Statistical Behaviors of Distributed Transition Planning

Ning Wu and Alva Couch

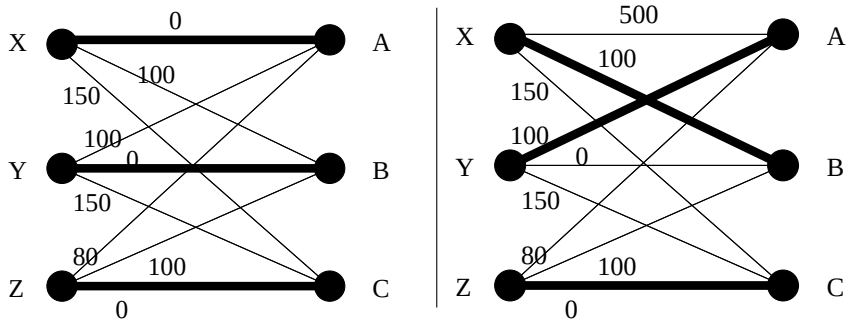
Tufts University, Medford, 02155, USA
{ningwu,couch}@cs.tufts.edu

Abstract. The transition planning problem is to move a system from an existing starting configuration to a desired final configuration at the lowest possible cost. Prior work shows that the transition planning problem can be reduced to a bipartite matching problem and the stable marriage algorithm can be used to approximate a minimum cost transition in a distributed environment. In this paper, we study the efficiency of this mapping, including its best-case, worst-case, and statistical behavior. We discuss the relationship between algorithm performance and the distribution of the input data. We show that while there are cases in which this algorithm performs poorly, works well when the costs of transitioning between distributed resources follow a normal distribution. We discuss the applicability of this algorithm to real-world situations in which resources are thus distributed.

1 Introduction

The transition planning problem for a system (such as a network) is to design a plan of operations that – when executed – will move the system from its existing configuration to a new configuration having desirable properties. In a previous paper [8], we discussed the distributed transition planning mechanism. Transition planning is reduced to a matching problem whose solution is approximated by solving the stable marriage problem. We apply distributed algorithms to compute the transition plan. We show that although these algorithms do not always achieve optimal solutions, they return acceptable results in most cases. Thus they can be used for system management.

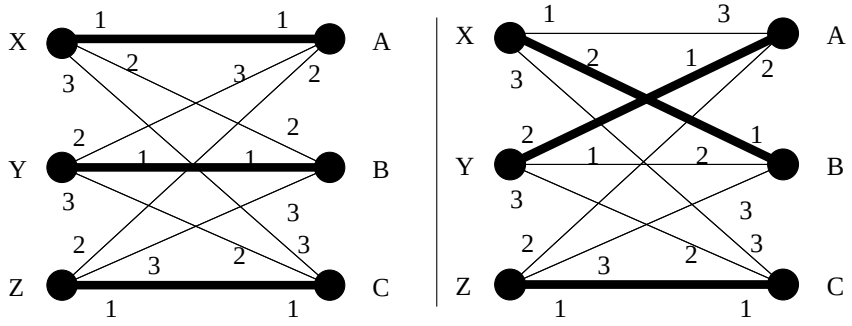
For example, suppose that there are three physical servers X, Y, and Z that act as web servers and serve three types of content that we call A, B, and C. Each server initially serves one form of content: server X serves content A, Y serves content B, and Z serves content C. We think of each kind of content as a *role* for the server. Initially, each server is matched to a unique role and the cost of changing roles is more than the cost of remaining at the same role. This is depicted as a bipartite graph, with X, Y, Z on one side and A, B, C on the other(Figure 1(a)). Edges represent cost of transition, which can be estimated by a variety of methods. In this bipartite graph, the cost of keeping the current server-role mapping is zero so we say that the system is at a *fixed point* [2, 1].



(a) Before catastrophe (b) After catastrophe

Fig. 1. The bipartite graph before and after the catastrophe.

When the environment changes (e.g., the users of content A increase), the processing power of servers serving content A may not be enough. We call such a change a transition catastrophe [8], to distinguish it from policy changes that do not incur configuration change and incur no cost. To provide the same level of service, some servers serving other content need to be transitioned to serve content A. The goal is to find a plan to meet the new requirements so that the transition happens at minimum cost. The difference between this and the prior state is that remaining at the current roles now has a nonzero cost, e.g., including the cost of violating a service-level agreement(Figure 1(b)). The minimum cost matching for this graph will not be the status quo. The Hungarian method [4] can be used to compute the minimum cost matching, which represents the minimum cost solution. The new minimum-cost matching is shown with bold lines. The roles have been rearranged so that SLAs are again met. The resulting reconfiguration achieves a fixed point again, because the cost of remaining in this state is again zero.



