RETRAiN: A REcommendation Tool for Reconfiguration of RetAil BaNk Branch

Rakesh Pimplikar and Sameep Mehta

IBM Research, New Delhi, India
{rakesh.pimplikar,sameepmehta}@in.ibm.com

Abstract. Customers in many developing regions (like India) use physical bank branch as primary and preferred banking channel, resulting in high footfall in the branch. This results in high wait time of customers and high pressure on organization's resources, impacting customer satisfaction (CSAT) as well as employee satisfaction (ESAT) adversely. A naive solution to handle this is to increase the service personnel to cater to the customers. However, this is an unviable alternative because this impacts top and bottom line of the bank. Therefore, organizations are strategically looking for intelligent systems which can help in fine tuning the overall business process to maximize their business objectives while incurring zero or very less investments. Towards this end, we present a system RETRAiN to enable such calibration of various components of bank operations. Based on real time data like waiting customers, service requests, availability of service personnel and business metrics, the system provides recommendations for *reconfiguration* of the operations. The reconfiguration includes selection of *scheduling policy*, *number of service personnel* and *configuration of service personnel.* We present the overall system along with analysis and optimization algorithms for generating the recommendations. To showcase the efficacy and usefulness of our system, we present results based on data collected over a period of four months from multiple branches of a leading bank in India.

Keywords: Applications and Experience, Retail Banking, Services Quality.

1 Introduction

In this paper, we present a framework to optimize the business process in retail banking through use of analytic and optimization techniques. Customers in many developing countries prefer to visit bank branch for their banking needs. Even though various alternate channels like ATM, Internet banking and mobile banking have evolved significantly in the last few years, the bank branch has still retained its position as the primary service delivery channel in many emerging economies. Due to various factors like literacy rate, lack of infrastructure, legacy of public sector banks, lack of trust in e-transactions, the alternate channels have not been adopted widely. While the private banks in India have nearly 35

© Springer-Verlag Berlin Heidelberg 2012

to 40%[1] transactions through alternate channels, this figure is in single digit for the public sector banks where the vast majority of population still bank. This results in high footfall in the bank making it difficult to maintain Customer Satisfaction (CSAT) at an acceptable level. Retail banking is one of the industries where CSAT plays a key role towards retention and growth of customer base. According to a study done by Financial Service Sector (FSS) consultants in our organization, *wait time*, *staff interaction*, *service time*, and *information availability* were discovered as important factors which influence CSAT. Of these *wait time* was found to be the most important factor. Moreover, due to huge volume of customers, the service personnel are also under constant scrutiny of customers and face tremendous pressure (often undue) to be more efficient. This affects the ESAT in an undesirable fashion. Please note that there is causal relationship between ESAT and CSAT, i.e., less pressure on employees will reflect itself in better interaction with customers, which in turn will positively impact CSAT.

A seemingly straightforward solution to the above mentioned challenges is to increase the number of service personnel. This will reduce wait time of customers (thereby increasing CSAT) as well as reduce workload for the service personnel (thereby influencing ESAT). The readers would note that every organization have multiple, sometime conflicting, business metrics. For example, even though CSAT and ESAT is important, the organizations always strive to increase gross profit. Therefore, adding personnel is an unviable solution because the bank has to incur cost for hiring, training, providing seats, procuring computers etc. Therefore, strategically, organizations are looking at intelligent integrated systems which help to meet diverse business objectives, minimize investments and increase Return on Investments (RoI).

In this paper, we provide a transformation framework **RE**commendation **T**ool for **R**econfiguration of Ret**Ai**l Ba**N**k Branch - **RETRAiN**, which generates various recommendations for the administrator or branch manager to reconfigure the branch operations. The framework does not change the business process associated with customer service but aims to optimize the process for various stakeholders. The system takes into account the real time information of customers, resources and service types to generate operational planning recommendations. Specifically, RETRAiN generates recommendations for the following questions: Given the real time mix of customers, service requests and resources[2]-

1. What is a *good* scheduling policy? Goodness depends on the efficacy of policy with respect to the business metrics. Our setting in non-preemptive, i.e., the customer service request is fulfilled in one go by assigned resource.
2. How many resources should be employed? The correct supply of resources helps in matching customer demand.
3. What should be the configuration of resources? Since the resources we consider are people, they display different proficiency towards different services. Therefore, given the demand, the most efficient resources should be chosen. The historical efficiency data is used for this assignment.

[1] This figure was quoted by bank officials during our meetings.

[2] In this article, we use resource and service personnel interchangeably.

Table 1. Comparison with BAU scenario and effect on KPM

Dimension	**Current (BAU)**	**Proposed (RETRAIN)**	**Optimized KPM**
Scheduling Policy	Agnostic to Demand Fixed, FIFO	Demand aware, Dynamic, non-FIFO	Profitability, Wait Time, CSAT
Number of Resources	Fixed or Ad-hoc Change	Computed	ESAT, Small Queue Customer Expectation
Resource Configuration	Universal (for all services) or Dedicated (for single service)	Any Subset of services	Reduction in Service Time

Table 1 highlights the key dimensions which are transformed by RETRAiN, comparison with Business-As-Usual (BAU) scenario and the impact on Key Performance Metrics (KPM) for a retail bank. We briefly describe the data in table.

