Online Stable Matching as a Means of Allocating Distributed Resources[1]

Hyunyoung Lee
Dept of Computer Science, Texas A&M University, College Station, TX 77843
E-mail: `hlee@cs.tamu.edu`

Abstract

In heterogeneous distributed systems, achieving optimality in both effective use of computational resources (e.g. throughput) and user satisfaction (e.g. response time) is an important unresolved problem. If the users of the system participate dynamically as consumers as well as donors of computational resources, the task of optimizing the exchange of these computational resources leads to a combinatorial problem. As a solution, we propose a novel algorithm, *Adaptive Online Stable Matching (AOSM)*. We present experimental data which compare the performance of AOSM with the performance of two alternative algorithms, First-come First-served (FCFS) and Fixed-k online.

Keywords: stable matching, distributed resources, online algorithm, approximation, adaptive windowing.

1. Introduction

The concept of *barter marketing* [5] has been proposed as a means of maximizing the utilization of the computational resources of heterogeneous distributed systems with varying computational power and demands (e.g. the Internet). In system architectures of this kind, the group of processors and processes dynamically forms two logical sets: donors and consumers. Each donor makes a certain amount of computational power available for a guest process, thereby accumulating credit. Each consumer wants to acquire a certain amount of computational power to run its process, reducing its credit. And as a whole, the system will always have zero credit [5]. From a global point of view, the purpose of such a mechanism is to maximize system utilization. From the point of view of the individual user, the system allows access to vast computational resources. If the user requires these resources only for short amounts of time, 'renting' them from the system is far more economical for the user than buying the corresponding hardware.

This paper describes a resource allocation algorithm which, in such a system environment, yields a desirable matching between donors and consumers. We assume that the two sets of donors and consumers are dynamically changing in time. Thus, the algorithm should run in real time. We design an online matching algorithm, which is based on the stable marriage matching algorithm [3] and its online version [7]. The notions of *approximated stability* and *degree of satisfaction* are defined and are used as measures of the performance of the algorithm. The algorithm computes a matching based on the events which happen within a certain *time window*. The window size is determined dynamically based on estimates of approximated stability and degree of satisfaction. These dynamic decisions are based on a prediction for the near future, which, in turn, is based on the performance of the system in the past and current information about donors and consumers.

[1] Journal of Systems Architecture. Vol. 45, pages 1345–1355. 1999

The contribution of this paper is twofold. Firstly, we develop a cooperative distributed system paradigm and design the system model. We apply a problem from traditional combinatorial matching theory to the resource allocation problem of our system. Secondly, we propose an online algorithm called *Adaptive Online Stable Matching (AOSM)* and analyze its behavior in terms of stability and degree of satisfaction.

The rest of this paper is organized as follows. Section 2 introduces notions and definitions regarding stability and our approach to windowing. The AOSM algorithm is introduced and analyzed in Section 3, and experimental evidence that AOSM performs better than FCFS and fixed-k online, is shown in Section 4. We conclude with a discussion of further research in Section 5.

1.1. Problem Definition

We consider a heterogeneous distributed system, for example, a group of processors and their users on the Internet. Processors form dynamically a set $\mathcal{D}$ of donors. Jobs arrive online, dynamically forming a set $\mathcal{C}$ of consumers. Thus, the entire set of users $\mathcal{U}$ of the system consists of the two subsets $\mathcal{D}$ and $\mathcal{C}$, such that $\mathcal{U} \supseteq \mathcal{D} \cup \mathcal{C}$ and $\mathcal{D} \cap \mathcal{C} = \{\}$ at any given point in time. Each donor and each consumer is characterized by a list of preferences. For example, a processor may prefer longer-executing jobs because this will allow the processor to accumulate more credit. Jobs may prefer faster processors etc.

