Recommending Resolutions for Problems Identified by Monitoring

Liang Tang and Tao Li
School of Computer Science
Florida International University
Miami, FL, USA
Email: {ltang002, taoli}@cs.fiu.edu

Larisa Shwartz
IBM T.J. Watson Research Center
Hawthorne, NY, USA
Email: lshwart@us.ibm.com

Genady Grabarnik
Dept. Math & Computer Science
St. John's University
Queens, NY, USA
Email: grabarng@stjohns.edu

***Abstract*—**

Service Providers are facing an increasingly intense competitive landscape and growing industry requirements. Modern service infrastructure management focuses on the development of methodologies and tools for improving the efficiency and quality of service. It is desirable to run a service in a fully automated operation environment. Automated problem resolution, however, is difficult. It is particularly difficult for the weakly-coupled service composition, since the coupling is not defined at design time. Monitoring software systems are designed to actively capture events and automatically generate incident tickets or event tickets. Repeating events generate similar event tickets, which in turn have a vast number of repeated problem resolutions likely to be found in earlier tickets. We apply a recommendation systems approach to resolution of event tickets. In addition, we extend the recommendation methodology to take into account possible falsity of some of the tickets. The paper presents an analysis of the historical event tickets from a large service provider and proposes two resolution-recommendation algorithms for event tickets utilizing historical tickets. The recommendation algorithms take into account false positives often generated by monitoring systems. An additional penalty is incorporated in the algorithms to control the number of misleading resolutions in the recommendation results. An extensive empirical evaluation on three ticket data sets demonstrates that our proposed algorithms achieve a high accuracy with a small percentage of misleading results.

I. Introduction

Today's competitive business climate, and the complexity of service environments, dictate efficient and cost-effective service delivery and support. This is largely achieved through service-providing facilities to collaborate with system management tools, combined with automation of routine maintenance procedures including problem detection, determination and resolution for the service infrastructure [18], [25], [6], [29], [30]. Automatic problem detection is typically realized by system monitoring software, such as IBM Tivoli Monitoring [3] and HP OpenView [1]. It actively captures the events from service infrastructures and generates incident tickets when certain alerts are detected. It is desirable to run a service in a fully automated operation environment, and employment of monitoring solutions is a first step towards achieving this goal. Automated problem resolution, however, is difficult. It is particularly difficult for the weekly-coupled service composition , since the coupling is not defined at design time. Traditionally Service Providers rely heavily on human manpower for such tasks as root cause analysis and resolution of incidents.

With the development of e-commerce, a substantial amount of research has been devoted to the recommendation systems. These recommendation systems determine items or products to be recommended based on prior behavior of the user or similar users and on the item itself. An increasing amount of user interactions has provided these applications with a large amount of information that can be converted into knowledge. In this paper we apply this approach to the resolution of incident tickets for maintaining service infrastructures. In addition, we extend the recommendation methodology to take into account possible falsity of the tickets. We focus on the event tickets, which are incident tickets generated by monitoring systems. We believe our work can help service providers to efficiently find appropriate problem resolutions and correlate related tickets resolved in the past. Most service providers keep track of a large amount of historical tickets with resolutions. The resolution is usually stored as a plain text which describes how this ticketed incident has been resolved. We analyzed historical event tickets collected from three different accounts managed by IBM Global Services. We consider an account as an aggregate of services using a common infrastructure. One observation is that many event tickets share the same resolutions. If two events are similar, then their triggered tickets probably have the same resolution. Therefore, we can recommend a resolution for an incoming ticket based on the event information and historical tickets.

In this paper, we propose two resolution recommendation algorithms for event tickets. An additional penalty is incorporated in the algorithms to minimize misleading resolutions. The contribution of this paper can be summarized as follows:

- We analyze historical event tickets from three production accounts and observe that their resolutions are recommendable for current event tickets on the basis of event information.
- We propose two resolution recommendation algorithms for event tickets capable of eliminating misleading resolutions.
- We conducted extensive experiments for our proposed algorithms on real ticket data. Their results show that our algorithms achieve the same accuracy as the traditional k-nearest neighbor (KNN) based algorithms [22] [24] with less misleading information in the results.

The rest of the paper is organized as follows: Section II briefly introduces the work flow of the infrastructure management of an automated service and shares our observations on three sets of event tickets. In Section III, we present our resolution recommendation algorithms for event tickets. Section IV discusses some detailed implementation issues. In Section V, we present experimental studies on real event tickets. Section VI describes related work about the automated service's infrastructure management and the recommendation system. Finally, Section VII concludes our paper and discusses the future work.

II. Background

In this section, we first provide an overview of automated service' infrastructure monitoring with ticket generation and resolution. Then we present our analysis on real ticket data sets.

A. *Automated Service's Infrastructure Monitoring and Event Tickets*

The typical workflow of problem detection, determination, and resolution in service's infrastructure management is prescribed by the ITIL specification [4]. Problem detection is usually provided by monitoring software which computes metrics for the hardware and software performance at regular intervals. The metrics are then matched against acceptable thresholds, a violation induces an alert, and if the latter persists beyond a specified period, the monitor emits an event. Events from the entire service infrastructure are accumulated in an enterprise console which uses rule-, case- or knowledge-based engines to analyze the monitoring events and decide whether to open an incident ticket in the ticketing system. The incident tickets created from the monitoring events are called event tickets. Additional tickets are created upon customer requests. The information accumulated in the ticket is used by the technical support for problem determination and resolution. In this paper, we consider tickets generated by event monitoring of Service Platform layer and Physical Layer see Figure 1 .

