Matching Theory and Virtual Machines

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

Ferdinando Romano
School of Electrical and
Computer Engineering
Virtual 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 matching, 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 facilitates ubiquitous IT services, ranging from online social networking services to infrastructure outsources, to be available to a wide variety of people. These cloud computing services are packaged in the form of virtual machines (VMs) through virtualization technology. This ability enables computing technologies to be virtualized by emulating processors, main memory, storage, and networking devices [5].

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 [4]. These VMs and cloud computing servers are still housed on powerful physical machines at this point. When each VM's resources are static, sourcing jobs to individual VMs involves optimizing job-VM matchings to maximize resource utilization and speed of job completion. This can happen due to time constraints or other restrictions to the VM 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 VM-optimal algorithm will be proposed to provide a solution that is better for both the VMs 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 $\mu : M \times W \rightarrow M \times W$ 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 individual 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. A better solution would be one where the results between the applicants and the institutions were equally optimal.

III. Method Description

The problem formulation and proposed algorithm are described in the following subsections.

A. Problem Formulation

This paper explores the problem of optimal assignement of jobs to separate VMs. Each of these jobs perform differently on different cores types where these core types can be graphical processors, computational processors, or something specifically for a certain program. Each of these VMs have different core types, each job can only be divided into a finite number of threads, and each job is assigned to one VM 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 VMs are viewed as 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. Perform a one-to-one matching using the Deferred Acceptance Algorithm;
  2. Determine most important matching;
  3. Save this match to the results matrix;
  4. 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 VMs (institutions). A job's preference for a particular VM is determined simply: given the processors available on each VM, if a job would peform faster on one VM than another, then the first VM 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 VM 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 VM. Then, the speeds of the chosen processors are summed together to give the VM's total speed at the job. A job prefers one VM over another if its total speed is higher than the other's.

A VM's preference for a particular job is based on the assumption that a VM 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 VM or the processor types and so each VM has the same preference ranking of jobs.

Step 2: Perform a one-to-one matching. The jobs are matched to the VMs according the calculated preferences using a college admissions algorithm where the quota of a VM 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 VM. 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 VM.

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 VM.

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 VM. Thus, that job will be matched with its first choice and its matching to a VM 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 always maximize processor utilization. However, it does in every iteration where the preferred VM 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 VM has. This situation is common because, often, a //vm with more available processor resources will outperform one with fewer. The exceptions occurs where there is a VM that has special purpose processors, such as ones that are developed for specific tasks. Those exceptions will significantly outperform those available at other VMs. This VM 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 alternative 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 applies for a VM without knowing how many others are also applying. This scheme disregards the availability of resources on the VM, which means some jobs may be matched to the a VM, but then are unable to complete in a reasonable time due to lack of resources. This is the scheme to which the other three variations of the deferred acceptance algorithm, including the proposed algorithm, are compared.

The second scheme involves a deferred acceptance routine where the quotas depend on how many jobs are listed and how many VMs are available. It takes these two values and divides the jobs among the available VMs. 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: there may not be enough resources for all of the matched jobs.

The third algorithm is another deferred acceptance algorithm variant. In each iteration of this scheme, the deferred acceptance algorithm matches a set of jobs to a set of VMs. 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 ends once all of the jobs are matched or all of the VMs are filled. This algorithm is expected to be the most fair out of the three because it does not match jobs to a VM that would not able to complete it.

Five different metrics characterize the performance of the proposed algorithm compared to the simple matching algorithm and the two different versions of the deferred acceptance algorithm. The metrics are average speed 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 represents how well the jobs could utilize their preferred cores. The average speed 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 single simulation. To enable comparisons across the different simulations, these values were converted to percent changes. The simple matching scheme was set as the baseline, which is why each of its sections for percent change are zero percent.

