Scheduling Service Tickets in Shared Delivery

Hari S. Gupta and Bikram Sengupta

IBM Research, Bangalore, India
{hsgupta1,bsengupt}@in.ibm.com

Abstract. We study the problem of optimally scheduling tickets in shared delivery of IT services. Such delivery models are characterized by a common pool of skilled agents who collectively support the service needs of several customers at a time. The ticket scheduling problem becomes interesting in this scenario due to the need to provide satisfactory experience to multiple customers with different Service Level Agreements (SLAs) in a cost-efficient and optimal way, by intelligently leveraging the available skill set and balancing workload across agents. We present a detailed description of the problem domain and introduce a novel metric for estimating the relative criticality of tickets from different customers at any point in time, taking into account several factors such as the distance from SLA breach, the SLA penalty and the expected volume of tickets during the rest of the service time window. This criticality measure is used within a Mixed Integer Programming (MIP) based solution approach to the ticket scheduling problem, where we consider the objectives of SLA penalty minimization, balancing breaches across customers, load balancing across agents, and maximizing skill match. Due to the complexity of the problem, optimization engines may not always return feasible or efficient solutions within reasonable time limits. Hence, we also develop a custom heuristic algorithm that returns acceptable solutions very fast. Detailed simulation experiments are used to compare these approaches and to demonstrate their efficiency in meeting the scheduling objectives of shared delivery.

1 Introduction

IT service delivery organizations (henceforth called service vendors) employ skilled practitioners to address service requests from customers. Many of these requests are relatively small, atomic tasks that require a specific skill and may be handled by a single practitioner within a short duration (e.g., a few minutes to a few days). Such requirements, which the vendor receives in the form of service *tickets*, may represent a specific IT problem experienced by the customer (e.g., “server down”, “transaction failed” etc.) or may be a request for a new capability (e.g., “create a new disk partition”). These requirements are generally of some business priority, with the expectation that it will be completed in a time-bound manner, as specified in Service Level Agreements (SLAs) between the customer and the vendor. If there are delays, then the customer’s business may be severely impacted, leading to significant financial losses. The consequent degradation in customer satisfaction may have implications on the vendor's future business with the customer. Moreover, vendors would need to compensate the customer by paying a penalty for any additional breach beyond the SLA threshold. Thus business imperatives and economic reasons call for timely, careful and intelligent handling of service tickets.

In this paper, we will study the problem of efficiently scheduling tickets in a service delivery organization. In particular, we study ticket scheduling in the context of the shared delivery or factory model [5,1], which is increasingly being used by IT vendors. What distinguishes these models is that instead of dedicated, customer-specific teams, a common pool of agents belonging to a specialization area (e.g., say a packaged application system) supports the needs of several customers who need services in that area. These customers may have different SLAs with the vendor, depending on the criticality of their business, and the price they are willing to pay for the vendor's services. Such shared delivery makes the assignment of tickets to agents an involved balancing exercise: for example, the vendor has to strive to reduce the aggregate SLA penalty it has to pay, while trying to maintain some parity between the service experiences provided to the different customers; at the same time, the vendor has to ensure that, to the extent possible, the workload of different agents are balanced (to prevent agent fatigue or reduce idle time), and tickets are assigned to agents with high levels of skill in the relevant areas (so that customers are satisfied with the quality of service provided). Naturally, as the number of tickets and customers scale up, balancing the various ticket assignment criteria and analysing the trade-offs, are way beyond what a human dispatcher will be capable of. There is a need to explore automated decision-making systems that can intelligently schedule tickets through an expert knowledge of shared service delivery, and that is encoded through a robust optimization or heuristic formulation.

Towards that end, this paper makes the following important contributions. After discussing related work (Sec. 2), we present a detailed description of the problem domain of scheduling tickets in a shared delivery system (Sec. 3), including its differences with the traditional model of delivery via customer-specific teams. We then develop a set of scheduling objectives for shared delivery, that attempts to balance the vendor's needs for profitability and delivery efficiency, with the need to ensure parity in the service experience of individual customers. To the best of our knowledge, this is the first work that investigates the complexities of work scheduling in a shared environment to this depth. After introducing the problem, over the next few sections, we present a number of solution approaches: a mixed integer programming (MIP) based formulation (Sec. 4) that attempts to optimize the scheduling objectives in a priority order; a heuristic algorithm that attempts to balance the objectives over multiple phases guided by several policies (Sec. 6); and two greedy algorithms (Sec. 5) that each tries to optimize on a specific objective, and represent baselines with which we can compare our approaches. In Sec. 7, we report on detailed simulation-based experiments to study the performance of all the approaches and analyse their scheduling effectiveness. We find that both MIP and heuristic approaches do very well in balancing the various scheduling objectives for moderate-sized problems, but the heuristic algorithm scales much better. Finally, Sec. 8 presents our conclusions and directions for future research.

2 Related Work

There is a rich body of literature related to the general problem of scheduling jobs on machines (job shop scheduling or JSS) [8,6], and different variants (e.g., online or offline scheduling, related or unrelated machines) have been investigated in depth. The most common scheduling objectives of JSS are minimization of makespan, total completion time, total tardiness, and load balancing. While these are standard objectives with wide applicability, scheduling of tickets need specific focus on the management of SLAs. In particular, shared delivery calls for scheduling objectives that are sensitive to the SLA and expertise needs of individual customer, even as they seek to maximize a vendor's profitability and delivery efficiency.