We are seeking an online algorithm which dynamically decides on a suitable point in time when the matching between donors and consumers is to become effective, that is, when jobs go for execution. We assume that processors are always able to accept jobs for execution. If a job is allocated to a processor in a matching, this means that the job is ready for execution on the processor right away. In this context, the criterion we are aiming at is to produce a stable matching, trying to maximize the system throughput as well as to minimize the waiting time of the users. Since the system is online, without any a priori information about $\mathcal{D}$ and $\mathcal{C}$, we cannot guarantee the same optimality as in an offline environment where all the information for both sets is known in advance. Instead, we pursue near-optimality with the help of the knowledge of previous inputs and the resulting matchings, as well as current input. Furthermore, based on the previous performance of the algorithm, we try to predict the behavior of the system in the near future in order to improve performance.

1.2. The Matching Approach

An important part of the system is the protocol which allocates processors to processes. This is different from just resource allocation. It has to ensure fair sharing of the resources and availability of the system even in peak times. In this work, we consider the combinatorial matching problem as a solution candidate.

A matching in a graph is a set of edges such that the degrees of all vertices are at most 1. We will consider only bipartite matchings, i.e. matchings in bipartite graphs. The sets $\mathcal{C}$ and $\mathcal{D}$ are the two vertex sets of the bipartite graph. There is an edge $\{c, d\}$ whenever a consumer $c \in \{\mathcal{C}\}$ could use the resources of a donor $d \in \{\mathcal{D}\}$.

There is a large body of theoretical results concerning the stable marriage matching problem. The problem was originally considered by Gale and Shapley in 1962, who established the concept of *stability*. The book by Gusfield and Irving [4] addresses related theories and analyses. In the context of the dynamic system model where participants are continuously arriving and/or leaving the system, and both parties are dynamically changing in time, Khuller et al. [7] consider an online version of the stable matching algorithm. Their algorithm is first-come-first-served (FCFS), in which all preferences of a party are initially known. When a member of the other party arrives, it is matched with its most preferred available partner.

We relax the restriction of matching a request *immediately* upon arrival. Thus, we obtain the flexibility to accumulate requests for a certain period of time, which may allow us to produce a better matching.

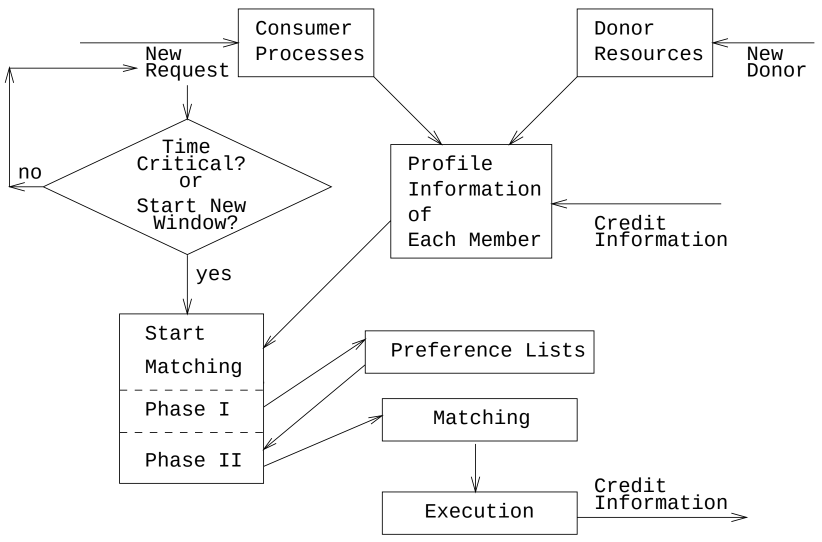

Figure 1: Software architecture

The time period constitutes our *window*. The window size is limited by the amount of time users can be expected to wait before they become dissatisfied with the system. In this setting, we aim to find the most profitable window size with respect to throughput and response time, rather than simply trying to maximize credit transfer. This approach yields better insight into the behavior of the system and results in a clearer way of analyzing the performance of the system.

The classical criterion in analyzing online algorithms is ‘competitiveness’, i.e. the *worst-case* ratio between the performance of the on-line algorithm and that of the optimal offline algorithm with respect to some performance measure. In this work, however, we show the superiority of our algorithm by other means, and consider additional criteria such as the overall stability of the algorithm.

