Auction Based Models for Ticket Allocation Problem in IT Service Delivery Industry

Prasad M Deshpande
IBM India Research Lab
Bangalore, India
prasdesh@in.ibm.com

Dinesh Garg
IBM India Research Lab
Bangalore, India
dingarg2@in.ibm.com

N Rama Suri
Indian Institute of Science
Bangalore, India
nrsuri@csa.iisc.ernet.in

Abstract

In this paper, we study the service request (ticket) allocation problem which arises in every IT service delivery organization. We refer to this problem as Ticket Allocation Problem *(TAP). We first show that TAP is an instance of the online scheduling problem on unrelated machines, which is known to be a hard problem. Next, we describe a baseline model, namely push model, that deals with the TAP. The push model is an industry wide standard and can be used with any known online scheduling algorithm for unrelated machines. To elaborate this further, we discuss a well known Generalized List Scheduling algorithm which can be used by the push model. We prove a bound for this algorithm's competitive ratio which beats all the known bounds. We show that push model suffers from an inherent inefficiency due to scheduler having incomplete and imprecise information regarding agents' proficiency. Finally, we show that if the scheduling algorithm used by the push model can be converted into a truthful auction mechanism then all the inefficiencies of the push model can be overcome. We refer to the resulting model as the pull model. To illustrate the idea, we map the Generalized List Scheduling algorithm into a truthful auction mechanism. Through simulation experiments, we show that the auction based pull model results in higher efficiency than the push model.*

1. Introduction

In this paper, we address the *Ticket Allocation Problem (TAP)*. The TAP is one of the most commonly faced problem as well as the most significant problem for any IT service delivery organization since it deals with an efficient utilization of human resources such as technical employees. We review state-of-the-art industry practice for tackling TAP and subsequently, we present auction based models for solving this problem. We show that these new models improve the efficiency and effectiveness of the human resource utilization.

The rest of the paper is organized as follows. In Section 2, we present the review of relevant work. We define the Ticket Allocation Problem (TAP) and show that the TAP is an instance of the online scheduling problem in Section 3. In Section 4, we describe the *push model* and describe the *Generalized List Scheduling algorithm* which can be used by the scheduler in the push model. In Section 5, we explain how inefficiencies of the push model can be overcome if one can convert the scheduling algorithm used by the push model into a truthful auction mechanism. In Section 6, we experimentally verify that the the pull model achieves better efficiency in allocation and higher employee satisfaction. Section 7 concludes the paper.

2 Related Work

In this paper we study the ticket allocation problem (which is an instance of the online scheduling on unrelated machines) and propose an auction based approach for dealing with this problem. Therefore, our research in this paper cuts across two areas - online scheduling algorithms and auction theory. Both of these areas are quite mature and well studied.

The survey of scheduling algorithms and taxonomy of scheduling problems can be found in [10, 17]. In particular, we have referred to Graham [11] and Susanne [1] for the algorithms of online scheduling on identical machines, Naveen Garg et al [8] for algorithms of online scheduling on related machines, and Nir Avrahami et al [5] for algorithms of online scheduling in order to minimize the total flow time instead of makespan. The problem of online scheduling on unrelated machines is the most general version of scheduling and is thus a difficult problem to solve. Aspnes et al [3] and Leonardi et al [14] have shown that it is possible to obtain $O(log\ m)$ competitive deterministic algorithm for the case of unrelated machines. Aspnes et al [3] analyzes the

978-0-7695-3283-7/08 $25.00 © 2008 IEEE
DOI 10.1109/SCC.2008.83

List algorithm for unrelated machines and shows that the competitive ratio is exactly n.

In auction theory, several papers have dealt with the problem of designing mechanisms for job scheduling problems in the presence of selfish machines and/or selfish jobs. One early piece of work is due to Nisan and Ronen [16] where they have analyzed one version of the scheduling problem on unrelated machines and have proposed a *Min-Work* mechanism for that. More recently, Archer et al [2] have proposed truthful mechanisms for several combinatorial problems including scheduling on related machines. Auletta et al [4] focus on general techniques to translate approximation/competitive algorithms into equivalent approximation/competitive truthful mechanisms and apply these to scheduling on related machines. The other strand of the work which is related to ours is regarding incentive compatibility properties of the auction mechanisms. The seminal work in that direction is due to Vickrey [18], Clarke [6], and Groves [12].

