Matching Theory and Virtual Machines

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

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

***Abstract*—The assignment of jobs to separate compute clusters can be approached using matching theory. The problem is modeled as a modified college admissions game where each institution has multiple quotas, each of which is of a specified type. An algorithm based on multiple iterations of the colleged admissions deferred acceptance algorithm is proposed. The algorithm is shown to terminate, result in a stable mathcing, and, under certain common conditions, approximate an optimal solution. The proposed algorithm is simulated and shown to result in either significant improvement or only minor regression compared to other approaches.**

I. Introduction

Because cloud computing is a cost-effective and flexible system for handling data and programs, it allows ubiquitous IT services, ranging from online social networking services to infrastructure outsources. These cloud computing services are packaged in the form of virtual machines (VMs) through virtualization technology, which allows for computing technologies to be virtualized by emulating processors, main memory, storage, and networking devices [resource needed].

A primary benefit of VMs is they can be configured to suit a specific application's needs, such as application isolation, security requirements, service level-agreements, and computational performance. [resource needed] These VMs and cloud computing servers are still housed on powerful physical machines at this point. When, due to timing constraints or other restrictions, each VMs resources are static, sourcing jobs to individual VMs involves optimizing job-VM matchings to maximize resource utilization and speed of job completion. The problem is similar to the assignment of jobs to distinct compute clusters present on a campus network where the clusters cannot exchange resources.

This project focuses on turning a simple VM job assignment problem into a college admissions game. The jobs act as applicants, and the VMs as institutions. The jobs apply for spots on the VMs until either there are no more jobs in the queue, or there are no more VMs created and availabe for the users at the time. It is assumed that VM creation is a significantly involved process such that a user would wait for an existing VM to become available rather than instantiate a new one. In this paper, a computer-optimal algorithm will be proposed to provide a solution that is better for both the virtual machines and the jobs.

The paper proceeds as follows: Section II covers background knowledge for matching games. Section III covers the problem formulation and proposed algorithm. Section IV discusses simulation results. Section V reviews literature similar to the work of this paper. Section VI concludes the present work and Section VII suggests areas of future research.

II. Background

A review of the stable marriage problem is presented here to help the reader understand the proposed solution. The stable marriage problem is a one-to-one matching model used to effectively match two groups of agents together, such as men and women for a marriage. These agents are two disjoint sets. Each agent has a complete, strict, and transitive preference over other individuals, which means the agent is not indifferent for any choices. In addition to this quality, each agent has a chance of being unmatched. For demonstration purposes, the two agent sets are men and women, whose sets are $W = \{w_1, w_2, \dots w_p\}$ and $M = \{m_1, m_2, \dots m_n\}$, where p does not have to equal n.

Their preferences are arranged and represented as ranked ordered lists. An example of such a list is $p_{mn} = w_2, w_4, \dots, \emptyset$, where w_2 is the man's first choice for a partner, w_4 is the second, and so on. In this case, the final choice represents when the man prefers to be single over his possible choices.

Definition 1: An outcome is a matching $\mu : M \times W \to M \times W$ such that $w = \mu(m)$ if and only if $m = \mu(w) \in W \cup \emptyset$, $\mu(m) \in M \cup \emptyset$ for all *m, w*.

This implies that agents from one set are matched to either the agents of the other set or to the null set. Agents' preferences over outcomes are determined only by their own preferences for certain partners.

Definition 2: A matching μ is stable if and only if it is individual rational and not blocked by any pair of agents.

Individual rational and blocking pair are defined as followed.

Definition 3: A matiching set is individual rational to all agents if and only if there does not exist an agent *i* who prefers being unmatched to being matched.

Definition 4: A matching μ is blocked by a pair of agents *(m,w)* if they prefer each other to the partner they receive at u. That is, $w \succ_m \mu(m)$ and $m \succ_w \mu(w)$. Such a pair is called a blocking set, where $\succ$ represents an agent's preference of one individual over another.

This means that as long as a matching is not blocked and a matching set is individually rational, a matched set will be stable. Therefore:

Theorem 1: A stable matching pair exists for every marriage market.

This theorem was proposed and proven by Gale and Shapley through their deferred acceptance algorithm, as demonstrated in [1].

The college applications game extends the concepts behind stable marriage. For a college applications game, the two sets of players are the institutions and the applicants. A single institution can be matched to multiple applicants at one time.

Theorem 2: Every applicant is at least as well off under the assignment given by the deferred acceptance procedure as he would be under any other stable arrangement.

The end result is a stable and optimal pairing between the institutions and the applicants. This is also proven in [1]. One of the issues with the stable marriage problem and the college admissions problem is that they are applicant optimal. In other words, the set of agents being applied to, such as the women in the earlier example, are not guaranteed their optimal choices. There may exist another stable matching where at least one woman receives a more preferred matching and no women receive a less preferred matching. If the men and women switched, and the women were the ones proposing to the men, then it would be the men who are not guaranteed their optimal choices.