2. The Conceptual Model

2.1. System Parameters

The software architecture of the proposed system is shown in Figure 1. Each member of the two sets of donors (resources) and consumers (processes) is specified by the following parameters, whose values may be provided by the member at the time of submission or by default: Each member provides its own information about preferences. Based on this information, Phase I in Figure 1, constructs sorted specific preference lists with currently available members in each set. Phase II uses those sorted preference lists to produce a matching. The matching algorithm is given in Section 3.2.2.

Each consumer process provides an (optional) deadline (delay bound) for execution, and an estimate of the required computational resources as a function of processor speed and execution time. Each donor specifies the amount of computational power as a function of processor speed and available service time. The computation of credit for time consumption is normalized to accrue weight fairly proportional to the processor speed. Credit information for both consumers and donors is provided by the name server. It plays the role of virtual money which consumers spend when they use other machines and which donors earn by ‘leasing’ their own machines to other processes. Accumulated credit is computed by computational power multiplied by the actual execution time.

2.2. Approximated Stability and Degree of Satisfaction

In the original stable marriage problem which was first formally discussed by Gale and Shapley, ‘stability’ is defined in terms of *stable* matchings. A matching is called *unstable* if there are two parties who are not matched with each other, each of which strictly prefers the other to his/her partner in the matching [3]. A stable matching is a matching that is not unstable. Following this definition of stability, Khuller et al. [7] count the expected number of instabilities to show the competitiveness of their online algorithm with respect to the optimal offline algorithm.

In our problem context, the ‘stability’ of a matching, produced solely based on each party’s preference, yields neither a sufficient nor a necessary condition for an optimal transfer rate of the overall system. The credit information does not appear in the preference list. However, it is an important factor in making a decision in producing a matching. Thus, the credit information is looked up by the matching protocol whenever competition occurs in any stage of the matching. Therefore, in the system environment discussed in this paper, a matching is produced based on both preferences and credit information.

We follow the original stable matching algorithm. The smaller of the two sets $\mathcal{C}$ and $\mathcal{D}$ will *propose* matchings to the larger set. (If $|\mathcal{C}| = |\mathcal{D}|$ either $\mathcal{C}$ or $\mathcal{D}$ may propose.) We modify the algorithm as follows: Whenever in the original algorithm two members A, B of the proposing set target the same element c of the other set, c is matched with the element (A or B) which ranks highest in c’s preference list. We substitute this decision by the following rule which combines preferences and credit information. For each of the two proposers, we multiply its credit with its rank in the preference list of c. We match c with the proposer whose value is higher.

Example 1 Let $|\mathcal{C}| = |\mathcal{D}| = 4$. The preference list of each donor and request and the credits of each donor are shown in Table 1. The stages of the execution of the matching algorithm are shown in Figure 2. In stage 1, resources 1 and 3 compete for request 1, and resources 2 and 4 compete for request 2. The preference list of request 1 shows that it prefers resource 3 to resource 1, and the credit information shows that resource 3 has higher credit than resource 1. Thus, request 1 is allocated to resource 3. The preference list of request 2 gives rank 4 to resource 4 and only rank 2 to resource 2. However, the credit of resource 2 is 5 times higher than the credit of resource 4. Thus, request 2 is allocated to resource 2.

In stage 2, request 2 is proposed by resource 1, but rejects since resource 1 has lower credit and preference values than its current match (resource 2). Request 3 is proposed by and allocated to resource 4. In stage 3, resource 1 competes with resource 4. Resource 1 has higher credit (2.5 times), but the rank of resource 4 in the preference list of request 3 is four times higher than the rank of resource 1. Thus, request 3 remains allocated to resource 4. In stage 4, resource 1 proposes to request 4 and is accepted without competition.

