Matching Theory and Virtual Machines

Kristen Hines
School of Electrical and
Computer Engineering
Virginia Tech
Email: kphines@vt.edu

Ferdinando Romano
School of Electrical and
Computer Engineering
Virginia Tech
Email: fmromano@vt.edu

***Abstract*—Insert abstract here.**

I. Introduction

This project is focused on applying matching theory to a virtual machine job assignment problem.

II. Related Work

Most similar topics: VM Cosheduling and VM Migration papers.

Related to VM Coscheduling: Distributed Selfish Load Balancing

Related to VM Migration: Seen As Stable Marriage and Online VM Shuffling (continuation of VM Migration work).

III. Method Description

High level description if needed.

A. Problem Formulation

The problem this paper is exploring is how to optimize job assigment to separate computer clusters. Each of these jobs perform differently on different cores types. These core types can be graphical processors, computational processors, or something else. Each of these clusters have different core types. Each job can only be divided into a finite number of threads, and each job is assigned to one computer at a time.

(probably a good place to introduce matching theory and the college admissions game). This problem can be formed as a college admissions game. The differences are that the institutions have multiple quotas, each applicant can fill multiple slots of different types, applicants prefer different slot types over others, and an applicant cannot be divided among multiple institutions.

Key assumptions for this problem are: the virtual machines will be treated like flexible computers, jobs are submitted at the same time, chosen jobs are completed simulatenously, unchosen jobs will be submitted with the next round, no indifference, and no externalities.

B. Proposed Algorithm

Things to do: Write the math and assumptions, state why we are doing this, and show that this will be stable as well.

The proposed algorithm is based on a deferred acceptance college admissions algorithm with special modifications to optimize it for the situation where the institutions have multiple quotas of different types and the applicants can fill multiple slots.

1) Algorithm: Step 1: Calculate the preferences of the jobs (applicants) and the computers (institutions). A job's preference for a particular computer is determined simply: given the processors available on each computer, if a job would peform faster on one computer than another, then the first computer is preferred over the second. Given the relative speeds of each processor at performing a particular job, the speed of that job on a particular computer is calculated by first choosing the fastest available processors until the job's processor limit is reached or there are no more processors available on the computer. Then, the speeds of the chosen processors are summed together to give the computer's total speed at the job. A job prefers one computer over another if its total speed is higher than the other's.

A computer's preference for a particular job is based on the assumption that a computer wants to maximize utilization of its resources. In this case, a job that can utilize more processors than another is preferrable. It is assumed that the number of processors that a job can use does not depend on the computer or the processor types and so each computer has the same preference ranking of jobs.

Step 2: Perform a 1-1 Matching. The jobs are matched to the computers according the calculated preferences using a college admissions algorithm where the quota of a computer is set to 1 if it has at least 1 processor still available. Otherwise, its quota is set to 0.

Step 3: Determine the most important matching. A job that can use the greatest number of processors is the most highly preferred and so it is matched to its first choice of computer. Thus this pair is stable and can be assigned to the finalized matching of the algorithm. This job and the processors it uses are no longer available so they are removed from future iterations of the algorithm.

Step 4: Return to Step 1. The algorithm is repeated either until all processors are assigned a job or until all jobs are matched to a computer.

2) Guaranteed of Termination: The algorithm is guaranteed to terminate because there is a finite number of jobs and each iteration of the algorithm matches one job to a computer.

3) Stability of Algorithm: The proposed algorithm produces a stable matching because in each iteration, the college admissions game is used to find a set of stable matchings. Out

of the jobs listed in the resultant set of stable matchings, the job that can use the most processors is preferred most by every computer. Thus, that job will be matched with its first choice and its matching to a computer is a stable matching. Thus, each pair produced by an iteration of the proposed algorithm is stable and therefore the final matching is stable.

4) Optimality of Algorithm: Whether the matching is optimal can be understood in multiple senses. In this section, three different approaches to optimality are discussed as they apply to the proposed algorithm.

Resource Utilization: A simple goal of the proposed algorithm would be to maximize processor utiliztion so that no computing resources go unused/wasted.

The proposed algorithm does not alway maximize processor utilization. However, it does in every iteration where the preferred computer of the job with the greatest possible processor utilization has at least as many processors available as either i) that job can use or ii) any other computer has. This situation is common because, often, a computer with more available processors will outperform one with fewer. The exceptions occurs where there is a computer that has special purpose procesors that significantly outperform those available at other computers and this computer does not meet either of conditions i) or ii) listed above.

Total Job Completion Time: The total computation time, i.e., the sum of total computation times for each job, is another good measure of the optimality of the proposed algorithm.

Assuming individual jobs cannot take advantage of processors previously used by other jobs that have completed, the proposed algorithm minimizes total computation time whenever jobs that use more processors are jobs that would take longer to complete than any other job. By 'take longer to complete', we mean take longer than other jobs if the other jobs were to use a subset or superset of the processors used by the first job. When this condition is met, the job that takes the longest is given the greatest speed possible, the job that takes the 2nd longest is given the next greatest speed possible for it, and so on. Thus, total computation time is minimized.

In the proposed algorithm, this condition that jobs use more processors take longer is not guaranteed. However, it is strongly encouraged by the proportional fairness of the algorithm: Jobs that would take longer to complete are incentivized to be able to use more processors.

Proportoinal Fairness: In the proposed algorithm, jobs' individual computation times/total required processing are not factored into the preferences and so have no bearing on the matchings. Instead, it is the processor utilization ability of a job that effects its ranking. This leads to a proportional fairness in which jobs that are shorter are still given a fair amount of processing power so that they will not take very long. On the other hand, jobs that require more processing power, i.e. would take longer, are incentivized to be able to use more pocessors than jobs that do not take as long.

For a job that would take time to complete $\tau_1 > \tau_2$, where τ_2 is the completion time for a second job, Job 1 would reduce its completion time by an absolute amount $\Delta\tau_1 = \tau_1 - \frac{\tau_1}{f}$ if it could increase its speed by a factor f. Similarly, for Job 2, $\Delta\tau_2 = \tau_2 - \frac{\tau_2}{f}$. Thus, $\tau_1 = \Delta\tau_1(1 - \frac{1}{f})$ and $\tau_2 = \Delta\tau_2(1 - \frac{1}{f})$. Since $\tau_1 > \tau_2$, we have $\frac{\Delta\tau_1}{1-\frac{1}{f}} > \frac{\Delta\tau_2}{1-\frac{1}{f}} \implies \Delta\tau_1 > \Delta\tau_2$. Therefore, Job 1 has more to gain by increasing its speed by a given factor than Job 2 does and so Job 1 has a greater incentive to be able to use more processors.

IV. Results and Discussion

Show results and discuss what they represent.

V. Conclusion

The conclusion goes here.

VI. Future Work

This paper studies an interesting extension of the college admissions algorithm where the institutions have multiple quotas of different types and the applicants can fill any of a particular instution's quotas. Other, similar modifications to the college admissions algorithm may be worthy of interest such as having each applicant only able fill certain quota types rather simply preferring some types over others. The proposed algorithm of this paper could also be extended to consider scenarios that are complicated by software licensing restrictions or the need to factor in job length and complexity. These considerations could lead to more realistic approximations so that the algorithm can more practically lend itself to real-world implementation.

VII. Individual Contributions

Enumerated individual contributions.

Acknowledgment

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.