The next metric, the average percent of jobs assigned to a VM, represents the number of jobs successfully assigned to a VM before the available cores were depleted. The average percent of available threads utilized for a job represents how efficient each scheme is. A job does 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 represents how effectively the VM resources are used. In this work, the VMs prefer to be completely utilized. This score would also reflect whether a VM is matched to any jobs at all. Finally, the last score is the resource score, which represents how well the jobs are distributed among the different core types. It reflects how frequently each job is executed the best it could be on a given VM.

These algorithms were compared in three different tests. For each test, a set of randomly defined inputs were entered into each algorithm for each iteration. During these iterations, the metrics were sampled, averaged, and stored until all of the simulations were finished. All random numbers were generated using the uniform random function in MATLAB. The parameters for the inputs of the comparison function for each iteration can be found in Table I.

The number of simulations represents how many simulations the comparison program executed. The number of VMs parameter decided the minimum and maximum number of VMs, from 1 to a positive integer, during an iteration. The number of jobs parameter decided the maximum and minimum number of jobs during each iteration. The number of core types parameter decided the minimum and maximum number of different core types that would be available for the user. The maximum number of cores per type parameter decided how many of which type of core were available for a job. The maximum number of threads per job parameter decided how big the job could be and how many threads could be matched to available cores. The speed ratios parameter decided how fast each job would be on each core type.

Each algorithms is referred to according to the following abbreviations: S.M. refers to the simple matching algorithm. D.A. 1 refers to the deferred acceptance algorithm with evenly distributed quotas. D.A. 2 refers to the deferred acceptance algorithm which performs a one-to-one matching each iteration. D.A. 2 is the algorithm whose preference matrices change each iteration due to jobs being matched and cores being used. Prop. Alg. refers to the proposed algorithm introduced in this paper.

The test cases represent three possible scenarios: one where the VMs and jobs are evenly matched, one where there are many available VM resources, and one where the number of jobs typically outnumbers the number of available VMs. For these cases, the job performance is measured by the percent jobs assigned, average available threads used, average resource score, and average percent change in the speed score compared to the simple matching scheme. The VM performance is based on the average percent of cores used because its main requirement was to have all of the resources among all of the VMs used. All values used for comparisons are defined as percentages.

	Comparison		
	I	II	III
Number of Simulations	500	100	100
Number of VMs	[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. Speed 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. Speed Score Change from Simple Matching	0.00%	-7.496%	20.82%	26.78%

TABLE II. COMPARISON I

A. Comparison I: Normal amount of VMs to jobs

Table II shows that, in this comparison case, the simple matching performed worse compared to the other approaches on average available threads used, average cores used, and average resource score. The first deferred acceptance algorithm performed similarly to simple matching. Both the second deferred acceptance algorithm and the proposed algorithm performed similarly to each other and better on average than the other two for this example. The first deferred acceptance algorithm performed better in terms of resources used and threads used for this example. The proposed algorithm performed its jobs the fastest.

B. Comparison II: More VM resources compared to jobs

For this comparison, the second deferred acceptance algorithm does as well as or better than the other algorithms. As can be seen in Table III on page 5, the proposed algorithm and the first deferred acceptance algorithms performed similarly on most metrics except average resource score and average percent change in the speed score. The proposed algorithm did better for average percent change in the speed score, but worse for average resource score. The simple matching algorithm performed similarly to the first deferred acceptance algorithm on everything except percent of jobs assigned.

C. Comparison III: More jobs than VM resources available

For this case, the second deferred acceptance algorithm performed the best in everything except percent jobs assigned and average resource score, as shown in Table IV on page 5. The first deferred acceptance algorithm performed the best in those two categories. The proposed algorithm's results lie between the first and second deferred acceptance results. The simple matching performed worse than the other algorithms on average for this case.