The allocation of tickets to agents also has (superficial) similarities with the routing of calls in a call-center. Skill-based routing (SBR) [3,11] segments calls based on the skills needed to handle them and routes a call to an available agent with appropriate skill. Given the low limits of caller patience, the objective is to minimize waiting time and calls drop rate, and thus there is no scope for batch scheduling. In contrast, service tickets can be batched together at the start of a shift and then at specific intervals, and dispatched to agent queues. Moreover, unlike a call that will naturally drop off beyond a waiting time, each ticket that arrives has to be serviced, even if its service level objective (SLO) has been breached; these tickets thus impose an additional burden that can potentially delay other tickets.

In recent years, there has been considerable interest on automation, measurement and optimization of service delivery. For example, [4] proposes a solution for efficient seat utilization to reduce infrastructure costs. [12] models the cost overheads in distributed service delivery and proposes relevant metrics to assess the same. Our work combines automation and optimization is scheduling of service tickets in shared delivery, using the novel metric of relative ticket criticality. There has been limited research on the topic of optimal allocation of service tickets, and the few papers in the area mostly address issues that are complementary to our work. For example, [10] studies game theoretic aspects of the problem and designs incentive mechanisms for ticket complexity elicitation in a truthful way from agents. In [2], an auction-based pull model for ticket allocation is proposed, along with incentive mechanisms for agents to be truthful about expected resolution time when bidding for tickets. While theoretically appealing, it is unlikely that large-scale service delivery bound by SLAs can function effectively when it completely depends on agents to bid for tickets. [9] develops a Markov model by mining ticket resolution sequences of historical tickets that were transferred from one group to another before being resolved. This model is then used to guide future ticket transfers and reduce mis-routing. In [7], supervised learning techniques are used to determine the likelihood that a service request corresponds to a specific problem symptom; prior performances of support groups are then analyzed to identify groups that have the necessary skills to resolve these problems. While expertise is indeed an important consideration for ticket assignment, there are other practical objectives (e.g., manage SLAs, balance load etc.) that need to be taken into account in a real-life delivery organization.

3 Domain Description

The traditional mode of delivery of IT services has been through a dedicated team of agents for every customer. In practice, this simplifies the task of assigning and scheduling tickets within the team. All agents work towards a common goal (of serving the customer's interests in the best possible way) and this synergy of purpose helps in prioritization of tasks, often in consultation with the customer. However, while this model is responsive to the needs of individual customers, it has a number of drawbacks from the vendor's perspective. The most significant of these are that utilization of agents may be poor/uneven across teams due to variability in demand, and that dedicated teams tend to operate in silos making knowledge sharing and deployment of best practices difficult. For these reasons, many vendors [5,1] are now adopting a shared delivery model, wherein groups of agents who specialize in a particular domain are pooled together and made responsible for addressing the needs of multiple customers who require services in that domain. What makes this approach feasible is that for a significant body of IT services, the skills needed are more domain-specific than customer-centric. Such a model can translate to reduced service costs (due to increased sharing of resources and practices), whose benefits may then be shared with the customer. However, in a shared delivery model, the vendor needs to assume greater ownership of scheduling work. Tickets from different customers would compete for the attention of the same set of agents, and this would introduce a natural conflict into the system. How well the vendor manages this conflict will determine the service experience of individual customers from the shared delivery system, as well as the financial returns of the vendor. It is in this context that we study the ticket scheduling problem.

We begin with an informal description of the problem domain, introducing the main concepts and their relationships. An IT service vendor will be employing a pool of *agents* for each specialization area that it supports (for example, management of servers, packaged application systems etc.) Within the specialization area, different *categories of service requests* may arise. An agent will be *skilled* in handling one or more of these request categories. Each pool of agents from a given specialization area will be servicing tickets received from a set of *customers.* A *ticket* usually contains a description of the problem or request, based on which it may be assigned the appropriate service category at the time of its creation. A ticket will also contain a *priority level*, which reflects its business urgency from the perspective of the customer. The service vendor and customer would have entered into a *Service Level Agreement (SLA)* for resolving tickets within a specific time limit, based on their priority. For example, for priority 1 tickets (having the highest urgency) from customer X, the agreement may be that 90% of such tickets within a given service delivery time window (e.g., a month) will be resolved within 4 hours. If the SLA is breached, the vendor is generally required to pay a *penalty* to the customer, per additional ticket that exceeds the time limit. A *Service Level Objective (SLO)* represents the manifestation of a SLA at a per-ticket level and is the time-limit by which a vendor attempts to resolve each ticket of a given priority from a customer. Thus for the SLA example introduced above, the SLO for priority 1 tickets from customer X would be 4 hours. While an agent needs to have specific skill(s) to be eligible to resolve a ticket of a given category, the *resolution effort* i.e., the time spent by the agent on the ticket to resolve it, would depend on the agent's degree of expertise for the ticket (higher the expertise, lower the effort). There may be different ways to estimate expertise e.g., by number of relevant tickets successfully resolved, level of training received, number of years of experience in the area etc. In this paper, we will assume that expertise values are available, without going into specific computation methods. We will consider the following *states* of a ticket: *New Arrival* (a ticket has just arrived), *Queued*, (it has been placed in an agent's queue), *Work-in-Progress or WIP* (an agent is working on it), and *Closed* (it has been resolved).