Scheduling Policy. Currently, all banks use FIFO to schedule and serve customers. While FIFO is a fair policy but it does not lend itself to customer and service differentiation [1]. Given the number of customers coming in branch, the banks are increasingly looking at way to pay more attention to more important customers (High Networth Individuals -HNI) or high profitable services (like Demand Orders) and provide better quality service (low wait times). However, it is not always advisable to use non-FIFO policy. For example, consider the following scenarios:

Scenario 1: Only 10% of customers waiting in branch are HNI.
Scenario 2: 60% of customers waiting in branch are HNI.

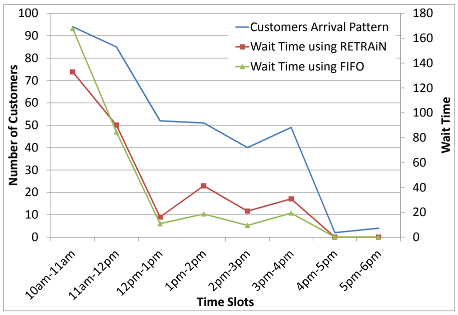
It is clear that a priority based scheduling will work very well in Scenario 1 and help server HNIs in a better fashion. However, in Scenario 2, a FIFO based policy might be better because of large number of HNIs in the branch.

Number of Resources. Currently, the number of resources are fixed in the bank or resources are added after manually observing the queue in the bank. In our system, the choice of addition or removal of a resource is taken by analytically computing the impact of such action on the business metrics.

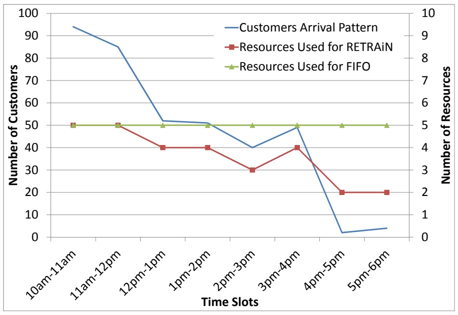
Configuration of Resources. The banks configure the service personnel in two ways either the personnel can provide all services or she provides only one service (like Account Opening or investment advice). However, our system analyzes the current demand of customers while taking into account the individual proficiencies of resources to generate a configuration which allows any subset of services to be assigned to a resource.

To re-iterate, the crux of our framework is to leverage the current customer demand, proficiencies of resources, priority of customers and profitability of services to generate recommendations by employing novel algorithms which will optimize various business metrics.

To further motivate the need and importance of such framework, we present few results derived from data which was collected from multiple branches while deploying some functionality of RETRAiN. Figure 1(a) shows the wait time of customers in each hourly slot for both FIFO as well as RETRAiN. Please note that the wait time is almost equal for both cases. Overall, the FIFO provides a slightly better performance. The wait time using FIFO is 2% less than RETRAiN. However, our system does provide the differentiation between customers by recommending non-FIFO based policies. Now, lets look at the number of resources used as shown in Figure 1(b). The number of resource hours used by our system is 29 whereas in static system the corresponding number is 40. Therefore, our system is able to reduce 25% resource hours while increasing the wait time by a mere 2%. Moreover, in few slots, our system recommended non-FIFO policy which helped in focusing on important customers and profitable services, thereby, optimizing metrics which matter the most.



(a)



(b)

Fig. 1. Comparison of wait time and number of resources

Section 2 describes the related work in this field. Due to real time nature of the system, we chose to develop some polynomial time approximation algorithms to handle the technical challenges. We provide those details in Section 3. Finally, we present results on data in Section 4. The data was collected over a period of four months from multiple branches of a leading bank in India.

2 Related Work

In the proposed system, different (human) resources have different efficiency. The problem of scheduling in such settings is known as unrelated machine problem, which has already been proved to be NP-Hard by [2]. Since then, several approximation algorithms like [2–8] have been proposed for solving this problem. [9] describes scheduling on unrelated parallel machines where every job is associated with a weight. Our system also has a weight/priority assigned to every customer. But unlike the standard problem where number of machines/resources are fixed, number of resources may vary in our system. Apart from finding a *good* schedule, we also need to find *correct* number of resources to be used in the schedule. Moreover since all resources are not equally efficient, we also need to find which resources are best to be used given the real time state of the system. Our schedule depends not only on makespan but also on business metric. Therefore, solutions mentioned in above literature are not suitable for our system. We propose a polynomial time approximation algorithm for this problem. We point interested readers to some excellent surveys on scheduling [10–12].

The work most pertinent to our work is by [13]. The authors describe a hybrid scheduling policy obtained from the integration of real time scheduling algorithms. Based on the characteristics of jobs, hybrid scheduling policy gets reduced to one of the scheduling algorithms. Their choice of real time scheduling algorithms and formulation of hybrid scheduling policy are very much specific to the embedded real time monitor and control system requirements. Hence it cannot be extended for our system. Moreover, working of a scheduling policy is supposed to be unknown to our system. We need to pick the best scheduling policy based on the business metrics returned by various scheduling policies. This objective of our system overrules the idea of having a hybrid policy.