3 Ticket Allocation Problem (TAP)

In a typical IT service delivery organization, a significant fraction of the technical staff is engaged in operational services such as: IT infrastructure management services, contact centers, remote server support management, etc. The employees engaged in offering these services are popularly known as *agents*. A typical way in which these services are offered is as follows. An end user raises a request for the service which results in one discrete unit of the job for an agent, for example, create a new user account, create a disk partition, diagnose a network related problem, optimize a database for a specific query workload, etc. Such a service request is captured in the form of what is known as *ticket*. These tickets arrive at significantly high rate and each arriving ticket needs to be allocated to an agent in an online fashion. This problem of allocating the tickets to the agents in an online fashion is knows as *Ticket Allocation Problem (TAP)*. TAP is a non trivial problem due to several reasons as mentioned below.

The tickets vary widely in terms of their complexity and the time it takes to complete them and moreover, each ticket requires a different set of skills to be handled. On the other hand, agents also differ in their skill sets and expertise levels. Finally, an IT service delivery firm typically signs up a Service Level Agreement (SLA) with each of its client organization. These SLAs classify tickets into different level of severities and for each severity level there is certain time constraints within which the ticket should be resolved.

3.1 Modelling TAP as an Online Scheduling Problem

In what follows, we first define a generic *scheduling problem* and introduce different classes of this problem including online scheduling. Next, we show that TAP is an instance of *online scheduling problem with unrelated parallel machines*.

Consider a set of n independent jobs $\{J_1, J_2, \ldots J_n\}$ to be scheduled on m machines, namely $M_1, M_2, \ldots M_m$. Each job J_j can be assigned to any one of the machines and requires a processing time t_i^j on machine M_i. The *scheduling problem* deals with all aspects of scheduling the jobs on these machines so that some predefined objective is achieved. There are many variations of this problem depending on the characteristics of the jobs, the machines, the processing constraints, and the optimality criterion. A few key criteria are given below:

Speed of the Machines - machines could be identical, related or unrelated. We call machines are identical if $t_i^j = t^j$ for each job J_j and for each machine M_i. If $t_i^j = t^j/s_i$, where s_i is the speed of machine M_i, the machines are uniform. In the most general case the machines are unrelated [9, 10].

Optimality Criterion - the goal of the scheduling task is to minimize an objective function. There could be various objective functions such as the processing time of the entire batch (makespan), the average completion time, the total flow time or the maximum lateness for any job. Since the scheduling problem is known to be NP-hard in the strong sense [7], research has focused on developing approximation algorithms.

Off-line vs Online - if all the jobs are known beforehand at the time of scheduling, then the problem is referred to as off-line scheduling. On the other hand, an online scheduling problem is the one where the jobs arrive in an online fashion and the scheduler does not know the sequence in which the jobs will be arriving at the beginning of the run. This lack of knowledge about the input set in an online schedule makes it impossible for the scheduler to guarantee an optimal schedule for all input instances. So the research focus in the online scheduling literature is to find scheduling algorithms that are guaranteed to be not too far from the optimal off-line algorithm. The performance of an online algorithm is measured by the competitive ratio (w.r.t. some objective function). An online algorithm is σ-competitive if for each input instance, the objective value of the schedule produced by the algorithm is at most σ times larger than the optimal objective value. The competitive ratio may depend on m, but should be independent of n, which reflects the fact that the number of jobs is not known to the on-line algorithm.

Other factors such as non-preemptive versus preemptive scheduling, availability of information regarding processing times of the job on various machines, etc. add to the variations of scheduling problem.

If we consider tickets to be jobs and agents as machines, it is apparent that the TAP is an instance of the online scheduling problem. With regard to the criterion discussed previously, we can say that the TAP is a scheduling problem with following features - unrelated machines (since different agents might have different levels of expertise in solving different problems), stochastic online arrival of tickets, and non-preemptive scheduling. However, in reality, agents need not behave exactly like machines due to human factors. In our model, we use the average case behavior of agents to take into account such factors. The objective function of the TAP is not very well defined in practice but for the sake of our analysis we use the *makespan*. However, all the ideas and theories can be easily extended to other objective functions also.