Each event is stored as a database record which consists of several related attributes with values describing the system status at the time when this event was generated. For example, a CPU related event usually contains the CPU utilization and paging utilization information. A capacity-related event usually contains the disk name, and the size of disk used/free space. Typically, different types of events have different sets of related attributes. The problem resolution of every ticket is stored as a textual description of the steps taken by system administrator to resolve this problem. A false ticket contains comments stating falsity and possibly the reason for it and how to identify it.

B. *Repeated Resolutions of Event Tickets*

We analyzed ticket data from three different accounts managed by IBM Global Services. One observation is that many ticket resolutions repeatedly appear in the ticket database. For example, for a low disk capacity ticket, usual resolutions are deletion of temporal files, backup data, or addition of a new disk. Unusual resolutions are very rare.

The collected ticket sets from the three accounts are denoted by "account1", "account2" and "account3" respectively.

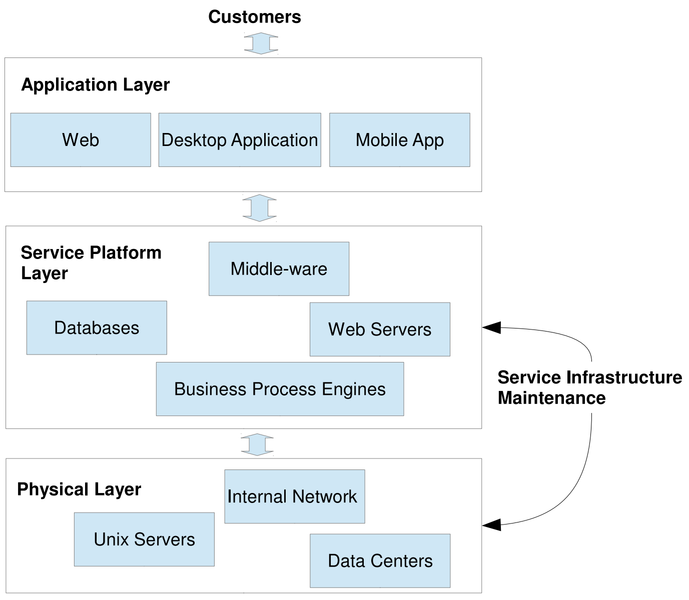


Fig. 1: Service Maintenance

TABLE I: Data Summary

Data set	Num. of Servers	Num. of Tickets	Time Frame
account1	1,145	50,377	55 days
account2	614	6,121	29 days
account2	391	4,066	48 days

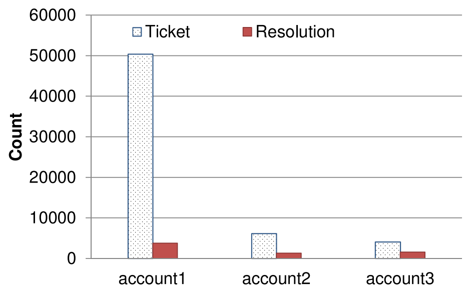


Fig. 2: Numbers of Tickets and Distinct Resolutions

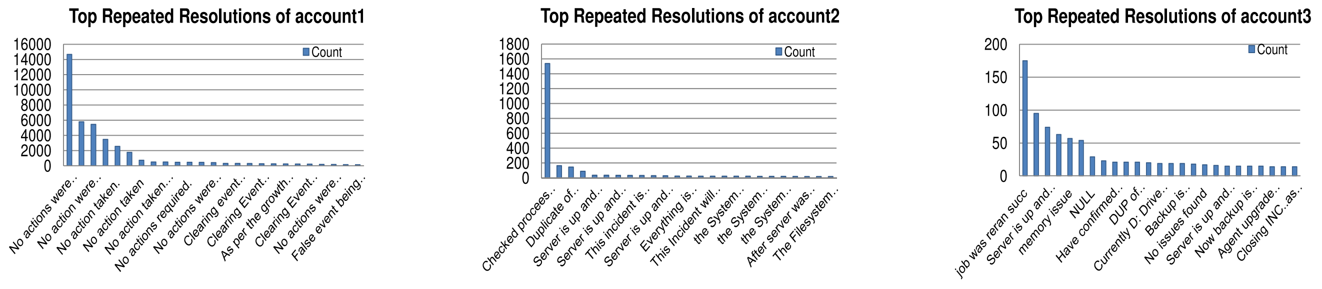
Table I summarizes the three data sets. Figure 2 shows the numbers of tickets and distinct resolutions and Figure 3 shows the top repeated resolutions in each data set. It is seen that the number of distinct resolutions is much smaller than the number of tickets - in other words, multiple tickets share the same resolutions. For example (Figure 3a) the first resolution, "No actions were...", appears more than 14000 times in "account1".

III. Resolution Recommendation

In this section, we first introduce the basic KNN-based recommendation algorithm, and then present our improved algorithms.

A. *Basic KNN-based Recommendation*

Given an incoming event ticket, the objective of the resolution recommendation is to find k resolutions as close as



(a) account1 (b) account2 (c) account3

Fig. 3: Top Repeated Resolutions for Event Tickets

possible to the the true one for some user-specified parameter k. The recommendation problem is often related to that of predicting the top k possible resolutions. A straightforward approach is to apply the KNN algorithm which searches the K nearest neighbors of the given ticket (K is a predefined parameter), and recommends the top $k \leq K$ representative resolutions among them [22], [24]. The nearest neighbors are indicated by similarities of the associated events of the tickets. In this paper, the representativeness is measured by the number of occurrences in the K neighbors.