Relative Criticality Measure

In order to help a vendor prioritize tickets coming in from different customers to the same pool of agents, we now introduce a notion of relative *criticality* of tickets. For a given customer c and priority p, let

- r_{cp} = % of tickets allowed to breach SLA
- q_{cp} = Penalty of a SLO breach beyond SLA
- u_{cp} = Number of tickets to be scheduled now
- v_{cp} = Number of tickets closed in the past
- w_{cp} = Number of tickets expected to come in future
- n_{cp} = Number of tickets breached SLO in the past (among the closed tickets)

Given the above information, we can compute the estimated number of tickets allowed to breach (at max) among present and future tickets (m_{cp}) by:

$$m_{cp} = \frac{(u_{cp} + v_{cp} + w_{cp}) * r_{cp}}{100} - n_{cp}$$

Now, we define a ticket's criticality belonging to the given customer-priority as:

$$C_{cp} = \begin{cases} \frac{(u_{cp}+w_{cp})*q_{cp}}{m_{cp}+1} & \text{if } m_{cp} < u_{cp} + w_{cp} \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, the criticality of a ticket is less when the number of allowed SLO breaches (m_{cp}) is more; criticality is more when the volume of tickets ($u_{cp} + w_{cp}$) over which these breaches are allowed is more; criticality is more when the SLA penalty is more; and criticality is negligible (zero) when the number of allowed SLO breaches that remain is more than or equal to the current and future ticket volume. It may be noted that the criticality measure depends on knowledge of the future volume of tickets. Usually the volume of tickets from a specific customer over a given service window can be reasonably estimated based on historical data, and this estimate is also used by the vendor to draw up staffing plans. The more accurate this information, the more refined will the criticality measure be. Of course, the vendor can also update this estimate at any time during a service window (e.g. based on some unanticipated event) and the criticality measure would be adjusted accordingly for subsequent tickets.

Problem Objectives and Constraints

Given this background, we consider the following problem: assume we have a pool of agents, where each agent has a queue of tickets, and may have a *WIP* ticket she is working on; given a set of new tickets that have arrived, how do we optimally allocate the new and queued tickets to the agents? Below, we introduce our scheduling objectives in the order of their (decreasing) relative importance, from the vendor's perspective.

The first objective (*SLA Penalty Minimization*) that the vendor will try to meet is minimization of the penalty it has to pay due to SLO breaches beyond the SLA limit. Assume there is a ticket t from a customer c with priority p, where the permitted number of breaches for priority p tickets from c under the SLA has already been reached or exceeded. The vendor would then want to schedule t in a way that it does not breach its SLO, since otherwise the vendor will have to pay a penalty. Note that, to achieve this, the vendor may have to breach (in some cases) other ticket(s) that are either within their respective SLA limits, or carry a lower penalty.

The above objective helps a vendor minimize financial losses *after a SLA has already been violated*. However, the vendor would want to schedule tickets in a way that minimizes the chance of a SLA breach in the first place. This leads to our second objective set *SLO Breach Balancing and Minimization*, which (i) helps a vendor reduce SLO breaches for high criticality tickets (thereby reducing the risk of an SLA breach) by minimizing the maximum criticality value of a ticket with SLO breach, and (ii) minimizes the total number of SLO breaches.

Our third objective (*Load Balancing*) is to ensure that some agents are not overloaded with work, while others are relatively idle. An inequitable load distribution need not always lead to an SLO breach, but it would adversely impact agent's motivation, thereby justifying a separate objective to ensure fairness. In general, of course, a balanced workload distribution also helps in reducing delays, thus this objective complements the first two introduced above.

Our fourth objective (*Expertise Maximization*) helps in assigning tickets to agents who are highly skilled in resolving the associated problem categories while maintaining the fairness across customers (achieved by a sub-objective). In particular, for the shared delivery system, we want to ensure this not only at an aggregate level, but for each customer being supported, since it will help ensure that tickets are resolved faster and the solution quality is high, both of which will positively impact customer satisfaction.

As is usually the case with multi-objective optimization, the solutions motivated by individual objectives will differ, even conflict. For example, expertise maximization without load balance may lead to some highly skilled people being assigned excessive work, which can de-motivate them as well as lead to delays and SLO breaches. Hence it is essential to arrange the objectives in the right sequence (as introduced above), so as to address the vendor's scheduling goals most optimally.

In addition to finding an efficient solution for the given objectives, we adhere to a few constraints, a couple of which need some explanation. First, once a ticket has been assigned to an agent, we do not re-assign it to another agent subsequently (even if it improves the scheduling objectives), although, as new tickets arrive, we may change its position in the queue. This is to prevent a ticket from hopping from one agent to another multiple times, which would be an irritant for the agents (since, they may have already reviewed tickets in queue) and will also confuse customers (who are notified of ticket assignments). There may indeed be a few cases where re-assignment is a practical necessity, but for now we leave it to the agents and supervisors to identify these instances and manually re-assign. Secondly, once a ticket has breached its SLO, we limit the number of tickets in the queue that arrived later than this ticket but are placed ahead of it. This ensures that no ticket starves, since otherwise such tickets can get indefinitely delayed as the scheduler tries to avoid further SLO breaches in new tickets.

4 MIP Formulation

We will now develop a Mixed Integer Programming (MIP) based formulation for optimal assignment of tickets in shared delivery. This will formalize the objectives and constraints introduced in Sec. 3.