4 Push Model

Since the TAP is an instance of the *online scheduling problem on unrelated machines*, it can be solved by having a scheduler (manual or automatic) that allocates each arriving ticket to an agent in an online fashion. Any known online scheduling algorithm can be used by such a scheduler. This mode of allocation is indeed used as a current industry practice. We prefer to call such a model for TAP as *Push Model*. In a typical push model, there is a queue associated with each agent and whenever scheduler allocates a ticket to an agent, the ticket joins the agent's queue. Each agent picks a new ticket to work on from the head of his queue whenever he completes his current ticket. Thus the agent works on the tickets in a non-preemptive fashion. We call this model as push model because problem tickets are pushed by the scheduling unit towards the agents.

There are several scheduling algorithms in literature to solve the online scheduling problem for unrelated machines [17]. Thus, the scheduler (manual or automatic) of the push model can use any of these algorithms. For any such scheduling algorithm, the input would be the processing time t_i^j required by each agent M_i to solve the incoming ticket J_j that need to be allocated. The output of any such algorithm will be the agent to whom the ticket should be allocated. In what follows we discuss one interesting scheduling algorithm - *Generalized List Scheduling Algorithm* that can be used by the scheduler in push model.

4.1 Generalized List Scheduling Algorithm

One of the oldest and most well known greedy online scheduling algorithm is the *List Scheduling algorithm* proposed by Graham [11] in 1966. Although, the List algorithm was originally proposed only for identical machines but it can be easily modified and made suitable for the unrelated machines also. The specific assumptions made by the original List Scheduling algorithm are as follows (1) machines are identical, and (2) information regarding processing time of a job on each machine is known.

The original List algorithm works as follows: *for each new job, assign it to the machine with the least amount of work assigned so far.* The competitive ratio of the original List Scheduling algorithm is $2-1/m$ [11]. The generalized version of the List algorithm for non-identical machines is as follows: *for each new job, assign it to the machine with the least expected completion time for that job.* For the case of identical machines, this is equivalent to the usual formulation that the next job is assigned to the machine with the smallest load.

Aspnes et al [3] have shown that the competitive ratio of the generalized List Scheduling algorithm for unrelated machines is n. This is not very desirable since it is dependent on n which can get very large. To get a more useful competitive ratio, we analyze the generalized List scheduling assuming an upper bound on the ratio of the best processing time and worst processing time for any job over all the machines. We can expect to have such a bound in practice since each agent has a minimum level of competence. Before we analyze the competitive ratio for the generalized List algorithm, we need to define the following quantity. As before, let t_i^j be the time required by machine i to process the job j. Let the time required by the best machine to process the job j be t_{opt}^j, thus $t_{opt}^j = min_i\{t_i^j\}$. We assume that the ratio of the best time and the worst time for any job j is bounded by r, thus $\forall i, \forall j, \forall k, t_i^j/t_k^j \leq r$. Now the competitive ratio of the generalized List Scheduling algorithm can be given as follows.

Theorem 4.1. *The competitive ratio of the generalized List Scheduling algorithm for m unrelated machines is bounded above by $2r-1/m$.*

Proof. Let $W = \sum_{j=i}^{n} t_{opt}^j$, be the sum of the optimal processing times for all the jobs. If the *makespan* for the optimal schedule is *Opt*, the total processing time available for the optimal schedule is $m.Opt$. So, $Opt \geq W/m$ and also $Opt \geq t_i^j, \forall i, \forall j$.

Let A be the *makespan* of the schedule provided by the generalized List scheduling algorithm. So there is a job k assigned to machine l with processing time t_l^k that ends at makespan time. For any other machine i, it cannot end all its tasks before $A - t_i^k$, because then the job k would have been assigned to that machine. So machine i is busy at least from time 0 through $A - t_i^k$. Let i_j be the machine to which task j has been assigned. Thus we have,

$$\sum_j t_{i_j}^j \geq \sum_i (A - t_i^k) + t_l^k \tag{1}$$

By definition of r, we can produce following inequality.

$$\sum_j t^j_{i_j} \leq \sum_j r.t^j_{opt} = r.\sum_j t^j_{opt} = r.W \qquad (2)$$

$$\text{Also } \sum_i (A - t^k_i) + t^k_l = m.A - \sum_i t^k_i + t^k_l$$

$$\geq m.A - \sum_i r.t^k_{opt} + t^k_{opt} = m.A - m.r.t^k_{opt} + t^k_{opt}$$

$$= m.A - (m.r - 1).t^k_{opt} \qquad (3)$$

Combining the inequalities 1, 2 and 3, we get the following

$$\begin{aligned} r.W &\geq m.A - (m.r-1).t^k_{opt} \\ A &\leq r.W/m + (r - 1/m).t^k_{opt} \\ &\leq r.Opt + (r - 1/m).Opt = (2.r - 1/m).Opt \end{aligned}$$

□

The above theorem suggests that modified List Algorithm is simple and moderately efficient scheduling algorithm to be used by the scheduler in push model.