TABLE II: Notations

Notation	Description
D	Set of historical tickets
$\lvert \cdot \rvert$	Size of set
t_i	i-th event ticket
$r(t_i)$	Resolution description of t_i
$e(t_i)$	Associate event of t_i
$c(t_i)$	Type of ticket t_i, $c(t_i) = 1$ indicates t_i is a real ticket,$c(t_i) = 0$ indicates t_i is a false ticket.
$A(e)$	Set of attributes of event e
$sim(e_1, e_2)$	Similarity of events e_1 and e_2
$sim_a(e_1, e_2)$	Similarity of a values of event e_1 and e_2
K	Number of nearest neighbors in the KNN algorithm
k	Number of recommended resolutions for a ticket, $k \leq K$

Table II lists the notations used in this paper. Let $D = \{t_1, ..., t_n\}$ be the set of historical event tickets and t_i be the i-th ticket in D, $i = 1, ..., n$. Let $r(t_i)$ denote the resolution description of t_i, $e(t_i)$ is the associated event of t_i. Given an event ticket t, the nearest neighbor of t is the ticket t_i which maximizes $sim(e(t), e(t_i))$, $t_i \in D$, where $sim(\cdot, \cdot)$ is a similarity function for events. Each event consists of event attributes with values. Let $A(e)$ denote the set of attributes of event e. The similarity for events is computed as the summation of the similarities for all attributes. There are three types of event attributes: categorical, numeric and textual (shown by Table III). Given an attribute a and two events e_1

TABLE III: Event Attribute Types

Type	Example
Categorical	host name, process name, ...
Numeric	CPU utilization, disk free space percentage, ...
Textual	event message,...

and e_2, $a \in A(e_1)$ and $a \in A(e_2)$, the values of a in e_1 and e_2 are denoted by $a(e_1)$ and $a(e_2)$. The similarity of e_1 and e_2 with respect to a is

$$sim_a(e_1, e_2) = \begin{cases} I[a(e_1) = a(e_2)], & \text{if } a \text{ is categorical,} \\ \frac{|a(e_1) - a(e_2)|}{max|a(e_i) - a(e_j)|}, & \text{if } a \text{ is numeric,} \\ Jaccard(a(e_1), a(e_2)), & \text{if } a \text{ is textual,} \end{cases}$$

where $I(\cdot)$ is the indicator function returning 1 if the input condition holds, and 0 otherwise. Let $max|a(e_i) - a(e_j)|$ be the size of the value range of a. $Jaccard(\cdot, \cdot)$ is the Jaccard index for *bag of words model* [21], frequently used to compute the similarity of two texts. Its value is the proportion of common words in the two texts. Note that for any type of attribute, inequality $0 \leq sim_a(e_1, e_2) \leq 1$ holds. Then, the similarity for two events e_1 and e_2 is computed as

$$sim(e_1, e_2) = \frac{\sum_{a \in A(e_1) \cap A(e_2)} sim_a(e_1, e_2)}{|A(e_1) \cup A(e_2)|}. \qquad (1)$$

Clearly, $0 \leq sim(e_1, e_2) \leq 1$. To identify the type of attribute a, we only need to scan all appearing values of a. If all values are composed of digits and a dot, a is numeric. If some value of a contains a sentence or phrase, then a is textual. Otherwise, a is categorical.

B. Division Method

Traditional recommendation algorithms focus on the accuracy of the recommended results. However, in automated service management, false alarms are unavoidable in both the historical and incoming tickets [25]. The resolutions of false tickets are short comments such as "this is a false alarm", "everything is fine" and "no problem found". If we recommend a false ticket's resolution for a real ticket, it would cause the system administrator to overlook the real system problem. and besides, none of the information in this resolution is helpful. Note that in a large enterprise IT environment, overlooking a real system problem may have serious consequences such as system crashes. Therefore, we consider incorporation of penalties in the recommendation results. There are two cases meriting a penalty: recommendation of a false ticket's resolution for a real ticket, and recommendation of a real ticket's resolution for a false ticket. The penalty in the first case should be larger since the real ticket is more important. The two cases are analogous to the *false negative* and *false positive* in prediction problems [24], but note that our recommendation target is the ticket resolution, not its type. A false ticket's event may also have a high similarity with that of a real one. The objective of the recommendation algorithm is now maximized accuracy under minimized penalty.

A straightforward solution consists in dividing all historical tickets into two sets comprising the real and false tickets respectively. Then, it builds a KNN-based recommender for each set respectively. Another ticket type predictor is created, establishing whether an incoming ticket is real or false, with the appropriate recommender used accordingly. The divide method works as follows: it first uses a type predictor to predict whether the incoming ticket is real or false. If it is real, then it recommends the tickets from the real historic tickets; if it is false, it recommends the tickets from the false historic tickets. The historic tickets are already processed by the system admin, so their types are known and we do not have to predict them.

The division method is simple, but relies heavily on the precision of the ticket type predictor which cannot be perfect. If the ticket type prediction is correct, there will be no penalty for any recommendation result. If the ticket type prediction is wrong, every recommended resolution will incur a penalty. For example, if the incoming ticket is real, but the predictor says it is a false ticket, so this method only recommends false tickets. As a result, all the recommendations would incur penalties.

C. Probabilistic Fusion Method

To overcome the limitation of the division method, we develop a probabilistic fusion method. The framework of the basic KNN-based recommendation is retained, with difference that, the penalty and probability distribution of the ticket type are incorporated in the similarity function.