Table 1: Credits and preferences

	Donor (resources)		Consumer (requests)
	credits	preferences	preferences
1	50	1 2 3 4	4 3 2 1
2	100	2 1 3 4	4 3 2 1
3	100	1 2 3 4	4 3 2 1
4	20	2 3 4 1	4 3 2 1

The matching protocol, by knowing not only the preferences, but also the credit information of each donor and consumer, produces a matching which is considered to be the most appropriate. This is, even

Stage	Proposals
1	$1 \to 1$ $\{2 \to 2\}$ $\{3 \to 1\}$ $4 \to 2$
2	$1 \to 2$ $\{4 \to 3\}$
3	$1 \to 3$
4	$\{1 \to 4\}$

Figure 2: Stages of matching; enclosed in braces are accepted proposals.

though there exist pairs such that two members of one party strictly prefer the other's partner to their own partner in the matching, the matching will be superior, as it takes credit information into account. The influence of credit information may cause members to be paired with less desired partners than would be the case without credit. We capture the degradation due to the credit information in the following definition:

Definition 1 An $\alpha\%$ *approximated stable matching* is a matching based on not only the preference lists but also credit information as described by the rule above, in which each element c is matched with a partner whose rank in c's original preference list is in top $\alpha\%$ of the preference list.

It is observed by the experimental measurements, that with increased window size, the chance that the system produces a matching with higher stability also increases. In terms of the satisfiability of the users of the system, increased stability indicates better balanced credit within tolerable delay.

Now, using the concept of approximated stability, *matching throughput for credit* and *degree of satisfaction* are defined. These concepts will be used to measure the performance of the matching algorithm.

Definition 2 *Matching throughput for credit (MT)* is the total credit transfer rate of all the donors participating in the matching such that $\mathcal{MT} = \sum_{i=1}^{k} \mathcal{C}(d_i)$ where $\{d_1, \ldots, d_k\}$ are the donors in the matching M, and $\mathcal{C}(d_i)$ is credit transfer rate of donor d_i in M. The credit transfer rate is computed as the amount of computing power multiplied by the time consumed by the consumer process c_i.

Definition 3 f_{DoS} is a function of $\mathcal{MT}$ and $\mathcal{R}$, to compute the *degree of satisfaction (DoS)* of the members in the matching such that $f_{\text{DoS}}(\mathcal{MT}, \mathcal{R}) = \frac{\mathcal{MT}}{(\alpha - \frac{1}{m})\mathcal{R}}$ where $\mathcal{MT}$ is the matching throughput, $\mathcal{R}$ is the sum of the waiting times of all consumers in M, α is the approximation factor, and m is the size of preference list. $f_{\text{DoS}}(\mathcal{MT}, \mathcal{R}) = \mathcal{MT}$ if $\alpha - \frac{1}{m}$ or $\mathcal{R}$ (or both) is zero.

We will analyze the fluctuations of f_{DoS} over time. This is important in the sense that when the system produces stable matchings within a certain approximation and in a steady manner, our prediction for future matchings will be more reliable than when the system is in a fluctuating state. The 'steady state of stability' will be explained in the following section.

2.3. Adapting the Window Size and Steady State of Stability

Unlike the first-come-first-served algorithm of Khuller et al. [7], our algorithm permits a delay before producing a matching, dynamically setting the size of window in which a matching is produced. The decision of setting a window is based on prediction which depends on previous inputs and the corresponding matching. After each matching is produced, the overall system performance is analyzed and will be used as a prediction metric for the next matching. We use two schemes to set the window size adaptively:

Predicted windowing determines the size of next window by prediction without any knowledge about incoming input. *Lazy windowing* postpones determining the size of window until some of the incoming inputs are known, but without delaying them beyond their deadlines. That is, by lazy windowing, closing the window is put off as late as possible without any harm to a process's execution. These two windowing schemes are to be used in our algorithm in a combined manner. During the initial stages, lazy windowing will be mainly used. As the system approaches a predictable and steady state, predicted windowing will allow immediate acceptance of matchings and so will render reliable results, as well as more efficient ones. Lazy windowing, however, enables deferred acceptance of matching and so yields more stable matchings.

The system is said to be in a *steady state of stability* when the variation of the measurements of f_{DoS} becomes negligible with time flow. When the system is in a steady state of stability, its performance can be correctly predicted with high accuracy, such that the algorithm can rely on predicted windowing. A steady state may be in either positive or negative growth.