III. Method Description

The problem formulation and proposed algorithm are described.

A. Problem Formulation

This paper explores the problem of optimal assignement of jobs to separate computer clusters. Each of these jobs perform differently on different cores types where 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.

The problem can be modeled as a college admissions game with a few key differences: The institutions have multiple quotas, each applicant can fill multiple slots of different types, and applicants prefer some slot types over others. As before, an applicant cannot be divided among multiple institutions.

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

B. Proposed Algorithm

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.

Data: Number of cores per VM available, Speed ratio matrix for jobs, Max threads used per job
Result: Matrix with jobs matched to VMs
Initialize preference matrices and quotas;
while *Either jobs or VMs are left* **do**
  1. Set quota to 1 for each non-full VM;
  2. Run Deferred Acceptance Algorithm;
  3. Choose job who used the most of the computer resources and keep it;
  4. Save this job to the results matrix;
  5. Update preference matrices and core availability matrix;
end
Compile results and send to user

Algorithm 1: Proposed Algorithm

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.

Proportional 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

The proposed algorithm was compared to three different algorithms. The first was a simple matching scheme, where jobs are matched with their preferred VM. This case would be similar to a VM lab where each user had to apply for a VM without knowing how many other people were applying. This scheme disregards the availability of resources on the computer, which means some jobs may be matched to the computer, but they are unable to complete in a reasonable time due to lack of resources. This is the scheme all of the other three variations of the deferred acceptance algorithm, including the proposed algorithm, are compared to.

The second scheme involved a deferred acceptance routine where the quotas depended on how many jobs were listed and how many VMs were available. It will take those two values and divide the jobs among the available computers. This scheme is intended to favor the jobs that are able to be serviced by the VM. However, this case faces the same issue as simple matching, where there may not be enough resources for all of the matched jobs.

The third algorithm is another deferred acceptance algorithm variant. For this scheme, the deferred acceptance algorithm will match a set of jobs to a set of VMs for each iteration. This is handled by setting the quotas to one for each iteration until a VM is filled. Once a VM is filled, its quota is set to 0, and no more jobs can be matched to that VM. Each iteration, the preferences for the VM and the jobs are updated in order to reflect which jobs are matched and what resources are still available. This process will end once all of the jobs are matched or all of the VMs are filled. This algorithm is expected to the most fair out of the three because it will not match jobs to a VM that is not able to complete it.

Five different metrics were used to observe the performance of the proposed algorithm compared to the simple matching algorithm and two different versions of the deferred acceptance algorithm. The metrics were average time scores, average percent of jobs assigned to each VM, average percent of available threads utilized for a job, the average number of cores used per VM, and a resource score that represented how well the jobs were able to utilize their preferred cores. The average time score is based on the speed ratios for each job depending on the core type. For this number, higher scores are better, but they can only be compared within a run. To enable comparisons across the different runs, these values were turned into percent changes. The simple matching scheme was set as the baseline.

The next metric was the average percent of jobs assigned to a VM. This percent represents the number of jobs that were successfully assigned to a VM before the number of available cores were depleted. The average percent of available threads utilized for a job was meant to represent how efficient each scheme was. A job did not need to have all of its threads used

to be completed, but the job would be finished faster if more threads were used. The average number of cores used per VM is meant to represent how effectively the VM resources were used. For this paper, the VMs prefer to be completely utilized. If a VM was unmatched with any jobs, then this score would reflect that. Finally, the last score is the resource score, which represents how well the jobs were distributed among the core types. This score reflects how often each job was executed the best it could be on a computer.

	Comparison		
	I	II	III
Number of Simulations	500	100	100
Number of Computers	[1,20]	[1,50]	[1,20]
Number of Jobs	[1,50]	[1,100]	[1,100]
Number of Core Types	[1,5]	[1,10]	[1,5]
Max. Number of Cores per Type	[1,25]	[1,50]	[1,50]
Max. Number of Threads per Job	[1,50]	[1,200]	[1,50]
Speed Ratios	[1,100]	[1,200]	[1,200]

TABLE I. RANDOMIZED INPUTS TO COMPARISONS

	S. M.	D. A. 1	D. A. 2	Prop. Alg.
Avg. Time Score	780	722	942	989
Percent Jobs Assigned	24.92%	41.67%	37.1%	34.49%
Avg. Avail. Threads Used	71.41%	78.96%	65.57%	64.83%
Avg. Cores Used	63.63%	87.94%	98.57%	98.53%
Avg. Resources Score	54.49%	62.41%	50.02%	49.5%
Avg. Time Score Change from Simple Matching	0.00%	-7.496%	20.82%	26.78%