Let λ be the penalty for recommending a false ticket's resolution for a real ticket, and $1-\lambda$ that for recommending a real ticket's resolution for a false one. λ can be specified by the system administrator based on the actual cost of missing a real alert, $0 \leq \lambda \leq 1$. Larger λ indicates a greater importance of real tickets. The penalty function is

$$\lambda_t(t_i) = \begin{cases} \lambda, & t \text{ is a real ticket, } t_i \text{ is a false ticket} \\ 1-\lambda, & t \text{ is a false ticket, } t_i \text{ is a real ticket} \\ 0, & \text{otherwise,} \end{cases}$$

where t is the incoming ticket and t_i is the historical one whose resolution is recommended for t. Conversely, an award function can be defined as $f_t(t_i) = 1 - \lambda_t(t_i)$. Since $0 \leq \lambda_t(t_i) \leq 1$, $0 \leq f_t(t_i) \leq 1$.

Let $c(\cdot)$ denote the ticket type. $c(t_i) = 1$ indicates t_i is a real ticket; $c(t_i) = 0$ indicates t_i is a false ticket. Since t is an incoming ticket, the value of $c(t)$ is not known. Using a ticket type predictor, we can estimate the distribution of $P[c(t)]$. The idea of this method is to incorporate the expected award in the similarity function. The new similarity function $sim'(\cdot,\cdot)$ is defined as:

$$sim'(e(t), e(t_i)) = E[f_t(t_i)] \cdot sim(e(t), e(t_i)), \qquad (2)$$

where $sim(\cdot,\cdot)$ is the original similarity function defined by Eq. (1), and $E[f_t(t_i)]$ is the expected award, $E[f_t(t_i)] = 1 - E[\lambda_t(t_i)]$. If t_i and t have the same ticket type then $E[f_t(t_i)] = 1$ and $sim'(e(t), e(t_i)) = sim(e(t), e(t_i))$, otherwise $sim'(e(t), e(t_i)) < sim(e(t), e(t_i))$. Generally, the expected award is computed as

$$\begin{aligned} E[f_t(t_i)] &= E[1 - \lambda_t(t_i)] = 1 - E[\lambda_t(t_i)] \\ &= 1 - \sum_{c(t), c(t_i) \in 0,1} P[c(t), c(t_i)] \lambda_t(t_i). \end{aligned}$$

We can assume that a new ticket t and historical ticket t_i are independent, i.e., $P[c(t), c(t_i)] = P[c(t)] \cdot P[c(t_i)]$. Then, the expected penalty is

$$E[\lambda_t(t_i)] = \sum_{c(t), c(t_i) \in 0,1} P[c(t)] \cdot P[c(t_i)] \cdot \lambda_t(t_i).$$

Since $c(t_i)$ is already fixed, substituting $\lambda_t(t_i)$, we obtain

$$E[\lambda_t(t_i)] = \begin{cases} P[c(t) = 0] \cdot (1 - \lambda), & t_i \text{ is a real ticket} \\ P[c(t) = 1] \cdot \lambda, & t_i \text{ is a false ticket} \end{cases}$$

Note that all factors in the new similarity function are of the same scale, i.e., $[0, 1]$, thus $0 \leq sim'(\cdot,\cdot) \leq 1$.

D. Prediction of Ticket Type

Given an incoming ticket t, the probabilistic fusion method needs to estimate the distribution of $P[c(t)]$. The dividing method also has to predict whether t is a real ticket or a false ticket. There are many binary classification algorithms for estimating $P[c(t)]$. In our implementation, we utilize another KNN classifier. The features are the event attributes and the classification label is the ticket type. The KNN classifier first finds the K nearest tickets in D, denoted as $D_K = \{t_{j_1}, ..., t_{j_k}\}$. Then, $P[c(t) = 1]$ is the proportion of real tickets in D_K and $P[c(t) = 0]$ is the proportion of false tickets in D_K. Formally,

$$\begin{aligned} P[c(t) = 1] &= |\{t_j | t_j \in D_K, c(t_j) = 1\}| / K \\ P[c(t) = 0] &= 1 - P[c(t) = 1]. \end{aligned}$$

IV. Implementation

In this section, we discuss several issues in implementing the resolution recommendation system.

A. Redundancy Removal in Recommendation

KNN-based recommendation algorithms recommend the top k representative resolutions in the K nearest tickets. However, since all of these are similar to the incoming ticket, the resolutions of the K tickets may also be similar to each other, so that there may be some redundancy in the recommended results. To avoid this, another validation step is applied. First, the K nearest tickets' resolutions are sorted according to their representativeness in descending order. Then, we go through all K resolutions, and check whether or not each of them is redundant to any previously selected resolution. If it is, we skip this resolution and jump to the next one; otherwise, we add it to the selection. Since the resolutions are textual descriptions, the redundancy of two resolutions is measured by the Jaccard index, $Jaccard(\cdot,\cdot)$, introduced in Section III-A. In practice, if the descriptions of two resolutions $r(t_1)$ and $r(t_2)$ have more than one half common words (i.e. $Jaccard(r(t_1), r(t_2)) > 0.5$), the two resolutions are quite likely to be the same.

B. Finding Nearest Neighbors

Finding the K nearest neighbors in a large collection of historical tickets is time-consuming. There are many standard indexing search methods, such as k-d Tree [8], R-Tree [14], VP-Tree [31], cover tree [9]. But the search space of our event tickets is not metric and the dimensionality is high. Therefore, locality sensitive hashing [13] is more practical. Another heuristic solution is the attribute clustering based method. Different system events have different system attributes, and the clustering algorithm can easily separate all tickets into categories based on their attribute names. If two events share very few common attributes, their similarity cannot be high. Therefore, in most cases, the nearest neighbors search only needs to access these tickets in the same category.