(a) Before catastrophe (b) After catastrophe

Fig. 2. The bipartite graph using rankings before and after the catastrophe.

However, in a distributed environment, the bipartite matching method for transition planning may not work properly for several reasons. Agents might not agree on how the costs are computed, or their units for cost may differ. We proposed in [8] to use a stable marriage algorithm, the Gale-Shapley algorithm [3], to solve the distributed transition planning problem. The first step in using this method is to convert costs to rankings. Each node ranks the edges connected to it in order of preference, from lowest cost to highest cost. For example, for Figure 1(a), the rankings are shown in Figure 2(a). If costs are at a fixed point, so are rankings. When a catastrophe arises, instead of considering costs (as in Figure 1(b)), agents will consider rankings, as illustrated in Figure 2(b). Thinking of agents as representing both sides, agents will attempt to form a set of *stable marriages*, so that there is no advantage to any server switching roles. Such a matching is shown again in bold.

2 Hospitals/Residents Problem

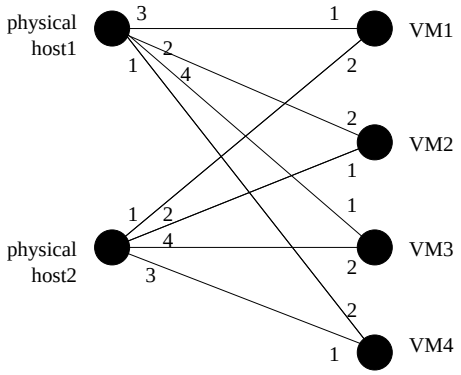


Fig. 3. Assign virtual machines to physical hosts

The stable marriage problem results in a one-to-one mapping. Another related problem is the Hospitals/Residents problem (a.k.a college admission problem) [3], which computes a one-to-many mapping. The hospitals/residents problem describes the matching process that is used to match medical students and hospitals having openings for residents; because a hospital only has limited positions, only the top candidates ranked by that hospital will be admitted. This problem is also called the college admission problem. The hospital/residents problem can again be solved by the Gale-Shapley algorithm [3]. In each round, each unmatched applicant proposes to the most preferred hospital that has not rejected his proposal.

We can investigate the resource allocation and transition problem by reducing it to the hospitals/residents problem. Let us use the virtual machine assignment problem as an example. In virtual machine (VM) management, a common task

is determining how to assign a VM to an appropriate physical machine. Figure 3 shows an example of assigning virtual machines to physical machines.

One challenge of applying the Gale-Shapley algorithm to VM assignment is how to determine the ranking of the VMs. An agent representing a physical host can rank VMs by many standards, such as the importance of the VM, the resource-utilization profile (e.g., CPU-bound, disk-bound) of the VM, or user preferences. We start from the simplest cases and build to a discussion of how to handle various kinds of constraints.

For example, suppose that there are four virtual machines VM_1, VM_2, VM_3, and VM_4, and two physical hosts PHY_1 and PHY_2 (Figure 3). The attributes of physical hosts are memory size, storage size, and network bandwidth. There are also several constraints that limit matching between the VMs and PHYs.

Suppose that our task is to deploy new applications to VMs that are hosted by physical hosts. First, the resources needed for applications are estimated, though these estimates can be adjusted later. The goal is to find steps (ideally in a distributed way) to provide resources for the new application and ensure minimum cost.

In the bipartite graph, the virtual machines are listed on the right side; the physical hosts are listed on the left side. Each node has a ranking of all the nodes on the other side, based on its own standard. The right side proposes to the agents representing the physical hosts and current VMs. As in the one-to-one case, once stabilized, the left side ranks current bindings the highest. The ranks among current bindings are decided by the physical host based on its own standard. The same three-step process applies here: first use the Gale-Shapley algorithm to get a stable matching; then change the bindings according to the matching; finally change the rankings on the left side to reflect the new bindings. This transition process is a deterministic fixed-point operator.