4.2 Inefficiencies of Push Model

Irrespective of whether the scheduling is done in an automated manner or manual fashion and independent of the scheduling algorithm used by the scheduler, the push model is always subjected to one serious limitation which restricts its efficiency and effectiveness. Any scheduling algorithm used by the scheduler needs to know the $t^j_i \; \forall i$ before it can come up with a schedule for the ticket j. Note that t^j_i depends on the proficiency of the agents in fixing ticket j. In practice, it is very difficult to estimate the proficiency of a human being and moreover, the proficiency of a human being keeps evolving with time because of several reasons such as training, experience, etc. This causes inaccuracies in the t^j_is leading to an inefficient solution.

The above limitation motivated us to study the alternative models for the TAP. In next section, we propose an auction based model which we call as *Pull Model* and we show that these kinds of models can overcome all the inefficiencies and limitations of the push model.

5. Pull Model

Note that an agent can himself better judge about his own skills. Thus, one possible way in which we can overcome the inefficiencies of the push model is that the scheduler of the push model directly asks each agent to reveal her skills and use that at the time of scheduling. However, the problem with such an approach is that the agents need not be truthful to the scheduler and no surprise that an agent misreports her skills because this may help her to get lesser and easier tickets. This is where auction based model can provide a desired solution of the problem where the scheduler need not explicitly ask for the information regarding skills of the agents. Instead, scheduler can extract this information in an indirect way through the bid values of the agents.

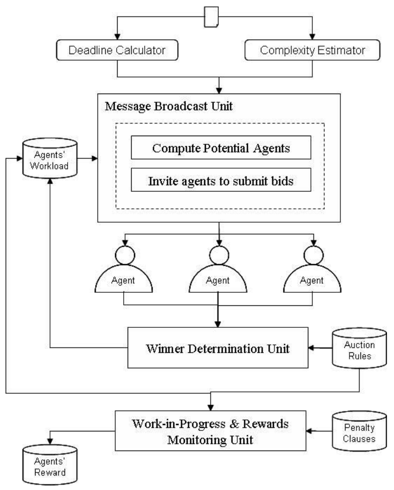

Figure 1. Auction House for Ticket Allocation

Thus, the main idea behind auction based models is that we convert any scheduling algorithm A into an equivalent truthful auction mechanism M such that the players (agents) reveal correct information about their skills indirectly in the form of bids and the assignment produced by the auction mechanism M becomes identical to that produced by A under complete information. Such a truthful auction mechanism enables fair estimation of the complexity of each job and allows the system to credit the agent based on the complexity of the tickets they are handling. It automatically leads to an efficient matching of the jobs with the agents based on the agents' skills and the skills required for each job. We refer to this model as the *Pull Model*.

In the pull model, each agent has access to the global ticket queue. The agents bid for the tickets in this queue and tickets are assigned to agents depending on their bid values. The *pull model* makes use of the techniques of classical *procurement (reverse) auctions*. Figure 1 depicts the schematic diagram of the proposed Pull Model for the ticket allocation problem. This model comprises a ticket queue which holds the problem tickets. There is a software entity namely *Ticket Auction House* (hereinafter, also referred to as *auction house*) which replaces the role of scheduler in the push model.

The Auction House comprises five major functional units - *deadline calculator*, *complexity estimator*, *message broadcast unit*, *winner determination unit*, and *work in progress & reward monitoring unit*. As soon as a ticket is taken for processing by the Auction House, it is sent to both the deadline calculator and the complexity estimator. The deadline calculator computes the deadlines by which this problem ticket must be fixed. This deadline depends on the Service Level Agreement (SLA) that the service delivery organization has signed with client organization. The complexity estimator estimates the time required by an agent to fix a given problem ticket, based on the description and other attributes of the ticket. In this paper, we will assume this component to be a black box and skip the details of its implementation.