3. The Adaptive Online Stable Matching Algorithm (AOSM)

3.1. Basic Policies

Firstly, only one process is scheduled to run on one processor in a matching. Because it is not guaranteed that the execution of a process will be finished in the time predicted by its submitter, the process may take more than its predicted time and, thus, there may not be enough time to execute other already-assigned processes. Therefore, allocating one processor to more than one process in a matching is avoided, as it may lead to unpredictable results.

Secondly, the matching algorithm is invoked based upon two conditions: (1) When the time interval between two arrivals of consumer processes is smaller than or equal to the time period of the window, the invocation is driven by the latter. (2) If no new processes arrive during the entire window period, the algorithm should be set to its initial status and wait for new arrivals of consumer processes.

Third, the algorithm is always donor-optimal and consumer-pessimal, in such a way that any donor and consumer can become the proposer, however, the policy for choosing the partner differs. When the donor is proposing, it proposes to the most preferred process in its preference list. When the consumer is proposing, it proposes to the worst possible processor in its preference list. It can be proved that in both cases the resulted matching is a stable matching.

Fourth, when a processor finishes the execution of the matched process in any previous matching, the processor becomes a new member of the donor set, which is different from the static input status of the offline algorithm. Therefore, even with as appropriate as possible matchings overall, the matchings produced cannot be considered to be the same for both offline and online algorithms.

3.2. Algorithms

The algorithms consist of two parts. One is to decide the size of window of each matching adaptively, using prediction or postponing the decision until some degree of convincing information is available. The other part performs the matching within the window.

3.2.1. Adaptive Online Windowing

The algorithm takes three factors into account when adaptively determining the window size: (1) Degree of satisfaction (DoS) of matchings, which is computed by use of f_{DoS}. When the relative values of DoS of consecutive matchings are getting better with increased window size, the algorithm increases the window size. (2) Delay-bound of matching. When a matching is produced, it may happen that not all requests are allocated to resources, which may cause unbounded delay until such requests are matched. To avoid such starving situations, the system defines a maximum possible delay-bound. Also, each request can have its own delay-bound (deadline). If a consumer doesn't specify a deadline, the delay-bound is given by the system. However, in principle, starvation cannot be avoided altogether if no donor becomes available for a very long time. (3) Estimate that lazy windowing will produce a better matching. Initially, this estimate is 0.5. The algorithm keeps track of three matchings, M_0, M_1, M_2. M_0 is the matching produced in the previous window W_0. M_1 is the matching produced in current window W_1. M_2 is the matching produced from the union of W_0 and W_1. In other words, given three points in time $t_0 < t_1 < t_2$, M_0 is produced with the members arrived between t_0 and t_1, M_1 between t_1 and t_2, and M_2 between t_0 and t_2. With the computed values of DoS of these matchings, the algorithm examines if M_2, which is produced by lazy windowing, yields a better result than the combined result of M_0 and M_1, which are produced separately by predicted windowing. If so, the estimate is increased by a factor of two, and otherwise, decreased by a factor of two.

1. Prelude: Initialize Max_delay which is maximum time a consumer should ever wait.

2. Initial Step: As soon as a donor d_i becomes available, and if there is any process submitted by a consumer, match this process c_j to its worst possible partner d_{c_j}. Let this base matching be M_0 and the time when M_0 is produced be t_0. Measure the time between the arrivals of the first and the second job submitted and use this time length as the initial window size W_0. Set $W_1 := W_0$. Set Max_delay as the deadline of c_j if c_j specifies any deadline, and set $P := \frac{1}{2}$.

3. While waiting for next arrivals of processes, if $accumulated\_delay$ reaches Max_delay, then let M_0 go for execution. Go to the initial step. Otherwise, continue at step 4 with the newly arrived processes.

