Problem Identification by Mining Trouble Tickets

Vikrant Shimpi, Maitreya Natu, Vaishali Sadaphal, Vaishali Kulkarni
Tata Research Development and Design Centre
{vikrant.shimpi, maitreya.natu, vaishali.sadaphal, vaishali.kulkarni}@tcs.com

ABSTRACT

IT systems of today's enterprises are continuously monitored and managed by a team of resolvers. Any problem in the system is reported in the form of trouble-tickets. A ticket contains various details of the observed problem. However, the knowledge of the actual problem is hidden in the ticket description along with other information. Knowledge of issues helps the service providers to better plan to improve cost and quality of operations. In this paper, we address the problem of extracting the issues from ticket descriptions. We discuss various challenges in issue extraction and present algorithms to handle different scenarios. We demonstrate the effectiveness of the proposed algorithms through two real-world case-studies.

1. INTRODUCTION

With the increasing reliance of business on IT, the health of IT systems is continuously monitored. The IT service providers manage these systems with a team of resolvers. These resolvers act on the reported problems and take corrective actions to ensure smooth functioning of IT and business. The system problems are reported to the resolvers in two ways:

- *System-generated issues:* Components such as business applications, processes, CPU, disk, and network interfaces are monitored to detect anomalies. The monitoring tools capture and report any abnormal behavior of system components in the form of alerts.
- *User-generated issues:* The users of IT systems report problems through calls, emails, or chats.

Both system-generated and user-generated issues are reported in the form of *tickets* to a team of resolvers. A ticket contains various details of the reported problem such as the reporting time, system-of-origin, severity, and other details. In addition to this, each ticket also contains a description field that contains the details of the observed problem. In case of system-generated tickets, this description is automatically generated by the monitoring and alerting tools. This description is often structured and is based on how the alerting tools have been configured to report a problem. However, in case of user-generated tickets, this description contains free-form text written by users. This description is unstructured and contains variations, ill-formed sentences, and spelling and grammar mistakes.

While the ticket description contains many details, it is important to extract the information of the actual problem referred by the ticket. We refer to the problem referred by the ticket as *issue*. For instance, consider the following ticket description

> PATROL alert for filesystem apps-orabase is 90% full on d-2hh4knn

The *issue* referred by this description is

> PATROL alert for filesystem full

Extraction of issues occurring in the IT system can provide crucial information to better understand and control the IT operations. It can assist in improving both the IT system and the human system involved in the operations. The IT system consists of the business functions, applications, and the infrastructure. The human system consists of the teams of resolvers that manage the IT systems. Below are some examples of how the knowledge of issues can assist in improving these systems.

1. *Improving the IT system:* (a) The knowledge of issues observed in the IT system helps in inferring problem trends and frequent problematic areas in the system. (b) The issues referred by the high-severity tickets can provide insights into the critical areas that cause customer unrest or business instability. These issues can be prioritized for taking corrective actions by problem management. (c) Knowledge of frequent issues observed in specific domains can assist the service providers to prepare a service catalog. A service provider uses the service catalog for knowledge acquisition, generating training plans, developing automation scripts, etc.
2. *Improving the human system:* (a) The knowledge of issues can be used to identify issues that consume maximum effort of the resolvers. These issues can be considered for full or partial automation. (b) The knowledge of frequent issues can be used to prepare training plans to train resolvers with appropriate skills. (c) The knowledge of arrival patterns of the issues can be used

The 20th International Conference on Management of Data (COMAD), 17th-19th Dec 2014 at Hyderabad, India.

to plan shifts with right number of resolvers with right skills.

Mining issue from the ticket descriptions presents several challenges. (a) The same problem is represented in different ways in different tickets. (b) The organizations customize the structure of the description according to their own preferences. (c) The descriptions are domain-dependent and contain domain-specific words. For example, Oracle issues have words such as tables, SQL query, filesystem, disk, and Oracle-specific error codes. (d) The descriptions in user-generated tickets are written in free-form text. They are often ambiguous, and contain spelling and grammar errors.