The set of inputs for the given problem are: **(a)** a set of customers S_C, a set of priorities S_P, a set of ticket categories T_C and a set of (Min, Max) resolution time tuples R_T for the different ticket categories; **(b)** a set of SLAs (SLA_{CP}) applicable over a specific service time window (e.g., a month) for different (customer, priority) combinations, from which we can further derive the sets, (i) SLO_{CP} representing the time-limits by which tickets of different (customer, priority) combinations are expected to be solved, (ii) R_{CP}, representing the maximum percentage of tickets that are allowed to breach the SLO time-limits for different (customer, priority) combinations, and (iii) Q_{CP}, which denotes the penalty per additional SLO breach beyond maximum allowed percentage limits, for different customers and priorities; **(c)** a set of agents S_A {1, 2, ..., M}, and for each agent $a \in S_A$, (i) a set of skills $SK_a \subseteq T_C$, comprising the set of ticket categories the agent can resolve, and (ii) existing load on the agent L_a due to *WIP* ticket's remaining resolution time in the agent's queue; **(d)** a set of (*New Arrival* and *Queued*) tickets S_T {1, 2, ..., N} that need to be assigned to the agents, where each ticket t_k contains, (i) information about the customer ($Cust(t_k)$), priority ($Pri(t_k)$) and category ($Cat(t_k)$) to which it belongs, (ii) time stamp representing the arrival time of the ticket into the system, denoted by TS_k, and

(iii) information about any existing agent assignment, which needs to be maintained in the next run; **(e)** expertise value f_{ij} for each ticket j and agent i; **(f)** resolution time t_{ij} for each ticket j and agent i (calculated from the min and max resolution times of the category to which the ticket belongs, and from f_{ij}).

Note: In the MIP model of the problem, for compactness we have used the logical expressions at several places, all of which are linearisable, and have been linearised for the implementation.

Before describing the MIP model, we first describe the set of decision variables used in the formulation, secondly the constraints on these variables, and finally the objectives.

A $N \times M$ decision variables matrix (A) is used to record assignment, in which each entry $a_{ij} \in \{0, 1\}$ represents whether a ticket j is assigned to an agent i or not. The value of any entry a_{ij} should result 1 if the ticket j is assigned to agent i, or 0 otherwise. Similarly, we define two more decision variables matrices S and E, in which each entry $s_{ij} \geq 0$ ($e_{ij} \geq 0$) represents the start time (end time) of the ticket j if it is assigned to the agent i, or 0 otherwise.

We maintain another $N \times M$ Assignment History Matrix (H) to constrain the decision matrix (A), and each entry $h_{ij} \in \{0, 1\}$ in H is 1 if ticket j was assigned to agent i (when j is a *Queued* ticket), or 0 otherwise. Therefore,

$$\forall j \in S_T, \sum_{i \in S_A} h_{ij} \leq 1$$

Let β_{ij} represents the number of tickets in *Closed* or *WIP* state, which arrived later than ticket j, but scheduled before ticket j on agent i. We define also, $\alpha_{ijk} \in \{0, 1\}$ as an indicator variable identifying whether a ticket k arrived later than a ticket j, and is assigned to agent i ahead of j, i.e.,

$$\alpha_{ijk} = \begin{cases} 1 \text{ if } a_{ij} = a_{ik} = 1 \wedge TS_k > TS_j \wedge s_{ij} \geq e_{ik} \\ 0 \ \ \text{otherwise} \end{cases}$$

The completion time for each ticket can be defined as the sum of its end times on all agents, as the end time of the ticket would be 0 for the agents to whom ticket is not assigned, i.e., $\forall j \in S_T, c_j = \sum_{i \in S_A} e_{ij}$

For a ticket j, we use $b_j \in \{0, 1\}$ to record if j will breach its SLO or not. The variable $b_j = 1$ if the ticket's completion time (c_j) is greater than its remaining SLO time, and 0 otherwise Let t_c is the current time (i.e., time when scheduling starts), then

$$b_j = \begin{cases} 1 \ \ \text{if } (Pr(j) = p) \wedge (Cust(j) = c) \\ \quad \wedge (c_j > SLO_{cp} - (t_c - TS_j)) \\ 0 \ \ \text{otherwise} \end{cases}$$

Let for a customer c and priority p, n_{cp} denotes the set of tickets with already breached SLO's, r_{cp} denotes the max percentage of tickets allowed to breach according to SLA, u_{cp} denotes the *New Arrival*s and the existing tickets which are in *Queued* state, v_{cp} denotes the tickets *Closed* in the past and the tickets in *WIP* state, w_{cp} denotes the expected number of tickets in future, m_{cp} denotes the maximum number of tickets allowed to breach in the current and future schedules, q_{cp} denotes the penalty for an SLO breach beyond SLA, and z_{cp} denotes the number of tickets breaching SLO beyond SLA. Also, an indicator values matrix G is given, in which each entry $\gamma_{jcp} \in \{0, 1\}$, is 1 if a ticket j belongs to customer c and is of priority p, or is 0 otherwise. Now, we can define m_{cp} and z_{cp} as follows:

$$m_{cp} = \lfloor (|u_{cp}| + |v_{cp}| + |w_{cp}|) * \frac{r_{cp}}{100} \rfloor - |n_{cp}|$$

$$z_{cp} = \max(\sum_{j \in S_T} (\gamma_{jcp} * b_j) - m_{cp}, 0)$$

Next we specify the set of constraints using the variables and other elements mentioned earlier.

