T-79.4101 Programming Assignment 1: Process Re-Assignment

Remy Rojas 486404

March 1, 2015

1 Problem Description

The problem consists of a given number of processes which need to be re-assigned in order to find the configuration with the smallest cost. Cost can be of two types:

- Moving a process from machine A to B generates a cost induced only by the process
- When a certain capacity is breached in a machine, a cost is induced by the excess resource consumption on the machine. Every machine has its own capacity.

Additionally 3 constraints must be satisfied for a configuration to be a valid solution:

- MCConst: Machines have a maximum capacity (always greater than the one mentioned previously). If this capacity is breached the machine crashes, so a new process can not be moved to a certain machine if the latter does not have enough capacity.
- SSConst: Machines are grouped into Locations and Processes are grouped into Services. The Minimum Spread of a service is a property each service has, which denotes the number of locations in which processes belonging to the service must be. In other words, each service must have representatives in at least X locations.
- SCConst: Stipulates two processes from the same service can not run in the same machine.

2 Algorithm

The local search is performed with steepest descent, taking the process with the least cost and probing every machine available looking for the best cost difference amongst neighbors. Neighbors are defined by two assignments with one differing process-machine assignment. Once all processes were moved to the best option (if available) we implement simulated annealing by generating random neighbors a vast number of times without checking costs (only constraint satisfaction). The number of random neighbors to which we move is twice the number of processes.

```
while True:

    while iterations < number_processes:

        process = find_minimum_cost_process()

        for machine in assignment.machines:
            if (constraints_satisfied(process, machine)) and (
                local_costs_improvement(machine, process)):
                 candidates.add(machine)

        if candidates not empty:
            best_machine = min(candidates)
            assignment.update(best_machine, process)

        if global_cost_improvement(assignment, original) <
            current_global_cost:
            dump(assignment)

        iterations ++

    randomize(assignment)
```

3 Experiments

The programming and experimentation were carried out on Cloud9 IDE: a cloud based service running a VM with 512MB RAM memory.

The program succesfully improves the intanced initial solution. Although the upper bound of the solution is not reached within 5min of testing for large instances. The checker provided in the assignment does not complain about constraint violation when probing solutions, and the global costs calculated in the program (outputed to the console) correspond to the ones returned by the checker.

4 Conclusions

Unfortunately the results do not reache the upper boundary expected. Still this greatly depends on the function charged to generate a random assignment. Given enough time it should get to a very good solution. The settings in the experiment (cloud VM with 512MB RAM) are not optimal and way below standart physical machines. Nevertheless, the algorithm can be improved by looking into machines with the largest free capacity and prioritizing ignoring those where soft capacities overflow rather than checking for the smallest move cost.