In our previous work [1], we presented a scheduling algorithm for reducing weighted average weight time of customers in a bank branch, while considering prioritization of customers based on several factors. Our system also considers a priority assigned to every customer. It additionally considers a priority assigned to every service as well requested by customers. The focus of the this paper is not on scheduling algorithm. The scheduling algorithm is one component of the our system. Currently, we use algorithm presented in [1], however, the proposed framework allows for replacing it with any other scheduling algorithm.

In commercial offerings, Adobe and IBM solutions for bank branch transformation [14] concentrates mainly on efficient processing of data and documents. It focuses on providing solutions with more secure, personalized, and compelling communications. It helps in setting up different processes in a bank branch ensuring compliance with government regulations. In the similar spirit, Oracle's Siebel Branch Teller solution [15] provides a comprehensive, customer-centric teller solution with the following features. It has 360-degree view of customer relationship which enables more relevant and targeted sales offers (up/cross sell) and improves customer experience. It streamlines transaction processing via an easy-to-use interface. It improves operational efficiencies driven by centralizing business processes and operational information that traditionally exists in each branch server. A branch transformation system by Talaris [16] focuses on maximizing revenue through increased sales of targeted financial products. It also mentions creating a branch experience to retain and develop customer relationships through exceptional service. The tool advises on strategies to position the bank branch as strategic delivery channel and their expertise assists clients to achieve specific outcomes through targeted investment. However, like our system, none of above mentioned bank branch transformation systems enhances customer experience by employing scheduling policy which is different from traditional FIFO scheduling policy.

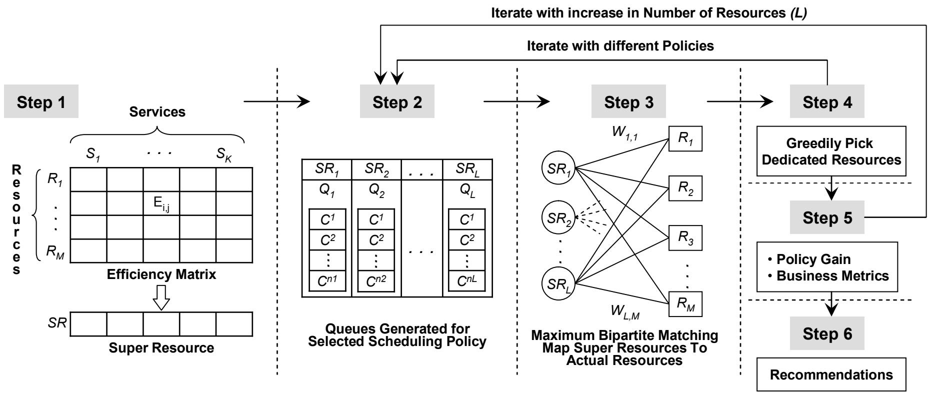

Iterate with increase in Number of Resources (L)
Iterate with different Policies
Step 1
Services
Resources
Efficiency Matrix
Super Resource
Step 2
Queues Generated for Selected Scheduling Policy
Step 3
Maximum Bipartite Matching Map Super Resources To Actual Resources
Step 4
Greedily Pick Dedicated Resources
Step 5
• Policy Gain
• Business Metrics
Step 6
Recommendations

Fig. 2. Depiction of algorithm

Optimizing resource usage for better demand and supply match is not supported by the solutions. Similarly, various Queue Management Systems exist in market to enforce discipline and schedule customers in some predefined order. The solutions generally work on a static fair play principle and schedule customers in a FIFO order. However, any advanced dynamic scheduling (e.g. based on multiple service parameters) and real time decision support system is missing from these products. To best of our knowledge, the proposed framework, associated technical problems and business use cases are novel.

3 Algorithms

Notations. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ denote the customers. Assume $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ be the list of services offered by the organization. Each customer C_i is associated with an arrival time AT_i, service request(s) $G_i \subseteq \mathcal{S}$ and a data packet D_i. The data packet can contain customer specific information like category, number of years with the bank, average quarterly balance etc. Please note that the focus of our current work is to recommend a policy and therefore, we do not delve into data requirements and the rules for individual policies. The assumption is that the abstract data packet would contain all relevant information which is needed by individual scheduling policies. For example, a policy can give more priority to customers with long running accounts. This priority can be calculated by using data packet. The list of scheduling policies is represented by $\mathcal{P} = \{P_1, P_2, \ldots, P_Q\}$. The list of M resources is denoted by $\mathcal{R} = \{R_1, R_2, \ldots, R_M\}$. E is a $M \times K$ matrix capturing the efficiency of resources. $E_{i,j}$ stores the time which R_i takes to serve a request of type S_j. Finally, $\mathcal{B} = \{B_1, B_2, \ldots, B_U\}$ represents the list of business metrics. Each policy P_i will generate a schedule for serving customers and also evaluate the generated schedule with respect to various business metrics. Some metrics which we consider are *wait time*, *wait time per category*, *wait time of important customers*, *wait time of customers with profitable services*, etc.

Policy Evaluation Function. This function generates a schedule Sch_{P_i} using policy P_i, subset of customers $\mathcal{C}'$, subset of resources $\mathcal{R}'$ and corresponding efficiency matrix E'. The definition is: `Sch = evalPolicy(`P_i`, C', R', E')`. If there are L resources, the schedule will have L ranked lists one corresponding to each resource. *Please note that generation of the optimal schedule, in presence of resources with different efficiencies, is a NP-Hard problem and is an area of active research.* However, in our proposed algorithm (Steps 1 and 2), we use this function with equally efficient resources. Therefore, the scheduling reduces to single resource scheduling problem (which can be solved in polynomial time) and schedule generation component simply picks the unserved customer from ranked list and assigns it to a free resource, thereby creating L ranked lists for L resources.