C_1 : Each ticket is *assigned to one and only one agent.*

$$\forall j \in S_T, \sum_{i \in S_A} a_{ij} = 1$$

C_2 : A ticket can only be assigned to an agent who has the required skill to resolve it.

$$\forall i \in S_A, j \in S_T, (a_{ij} = 1 \Rightarrow Cat(j) \in SK_i)$$

C_3 : The difference between the end time and start time of a ticket i on agent i should be equal to the resolution time t_{ij} if the ticket is assigned to the agent, and 0 otherwise:

$$\forall i \in S_A, j \in S_T, (e_{ij} - s_{ij} = t_{ij} * a_{ij})$$

C_4 : If a ticket j is assigned to an agent i then the start time (s_{ij}) would be $\geq L_i$, as agent cannot start working on the ticket without closing the ticket in *WIP* state with the agent, i.e.,

$$\forall i \in S_A, j \in S_T, (s_{ij} \geq a_{ij} * L_i)$$

C_5 : Since a ticket assigned to an agent is never transferred to another agent in a subsequent run of the scheduler, and to put this constrain we use the History Assignment Matrix (H) defined earlier.

$$\forall i \in S_A, j \in S_T, (h_{ij} = 1 \Rightarrow a_{ij} = 1)$$

C_6 : To prevent *starvation* of any ticket j breaching SLO, we put a bound MDC on the number of tickets which can be scheduled before ticket j and have arrived later than j in the system (i.e., the number of tickets having time stamp greater than the time stamp of ticket j):

$$\forall i \in S_A, \forall j \in S_T, (b_j = 1 \Rightarrow (\sum_{k \in S_T} \alpha_{ijk} + \beta_{ij}) \leq MDC)$$

C_7 : The constraint that the tickets do not have overlapping schedule (i.e., an agent works on one ticket at a time), is defined as follows:

$$\forall i \in S_A, j,k \in S_T (j \neq k \Rightarrow (s_{ij} \geq e_{ik} \vee s_{ik} \geq e_{ij}))$$

Subject to the constraints given above, the *MIP* tries to optimize for the following objectives. The objectives are defined in the order of their priority, as we discussed in Sec. 3.

Objective 1: Minimizing the total penalty due to SLO breaches beyond SLA's of different customer-priority tickets:

$$minimize(\sum_{c \in S_C} \sum_{p \in S_P} z_{cp} * q_{cp})$$

Objective 2: While the above objective is concerned with SLO breaches beyond the SLA limit, we have to try and minimize SLO breaches at each step, well before the SLA limit has been reached. In addition, we need to ensure that the SLO breaches, when they must occur, are balanced to the extent possible across different (customer, priority) combinations, taking into account the relative *criticality* of each breach. Since a further $m_{cp} + 1$ breaches will lead to a penalty of q_{cp} for a customer c and priority p, we estimate the criticality of each individual breach as $\frac{q_{cp} * (u_{cp} + w_{cp})}{m_{cp}+1}$. We thus have the following two goals:

$$minimize(\max_{c \in S_C, p \in S_P, m_{cp} > 0} (\frac{q_{cp} * (u_{cp} + w_{cp})}{m_{cp} + 1} * \sum_j b_j * \gamma_{jcp}))$$

$$minimize \sum_{j \in S_T} b_j$$

Objective 3: Load balancing across agents:

$$minimize(\max_{i \in S_A}(L_i + \sum_{j \in S_T} t_{ij} * a_{ij}))$$

Objective 4: To ensure that tickets are assigned to agents with high expertise whenever possible, we try to maximize the aggregate agent expertise across all tickets. However, in shared delivery, we also need to ensure that every customer individually receives a fair share of available expertise, hence we try to maximize the minimum average expertise of assignments received by any customer. This is handled through the following objectives:

$$maximize(\min_{c \in S_C} \frac{F_c + \sum_{j \in u_c} \sum_{i \in S_A} (a_{ij} * f_{ij})}{u_c + v_c})$$

$$maximize(\sum_{j \in S_T} \sum_{iinS_A} (a_{ij} * f_{ij}))$$

In the above, $u_c = \sum_{p \in S_P} u_{cp}$ and $v_c = \sum_{p \in S_P} v_{cp}$. F_c represents the aggregate expertise received by customer c for all closed and WIP tickets. For a ticket $j \in v_c$, let $Ag(j)$ return the agent who was assigned j. Then, $F_c = \sum_{j \in v_c, Ag(j)=i} f_{ij}$.

As the overall problem is multi-objective and there are trade-offs involved while optimizing for these objectives, we solve for the objectives in the order of their relative importance, and use the results from one solution as a constraint when solving the next objective. Although one can go for a dynamic prioritization of objectives considering the knowledge of their relative importance, but to our knowledge the presented prioritization of objectives best referred to the current service delivery environment.

5 Greedy Algorithms

In this section, we present two greedy algorithms which we have used to compare the MIP approach and the heuristic algorithm that we will propose next. The first algorithm (*GTMin*), tries to greedily minimize the number of tardy tickets (i.e., tickets breaching SLO's), while the second algorithm (*GEMax*), maximizes the average expertise of ticket assignments. In a way, these algorithms represent baselines for standard scheduling/assignment objectives.

Algorithm: Greedy Tardy Tickets Minimization (GTMin): First sort the tickets in the increasing order of SLO time limits; Pick the tickets one by one in the sorted order: (a) assign a *Queued* ticket to an agent whom it was assigned previously, and (b) assign a *newly arrived* ticket to an agent a such that the ticket's completion time is least, if is assigned to a. Whenever a ticket is assigned to an agent, the ticket is added in the end of the agent's ticket queue.