C. Parameter Selection

The best choice of K for the KNN algorithm depends on the data. A larger K may reduce of the effect of noise, but makes the boundaries between related resolutions and unrelated resolutions less distinct. Many heuristic approach are proposed for finding the best K. A widely used approach is the cross-validation [24], which enumerates all possible K and tests the average accuracy. Both too small K and too large K would degrade the accuracy. We choose K corresponding to the peak accuracy. $k \leq K$, is the number of recommended resolutions for each incoming ticket. In our recommendation system, we do not fix k for every ticket. A default k is provided by the system administrator, who can also enlarge the value of k for each ticket if more recommendations are needed.

V. Evaluation

A. Implementation and Testing Environment

We implemented four algorithms: KNN, weighted KNN [12], the division method and the probabilistic fusion method, which are denoted by "KNN", "WeightedKNN", "Divide" and "Fusion" respectively. Our proposed two algorithms, "Divide" and "Fusion", are based on the weighted KNN algorithm framework. We choose the KNN-based algorithm as the baseline because it is the most widely used Top-N item based recommendation algorithm. Certainly, we can use SVM to predict the ticket type to be false or real. But our core idea is not about classification, but to combine the penalty for the misleading resolution into the recommendation algorithm.

All algorithms are implemented by Java 1.6. This testing machine is Windows XP with Intel Core 2 Duo CPU 2.4GHz and 3GB of RAM.

B. Experimental Data

Experimental event tickets are collected from three accounts managed by IBM Global Services denoted later "account1", "account2" and "account3". The monitoring events are captured by IBM Tivoli Monitoring [2]. The ticket sets are summarized in Table I.

C. Accuracy

For each ticket set, the first 90% tickets are used as the historic tickets and the remaining 10% tickets are used for testing. Hit rate is a widely used metric for evaluating the accuracy in item-based recommendation algorithms [10], [15], [20].

$$\text{Accuracy} = \text{Hit-Rate} = |Hit(C)|/|C|,$$

where C is the testing set, and $Hit(C)$ is the set for which one of the recommended resolutions is *hit* by the true resolution. If the recommendation resolution is truly relevant to the ticket, we say that recommended resolution is *hit* by the true resolution.

Since real tickets are more important than false ones, we define another accuracy measure, the weighted accuracy, which assigns weights to real and false tickets. The weighted accuracy is computed as follows:

$$\text{Weighted Accuracy} = \frac{\lambda \cdot |Hit(C_{real})| + (1-\lambda) \cdot |Hit(C_{false})|}{\lambda \cdot |C_{real}| + (1-\lambda) \cdot |C_{false}|},$$

where C_{real} is the set of real testing tickets, C_{false} is the set of false testing tickets, $C_{real} \cup C_{false} = C$, λ is the importance weight of the real tickets, $0 \leq \lambda \leq 1$, it is also the penalty mentioned in Section III-C. In this evaluation, $\lambda = 0.9$ since the real tickets are much more important than the false tickets in reality. We also test other large λ values, such as 0.8 and 0.99. The accuracy comparison results have no significant change.

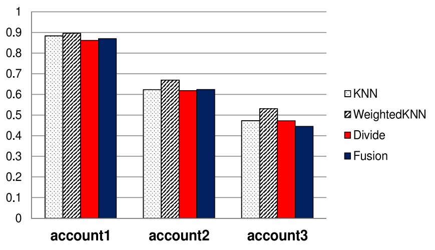
We vary K and k from 1 to 20 to obtain different parameter settings. Figures 4 and 5 are the testing results for $K = 10, k = 3$ and $K = 20, k = 5$. The comparison results for other parameter settings are similar to the two figures. It is seen that, the weighted KNN algorithm always achieves the highest accuracy in the three data sets. But for real tickets, our proposed probabilistic fusion method outperforms other algorithms (Figures 4b and 5b). As for the weighted accuracy in Figures 4c and 5c, the weighted KNN and the probabilistic fusion are still the two best algorithms, and neither of them outperforms the other in all data sets. Overall, the performances of all four algorithms are very close. For each comparison, the accuracy gap between the highest one and the lowest one is about 10%.

D. Penalty

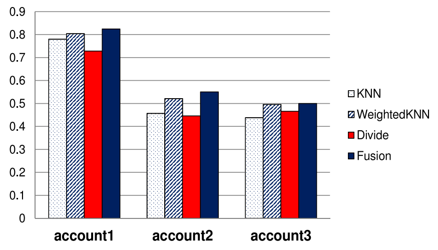
Figures 6 and 7 show the average penalty for each testing ticket. We assigned a higher importance to the real tickets, $\lambda = 0.9$. As shown by these figures, our proposed two algorithms have smaller penalties than the traditional KNN-based recommendation algorithms. The probabilistic fusion method outperforms the division method, which relies heavily on the ticket type predictor. Overall, our probabilistic fusion method only has about 1/3 of the penalties of the traditional KNN-based algorithms.

E. Overall Performance

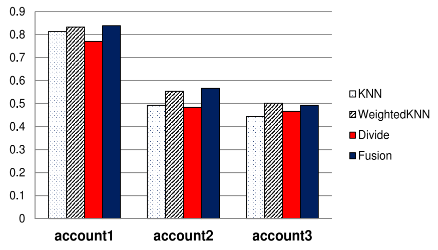
An overall quantity metric is used for evaluating the recommendation algorithms, covering both the accuracy and the average penalty. It is defined as overall score = weighted accuracy / average penalty. If the weighted accuracy is higher or the average penalty is lower, then the overall score becomes higher and the overall performance is better. Figures 11 and 12 show the overall scores of all algorithms for two parameter settings.