Schedule Evaluation Function. Given a schedule generated by P_k, this function returns values of different business metrics. The definition is `B = evalSch(`Sch_{P_k} `, R', E')`. For each resource, the corresponding ranked list is simulated by taking into account the efficiency of the resource.

Gain Evaluation Function. Given a schedule and two policies P_1 & P_2, the function computes the difference in the values of different business metrics. This difference can be construed as gain $\mathcal{G}$ which a candidate policy P_2 realizes over current in-use policy P_1. The function is implemented as: $\forall_{i=1}^{U}\ G(i) = \frac{B_i^{P_2} - B_i^{P_1}}{B_i^{P_1}}$.

3.1 Algorithms

As noted in related work, the problem of scheduling jobs on known number of unrelated machines is a NP-Hard problem. Our problem becomes much harder because the number of machines (resources) and which resources to be used are also unknown. The choice of resources depends upon the characteristics of services being requested by the customers. Moreover, due to complexity of problem arising from varying efficiency of resources, properties which could help to reduce search space do not hold. For example, P_1 can outperform P_2 on a business metric given L resources, however, with addition of one more resource, the performance may be reversed. Similarly, given customer and service data, assume we use a single resource to serve all customers. Let resource R_1 be the best performer followed by R_2 and R_3. However, the top resources together, i.e., $\{R_1, R_2\}$ can be outperformed by combination of $\{R_2, R_3\}$. This can happen if efficiency of R_2 and R_3 complement each other. We propose a solution which iterates over values of L from 1 to M, while picking best L resources and best scheduling policy every time. We recommend L along with the corresponding resources and policy, which perform the best as per the business metrics. The bottleneck here is to chose best L resources, which is a NP-Hard problem.

NP-Hardness Proof. We already know that it is a NP-Complete problem to decide if all edges of a graph can be covered by exactly $\mathcal{K}$ number of vertices.

The problem of selecting optimal set of resources in current setting can be proved to be NP-Hard by reducing Vertex Cover problem to resource selection problem in polynomial time as follows. Create a resource for every vertex in the graph. Create a customer for every edge in the graph who requires a unique service corresponding to the same edge. A resource provides a service in unit time if the vertex corresponding to the resource is one of the two vertices of the edge corresponding to the service. Otherwise resource requires infinite time to provide a service. Now we solve this resource selection problem to select exactly $\mathcal{K}$ resources so as to minimize the makespan. If the value of makespan is infinity, there is at least one customer who requires a service which is not provided by any of those $\mathcal{K}$ selected resources in unit time. It also indicates that all edges in the graph cannot be covered by exactly $\mathcal{K}$ vertices. If the value of makespan is any finite number, every customer can be served in unit time by one of the $\mathcal{K}$ selected resources. It also indicates that all edges of the graph can be covered by $\mathcal{K}$ vertices corresponding to the $\mathcal{K}$ selected resources. Thus we can use resource selection problem to solve vertex cover problem. Clearly, reduction of vertex cover to resource selection problem takes polynomial time. Hence resource selection problem is harder than vertex cover problem and can be included in NP-Hard category. Even if we are given $\mathcal{K}$ optimal resources, we cannot verify it in polynomial time. We need to enumerate all other combinations of $\mathcal{K}$ resources to verify if the given solution actually results in the smallest makespan.

Figure 2 presents the key steps of the proposed solution.

Step 1. For each service type S_j, we find the resource R_i which takes the least time to provide the service and store the service time, $E_{i,j}$, in SR. Formally, $SR_j = \min\{E_{*,j}\}$. SR can be conceived as a *Super Resource* which provides all services in minimum time possible.

Step 2. In this step, a schedule is generated given a scheduling policy $P_k \in \mathcal{P}$ and L resources where $1 \leq L \leq M$. The point to note is that all L resources are taken to be *super resources*. Moreover, with this setup we can use `evalPolicy` to generate schedule. This construction provides, hypothetically, the best performance (in terms of average wait time) which can be achieved by L resources. In next step we map the super resources to actual resources while incurring an increase in wait time. For a resource SR_i and Policy P_k, the average wait time $WT^{P_k}_{SR_i}$ of assigned customers (Q_i in Step 2 of Figure 2) as per policy P_k is computed by using `evalSch`.

Step 3. In this step we map L super resources to L actual resources while minimizing the increase in wait time. We pose this problem as a maximum bipartite matching with edge weights. Super Resources form one set of vertices while actual resources the other set. The graph is fully connected because every super resource can be replaced by any of the actual resource. The cost of replacing a super resource by actual resource is $C_{i,j} = WT^{P_k}_{R_j}$ - $WT^{P_k}_{SR_i}$. The cost/penalty captures the increase in wait time if SR_i is replaced by R_j. Since the objective is to find maximum weight matching, the weights are computed as $W_{i,j} = max\{C_{*,*}\} - C_{i,j}+1$. We use existing algorithm proposed by [17] to find the matching. Algorithm takes $O(mn \log_{\lceil m/n+1 \rceil} n)$ where n is the number of nodes and m is the number of edges.