When a catastrophe arises, some VMs will be chosen to be served by other physical servers; this is called "swinging the physical servers to serve new VMs". Once transitioned, these VMs may be adjusted to meet the new requirements. For example, one problem might be to find enough bandwidth for a new application (by moving some existing applications around in the physical hosts).

Failover can be characterized as a transition catastrophe: when a physical server fails, the VMs running on it will try to locate new physical servers for their host environment. Agents monitoring these VMs will notice the outage and reflect it in their stable marriage rankings. Then these agents will propose to the rest of the physical hosts and determine whether the matching is feasible, because the capacity of each physical server is limited.

3 Quality of Stable Marriage and Hospital/Resident Solutions

The Gale-Shapley algorithm always produces a stable marriage, but there is some question as to how well that marriage substitutes for the more ideal transition

plan based upon costs rather than ranks. We first develop a measure of how well stable marriage works compared to the theoretical best solution.

Definition 1. *The approximation ratio of a stable marriage solution is the ratio of the cost of the stable marriage solution to the cost of the ideal solution obtained through minimum bipartite matching.*

The ratio is always >= 1.0. Values near 1.0 indicate that the stable marriage algorithm is working well, while high values indicate that it is working poorly. A value of 1.0 means that the stable marriage algorithm did as well as a bipartite matching solution in which all weights (costs) are collected in a central location. The approximation ratio may be thought of as the "cost of distributing" the algorithm onto an agent network.

The approximation ratio is highly distribution dependent; it differs with the statistical distribution of the input weights.

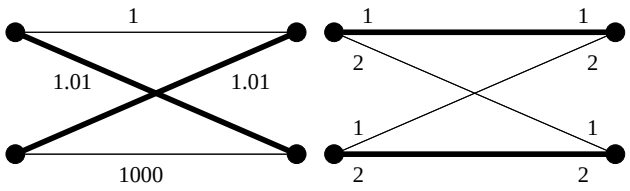
Theorem 1. *When all the weights in the bipartite graph are identical, the ratio of the total cost computed through stable marriage to the optimal solution is 1.0.*

Proof. Suppose that d edges need to be selected. When all the weights are the same, the total cost in any mapping is always $d \cdot weight$. Thus the approximation ratio is always one. □

Theorem 2. *Suppose the maximum ratio of two weights in the bipartite graph is k. In the worst case, the ratio of the total cost computed through stable marriage to the optimal solution (i.e. the approximation ratio) is less than or equal to k.*

Proof. Let $max(weight)$ and $min(weight)$ represent the maximum and minimum weights in the graph. Let $k = max(weight)/min(weight)$. In the worst case, the stable marriage or the hospital/resident algorithm selects the worst solution possible. Suppose that d edges need to be selected in the solution. The optimal solution has a total weight greater than or equal to $d \cdot min(weight)$, and the stable marriage or hospital/resident solution has a total weight less than or equal to $d \cdot max(weight)$. Thus the maximum ratio is less than or equal to k. □

The worst scenario is illustrated in the following example (Figure 4).



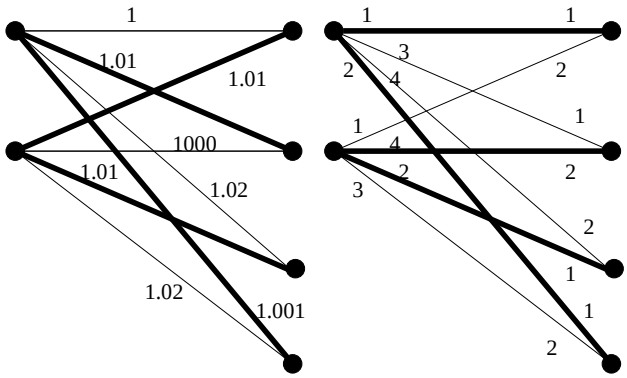
(a) weights (b) ranking

Fig. 4. The solution of the stable marriage problem vs. the solution of the assignment problem. Bold lines represent the matchings derived in each case.

Theorem 3. *Suppose that for one side of the bipartite graph, we select the maximum and minimum labels max_i and min_i for each node i on that side. Suppose that max and min are the sums of max_i and min_i over all i, respectively. Then the approximation ratio cannot be greater than max/min.*

Proof. The best possible outcome of a bipartite matching includes the minimum edge taken from every node, yielding a minimal score. The worst outcome of stable marriage includes the maximum edge taken from every node, yielding a maximum score. □

In this example, the optimal solution has a total cost of 1.01 + 1.01 = 2.02, whereas the stable marriage algorithm yields a total of 1000 + 1 = 1001. The ratio is almost 500, which is less than 1000/1 = 1000.