Algorithm: Greedy Expertise Maximization (GEMax): First sort the tickets in the increasing order of SLO time limits; Pick the tickets one by one in the sorted order: (a) assign a *Queued* ticket to an agent whom it was assigned previously, and (b) assign a *newly arrived* ticket to an agent a such that a's expertise level is maximum for the category to which the ticket belongs. Whenever a ticket is assigned to an agent, the ticket is added in the end of the agent's ticket queue.

6 Heuristic Algorithm

The heuristic algorithm runs in two phases. The first phase is a variant of *GTMin*, in which the heuristic tries to minimize the number of SLO breaches along with load balancing (lines 1 to 6), and additionally tries to increase the expertise if possible (line 6). In the second phase, it tries to reposition a ticket in the same agent's queue to whom the ticket is assigned (line 13) , or swap the tickets across agents with repositioning after swapping (line 15), so that the total penalty decreases. It is guided by 3 policies: *pick-ticket* defines the order of selecting tickets one at a time and either repositioning it in the same queue using *tuning-policy* or swapping it with some other ticket along with repositioning according to *swap-policy*. Overall, the heuristic considers all objectives during its run, while meeting constraints.

```
Algorithm: Heuristic
Phase 1:
1.  Sort the tickets in the increasing order of SLO time limits (deadlines);
2.  For each ticket $t$ in sorted order, do
3.      $C_T$ = minimum completion time, if $t$ is assigned to any agent at the last
        position in the agent's queue;
4.      $s_a$ = set of agents: $\forall a \in s_a$, $t$'s completion time is within $x\%$ (we use
        $x = 20$) of $C_T$, if $t$ is assigned to $a$ at the last position in $a$'s ticket-queue;
5.      If $t$ is in Queued state and previously assigned to agent $a'$, then
            assign $t$ back to agent $a'$ at last position in $a'$'s ticket-queue;
6.      Otherwise, assign $t$ to an agent $a \in s_a$ such that $a$'s expertise is
        maximum among all the agents in $s_a$ for the category to which $t$ belongs;
Phase 2:
7.  $P_T$ = total penalty due to SLO breaches so far including past breaches,
        beyond SLA's for all customer-priority pairs;
8.  For $N = 1$ to $\#tickets$, do
9.      Pick a ticket $t$ using algorithm pick-ticket;
10.     $P_{tuning}$ = total penalty on tuning the position of $t$ using tuning-policy;
11.     $P_{swap}$ = total penalty on swapping $t$ with other tickets using swap-policy;
12.     If $P_{tuning} < P_T$ AND $P_{tuning} \leq P_{swap}$
13.         Readjust the positions of the tickets as suggested by tuning-policy;
14.     Else If $P_{swap} \leq P_T$
15.         Readjust the positions and assignments of the tickets as suggested by
            swap-policy;
```

Pick-Ticket: Choose a ticket from the unpicked tickets abide by the following rules: (a) choose a ticket which is breaching SLO and for corresponding customer-priority pair the percentage of SLO breaches is more than the SLA, (b) in case of a tie from the rule-a, then choose a ticket, the penalty of which is maximum among all the penalized breaching unpicked tickets, (c) to break a tie further, choose a ticket having highest criticality value, (d) in case of further tie, choose a ticket having least deadline (SLO time limit) which can be negative also, (e) if more than one ticket follow all the three rules, then pick any of them and return.

Tuning-policy(t): Shift the ticket t backward and forward in the same agent's queue to whom the ticket is assigned, by shifting other tickets in the queue appropriately. Pick a shift among all the positions tried, such that the total penalty after the shift is minimum among the total penalties after other shifts, and is also less than the total penalty before the shift. If more than one shifts induce same amount of reduction in the total penalty, then choose the position such that the shift is minimum from the original position. While shifting the tickets maintain the starvation constraints (i.e., any *Queued* ticket is not delayed by more than *MDC* number of tickets arrived later than t). Suggest the new positions of the tickets.

Swap-policy(t): Swap the ticket t with each of the other tickets t', one by one. Let t is assigned to agent a and t' is assigned to agent a'. If t and t' are assigned to same agent (i.e., $a = a'$) then do nothing. Otherwise, swap the positions and assignments of the tickets (i.e., assign t to a' and t' to a). Tune the positions of t in a''s queue and t' in a's queue respectively, such that the reduction in total penalty after tuning the positions is maximum. Restore the assignments and positions of the tickets, and repeat the swapping of t with other tickets. Among all the swaps, pick a swap such that the total penalty is minimum among all the swaps with different tickets. Suggest the new assignments and positions of the tickets.