If all resources are equally efficient or if we can order them in decreasing order of their efficiencies such that R_i is more efficient than R_j in providing all services for $i < j$, then resources selected in step 3 are nothing but the optimal resources. Step 3 fails to get optimal resources when efficiencies of resources complement each other for different services, for example, $E_{1,1} < E_{2,1}$, but $E_{1,2} > E_{2,2}$. Step 4 provides a greedy solution to fix this problem.

Step 4. Customers are scheduled in Step 2 considering super resources. It results in service requests to be uniformly distributed among all resources. This overrules the inclusion of resources who are extremely efficient in providing one service, but equally bad in providing other services. In optimal solution, these resources might have been selected to provide that one service dedicatedly, resulting in reduction of total wait time of customers. Figure 3 represents an algorithm to greedily select such dedicated resources.

```
Require: $\mathcal{R}^L$, set of $L$ resources selected in Step 3;
    $\mathcal{R}^C$, set of all available resources to be considered;
    $E$, efficiency matrix; $RC_k$, request count for service $S_k$;
    $RS_{i,k}$, number of requests of service $S_k$ to be served by $R_i$;
    $P$, policy in consideration; $\mathcal{C}'$, waiting customers; $WT_{min}$, total wait time using $\mathcal{R}^L$
1: while $\mathcal{R}^C \neq \emptyset$ && $\max_i RC_i > 0$ do
2:     Find a service with maximum requests, $m \leftarrow \text{argmax}_k\ RC_k$
3:     Find the most efficient resource for service $S_m$, $d \leftarrow \text{argmin}_{i \in \mathcal{R}^C}\ E_{i,m}$
4:     $\mathcal{R}^C \leftarrow \mathcal{R}^C - \{R_d\}$
5:     if $R_d \notin \mathcal{R}^L$ then
6:         for all $R_i \in \mathcal{R}^L$ do
7:             Prepare a new set by replacing $R_i$ with $R_d$, $\mathcal{R}^L_{new} \leftarrow \big(\mathcal{R}^L - \{R_i\}\big) \cup \{R_d\}$
8:             $Sch = \texttt{evalPolicy}(P, \mathcal{C}', \mathcal{R}^L_{new}, E)$
9:             Compute total wait time, $WT_i = \texttt{evalSch}(Sch, \mathcal{R}^L_{new}, E)$
10:        end for
11:        if $\min_i WT_i < WT_{min}$ then
12:            Find a resource to be removed from $\mathcal{R}^L$, $n \leftarrow \text{argmin}_i WT_i$
13:            Update minimum total wait time, $WT_{min} \leftarrow WT_n$
14:            Replace $R_n$ with $R_d$ in $\mathcal{R}^L$, $\mathcal{R}^L \leftarrow \big(\mathcal{R}^L - \{R_n\}\big) \cup \{R_d\}$
15:            Decrease request count for $S_m$, $RC_m \leftarrow RC_m - RS_{d,m}$
16:        end if
17:    else
18:        Decrease request count for $S_m$, $RC_m \leftarrow RC_m - RS_{d,m}$
19:    end if
20: end while
```

Fig. 3. Algorithm to select dedicated resources

'IF' condition on line 11 ensures that whenever a resource is replaced by a new dedicate resource on line 14, our solution always results in decreased total wait time and we approach towards the optimal solution. Though we could not prove the approximation bound for our algorithm because of its complexity, it never deviated from optimal solution by a factor more than 2 during our experiments.

Steps 2-4 are repeated by keeping the resources fixed at L and changing the policy. At the end of this iteration, we would have identified L *best* resources for each P_k.

Step 5. In this step we compute the gain $\mathcal{G}$ and other business metrics for each policy and use a rule based system to generate candidate recommendations. Next, we enumerate the rules and also describe intuition behind them:

- *Rule 1.* The Gain ($\mathcal{G}$) over current configuration should be greater than θ_1 where θ_1 defines the improvement which the organization would like to witness. All policies with corresponding gain greater than θ_1 are chosen to generate candidate recommendations. In current deployment, average wait time of all customers is used to compute gain. If gain is too less, it implies that the organization is changing a business process with a new one without substantial improvements.
- *Rule 2.* RETRAiN generates configuration recommendations after every F minutes. Given policy (selected by Rule 1) and set of resources, we compute how many customers can be served in next F minutes. Subtracting this from the total waiting customers C' (used in Policy Evaluation), gives the number of unserved customers UC at the end of next F minutes. If $\frac{UC}{C'} \leq \theta_2$, then the configuration gets added to candidate recommendations.

The configurations which satisfy both rules are tagged as candidate recommendations with the following details $\{P_k, \mathcal{C}', L, \mathcal{R}', E', \mathcal{G}, \frac{UC}{C'}\}$. If no configuration satisfies rules, then number of resources is increased by 1 and Step 2 is repeated. The process is continued till $L \leq M$.