(a) weights (b) ranking

Fig. 5. The solution of the hospitals/residents problem vs. the solution of the matching problem. Bold lines represent the matchings derived in each case.

In the worst case, the approximation ratio can also be arbitrarily large in the hospitals/residents problem (Figure 5). In the figure, on the left side, each physical host has two slots and there are four VMs. The approximation ratio is almost 250.

In practice, the upper bound is tighter because there is no case in which the optimal solution chooses all minimum weights and the stable marriage solution chooses all maximum weights, because the stable marriage algorithm chooses at least one edge that has minimum weight.

4 Simulations

In the following we study the approximation ratio of the Gale Shapley algorithm and that of a variant utilizing a heuristic strategy.

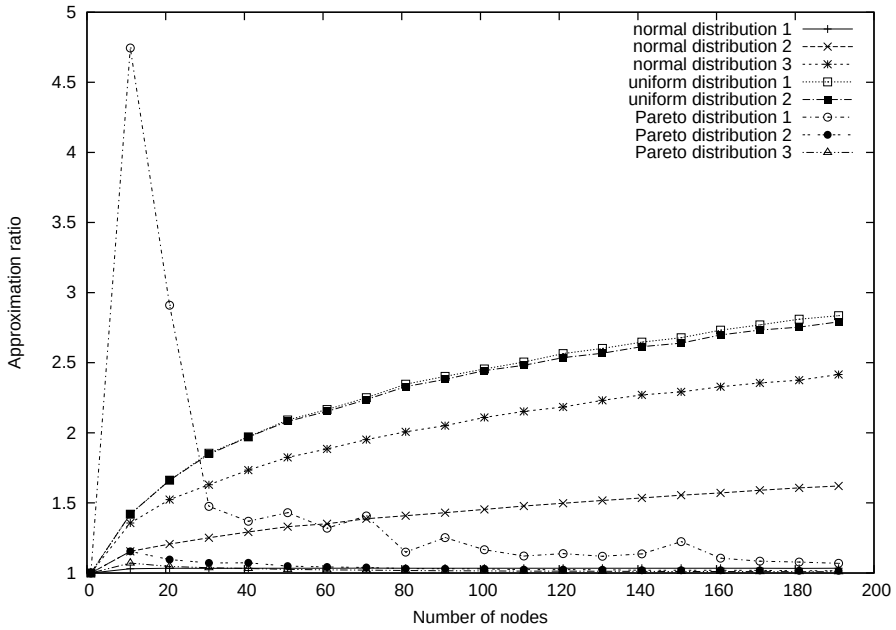


Fig. 6. The approximation ratio for weights following the uniform, normal, and Pareto distributions.

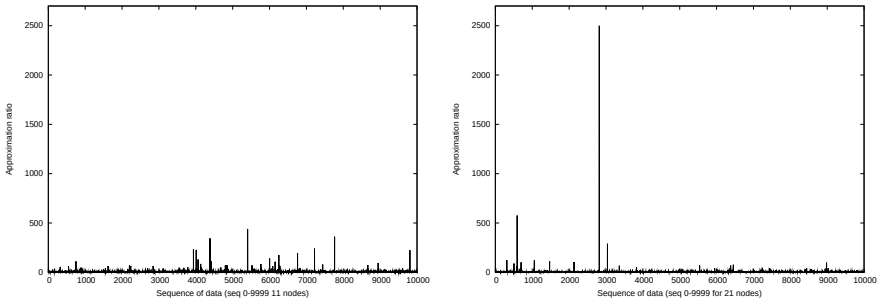
4.1 The Gale-Shapley Algorithm

In the previous section, we showed several examples where the approximation ratio for stable marriage depends upon the choice of weights. In this section, we consider how the approximation ratio for stable marriage changes as we change the statistical distribution of the weights in the original bipartite matching problem. For some distributions, stable marriage yields low approximation ratios that are nearly optimal. For others, high approximation ratios are observed. We tested the stable marriage transition planning scheme in simulation and compared the resulting matching to that obtained from bipartite matching. We used different probability distributions to determine the weights of the transition edges. The results for the uniform, normal, and Pareto distributions are shown in Figure 6. We tried two uniform distributions. The first setting (uniform distribution 1) has a mean of 1000 in range (0,2000], and the second setting (uniform distribution 2) has a mean of 10000 in range (0,20000]. These yield almost identical results.