Currently, service providers lack a systematic approach to capture the knowledge of issues. They either rely on the coarse-grained information from the ticketing tools or on the intuition-driven inputs provided by the resolvers.

- Ticketing tools such as ServiceNow [1] and BMC Remedy [2] allow resolvers to classify each ticket in various categories and subcategories. These categories are often referred as Category, Type, Item, and Summary (CTIS) [3]. However, most often these tools are not configured to sufficient level of details. For instance, various disk-related issues such as *disk full*, *disk failure,* or *disk corruption* are all classified as *Category = Infrastructure, Type = Hardware, Item = Storage, Summary = Disk issue*. In addition to that, resolvers and users make mistakes in rightly classifying the issue. As a result, the analysis based simply on this classification can often lead to incorrect and insufficient knowledge of the issues.

- Another approach that is adopted to infer the issues is by analyzing the fixlogs used for ticket resolution. A fixlog of an issue is a document that contains the details of the resolution steps to act on the issue. A resolver after acting on an issue makes an entry in the ticketing tool of the fixlog used to resolve a ticket. However, there are several practical challenges faced in this approach. Often the fixlogs are not available for all issues. Many times, the fixlogs are too generic and are used for many issues. In addition to this, the resolvers often do not make an entry of the fixlog in the ticketing system.

- The most common approach used to infer issues is the manual intuition-driven approach. The service providers infer the issue information either from the monthly reports manually produced by the teams or by talking to the domain experts. This approach carries the risk of being incomplete, inaccurate, and of variable quality.

In this paper, we propose an analytics-driven approach to mine the textual descriptions of tickets to extract issues. We propose the following approach of issue extraction.

- We propose to use information retrieval techniques along with domain knowledge to extract the issues from tickets.

- We propose two different algorithms to extract issues from system-generated and user-generated tickets. There is an inherent difference in the structure and heterogeneity in the descriptions of system-generated and user-generated tickets. The system-generated tickets have structured and consistent descriptions. We propose a clustering-based technique for system-generated tickets. On the other hand, the user-generated tickets have verbose and ambiguous descriptions. We propose a keyword-discovery-based approach for user-generated tickets.

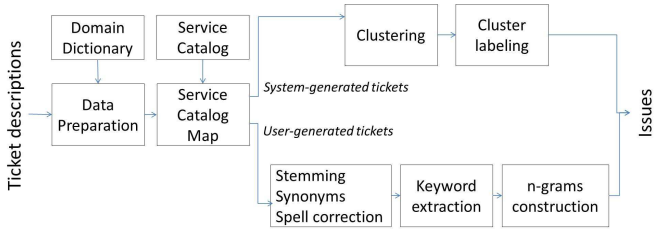

Figure 1: Functional diagram of the proposed approach.

- We capture the knowledge of IT and business domains in the form of dictionaries of include and exclude list of words and regular expressions. The knowledge consists of words and patterns pertaining to (a) the technology domain such as Oracle, Windows, and Linux, (b) business domain such as banking, retail, and finance, (c) monitoring tools such as ControlM, Autosys, and BMC Patrol, and (d) organization-specific naming conventions

- Finally, we present validation of the effectiveness of our techniques through two real word case-studies.

In Section 2, we present design rationale of the proposed issue extraction techniques. In Section 3, we present the proposed approach for extracting issues for system-generated and user-generated tickets. In Section 4, we demonstrate the effectiveness of the proposed approach using two real-world case-studies. We present related work in Section 5 and conclude in Section 6.

2. DESIGN RATIONALE

In this section, we present the proposed approach for extracting issues from ticket descriptions. Various factors need to be addressed while extracting issues from ticket descriptions.

Consider the following description:

> *ControlM Alert job=orahAAAA04 memname=orahotback.sh node=aaaa04 msg=sev3 job failed AAAA04*

The issue referred by this description is as follows:

> *ControlM Alert job failed*

The description contains other details such as name of the application *job=orahAAAA04*, name of the script *memname=orahotback.sh*, and severity level *msg=sev3*. To accurately extract the issues, it is critical to separate the factual knowledge of the issue from the specific details of its individual manifestations.

