317 circles to reach the balance for 10 processors
Took -1 circles to reach the balance for 100 processors

The estimated time for 5 processor is around 7000 time units, for 10 processors is around 30000 time units, and it is not working for 100 processors. This strategy is not working because it can't handle the case where the load units of a processor p is lower or equal than the average of it and its neighbors. For example,
The load unit for processor 30 is 427,

The load unit for processor 31 is 416
The load unit for processor 32 is 406
The load unit for processor 33 is 396
The load unit for processor 34 is 395
In this case, for processor 31, it reaches the average of its neighbor and its values, thus it cannot make any changes on its two neighbor. For processor 32, same reason, it can't change the value of its two neighbor too. To solve this question, we need to modify the strategy where the two neighbors should be able to give and request as well as the processor. Here is the result for the new strategy.

After new load balancing strategy:
Took 2414 circles to reach the balance
Took 3929 circles to reach the balance
Took 107916 circles to reach the balance

3 Full Code

```
#include <iostream>
#include <vector>
#include <random>
#include <chrono>
#include <unordered_map>


using namespace std;

/*
Test different processor number {5,10,100}
Load units [10,1000]
Time interval [100,1000]

program logic:
Initial a random load unit number for each processor
Run 1001 loop and treat each loop as time unit
    if the time match with schedule_time,
        using the balance strategy
        check if all the processors reaches the balance state
            stop and return the time if it does

Definition of balance state:
    if all of the processors has same or -1 or +1 units with their two neighbors, then balance
balance strategy:
    Add the load units of three processor togather, and calculate the differece between averag
    Possible cases need to be handled for processors p1,p,p2 with average load unit x.
        p>x, p1<x, p2<x : give the extra unit to p1 to help it reaches x, if still have left u
        p>x, p1<x, p2>x : give the extra unit to p1 to help it reaches x
        p>x, p1>x, p2<x : give the extra unit to p2 to help it reacheds x
*/

int mod(int a,int b) {
    int res = a % b;
    if (res < 0){
        res += b;
    }
    return res;
}


struct processor{
    int load_units=-1;
};
int rand_uniform(int low, int high){
    /*
    Credit to the stackoverflow answer
    https://stackoverflow.com/questions/32575167/generate-random-numbers-with-uniform-distribu
    */

    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    //unsigned seed = 0;
    static std::default_random_engine rand_engine(seed);

    std::uniform_int_distribution<int> res (low,high);
    int r = res(rand_engine);
    return r;
}

void new_load_balance(processor &p1, processor &p, processor &p2){
    int average = (p1.load_units + p.load_units + p2.load_units)/3;
    p.load_units = average;
    p1.load_units = average;
    p2.load_units = average;
    if( (average*3) < (p1.load_units + p.load_units + p2.load_units)){
        p2.load_units +=1;
    }
}

void load_balance(processor &p1, processor &p, processor &p2){
    int average = (p1.load_units + p.load_units + p2.load_units)/3;
    //cout<<"Load balancing ac for "<< p1.load_units << " : " << p.load_units << " : "<< p2.lo
    if(p.load_units > average){
        int diff = p.load_units - average;
        p.load_units -= diff;
        if(p1.load_units < average){
            int diff1 = average - p1.load_units;
            if(diff1 >= diff){
                p1.load_units += diff;
                diff = 0;
            }
            else{
                diff = diff - diff1;
                p1.load_units += diff;
            }
        }
        if(p2.load_units < average){
            int diff2 = average - p2.load_units;
            if(diff2 >= diff){
                p2.load_units += diff;
                diff = 0;
            }
            else{
                diff = diff - diff2;
                p2.load_units += diff;
            }
        }
        if(diff > 0){
            p.load_units += diff;
        }

    }
    //cout<<"After : Load balancing ac for "<< p1.load_units << " : " << p.load_units << " : "<

}
bool is_balance(vector<processor> processors,int k){
    for(int i =0;i< processors.size();i++){
        if(abs(processors[i].load_units - processors[mod(i-1,k)].load_units) > 2 || abs(proces
            return false;
        }
    }
    return true;
}
void print_processors(vector<processor> processors){
    for(int i =0;i < processors.size();i++){
        cout<<"The␣load␣unit␣for␣processor␣"<< i <<"␣is␣"<<processors[i].load_units<<endl;
    }
}
int process(int p_num){
    vector<processor> processors;
    unordered_map<int, int> map;
    //initial a random load unit number and scheduling the load activity time for each process
    for(int i =0;i<p_num;i++){
        processor p;
        p.load_units = rand_uniform(10,1000);
        int schedule_time = rand_uniform(100,1000);
        map[schedule_time] = i;
        processors.push_back(p);
    }
    int time_count = 0;
    for(int r = 0; r < 10000; ++r){ // round
        for(int t = 0; t<=1000; ++t){ // time loop
            time_count++;
            if(map.count(t)!=0){
                int processor_index = map[t];
                new_load_balance(processors[mod(processor_index-1,p_num)],processors[processor
                if(is_balance(processors,p_num)){
                    print_processors(processors);
                    return time_count;
                }
            }
        }
    }
    print_processors(processors);
    return -1;
}
int main(){
    int K[3] = {5,10,100};
    vector<int> res;
    for (auto p_num:K){
        int time = process(p_num);
        res.push_back(time);
    }
    for (auto r:res){
        cout<<"Took "<< r <<" circles to reach the balance " <<endl;
    }
}
```

Text after it . . .