7 Experiments

7.1 Methodology

To test the efficacy of our scheduling approaches (MIP, heuristic) through experiments, we have designed *an event driven simulator*, which mimics the service delivery system by tracking events such as: ticket generation, dispatch event, start ticket event (which occurs when an agent picks a ticket from queue), and ticket departure event (when a ticket is closed). For our experiments, data sets were designed with utmost care, by studying historical data sets and through interactions with service delivery practitioners. We consider 10 different ticket categories, 5 different customers and 4 priority levels P_1 (highest priority) through P_4 (lowest priority). Approximate distribution of tickets across different priority levels is as follows: P_1 : 5%, P_2 : 10%, P_3 : 35%, P_4 : 50%. For each (customer, priority) combination, the SLOs are generated in the range of 3 to 40 time units, the maximum allowed breaches are varied from 5% to 20%, while SLA penalties are generated in the range of 1 to 20 monetary units. To ensure realistic SLAs, the data generation procedure was constrained so that for each customer, SLOs for higher priority tickets are smaller than for lower priority tickets, maximum allowed breach level %s are lower, while SLA penalties are higher, reflecting the higher criticality associated with higher priority. For each category c, we assume we have a minimum ($t_{c_{min}}$) and a maximum ($t_{c_{max}}$) observed resolution time (ORT), through historical analysis of tickets of that category. With each agent, we associate between 2 to 6 categories reflecting his/her skills, generated from a Gaussian distribution. For each (agent, category) pair, we set an expertise level = 0 if the agent does not have the corresponding skill, otherwise an expertise value is assigned in the range of 0.1 to 0.9, using a Gaussian distribution. We define the Estimated Resolution Time (ERT) t_{ij} for a ticket j (of category c) and agent i as follows, where α the expertise of the agent for the category c:

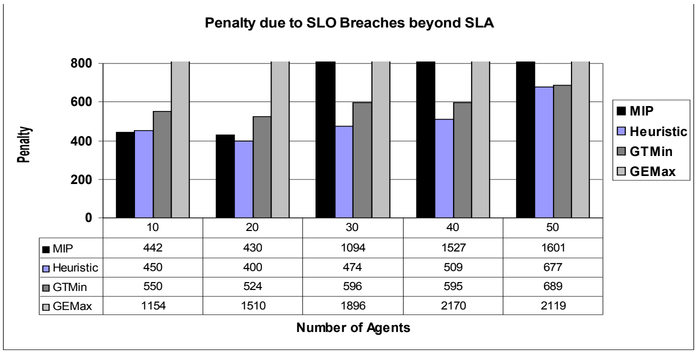
$$t_{ij} = t_{c_{max}} - \frac{t_{c_{max}} - t_{c_{min}}}{2} * \alpha^2$$

For an agent with very high expertise, this means that the resolution time is estimated to be around the middle of the (min, max) ORT range. This is a realistic approach, since the historical minimum time observed for a ticket of a given category cannot be directly and completely linked to the expertise of the agent who had resolved it. Also, we use α^2, so that ERT grows sub-linearly with decrease in expertise. To predict the future volume (used in computing the criticality value) corresponding to a customer and priority, we perturb the future volume from the dataset randomly within ±20% for a period of 1000 time units. We do all the experiments for a stream of 1000 tickets, and with number of agents varying from 10 to 50. The frequency of scheduling is set to 0.3 time units, i.e., after every 0.3 time units the scheduler (*MIP*, *heuristic*, *GTMin*, or *GEMax*) is invoked to schedule and reposition, the *new arrivals* within this period and *Queued* tickets respectively.

We consider a Poisson model for tickets arrival, where the inter-arrival times follow an exponential distribution, with bursts following Gaussian distribution (as bursts are periodic, occurring mostly at the start of a service shift). To generate the ticket arrivals, (i) first, we generate a large number of tickets (normal stream) with exponentially distributed inter-arrival times (mean arrival rate $\lambda = \frac{1}{avg.\ ERT}$), (ii) then, generate a sequence of bursts (burst-stream) following Gaussian distribution with independent exponentially distributed inter-arrivals within each burst, (iii) for each experiment, as the number of agents (m) is varied, the inter-arrival times in the normal stream is adjusted uniformly to match the arrival rate of the merged stream (normal-stream merged with burst-stream) with the service rate (y), where $m = y + y^{1/3}$. This makes the overall utilization more aggressive than the well-known square-root staffing approach [11]. Finally, we set maximum delay count (MDC) to 8, to avoid starvation for tickets that breach SLO. To solve the *MIP* formulation, we used Java APIs of ILOG-CPLEX, with a time bound of 3 min. for each of the objectives.

7.2 Results

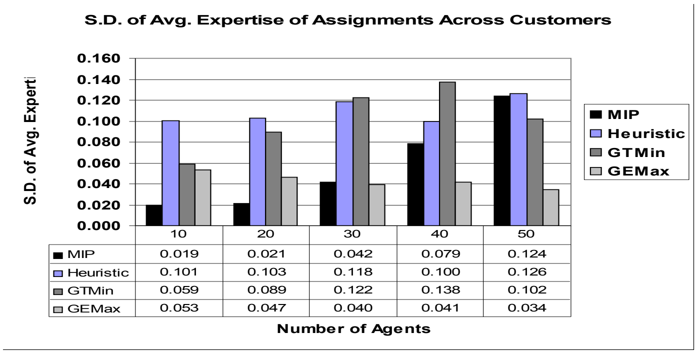
To analyze the efficacy of our scheduling algorithms for shared delivery, we measure a set of metrics in course of our simulation experiments. These metrics follow directly from the scheduling objectives introduced earlier.