D. Discussion of Results

In all cases, the first deferred acceptance algorithm average resource score and percent of jobs assigned was usually the

	S.M.	D.A. 1	D.A. 2	Prop. Alg.
Avg. Speed 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. Speed 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. Speed 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. Speed Score Change from Simple Matching	0.00%	-7.32%	35.79%	28.21%

TABLE IV. COMPARISON III

highest or second highest compared to the other algorithms. It was also the slowest at job completion compared to the others. The second deferred acceptance algorithm performed generally the best in the rest of the categories, with the proposed algorithm measuring close to it. The simple matching algorithm performed normally as well as the first deferred algorithm. Its main weakness was that it normally did not have the resources to service many of the available jobs, which would make users very unhappy.

Compared to the other cases, the proposed algorithm did not have a significant strengths over the other algorithms. On average, it performed better than the simple matching and first deferred acceptance algorithm for most of the metrics. In addition, it was normally as good as the second deferred acceptance algorithm. This algorithm was as VM-optimal in terms of cores used as the second deferred acceptance algorithm, and more optimal compared to the other two algorithms. As for job-optimal, it was not as good as the first deferred acceptance algorithm. However, the proposed algorithm created better matches for jobs than simple matching.

V. RELATED WORK

Variations of this work can be found, handling a variety of problems in multiple areas of virtualization and cloud computing. These problems include topics such as VM co-scheduling, general networking situations where defining utility functions may be difficult, VM migration in cloud computing, distributed loads for VMs, and VM shuffling for congestion reduction. A few of these examples these works are discussed here. [7] [6]

In [2], the authors prepared the background to apply matching theory to networking problems, such as those where utility functions are difficult or impossible to find. Rather than pursuing optimality, they aimed for stability and 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, their work is polarized towards the proposing side of the two parties.

This issue was addressed in their next work [3] where the authors sought to reduce the polarization issues they had in the previous paper, similar to the approach in this project. The problem they examined was server a 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 used to encourage a more egalitarian solution. This dissatisfaction score was derived from each agents individual dissatisfaction with its current pair. The results showed that the overall dissatisfaction was reduced while maintaining stability. The drawback behind this work is that their algorithm performed poorly when the number of 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 that host the VMs. Dhillon, Purini, and Kashyap [4] attempt mitigate this performance degradation using matching theory to create a co-scheduling algorithm. They defined the 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 and showed that the stable marriage and stable roommates problems provide no improvement over current technology available for co-scheduling. However, their work opens the way for approaches based on variants of the stable matching and stable roommates problems.

VI. CONCLUSION

Though it does not prove to be a catch-all solution to matching jobs to VMs, the proposed algorithm does demonstrate itself to be the most advantageous of those considered here when working with situations similar to Comparison I: there are more jobs than VMs but the number of cores is limited and the speeds of each core type for a particular job are more similar. Even when the situation does not meet these conditions, however, the algorithm does not always perform too poorly compared to the best alternative. So, we can say that using the proposed algorithm is a reasonable approach in other scenarios as well. The scenarios considered in this work are not exhaustive and situations where the parameters are set to other ratios are worth consideration.

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. In addition, this work can be applied to the exampled mentioned in the related work section.

VIII. Individual Contributions

Kristen Hines formulated the problem statement with Ferdinando Romano. In addition, she wrote the code for the college admissions program. Once the proposed algorithm was written, she tested it against other approaches. These approaches were: simple matching, deferred acceptance with quotas that depended on the number of jobs and VMs, and deferred acceptance that matched jobs iteratively. Ferdinando Romano and Kristen also developed the metrics for job speed. Kristen created and implemented the resource metrics. She wrote a comparison function to attain the results for this paper. She also researched the background literature related to the project and contributed to this written report.

Ferdinando Romano formulated the problem statement with Kristen Hines. He developed and implemented the proposed algorithm. He also designed and implemented the functions to create the preference matrices for the VMs and jobs. Ferdinando Romano and Kristen Hines developed the metrics for job speed. He derived the results about the algorithm's properties relating to termination, stability, and optimality. He researched the problem formulation part of this paper and contributed to the 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] 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.

[5] 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.

[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.