(a) Accuracy

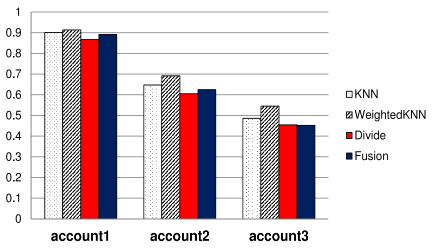


(b) Accuracy for Real Tickets

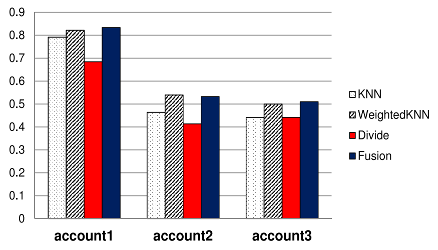


(c) Weighted Accuracy

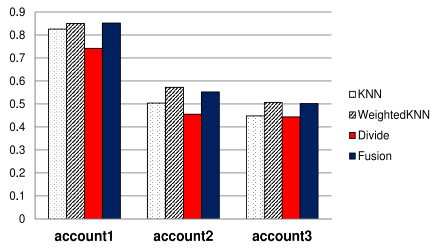
Fig. 4: Test Results for $K = 10$, $k = 3$



(a) Accuracy

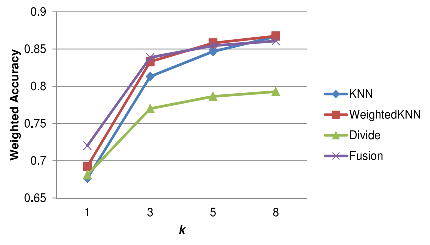


(b) Accuracy for Real Tickets

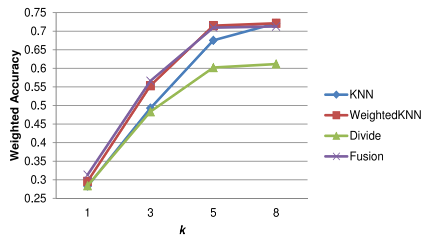


(c) Weighted Accuracy

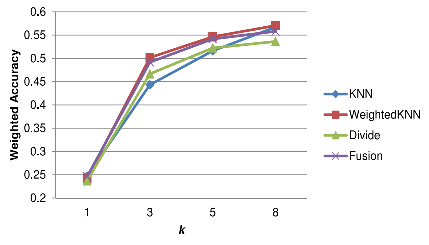
Fig. 5: Test Results for $K = 20$, $k = 5$



(a) Weighted accuracy for account1

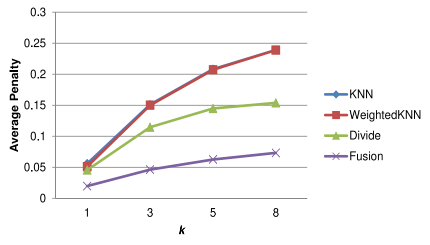


(b) Weighted accuracy for account2

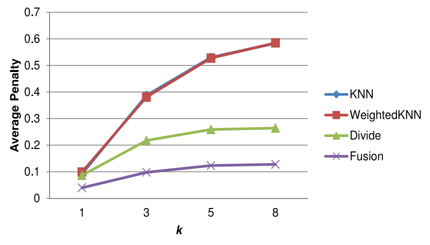


(c) Weighted accuracy for account3

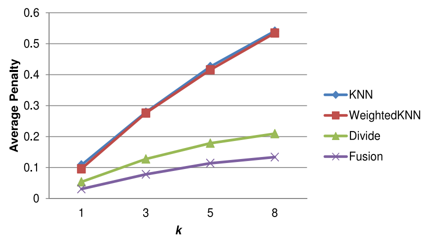
Fig. 8: Weighted Accuracy by varying k, $K = 10$



(a) Average penalty for account1



(b) Average penalty for account2



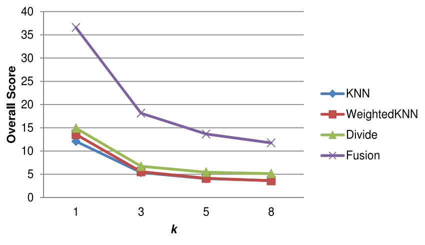
(c) Average penalty for account3

Fig. 9: Average Penalty by varying k, $K = 10$

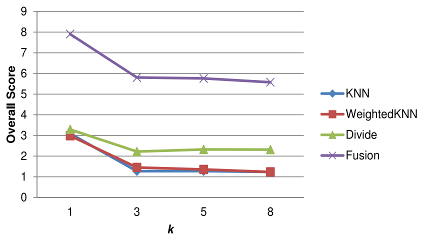
It is seen that, our proposed algorithms are always better than the KNN-based algorithms in each data set.