Step 6. All the candidate recommendations are presented to the administrator to choose from. The chosen recommendation then replaces the current configuration and is used for next F minutes. At this point, we would like to share the key motivation of involving domain expert as opposed to the completely automated system. Consider an example where recommendation CR_1 results in gain of 5% whereas CR_2 shows gain of 4.9%. An automated system will choose CR_1 as new configuration. However, the domain expert can investigate respective policies and conclude that the policy used in CR_2 enforces fairness whereas CR_1 gives high preference to a set of customers. In this case, based on business logic and other real time factors, she may decide to choose CR_2. Moreover, in such scenarios, the rules itself keep changing based on state of the system including number of customers, number of available resources, time of month, special promotion season etc. Therefore, the same expert can take different decisions based on real time state of the physical system. It would be very challenging to encode and prioritize full expert knowledge base with different permutations of system state.

4 Results

The experiments are conducted on real data collected during (partial) deployment of RETRAiN at multiple branches of a leading bank in India. Over the deployment period, our system scheduled around 25000 customers. Category 2 (important customers) accounted for around 60% of customers whereas around 1500 customers were in Category 1 (most important customers). Out of 30000 service requests, around 75% were deemed to have positive value for the bank. For each branch we collected one week of data with fixed resources and FIFO policy. This data was used to learn the model and also baseline various metrics.

We demonstrate the working of our algorithm on different scenarios to showcase the efficacy of the proposed solution. We consider four different categories (priorities) of customers, denoted as $Cat_1, \ldots, Cat_4$. These categories scaled by appropriate weight vector are used in ranking customers with Weighted Shortest Job First (WSJF) policy. Different weighing vectors result in different policies. For example, weight vector (0.5, 1, 1, 1) suggests that Cat_1 customer is twice more important than any other category. Similarly, vector (0, 1, 1, 1) implies that Cat_1 customers should be served as soon as a resource is free by pushing it ahead of all other customers. Another weighing vector (1, 1 ,1, 5) captures that all customers except Cat_4 are equal and Cat_4 would have to wait much longer. Similarly we consider five different service categories as $S_1, \ldots, S_5$. For both customer and service categories, lower the category id, higher is the importance. There are five resources in our setup with different proficiencies for different services as mentioned in Table 2. Every cell represents the average time (in seconds) required by corresponding resource to provide the corresponding service. We computed the efficiency matrix from the collected data.

Table 2. Efficiency matrix

Resources\Services	S_1	S_2	S_3	S_4	S_5
R_1	258.17	275.49	250.86	180.54	236.64
R_2	199.70	201.89	185.73	145.60	191.91
R_3	312.95	548.07	309.27	189.34	317.12
R_4	453.57	300.62	123.95	240.66	253.90
R_5	308.67	312.56	220.17	168.18	135.09

Key Results. The highlights of our deployment are:

- The wait time of Cat 2 customers reduced by 30% while the most important customers (Cat 1) experienced a wait time reduction of 83%. This can be attributed to non-FIFO policies. Due to large chunk of customers (65%) in these two categories, the overall wait time over all customers went down as compared to pure FIFO policy.

- The wait time of least important customers (10% by volume) increased by 25%.
- For the above mentioned results, the number of resources were kept fixed (same as in baseline data). When the number of resources and their configuration was optimized, we found that by using ≈ 25% less resources we can maintain same wait time (within ± 2%) as in baseline data.
- We conducted an informal survey of customers to get their feedback on the system. Around 70% customers felt reduction in wait time. Around 19% customers had difficulty to understand the new system. Overall 82% customers felt system has made a positive impact and thereby increasing CSAT.
- Informally, the bank staff also acknowledged the impact of the system. The support was exemplary.

Next, we show some of the expository results. We consider FIFO as the baseline policy. Different variants of WSJF constitute the policy bank. The experiments use RETRAiN frequency F as 30 minutes. From real data we observed that during peak hours, around 60 customers come to branch in an hour. Therefore, in our experiments we use 30 customers. Please note the arrival frequency changes through the day. However, we choose the peak period because RETRAiN is motivated to help banks in peak periods. We set $\theta_1 = 0.1$, i.e., a policy with improvement of at least 10% over FIFO should be selected for the current time slot.

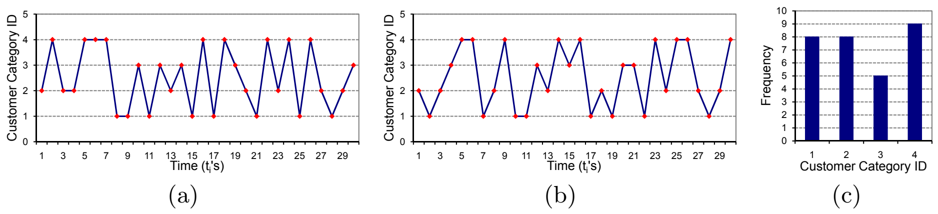


Fig. 4. Uniform ordering, (a) WSJF recommended (b) FIFO recommended (c) Customer category distribution

In our first setting we study the effect of arrival patterns of customer categories on policy selection. Figure 4(a) shows a pattern where customer categories are uniformly ordered and improvement of WSJF over FIFO is computed to be 21% ($> \theta_1$). Therefore, WSJF policy is selected as a candidate recommendation. Figure 4(c) shows the histogram of different customer categories in the pattern. Figure 4(b) shows a different arrival pattern for the same frequencies of categories. However, improvement of WSJF over FIFO in this case is very less, 9.3% ($< \theta_1$). Therefore, FIFO is selected. It is evident that policy selection depends on the arrival pattern of customer categories.