4. (a) $t_1 \leftarrow t_0 + W_1$;

   (b)
```
If ($(accumulated\_delay + W_1) > Max\_delay$)
then  if $P < \frac{1}{2}$
      then  1. go for execution with $M_0$ right away
            2. compute new $M_0$ with the jobs that have arrived between $t_0$ and $t_1$
            3. $t_0 \leftarrow t_1$; $t_1 \leftarrow t_0 + W_1$; go to the beginning of step 4(b)
      else  1. wait until $Max\_delay$
            2. compute $M_2$ with $M_0$ and the jobs which have arrived since $t_0$
            3. let $t_2$ be the time when $M_2$ is produced; go for execution with $M_2$
            4. $t_0 \leftarrow t_2 + W_1$; compute $M_0$ with the jobs which arrived between $t_2$ and $t_0$
            5. $t_1 \leftarrow t_0 + W_1$; go to the beginning of step 4(b)
```

   (c) Compute M_1 only with the jobs which have arrived between t_0 and t_1. Compute M_2 including members in M_0. Compute degree of satisfaction[2] s_0, s_1, and s_2 for M_0, M_1, and M_2, respectively.

   (d) Compute the new predicted window size W_1 by use of $\frac{s_2}{s_0}$ as a parameter. $\frac{s_2}{s_0}$ provides a reasonable indication of how better (or how worse) is the matching that the larger window produced compared to the matching of the smaller window.

   $W_0 \leftarrow W_1; W_1 \leftarrow W_1 \times \frac{s_2}{s_0}$;

[2]computed by use of f_{DoS} function

(e) Based on the degree of satisfaction

- Case 1. If $\frac{s_2}{s_0} > 1$: M_2 is a better matching than M_0, that is, increasing the window size is profitable. Thus, let M_2 be the base matching of next phase and increase the estimate that lazy windowing will produce a better matching.

  $M_0 \leftarrow M_2$; $P \leftarrow 2P$; $t_0 \leftarrow t_1$; Go to the beginning of step 3.
- Case 2. If $\frac{s_2}{s_0} = 1$: M_0 and M_2 have the same degree of satisfaction, i.e. lazy windowing did not do better, because the profit in throughput achieved by increased size of window, is offset by latency in response time. M_0 is obsolete due to the delay. Thus, use M_2 as the base matching of next phase, but keep the estimate unchanged.

  $M_0 \leftarrow M_2$; $t_0 \leftarrow t_1$; Go to the beginning of step 3.
- Case 3. If $\frac{s_2}{s_0} < 1$ and $\frac{s_2}{s_1} < 1$: M_0 has better degree of satisfaction than M_2. Since lazy windowing has produced a poorer matching, decrease the estimate. M_1 is also better than M_2, thus let M_1 be the base matching of next phase.
  i. Let M_0 go for execution.
  ii. $M_0 \leftarrow M_1$; $P \leftarrow \frac{1}{2}P$; $t_0 \leftarrow t_1$; Go to the beginning of step 3.
- Case 4. If $\frac{s_2}{s_0} < 1$ and $\frac{s_2}{s_1} \geq 1$: M_0 is better matching than M_2, and so is M_2 in regards of M_1. M_2 will yield the best result, however, the estimate is not changed. Compute new M_0.
  i. Let M_2 go for execution.
  ii. Compute M_0 with the jobs which have arrived between t_1 and t_0 where $t_0 = t_1 + W_0$. Go to the beginning of step 3.

3.2.2. Matching

Each of donors and consumers which arrives comes with a sorted preference list which is input to the matching procedure. The matching procedure functions as a gateway to and from the name server to deal with credit information, and is to produce a stable matching[3] to have the processes scheduled to run. After the matching is done, the credit of the members involved in this window should be updated properly through the name server. This matching procedure requires O(n^2) in time, where n is the size of proposing set.

```
/* compute a stable matching, and following the matching, update credit history */
assign each donor and consumer process to be free;
assign Donor set or Consumer set as proposer, which is smaller in size;
while some proposer p is free do
        q := first member on p's preference list to whom p has not yet proposed
```

[3] The algorithm is based on Gale-Shapley's stable marriage matching algorithm; the changes are to handle different sizes of the two input sets and credit information.

```
          if q is free
          then assign p and q to be paired /* to each other */
          else if p wins over q's partner p' following the rule in section 2.2
                  then assign p and q to be paired and p' to be free
                  else q rejects p /* and p remains free */
output the stable matching consisting of the n pairs;
update the credit history of each member involved in the matching;
```

3.3. Stability and Bounded Delays

Steady State of Stability: We aim for a "steady state" of stability which shows the long term proportion of stable matchings during a certain period T. Since the arrivals of donors and consumers and their inputs are assumed to be predictable in some degree, it is expected that the steady state of stability will eventually be reached.