The outputs of the deadline calculator and the complexity estimator are given into the message broadcast unit. The message broadcast unit comprises two subcomponents namely *compute potential agents* and *invite agents* to submit bids. The component *compute potential agents* takes three inputs: (i) deadline value, (ii) estimated time to fix the ticket, and (iii) the current workload of all the agents from agents' workload database. This component adds up the estimated time to the current workload and computes the expected time by which an individual agent would be able to fix the ticket if the ticket is assigned to that agent. Making use of these numbers, the component selects the agents who would be able to fix this problem ticket before the estimated deadline. Then all these potential agents are sent a broadcast message describing the problem ticket, estimated time, and the deadline. Next, all these potential agents are invited to submit bids for that ticket. The bid of an agent is the time that is required to fix this ticket if that agent works on this ticket. The estimated time for the ticket can be used as a reserve value to filter the bids received above this value, so that we don't consider spurious bids.

The bids of the agents are given as input to the winner determination unit. The winner determination unit decides that among these potential agents who should be assigned the ticket to work upon. The winner determination unit makes such decision based on the auction rules that are designed to achieve the allocation as per the chosen scheduling algorithm. Apart from winner determination rule, the auction mechanism also consists of payment (reward rule). The payment rule is used by the work-in-progress & rewards monitoring unit for the purpose of deciding the reward points of the agents and also penalty in case they fail to fix the ticket or take more time than allocated.

This model is called as Pull Model because unless an agent shows an interest to work on a problem ticket, the ticket can not be allocated to him. It is easy to observe from the discussion so far that in order to convert a push model into an equivalent pull model, one needs to convert the scheduling algorithm of the push model into an equivalent auction mechanism which is truthful. An interesting question to ask is whether it is possible to map any scheduling algorithm into a truthful mechanism. Although, in this paper we don't answer this general question, in the next section we show that we can design a truthful auction mechanism using reverse auctions for the generalized List scheduling algorithm.

5.1. Auction Mechanism for Generalized List Scheduling Algorithm

In auction theory literature, it is well established fact that any auction mechanism can be described by a winner determination rule and a reward rule [15]. Also, any auction mechanism induces a *Bayesian game* among the bidders and the bidders will bid as per the Bayesian Nash equilibrium strategy. A Bayesian Nash starategy need not be truthful in which case the bids of the bidders would not reflect their true estimates about the time they think they will take to work on the ticket. However, we always prefer those auction mechanisms for which truth telling is a Bayesian Nash equilibrium. Such auction mechanism will eliminate the need of knowing the agents' skills for the scheduler. The bids themselves will reflect the accurate information regarding skills of the agents. The auction mechanisms in which truth telling is Bayesian Nash equilibrium are said to satisfy the *incentive compatibility* property. [1]

In what follows, we describe a winner determination and reward rule that induces truthful bidding and achieves the same allocation as the generalized List scheduling algorithm. It can be shown that though this auction mechanism appears to be the well known VCG (Vickry-Clarke-Groves) mechanism [18, 6, 12] but in reality it is not. In this case, the auction of each ticket is not an independent event, but it is affected by other tickets auctioned before this since they contribute to the workload of the agents.

Let $N = \{1, 2, \ldots, n\}$ be the set of potential bidders for a new ticket.

- **Bid Structure:** A bid of an agent i represents the time that the agent thinks he will require to solve the ticket. We denote the bid of agent i by b_i.

- **Busy-Till-Period** The Busy-Till-Period of an agent i represents the time till when this agent is busy with his current work load. We denote the Busy-Till-Period of agent i by B_i. Our auction mechanism keeps track of the busy till period of each agent and keeps updating based on events such as agents finishing a ticket and agents getting allocated a ticket.

- **Winner Determination Rule:** Making use of Busy-Till-Period B_i and the bid b_i for each potential agent i, we compute the expected completion time E_i of the ticket. This is the time by which we expect the ticket to be completed if it is assigned to the agent i. The

[1]Please refer to [15] and [13] more details about the auction mechanisms, Bayesian games, and incentive compatibility property.

winner of the auction is the one for which E_i is the minimum. That is,

$$E_i = b_i + B_i, winnerID = argmin_{i \in N}\{E_i\} \quad (4)$$

The ticket under auction goes into the queue of the agent whose ID is $winnerID$.

- **Reward Rule:** Each winning agent also gets some reward points, which is a function of bid values of all the agents. Let $w_{(1)}$ be the ID of the winning agent and $w_{(2)}$ be the ID of the best losing agent. Then the reward $r_{w_{(1)}}$ for the winning agent is defined as

$$r_{w_{(1)}} = \alpha\left(b_{w_{(1)}} + E_{w_{(2)}} - E_{w_{(1)}}\right) \quad (5)$$

  where α is a constant that we call as *reward rate* and it basically converts the units from time to money or reward points. The exact details of the number α is immaterial for the rest of the discussion here. A losing agent does not get any reward.