F. Variation of Parameters

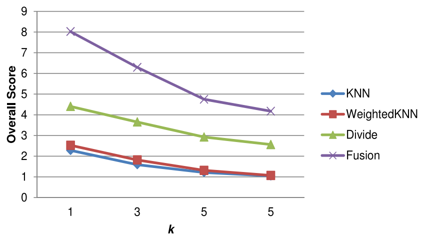
To compare the results of each algorithm, we vary the number of each recommendation resolutions, k. Figures 8, 9 and 10 show the weighted accuracies, average penalties and overall scores by varying k from 1 to 8, with $K = 10$. For other values of K, the comparison results are similar to the three figures. As shown by Figure 8, when we increase the value of k, the size of the recommendation results becomes larger. Then the probability of one recommended resolution being *hit* by the true resolution also increases. Therefore, the weighted accuracy becomes higher. Except for the division method, all algorithms have similar weighted accuracies for



(a) Overall score for account1



(b) Overall score for account2



(c) Overall score for account3

Fig. 10: Overall Score by varying k, $K = 10$

TABLE IV: A Case Study for $K = 10$, $k = 3$

Algorithm	**Recommended Resolution**	**Is Hit**	**Is Real Ticket's Resolution**	**Penalty**
KNN	No actions were taken by GLDO for this Clearing Event...	no	false	0.9
	Clean up the backup filesystem. Filesystem kbytes used avail capacity...	no	true	0
	Duplicated 28106883...	no	true	0
WeightedKNN	No actions were taken by GLDO for this Clearing Event...	no	false	0.9
	I cleaned up the FS using RMAN retention policies...	yes	true	0
	Duplicated 28106883...	no	true	0
Divide	Duplicated 28106883...	no	true	0
	Another device failure has been reported for this node...	no	true	0
	I cleaned up the FS using RMAN retention policies...	yes	true	0
Fusion	Duplicated 28106883...	no	true	0
	Another device failure has been reported for this node...	no	true	0
	I cleaned up the FS using RMAN retention policies...	yes	true	0

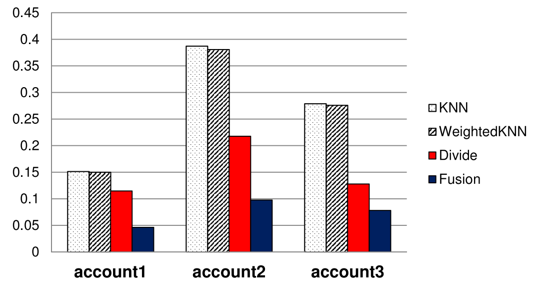


Fig. 6: Average Penalty for $K = 10$, $k = 3$

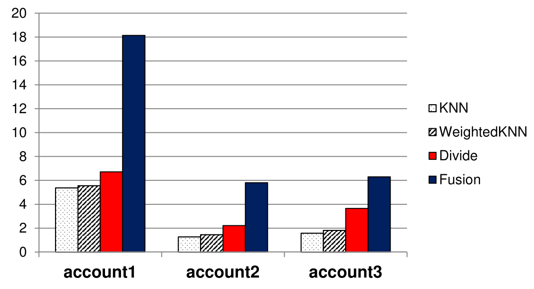


Fig. 11: Overall Score for $K = 10$, $k = 3$

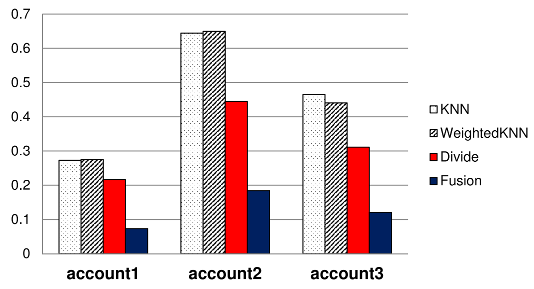


Fig. 7: Average Penalty for $K = 20$, $k = 5$

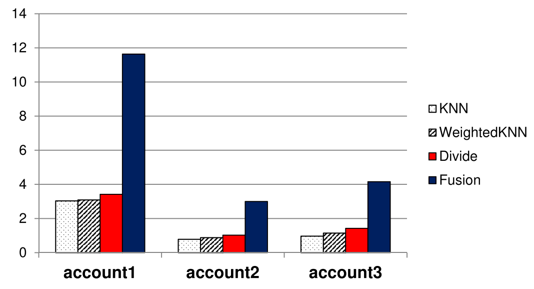


Fig. 12: Overall Score for $K = 20$, $k = 5$

each k. However, as k is increased and there are more recommended resolutions, there are more potential penalties in the recommended resolutions. Hence, the average penalty also becomes higher (Figure 9). Finally, Figure 10 compares the overall performance by varying k. Clearly, the probabilistic fusion method outperforms other algorithms for every k.

G. A Case Study

We select an event ticket in "account1" to illustrate why our proposed algorithms are better than the traditional KNN-based algorithms. Table IV shows a list of recommended resolutions given by each algorithm. The testing ticket is a real event ticket triggered by a low capacity alert for the file system. Its true resolution of this ticket is: "cleaned up the FS using RMAN retention policies..." RMAN is a data backup and recovery tool in Oracle database. The general idea of this resolution is to use this tool to clean up the old data.

As shown by Table IV, the first resolution recommended by KNN and WeightedKNN is a false ticket's resolution: "No actions were taken by GLDO for this Clearing Event..." It might be caused by a temporal file generated by some application, which would clean up the temporal file automatically after its job was done. When the system administrator opened that ticket, the problem was gone, and that ticket is seen as false. However, the testing ticket is real and would not disappear unless the problem was actually fixed. This resolution from the false ticket would have misled the system administrator to overlook this problem. Consequently, a penalty of $\lambda = 0.9$ is given to KNN and WeightedKNN.

WeightedKNN, Divide and Fusion all successfully find the true resolution of this testing ticket, but WeightedKNN has one false resolution, so its penalty is 0.9. Our proposed methods, Divide and Fusion, have no penalty for this ticket. Therefore, the two methods are better than WeightedKNN.