The variable P in the algorithm in section 3.2.1 contributes toward reducing fluctuations in degree of satisfaction: P is modified depending on whether the matching yields higher degree of satisfaction or not and whether the already-assigned processes have started their execution or not. These categories are shown in the cases of step 4(e) of the algorithm.

Guaranteed delay bound: Max_delay is guaranteed for each process $p_i \in M_i$ for all i within T, i.e., no process should be revoked or starved, once it is submitted and as far as there are available processors, and further, once it is scheduled to run by any matching. This can be proved by the fact that the execution of the process will not be indefinitely put off, with the help of the maximum permissible delay bound (Max_delay) which is known to every consumer upon submission and checked at step 3 and 4 in every iteration of the algorithm.

4. Experimental Measurements

In this section, we will perform experiments in order to compare the performance of AOSM with that of two other matching algorithms: FCFS and fixed-k online algorithm where k is initially given and fixed until it is changed explicitly through an external mechanism. The latter two algorithms can be considered as special cases of AOSM. If the window size of AOSM is kept constant then AOSM is just fixed-k online. If the window size is set such that each window contains only one arrival event, thus matching any donor or consumer as soon as it arrives, we obtain FCFS.

We have run the three algorithms on random input data. The inter-arrival times of donors and consumers were generated uniformly at random from the set $\{1, \ldots, 10\}$ seconds. The preference lists, credit and deadline information, processor speed, and execution time of the arriving processes were also generated at random. (Details omitted due to space restrictions.)

We measured matching throughput and cumulative response time. The values of f_{DoS} were computed following the formula given above. We simulated the system over a total of 1000 seconds, gathering data every 100 seconds. This procedure was run 100 times and the results were computed by averaging them.

We have used two measures to compare the three algorithms. The first measure is matching throughput $\mathcal{MT}$. As shown in Figure 3, overall, AOSM produced a higher $\mathcal{MT}$ value than FCFS and Fixed-k online. The second measure is the delay incurred by the algorithm. FCFS, in theory, incurs no delay since the request is matched upon its arrival. However, in practice, in case that there is no available resource for newly arrived request, FCFS also suffers from delay, waiting for new resources to become available. For all three algorithms, we assume that an indefinite waiting situation should not happen, and that some partner will always become available after a tolerable delay. Therefore, consumers always become proposers.

Depending on the distribution of the inter-arrival times, statements of this kind can be shown to hold with high probability. Fixed-k online yields $\mathcal{T}(k)$ seconds of delay on average, where $\mathcal{T}$ is the function to compute the average delay for all pairs in a matching. Assuming periodic arrivals of requests, this function should be proportional to the window size. Let $\mathcal{R}(W_j)$ denote the total delay under AOSM for consumer processes within the window W_j, until they are matched.