The above four components together completely defines an auction mechanism which is truthful and achieves the same allocation as the generalized List scheduling algorithm. In what follows, we formally discuss these properties of this auction mechanism.

Theorem 5.1. *The proposed auction mechanism is truthful or strategy-proof.*

Proof. Let $U_i(x)$ be the utility function, i.e. the utility that agent i will derive by bidding a value x. Let t_i be the actual time required by the agent to solve the ticket. We need to show that $U_i(t_i) \geq U_i(b_i), \forall b_i \in [0, \infty)$. If this holds, then there is no incentive for the agent to bid untruthfully.

From Equations 4 and 5, it follows that

$$\begin{aligned} r_{w_{(1)}} &= \alpha\left(b_{w_{(1)}} + E_{w_{(2)}} - (B_{w_{(1)}} + b_{w_{(1)}})\right) \\ &= \alpha\left(E_{w_{(2)}} - B_{w_{(1)}}\right) \end{aligned}$$

The winning agent, $w_{(1)}$, gets a reward $r_{w_{(1)}}$ and needs to spend time $t_{w_{(1)}}$ to solve the ticket. Thus, the utility for the winner $U_{w_{(1)}}(x) = r_{w_{(1)}} - \alpha t_{w_{(1)}} = \alpha\left(E_{w_{(2)}} - B_{w_{(1)}} - t_{w_{(1)}}\right)$. The utility for any other non-winning agent i is 0 since their reward is 0 and they don't spend any time on the ticket since its not allocated to them. Now consider an agent i. Let E_{-i} be the earliest expected completion time among all the agents other than i. Consider the following two cases:

1. $B_i + t_i \leq E_{-i}$. For any bid $0 \leq b_i \leq (E_{-i} - B_i)$, agent i will be the winner, with $U_i(b_i) = \alpha\left(E_{-i} - B_i - t_i\right) \geq 0$, which is independent of b_i. Since, t_i is in the same range, $U_i(t_i)$ also has the same value. If $b_i > E_{-i} - B_i$, i will lose, leading to utility of 0. Thus $U_i(t_i) \geq U_i(b_i), \forall b_i$.

2. $B_i + t_i > E_{-i}$. For any bid $0 \leq b_i \leq (E_{-i} - B_i)$, agent i will be the winner, with $U_i(b_i) = \alpha\left(E_{-i} - B_i - t_i\right) < 0$, which is independent of b_i. Since the utility is negative, its not worth bidding in this range. If $b_i > E_{-i} - B_i$, i will lose, leading to utility of 0. In particular, t_i is also in this range, thus $U_i(t_i) = 0$. Thus $U_i(t_i) \geq U_i(b_i), \forall b_i$.

In either case, $U_i(t_i) \geq U_i(b_i), \forall b_i$, making truth telling the dominant strategy. □

Since the bids are truthful, it is obvious that the allocation produced by the winner determination mechanism is identical to that of the generalized List scheduling algorithm. To recap, the generalized List scheduling algorithm is as follows: *for each new job, assign it to the machine with the least expected completion time for that job.* This corresponds directly to the winner determination rule in the above auction which leads to the following lemma:

Lemma 5.1. *The competitive ratio of the auction mechanism for m agents is $2r - 1/m$.*

Proof. From Theorem 5.1, it follows that the agents will bid their true times for the tickets. The outcome of the auction mechanism is then identical to the generalized List algorithm for non-identical processors, from which the bound follows (Theorem 4.1). □

In spite of the efficiency of the Pull model, the Push model also has some advantages over the Pull model in terms of giving better control to the manager over ticket allocation. We can combine the best features of both models in a hybrid model. In this model, most of the tickets go through the pull model. Tickets that are urgent or critical are sent through the push model to reduce the overhead and give more control. We will skip the details of this model due to space constraints.

6. Experiments

We performed a detailed simulation study to compare the proposed Pull model with the traditional Push model. We simulated the environment in a typical service delivery organization. The simulation system consists of a set of agents. Tickets enter the system at a specified rate. A ticket could belong to any one of the several categories based on the content of the ticket. For each category, there is a range of time required to solve a ticket in that category. Each agent can have different levels of expertise for different categories of tickets.