Consider a special case as shown in Figure 5 where there are maximum customers of the same category Cat_2 and others are distributed among other categories. Table 3 shows the two different WSJF and FIFO numbers (in seconds).

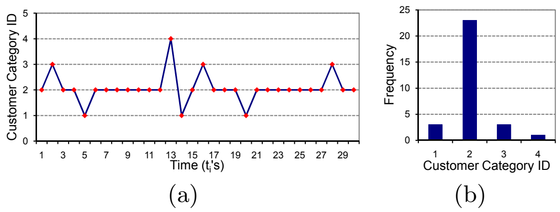


Fig. 5. Skewed distribution

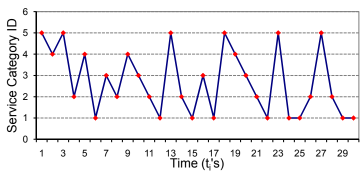
Table 3. Comparison of different policies

Policy	All Customers	Cat_1	Cat_2	Cat_3	Cat_4
FIFO	309.1	309.1	309.1	309.1	309.1
WSJF$_1$	304.70	27.89	309.73	395.60	848
WSJF$_2$	309.95	235.07	304.27	384.34	511

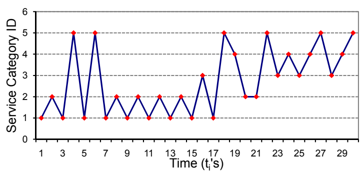
Since FIFO does not differentiate between customers, average wait time over all customers is taken to be wait time in each category as well. WSJF$_1$ drastically improves the wait time of Cat_1 customers, however, customers in Cat_4 are penalized heavily. Moreover, the improvement in overall wait time (over FIFO) is marginal. This small improvement for Cat_1 may not justify heavy increase in the wait time for other customers. Moreover, in such scenarios, the fairness provided by FIFO also plays a role in the decision. However, WSJF$_2$ provides a viable alternative where wait time of important customers Cat_1 decreases (by 23%) with small increase for Cat_4 customers. Based on domain knowledge, the admin can choose between FIFO or WSJF$_2$. Wait time of Cat_2 remains almost unchanged in all three settings.

In an another setting, we study the effect of arrival patterns of service requests on number of resources. Figure 6(a) and Figure 6(b) show two different arrival patterns of service categories but with same frequencies as depicted in Figure 6(c). In Figure 6(a) all services are uniformly ordered over the current time slot and three best resources are suggested to be $\{R_1, R_2, R_5\}$. Proficiency matrix mentioned in Table 2 is used in selecting these resources. If we observe this matrix, services S_1 and S_2 are most time consuming. Clearly if these two services are clustered in the earlier part of the slot as shown in Figure 6(b), then overall wait time of all customers increases. To reduce this wait time, system is reconfigured and 4 resources are suggested as $\{R_1, R_2, R_4, R_5\}$ instead of just 3 in previous case. Thus order of services and time required to process them are important factors to suggest *correct* resources.

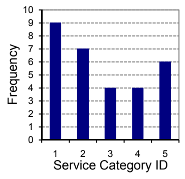
As noted in algorithm, we select minimum number of resources where $\frac{UC}{C'} \leq \theta_2$. In our setup, we have $\theta_2 = 0.1$. For a pattern shown in Figure 6(a) where three resources are suggested, unserved customers are just 3 out of 30 ($\frac{UC}{C'} = 0.1$) and average wait time is 313s. Now if we add one more resource, the average wait time decreases to 205s which is good for CSAT, but at the end of the slot all four resources remain idle for 5 to 10 minutes. To avoid this under utilization, θ_2 plays an important role.



(a)



(b)

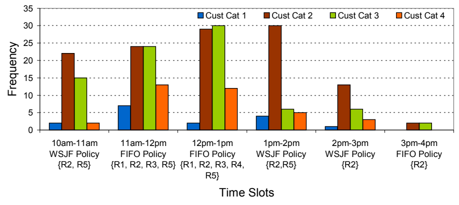


(c)

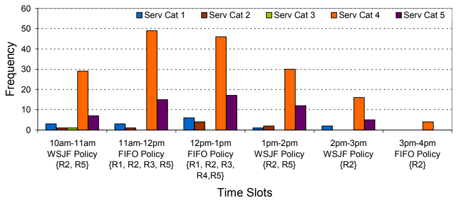
Fig. 6. (a) Uniform ordering, three resources recommended (b) Non-uniform ordering, four resources recommended (c) Service category distribution

We also present results of our system when run for an entire day over real time data. Figure 7 shows recommendations given by our system for different time slots of a day in a branch of a leading bank in India. RETRAiN frequency F is set to be 1 hour. Horizontal axis shows slot durations along with recommended policies and selected resources. Figure 7(a) shows distributions of customer categories for different time slots, while Figure 7(b) shows distributions of service categories for corresponding times slots. Initially, when customers are small in number in first slot, only two resources are selected. As we can observe services S_3 and S_4 are predominantly required by customers in first slot. So resources R_2 and R_5 are selected, because they are more efficient than others in providing S_3 and S_4, which is clear from efficiency matrix shown in Table 2. WSJF policy is recommended by our system for the first slot, as we are getting improvement greater than θ_1 over FIFO. FIFO is recommended for second and third slots, because WSJF has smaller improvement over FIFO in these slots. 4 and 5 resources are selected respectively for these slots, because our system observed increased in number of customers. When branch load decreases in later slots, resources are removed and appropriate policies are recommended. You can observe in Figure 7(b) that service S_4 is having high demand through out the day. So resource R_2 is selected for all time slots, as he is the most efficient in providing service S_4.