Next we simulated approximately normally distributed weights with mean 10000 and 3 different standard deviation parameters. The first setting (normal distribution 1) has a standard deviation of 2000; the second setting (normal distribution 2) has a standard deviation of 5000; the third setting (normal distribution 3) has a standard deviation of 10000. In the experiments, when a negative random number is generated following these settings, we discard it and choose another with the same distribution function. So the actual distribution

function slightly tilts toward positive numbers. The coefficient of variation (the ratio of the standard deviation to the mean) is an important factor. When the coefficient of variation is small (for example, 0.2 in normal distribution 1), the weights are relatively clustered and the performance is near-optimal. When the coefficient of variation increases, the approximation ratio moves towards that of the uniform distribution.

We also tried the Pareto distribution. The Pareto distribution function is $P(x) = ab^a/x^{a+1}$ [7]. We use several cases: distributions with $a = 1$ $b = 0.6$ (Pareto distribution 1), $a = 2$ $b = 0.6$ (Pareto distribution 2), and $a = 3$ $b = 0.6$ (Pareto distribution 3). The larger the value of a, the shorter the tail is in the PDF function. In the case of Pareto distributions 2 and 3, the efficiency is near optimal, but in the first case (Pareto distribution 1), the approximation ratio is undesirable when the number of nodes is small.



(a) 11 nodes (AVG=2.21) (b) 22 nodes (AVG=1.97)

Fig. 7. 10000 data points collected from simulation for approximation ratio.

For the above scenarios, the approximation ratio is good in most case, but some cases have a high approximation ratio. Because of the long tail of the Pareto distribution (especially with $a = 1$, where the probability distribution function curve is relatively flat), a weight can be far away from other weights, and as we have shown, this may results in a large ratio. In our experiment, because there is no upper limit for the weights in the Pareto distribution, one case can generate a large approximation ratio. Figure 7 shows the raw simulation data. While most of the points in Figure 7(a) have low approximation ratios, there are many with ratios that reach into the hundreds. By contrast, Figure 7(b) shows much fewer ratios in the hundreds, but also uncovers a ratio high in the thousands. For 21 nodes, there are less points with approximation ratios in the hundreds but there is one data point whose approximation ratio is in the thousands. The figure demonstrates that the long-tail distribution leads to poor approximation ratios.

4.2 The Gale-Shapley Algorithm Plus a Greedy Heuristic

To address the issue that occurs in the worst-case scenario, we propose a simple heuristic that could improve the efficiency of the stable marriage approximation. After running the Gale-Shapley algorithm once, the most expensive edge in the result is marked "infeasible", and the right side of that edge is forced to take its first preference as its match. Then the rest of the nodes use the Gale-Shapley algorithm to determine a stable marriage of the remaining bipartite graph. This process continues until the total cost can no longer be decreased. In practice, each agent will determine its own criteria for marking edges as expensive. The exclusion of edges will not be as close to optimal as that obtained in our simulation, which utilizes global knowledge. The simulation results can be viewed as the best-case result for this heuristic.

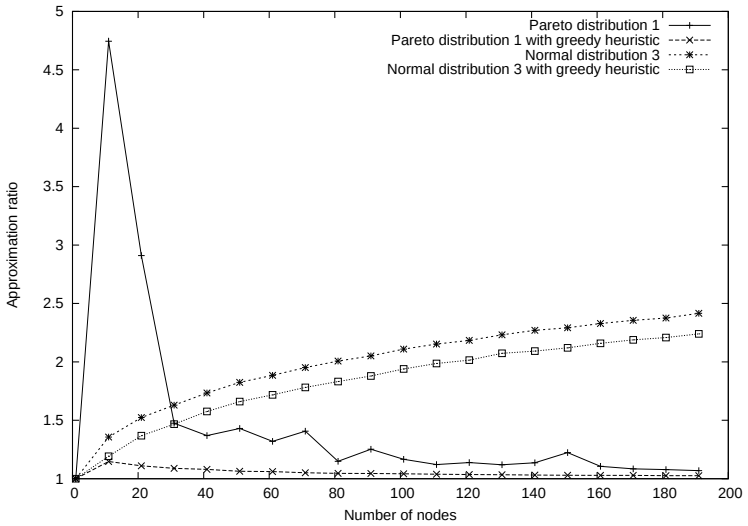


Fig. 8. The approximation ratio with and without using the greedy heuristic.