	10	20	30	40	50
■ MIP	442	430	1094	1527	1601
□ Heuristic	450	400	474	509	677
▨ GTMin	550	524	596	595	689
□ GEMax	1154	1510	1896	2170	2119

Fig. 1. Total penalty due to SLO breaches beyond SLAs of customer-priority pairs

The first metric (corresponding to SLA Penalty Minimization) is the *total penalty induced by the SLO breaches beyond SLAs*, the results of which are shown in Fig 1 along with the table of values in the bottom. As we see, for the first two experiments (with 10 and 20 agents), *MIP* and *Heuristic* induce the least penalty. *GTMin* causes a penalty increase of 24.4% and 21.8% over *MIP* penalties for the two experiments respectively. The *GEMax* algorithm does not perform well, as it is concerned mainly with the maximization of expertise in assignments. Starting from the third experiment (with 30 agents), we found that *MIP* fails to return any feasible assignment within the allotted time. On deeper investigation, we discovered that due to bursty arrival and queued tickets, the number of tickets to be scheduled in some runs was around 50. This led to a huge increase in the number of internal variables and constraints (>70,000), since these are of the order of $O(MN^2)$ where M and N refer to #agents and #tickets respectively. To circumvent this problem, for *MIP* experiments with (30, 40, 50) agents, we did not consider all the *Queued* tickets at every scheduling run, but fixed many of them at their existing queue positions. This made scheduling tractable, but led to a sharp increase in SLA penalty for *MIP*, as seen in Fig 1. This demonstrates that by usually allowing ticket positions to change in a queue, we are able to save on a lot of SLO breaches and SLA penalty.



	10	20	30	40	50
■ MIP	0.019	0.021	0.042	0.079	0.124
■ Heuristic	0.101	0.103	0.118	0.100	0.126
■ GTMin	0.059	0.089	0.122	0.138	0.102
■ GEMax	0.053	0.047	0.040	0.041	0.034

Fig. 2. Standard deviation of average expertise of assignments of tickets belonging to different customers

Next, we measure the S.D. of agent load at every run (corresponding to the load balancing objective). *MIP*, *heuristic* and *GTMin* all did well on this metric, showing that load is well-balanced by all of them. For example, for 20 agents, the S.D. value for *MIP* ranges from (1.78 to 36.7), and is only slightly higher for the *heuristic* and *GTMin*. Expectedly, *GEMax* returns poor results, with a S.D. range of (4.09 to 631.19) across all the runs for 20 agents. This is because there are a few multi-skilled people with high expertise in our experimental set-up, who attract many tickets due to expertise maximization, irrespective of their current load (which also leads to high SLA penalty, as seen above).

We then compute the S.D. of average expertise of assignments across all customers (Fig. 2). We find that while *GEMax* performs consistently well on the fourth objective, the other approaches show slightly higher S.D., which means that each customer is given a lesser balanced share of the available expertise. We also measured the average expertise of all assignments across all customers, and found that it ranges between [0.73, 0.86] for *GEMax* when aggregated over all runs, while the corresponding values for *GTMin* are [0.59, 0.63]. *MIP* and *heuristic* performed comparably with ranges from [0.60, 0.67] and [0.63, 0.71] respectively.

Finally, we measure the average time taken by various algorithms per run. *MIP* takes comparatively large time (235.8 - 416.9 sec.) per run for all the experiments across different numbers of agents, while all other approaches take negligible time ($<$ 6 seconds at most). It should be noted that for the *MIP*, we accepted whatever feasible solution it provided in cases where optimal solution could not be reached in the bounded time.

8 Discussions

Our experimental results demonstrate the effectiveness of our scheduling approaches for shared service delivery. Compared to *GTMin* which greedily reduces *SLO breaches*, the *MIP* and *heuristic* approaches give comparable or better results for SLA penalty minimization and load balancing. Another highlight of the results is that we are able to balance individual customer interests very well, both in terms of criticality of SLO breaches and the sharing of expertise. We also observe that the MIP-based approach works well for moderate sized problems but its performance degrades with scale, while the heuristic scales very well and can rapidly generate solutions of acceptable quality. Thus a combination of the two approaches can allow us to traverse a large problem space very effectively. In future, we plan to empirically study a service delivery system to determine how well we can estimate ticket resolution effort, agent expertise, and the relationship between the same.

References

1. Application assembly optimization: a distinct approach to global delivery. White Paper, IBM GBS (2010)
2. Deshpande, P.M., Garg, D., Rama Suri, N.: Auction based models for ticket allocation problem in it service delivery industry. In: IEEE Intl. Conf. on Ser. Comp., SCC (2008)
3. Gans, N., et al.: Telephone call centers: Tutorial, review, and research prospects. Manf. & Ser. Op. Mgmt. 5 (2003)
4. Gupta, P., Parija, G.R.: Efficient seat utilization in global it delivery service systems. In: IEEE SCC, pp. 97–103 (2009)
5. Kakal, C.S.: Global shared support service: Leveraging expertise, sharing costs, and deriving value. White Paper, Infosys (May 2005)

6. Karger, D., Stein, C., Wein, J.: Scheduling algorithms. In: Algorithms and Theory of Computation Handbook (2010)
7. Khan, A., et al.: Aim-hi: a framework for request routing in large-scale it global service delivery. IBM J. Res. Dev. 53 (2009)
8. Lawler, E.L., et al.: Sequencing and scheduling: Algorithms and complexity. In: Logistics of Production and Inventory, vol. 4, pp. 445–522. Elsevier (1993)
9. Shao, Q., Chen, Y., Tao, S., Yan, X., Anerousis, N.: Efficient ticket routing by resolution sequence mining. In: KDD (2008)
10. Subbian, K., et al.: Incentive compatible mechanisms for group ticket allocation in software maintenance services. In: APSEC (2007)
11. Wallace, R.B., Whitt, W.: A staffing algorithm for call centers with skill-based routing. Manufacturing & Service Operations Management 7 (2005)
12. Zhou, N., Ma, Q., Ratakonda, K.: Quantitative modeling of communication cost for global service delivery. In: IEEE SCC, pp. 388–395 (2009)