In our experiments, we had 25 agents with tickets arriving randomly at an average rate of 1 per minute. Tickets are uniformly distributed across 4 different categories. The time ranges for tickets in each category are listed in

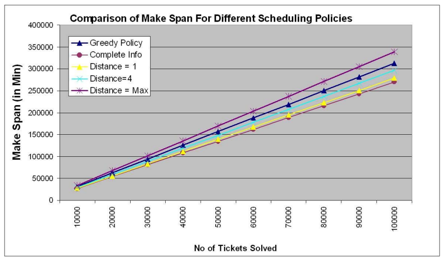

Figure 2. Makespan vs Number of Tickets

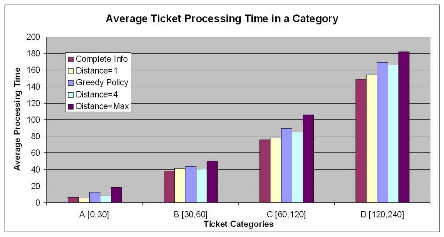

Figure 3. Average Ticket Processing Time

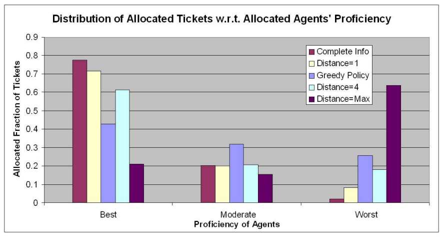

Figure 4. Distribution of Allocated Tickets

Category	1	2	3	4
Time Range (mins)	0-30	30-60	60-120	120-240

Table 1. Experimental Setting

Table 1. Proficiency of an agent in solving a ticket of any one of these 4 categories can be at any one of the 3 possible proficiency levels. If an agent is at level 1 for a particular category, say category 1 with range 0-30 minutes, the time taken by that agent will be a random number in the first third of that range, in this case 0-10 minutes. Similarly, an agent with proficiency level 2 for category 1 will take time in the range 10-20 minutes and an agent with level 3 will take time in the range 20-30 minutes. The proficiency matrix of an agent captures the proficiency level of the agent for each of these 4 categories. For each agent, we generate the proficiency matrix in a random manner. The experiments were performed on a Dual Pentium T2600, 2.16 GHz machine with 2 GB of RAM running Windows XP Professional. These machine configuration parameters are not relevant to our results, but we mention them for the sake of completeness.

As shown previously, the auction mechanism ensures truthful bids by the agents. Thus, the winner determination algorithm has perfect knowledge of the time any bidding agent will take to solve the given ticket. We refer to this as *Complete Information* scenario. In the Push model, there are several different possibilities, depending on the information available to the allocating manager. If the allocating manager has no idea about the agents' proficiency matrices, she will assume that all agents will need the same time for any given ticket. Thus she will assign the ticket to agent who is least loaded currently. We refer to this scenario as *Greedy Policy*. It may also happen that the manager has some idea about agents' proficiency matrices, but they are not accurate. We refer to this as *Distorted Information* scenario. We can measure the distortion between the actual proficiency matrix of an agent and the proficiency matrix perceived by the manager. There can be several ways to measure this distortion but we present one measure that we have used here. Let for an agent i, we have

$$\begin{aligned} \text{Actual Proficiency Matrix} &= [x_1, x_2, x_3, x_4] \\ \text{Perceived Proficiency Matrix} &= [y_1, y_2, y_3, y_4] \end{aligned}$$

where $x_j, y_j \in \{1, 2, 3\}$, then we define the *distance* D between actual proficiency matrix and the perceived proficiency matrix as follows: $D = \sum_{j=1}^{4} |(y_j - x_j)|$.

Note that here numbers 3 and 4 are special due to the experimental setting but we can modify them appropriately. In view of this definition to measure the amount of distortion in managers' information, we experiment with the settings of $D = 1$, $D = 4$ and the maximum possible distance.

6.1 Efficiency Measures

We use two different measures of efficiency – makespan and average time. For a given stream of tickets, we calculate the makespan as the time by which all tickets are completed over all agents. We varied the number of tickets from 10000 to 100000 and measured the makespan as shown in Figure 2. The results show that the makespan of the Complete Information case is always better than the other three, with an average savings of about 16% over the Greedy policy. As the distance between the truth and the perceived proficiency increases, the efficiency reduces. It is interesting to note that the Greedy policy performs better than the 'Distance = Max case. This implies that it is better to assume no information about the agents' proficiency and do the allocation accordingly rather than do it based on information that is very imprecise. Figure 3 shows the average time required to solve the tickets in various categories. Again, the Complete Information case always outperforms the Greedy policy by a margin of 16% or more.