We selected two weight distributions to verify the effectiveness of this heuristic. First we use the long-tail Pareto distribution (Pareto distribution 1). Then we use the normal distribution with a large coefficient of variation (normal distribution 3). The greedy heuristic works very well for the Pareto distribution, but only slightly improves the result in the case of normal distribution. The result is shown in Figure 8.

4.3 Simulating Real Cases

In the first experimental simulations, the left and right sides of the graph share the same weight. In the next simulations, we try to evaluate the stable-marriage-based scheme under a more realistic environment. We now estimate their cost

based on different formulas and use a cost function to derive the cost incurred by the bipartite matching.

We estimate the cost of a transition in two parts: file transfer cost and usage time lost due to downtime during the transition. The first part can be estimated based on the size of files transferred, and the second part can be estimated based on CPU cycles. It is known that in practice, file sizes follow the Pareto distribution. CPU speed follows a Moore's law [5] growth pattern. We estimate the speed of computer servers through a 3-year cycle. One third of the CPUs are the latest fastest, one-third are middling, and the last third are the slowest. The total cost formula is as follows:

$$totalCost = fileSize * coefficient1 + CPUSpeed * coefficient2 \quad (1)$$

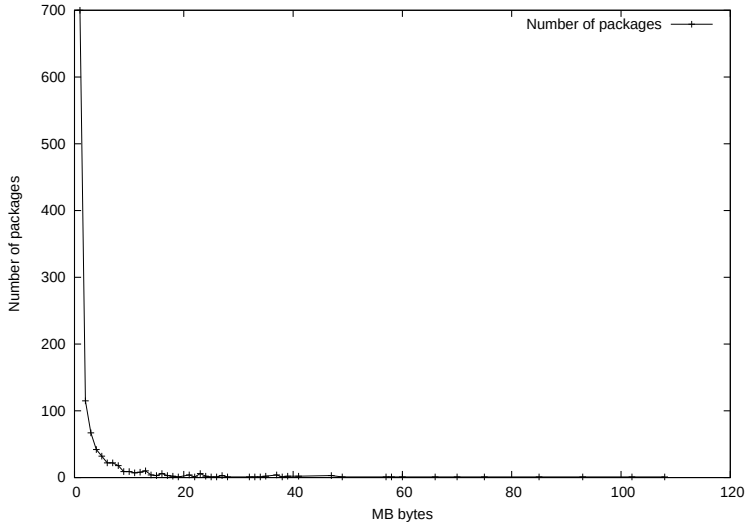


Fig. 9. The package size distribution on a typical Linux server.

We collect the RPM package size on a typical Linux server and show the raw sizes and their counts in Figure 9. The X axis shows the package size and the Y axis shows the number of packages of that size. The distribution is Pareto; we use another distribution function $0.6/x^2$ to fit the curve. The Pareto distribution function is $P(x) = ab^a/x^{a+1}$, where $a = 1$, $b = 0.6$. Figure 10 shows these two PDFs.

For CPU, we select three products: AMD Athlon 64 3800+ X2 (Dual Core) (14,564 MIPS), Intel Core 2 X6800 (27,079 MIPS), and Intel Core 2 Extreme QX6700(57,063 MIPS) [6]. To normalize file sizes and CPU speeds, we multiply file size by 10 and average with CPU MIPS. For example, a 3MB file and a Intel

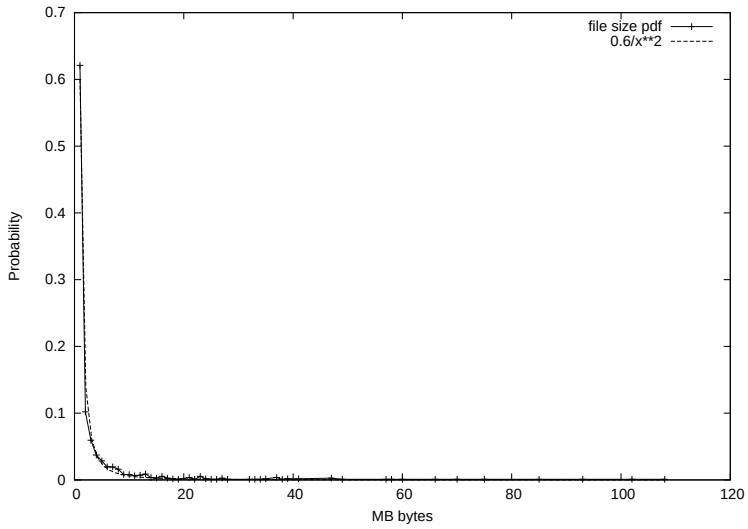


Fig. 10. The probability distribution function and the fitted curve of the package size distribution.