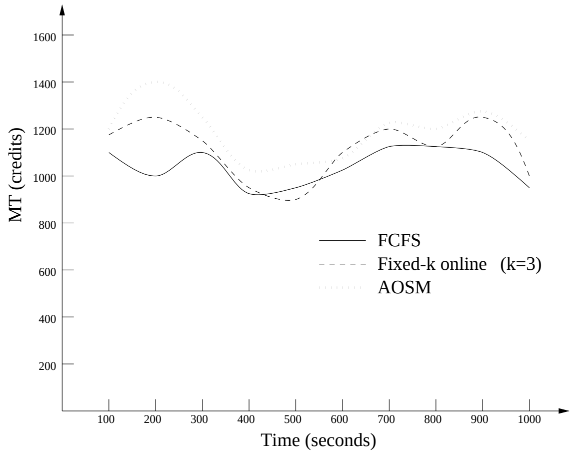

Figure 3: Measurement of matching throughput

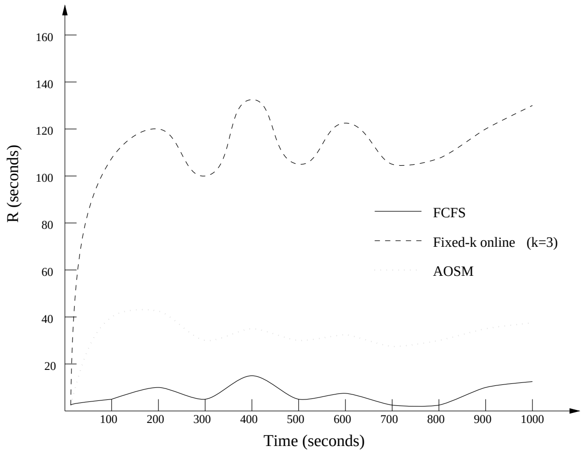

Figure 4: Measurement of response time

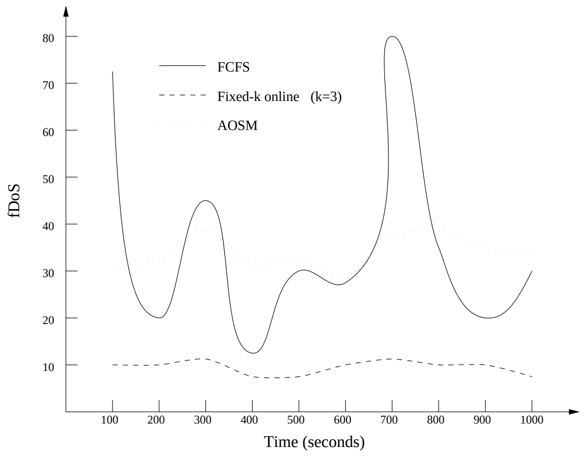

Figure 5: Measurement of degree of satisfaction

The degree of satisfaction can be computed from $\mathcal{MT}$ and $\mathcal{R}$ by use of f_{DoS}. Figure 5 shows that FCFS produces fluctuating results of matchings. Fixed-k online algorithm produces steady results but suffers from large delays. AOSM results in a steady state of stability with small variation and provides predictability for the system.

5. Conclusion and Future Work

We have presented an analysis of online matching algorithms including FCFS, fixed-k online, and AOSM, with respect to stability and degree of satisfaction. We have shown experimental measurements which provide evidence that AOSM performs in a more steady manner, producing better matchings in terms of stability and reliable performance of the system. Achieving a steady state of stability is important in that when a group of users wants to cooperate in sharing computational resources, fairness and availability are the main concerns. Users can easily determine their trust in a distributed system, using the measurements introduced and the metrics adopted in this paper.

The problem of DoS can be transformed to the problem of Quality of Service (QoS) in a communication network such as ATM, where we consider communication channels on the network links as distributed resources. With that, an applicable system model on communication networks can be established. Another issue is the construction of a real-world system model which will produce more robust and realistic execution results of the algorithms, which can therefore provide solutions for policies for some yet unresolved problems, such as cheating about the computational power, demand exceeding supply, etc.

Acknowledgments

This work was performed while the author was at Boston University. The author would like to thank Abdelsalam Heddaya who provided motivation of and important guidance for the work overall, and Steve Homer who gave a direction to online stable matching algorithms. The author also wishes to thank Jennifer Welch and the anonymous referees for their helpful comments.

References

[1] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *25th ACM STOC*, pages 623–631, 1993.

[2] E. Bernstein and S. Rajagopalan. The roommates problem: On-line matching and general graphs. Technical Report UCB//CSD-93-757, Computer Sci. Division, Univ. California, Berkeley, CA, 1993.

[3] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 69:9–15, 1962.

[4] D. Gusfield and R. W. Irving. *The stable marriage problem: structure and algorithms*. MIT Press, Cambridge, Mass, 1989.

[5] A. Heddaya. On the exchange of computational value. Notes for Distributed Systems Seminar, Dept. Computer Sci., Boston Univ., Fall 1996. Unpublished.

[6] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.

[7] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127:255–267, 1994.

[8] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.