TABLE II. COMPARISON I

	S. M.	D. A. 1	D. A. 2	Prop. Alg.
Avg. Time Score	8668	8185	10524	9161
Percent Jobs Assigned	57.93%	75.15%	74.45%	74.63%
Avg. Avail. Threads Used	88.65%	85.81%	85.00%	79.91%
Avg. Cores Used	72.25%	87.51%	99.21%	86.50%
Avg. Resources Score	67.09%	66.49%	61.90%	53.64%
Avg. Time Score Change from Simple Matching	0.00%	-5.57%	21.41%	5.69%

TABLE III. COMPARISON II

	S. M.	D. A. 1	D. A. 2	Prop. Alg.
Avg. Time Score	2742	2541	3724	3516
Percent Jobs Assigned	48.12%	68.78%	57.06%	59.73%
Avg. Avail. Threads Used	89.65%	95.17%	92.89%	84.35%
Avg. Cores Used	47.48%	86.52%	89.93%	88.44%
Avg. Resources Score	72.73%	80.30%	74.21%	72.60%
Avg. Time Score Change from Simple Matching	0.00%	-7.32%	35.79%	28.21%

TABLE IV. COMPARISON III

V. RELATED WORK

Variations of this work can be found, and those variations handle a variety of problems in multiple areas of virtualization and cloud computing. These problem include topics such as virtual machine co-scheduling, general networking situations were defining utility functions may be difficult, VM migration in cloud computing, distributed loads for VMs, and VM shuffling for congestion reasons. A few of these examples will be discussed.

In "Seen As Stable Marriages," the authors prepared the background for matching theory to be applied to networking problems, such as ones where utilities difficult or impossible to find. Instead of trying to pursue optimality, they aimed for stability. They developed a possible solution for non-centralized coordination between ISPs in an ISP peering example. The simplicity that Matching Theory affords, along with privacy benefits, makes this approach interesting. However, the issue with their work is that it is polarized towards the proposing side of the two parties.

This issue was addressed in their next work, "Egalitarian Stable Matching for VM Migration in Cloud Computing," the authors sought to reduce the polarization issues they had in the previous paper, similar to what was done in this paper. The problem they examined was server maintenance scenario, where VM migration is triggered by periodic upgrades, maintenances, and hardware failures. They introduce a current match dissatisfaction score, a metric for each agent, which was used to encourage a more egalitarian solution. This dissatisfaction score was derived from how dissatisfied each agent was in its current pair. The results from this work showed that the overall dissatisfaction was reduced while keeping the stability matching theory offers. The drawback behind this work is that their algorithm performed poorly when the total quotas of the servers was close to the total number of migrating VMs.

A problem faced in virtualization technology is the limits introduced by the physical machines the virtual machines have to run on. Dhillon, Purini, and Kashyap attempt to handle this performance degradation by using matching theory to create a co-scheduling algorithm. They defined this degradation issue as a stable roommate problem, where the VMs are paired together on a machine based on their compatibility and their likelihood of interfering with each other. Their work was then compared with other algorithms that handled co-scheduling. Their work showed that the stable marriage and stable roommates problems provide no improvement over current technology available for co-scheduling. However, this work does give a starting point for possible variants of the stable matching and stable roommates problems.

VI. CONCLUSION

The conclusion goes here.

VII. 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.

VIII. Individual Contributions

Kristen Hines formulated the problem, and tested the proposed algorithm against other approaches which she implemented along with the college admissions algorithm to produce the results of the project. She also researched the background literature related to the project and contributed to this written report.

Ferdinando Romano formulated the problem, and developed and implemented the proposed algorithm. He developed other implementations against which the proposed algorithm was tested and he also derived the results about the algorithm's properties relating to termination, stability, and optimality and contributed to this written report.

References

[1] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, p. 9, Jan. 1962.

[2] H. Xu and B. Li, "Seen as stable marriages," in *2011 Proceedings IEEE INFOCOM*, Apr. 2011, pp. 586–590.

[3] ——, "Egalitarian stable matching for VM migration in cloud computing," in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2011, pp. 631–636.

[4] X. Wen, K. Chen, Y. Chen, Y. Liu, Y. Xia, and C. Hu, "VirtualKnotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter," in *2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2012, pp. 12–21.

[5] J. Dhillon, S. Purini, and S. Kashyap, "Virtual machine coscheduling: A game theoretic approach," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2013, pp. 227–234.

[6] C. P. Adolphs and P. Berenbrink, "Distributed selfish load balancing with weights and speeds," in *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, ser. PODC '12. New York, NY, USA: ACM, 2012, pp. 135–144.

[7] Y. Jiang, X. Shen, J. Chen, and R. Tripathi, "Analysis and approximation of optimal co-scheduling on chip multiprocessors," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 220–229.

[8] D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, and Y. Morita, "Hard variants of stable marriage," *Theoretical Computer Science*, vol. 276, no. 12, pp. 261–279, Apr. 2002.

[9] A. E. Roth, "Stability and polarization of interests in job matching," *Econometrica*, vol. 52, no. 1, p. 47, Jan. 1984.