Thus resources are better managed by our system and now there is no need for all 5 resources to be engaged for all the slots. This directly leads to reduction of human hours from 30 to just 14 (53%).



(a) Distribution of customer categories



(b) Distribution of service categories

Fig. 7. Recommendations for entire day

Algorithmic Complexity and Timing Results. Scheduling customers using a policy with a single resource is typically a sorting task. The complexity is $O(N \log N)$. Generation of actual schedule by assigning customers to super resources is $O(N \log L)$ where $1 \leq L \leq M$. Therefore, the overall complexity of generating schedule for super resources is $O(N \log N) + O(N \log M)$. Finally, we create a bipartite graph which can have maximum $M * M$ edges and $2 * M$ nodes (if $L = M$). So, complexity of finding maximum bipartite matching is $O(M^3)$ using [17]. The steps are repeated for every policy (1 to Q) and number of resources (1 to M). So overall complexity of running our algorithm is dominated by $O(Q \times M \times M^3)$. With $Q = 10$ (number of candidate policies), 90 customers and 7 resources, our system generates recommendations in approximately 2 seconds. Due to the limit on number of resources which the branch can accommodate (typically 3 to 7), M^4 is manageable. The time increases linearly with the increase in number of policies.

5 Conclusions and Future Work

In this paper, we presented an integrated system RETRAiN which analyzes the real time mix of customers, service requests and resources and recommends a good configuration for optimizing the retail bank branch operations. The system tries to use minimum resources and strives to improve business metrics. We presented an approximation algorithm which discovers how many and which resources should be used. We presented some results on real data collected from a leading bank in India. Currently, we are conducting more experiments to study the quality of our algorithm vis-a-vis optimal algorithm. The problem is modeled as math program and solved using existing solver for optimal recommendation. Solving math program for large number of instances and comparing it with our algorithm will enable us to perform a gap analysis. Finally, we are also exploring the possibility of deploying the complete RETRAiN system in live customer environment as well as other domains like call centers.

References

1. Mehta, S., Chafle, G., Parija, G.R., Kedia, V.: A system for providing differentiated qos in retail banking. In: IJCAI, pp. 2494–2499 (2011)
2. Lenstra, J.K., Shmoys, D.B., Tardos, É.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. 46, 259–271 (1990)
3. Martello, S., Soumis, F., Toth, P.: Exact and approximation algorithms for makespan minimization on unrelated parallel machines. Discrete Applied Mathematics 75(2), 169–188 (1997)
4. Jansen, K., Porkolab, L.: Improved approximation schemes for scheduling unrelated parallel machines. In: Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, STOC 1999, pp. 408–417. ACM, New York (1999)
5. Efraimidis, Spirakis: Randomized approximation schemes for scheduling unrelated parallel machines. In: ECCCTR: Electronic Colloquium on Computational Complexity, technical reports (2000)

6. Efraimidis, Spirakis: Approximation schemes for scheduling and covering on unrelated machines. TCS: Theoretical Computer Science 359 (2006)
7. Gairing, M., Monien, B., Woclaw, A.: A faster combinatorial approximation algorithm for scheduling unrelated parallel machines. Theor. Comput. Sci. 380(1-2), 87–99 (2007)
8. Verschae, J., Wiese, A.: On the Configuration-LP for Scheduling on Unrelated Machines. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 530–542. Springer, Heidelberg (2011)
9. Chudak, F.A.: A min-sum 3/2-approximation algorithm for scheduling unrelated parallel machines. Journal of Scheduling 2, 73–77 (1999)
10. Sgall, J.: On-line Scheduling. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms 1996. LNCS, vol. 1442, pp. 196–231. Springer, Heidelberg (1998)
11. Vazirani, V.: Approximation Algorithms. Springer (2001)
12. Karger, D., Stein, C., Wein, J.: Scheduling algorithms. In: Handbook of Algorithms and Theory of Computation. CRC Press (2010)
13. Deng, Q., Lv, M., Yu, G.: Selecting a Scheduling Policy for Embedded Real-Time Monitor and Control Systems. In: Wu, Z., Chen, C., Guo, M., Bu, J. (eds.) ICESS 2004. LNCS, vol. 3605, pp. 494–501. Springer, Heidelberg (2005)
14. Adobe, IBM: Solutions for bank branch transformation, `http://www.adobe.com/enterprise/partners/ibm/banking.html`
15. Oracle: Siebel branch teller, `http://www.oracle.com/us/industries/financial-services/046715.html`
16. Talaris: Branch transformation, `http://www.talaris.com/en-gb/solutions/talaris-consulting/branch-transformation.aspx`
17. Galil, Z.: Efficient algorithms for finding maximum matching in graphs. ACM Computing Surveys 18(1), 23 (1986)