Core 2 CPU will give us (3000*10 + 27079)/2 = 28539.5. We use this model to explore the relationship between a cost function and individually decided rankings until we can find a more accurate cost model.

In this experiment, each side of the bipartite graph ranks the other side differently; the left side ranks using the file size, and the right side ranks using the CPU speed. The minimum cost transition uses weights computed from the cost function. The result is shown in Figure 11.

5 Conclusion and Future Work

In this paper, we investigate the statistical behaviors of transition planning mechanisms in a decentralized environment. The Gale-Shapley algorithm can be used to find transition plans. Theoretically, we can create an arbitrarily large approximation ratio by using the Gale-Shapley algorithm to simulate the optimal solution. But in practice, the worst case rarely arises and can be prevented via heuristics. We use simulation to study the impact of different probabilistic distributions. The result shows that the normal distribution has a very good performance when the coefficient of variation is low. When the coefficient of variation is high (for example, 1) the performance is similar to that of the uniform distribution. The uniform distribution has an average performance. But in a probability distribution that is flat and has a long tail, such as the Pareto, some settings (for example, $a = 1$) will produce a poor approximation ratio when the number of nodes is small. The performance improves dramatically when the number of

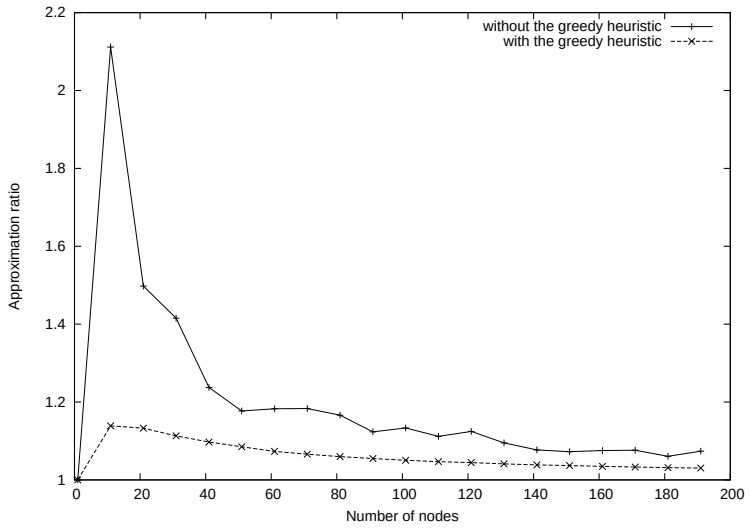


Fig. 11. Simulated approximation-ratio results when each side ranks according to a different cost standard, comparing that result to the minimum-cost transition based on a combined cost function.

nodes is increased. Thus, a minimum node size above 50 is recommended or the greedy heuristic should be used.

Although better stable marriage algorithms exist, which might address these issues, these algorithms take more time to settle than the Gale-Shapley algorithm. We propose a simple heuristic that greedily removes the most expensive edge from the mapping, and thus improves the stable marriage performance. In the future, heuristics can be investigated, that leave out some options in order to avoid obviously bad marriages.

6 Acknowledgement

We wish to thank Marc Chiarini for reading several versions of the paper and making timely comments.

References

1. Mark Burgess. An approach to understanding policy based on autonomy and voluntary cooperation. In *DSOM 2005*, pages 97–108, 2005.
2. Mark Burgess and Alva Couch. Autonomic computing approximated by fixed-point promises. In *1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2006)*, 2006.
3. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 1962.

4. Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 1955.
5. Gordon Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8), 1965.
6. Frank Voelkel. Intel's core 2 quadro kentsfield: Four cores on a rampage. `http://www.tomshardware.com/reviews/cores-rampage,1316-13.html`, 2006.
7. Eric W. Weisstein. "pareto distribution." from mathworld–a wolfram web resource.
8. Ning Wu and Alva Couch. Transition planning via matchings and marriages. In *The 2nd IEEE International Workshop on Modelling Autonomic Communications Environments (MACE 2007)*, 2007.