VI. Related Work

This section reviews prior research studies related to the automated IT Service management and the recommendation system. System monitoring, as part of the automated Service management, has become a significant research area of IT industry in the past few years. There are many commercial products such as IBM Tivoli [2], HP OpenView [1] and Splunk [5] focusing on system monitoring. The monitoring targets include the components or subsystems of IT infrastructures, such as the hardware of the system (CPU, hard disk) or the software (a database engine, a web server). Once certain system alarms are captured, the system monitoring software will generate the event tickets into the ticketing system. Automated ticket resolution is much harder than automated system monitoring because it requires vast domain knowledge about the target infrastructure. Some prior studies apply approaches in text mining to explore the related ticket resolutions from the ticketing database [23], [28]. Other works propose methods for refining the work order of resolving tickets [23], [19] and discovering the dependency of tickets [26].

With the development of e-commerce, a substantial amount of research has been devoted to the recommendation system. Lots of recommendation algorithms are proposed for promoting products to online users [7], [11], [16], [17]. Recommendation algorithms can be categorized as item-based [22], [15], [20] and user-based algorithms [27], [16], [7], [11]. The difference with our work is that, in e-commerce, products are maintained by reliable sellers. The recommendation algorithms usually do not need to consider the problem of fake or low-quality products. But in service management, false tickets are unavoidable. The traditional recommendation algorithms do not take into account the types of tickets and as a result would recommend misleading resolutions.

VII. Conclusion

This paper studies recommendation of problem resolutions for event tickets in automated service management. It analyzes three sets of event tickets collected from production service infrastructure, and identifies vast number of repeated resolutions for event tickets. Based on this observation, this paper proposes two resolution recommendation algorithms for event tickets. An additional penalty is incorporated in the algorithms to avoid misleading resolutions. In our future work, we will investigate and develop more advanced and efficient resolution recommendation algorithms with improved accuracy and efficiency.

VIII. Acknowledgement

The work is partially supported by US National Science Foundation under grants IIS-0546280, HRD-0833093, and IIS-1213026.

References

[1] HP OpenView : Network and Systems Management Products. http://www8.hp.com/us/en/software/enterprise-software.html.

[2] IBM Tivoli : Integrated Service Management. http://ibm.com/software/tivoli/.

[3] IBM Tivoli Monitoring. http://ibm.com/software/tivoli/products/monitor/.

[4] ITIL. http://www.itil-officialsite.com/home/home.aspx.

[5] Splunk: A commerical machine data managment engine. http://www.splunk.com/.

[6] N. Ayachitula, M. J. Buco, Y. Diao, M. Surendra, R. Pavuluri, L. Shwartz, and C. Ward. IT service management automation - a hybrid methodology to integrate and orchestrate collaborative human centric and automation centric workflows. In *IEEE SCC*, pages 574–581, 2007.

[7] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, pages 43–52, 2007.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[9] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

[10] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, Jan. 2004.

[11] Y. Ding and X. Li. Time weight collaborative filtering. In *ACM CIKM*, pages 485–492, 2005.

[12] S. A. Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems Man and Cybernetics*, SMC-6(4):325–327, april 1976.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of VLDB*, pages 518–529, Edinburgh, Scotland, UK, September 1999.

[14] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *ACM SIGMOD*, pages 47–57, 1984.

[15] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *CIKM*, pages 247–254, 2001.

[16] Y. Koren. Collaborative filtering with temporal dynamics. In *KDD*, pages 447–456, 2009.

[17] L. Liu, N. Mehandjiev, and D.-L. Xu. Multi-criteria service recommendation based on user criteria preferences. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 77–84, 2011.

[18] P. Marcu, L. Shwartz, G. Grabarnik, and D. Loewenstern. Managing faults in the service delivery process of service provider coalitions. In *IEEE SCC*, pages 65–72, 2009.

[19] G. Miao, L. E. Moser, X. Yan, S. Tao, Y. Chen, and N. Anerousis. Generative models for ticket resolution in expert networks. In *KDD*, pages 733–742, 2010.

[20] X. Ning and G. Karypis. SLIM: Sparse linear methods for top-n recommender systems. In *ICDM*, pages 497–506, 2011.

[21] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1984.

[22] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl. Application of dimensionality reduction in recommender system – a case study. In *ACM WebKDD Workshop*, 2000.

[23] Q. Shao, Y. Chen, S. Tao, X. Yan, and N. Anerousis. EasyTicket: a ticket routing recommendation engine for enterprise problem resolution. *PVLDB*, 1(2):1436–1439, 2008.

[24] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[25] L. Tang, T. Li, F. Pinel, L. Shwartz, and G. Grabarnik. Optimizing system monitoring configurations for non-actionable alerts. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012.

[26] L. Tang, T. Li, and L. Shwartz. Discovering lag intervals for temporal dependencies. In *ACM KDD*, pages 633–641, 2012.

[27] L. Terveen and W. Hill. Beyond recommender systems: Helping people help each other. In *HCI in the New Millennium*, pages 487–509, 2001.

[28] D. Wang, T. Li, S. Zhu, and Y. Gong. iHelp: An intelligent online helpdesk system. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 41(1):173–182, 2011.

[29] B. Wassermann and W. Emmerich. Monere: Monitoring of service compositions for failure diagnosis. In *ICSOC*, pages 344–358, 2011.

[30] Y. Yan, P. Poizat, and L. Zhao. Repair vs. recomposition for broken service compositions. In *ICSOC*, pages 152–166, 2010.

[31] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, pages 311–321, 1993.