6.2. Satisfaction Measures

Typically agents who are proficient in certain categories of tickets would like to work on those tickets. That way they can get more work done and also improve their reward. This will lead to better agent satisfaction. To measure this, we compute the ratio of tickets allocated to agents according to the proficiency levels. Figure 4 plots the fraction of tickets for each proficiency. In the case of Complete Info, almost 80% of the tickets get allocated to the most proficient agent. This ratio is much lower for the other cases. Thus complete information achieves a ticket allocation that is most acceptable to the agents as well. Each agent gets to work on tickets that they are most comfortable with, leading to better satisfaction.

6.3. Discussion

The critical factors in the above simulation study that can influence the outcome of the study are as follows - ticket arrival rate, variability in the complexity of the tickets and agents expertise levels, and overheads in conducting the auction. If ticket arrival rate is very low, then agents will be free most of the time and efficiency measures will be similar in both Push and Pull model. Similarly, if there is not much variability in the complexity and agents' skill levels then the allocation of tickets to agents does not matter much and both Push and Pull will behave similarly. In our simulation, we have ignored the overhead component of the Pull model. While this is justified for long running tickets, it will affect the outcome of the study if majority of the tickets are of very short duration.

7. Conclusions and Future Work

In this paper, we have addressed the Ticket Allocation Problem (TAP) that arises in services industries and showed that it is an instance of the online scheduling problem. We reviewed the current industry practice, Push model, and pointed out the inefficiencies of the model arising due to incomplete and/or imprecise information. We proposed a new auction based model, Pull model, to overcome these inefficiencies. We showed that one can convert a push model into a pull model just by converting the corresponding scheduling algorithm into a truthful auction mechanism which achieves the same allocation as the scheduling algorithm. As a proof of concept,we have proposed an auction mechanism that leads to truthful bidding and achieves the same allocation as suggested by the List scheduling algorithm. We experimentally verified that the Pull model does achieve better efficiency and agent satisfaction compared to the Push model. We have still not answered the bigger question whether every online scheduling algorithm can be converted into a truthful mechanism. If not all then is it possible that some subset of the online algorithms can be converted. We feel that all such questions open up a rich opportunity for further investigations.

References

[1] S. Albers. Better bounds for online scheduling. In *29th Annual ACM Symposium on Theory of Computing (STOC'97)*, 1997.

[2] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *42nd IEEE Symposium on Foundations of Computer Science (FOCS'01)*, 2001.

[3] J. Aspnes, Y. Azar, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computation*, pages 623–631, 1993.

[4] V. Auletta, R. D. Prisco, P. Penna, and G. Persiano. On designing truthful mechanisms for online scheduling. In *Proc. of SIROCCO*. Lecture Notes in Computer Science, 2005.

[5] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *15th annual ACM symposium on Parallel Algorithms and Architectures*, 2003.

[6] E. Clarke. Multi-part pricing of public goods. *Public Choice*, 11:17–23, 1971.

[7] M. Garey and G. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. Freeman, San Francisco, 1979.

[8] N. Garg and A. Kumar. Minimizing average flow time on related machines. In *ACM Symposium on Theory of Computing (STOC'06)*, 2006.

[9] M. Ghirardi and C. N. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 127(2):457–467, September 2005. available at http://ideas.repec.org/a/eee/ejores/v165y2005i2p457-467.html.

[10] R. Graham, E. Lawler, J. Lenstra, and A. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Maths*, 5:287–326, 1979.

[11] R. L. Graham. Bounds for certain multi-processing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[12] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[13] V. Krishna. *Auction Theory*. Academic Press, 2002.

[14] S. Leonardi and A. Marchetti-Spaccamela. On-line resource management with application to routing and scheduling. *Algorithmica*, 24(1):29–49, 1999.

[15] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[16] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. 31st ACM Symp. on Theory of Computing*, pages 129–140, 1999.

[17] J. Sgall. Online scheduling – a survey. In A. Fiat and G. Woeginger, editors, *On-Line Algorithms*, Lecture Notes in Computer Science, pages 196–231. Springer-Verlag, Berlin, 1997.

[18] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, March 1961.