Different tickets describe the same issue in different ways. For instance, the above example can be reported as

> *ControlM Sev 3 Alert on Job Failure: job=orahAAAA04, script orahotback.sh*

In case of user-generated tickets, this problem becomes more challenging because of lack of structure, too many variations, and spelling and grammar errors. For instance, consider the following issue.

application not responding

This issue is addressed by users in a variety of ways, such as,

application hung,
application dead,
No response from application since last 5 mins,
application not responding for the fifth time since morning.

While extracting issues, it is important to address such variations and generate a uniform issue for its various manifestations.

As discussed in Section 1, there is an inherent difference in the structure and heterogeneity in the descriptions of user-generated and system-generated tickets. Hence, we propose two different approaches to extract issues. Figure 1 presents the functional diagram of the proposed approach. The input is a set of ticket descriptions and the output is the issue extracted for each description.

Below we present the details of different functional elements of the proposed approach:

- *Data preparation:* The first step is to clean the descriptions, remove unwanted details, and tokenize. This step needs to be customized by knowledge of specific domains to ensure that all domain-specific words, patterns, and phrases are retained during data preparation.

- *Mapping to service catalog:* The commonly occurring issues in a domain are often captured in the form of a service catalog. For instance, service catalog of a technology domain of Oracle contains the list of commonly occurring issues in Oracle such as tablespace full, query running slow, index not built, etc. We use service catalog to mine these commonly occurring issues from the ticket descriptions. For the remaining ticket descriptions, we adopt the following approach.

- *System-generated tickets:* System-generated descriptions have a fixed structure and limited variations. Hence, we use clustering to group similar issues. We form label that best represents the descriptions in a cluster.

- *User-generated tickets:* On the other hand, user-generated descriptions demonstrate too many variations. Here, clustering is not effective. Hence, we first apply various techniques to address these variations such as use of stemming, synonym detection, and spelling corrections. We then extract keywords and group similar issues based on the commonality of keywords. We represent each group using an n-gram of keywords.

- *Knowledge cartridge:* Note that, the proposed approach is independent of any specific technology domain, and is designed to be configurable to extract issues for ticket descriptions of any domain. The effectiveness of the proposed approach can be significantly increased by domain-specific customizations of data preparation and service catalog mapping. We propose to make cartridges for technology domains (Unix, Oracle, Windows, etc.), business domains (banking, retail, finance, etc.), and tool domains (BMC Patrol, Control M, Tivolli TWS, Autosys, etc.) For each domain we maintain a dictionary of domain-specific words and patterns to include or exclude while extracting issues. We also maintain a service catalog of known issues within the domain.

In the rest of paper, we present details of these functional elements.

3. PROPOSED APPROACH

In this section, we present the approach of issue extraction from descriptions in system-generated and user-generated tickets.

3.1 Data preparation

As discussed earlier, *issue* is the problem reported in the ticket description. Ticket description contains lot of additional information such as timestamp, location, threshold, system-of-origin, severity, etc. This additional information changes as the same issue occurs at different timestamp, location, system-of-origin, etc. Though this information is important, it misleads issue extraction and hence must be separated. For example, consider a description such as

Unix server dt2n1g1 down.

Here, the actual issue is

Unix server down.

The name of system-of-origin *dt2n1g1* is additional information. Similarly, consider a description such as

filesystem 01hw4884 80% full.

Here, the actual issue is

filesystem full.

Here, the name of the system-of-origin and the current value of occupancy of space in the filesystem is additional information.

The challenge is to find the right set of words that should be removed or retained. Removing and retaining of words could have significant impact on subsequent issue extraction steps. We approach this problem by making following observations.

- *Issue is made up of dictionary words:* Consider following examples of description.

  1. *A-IM002930432.Pl reset the Unix password for login id bchho02 @ 156.5.238.69,*
  2. *27488732 CREATED unable to backup data after upgrade to windows 7,*
  3. *A-IM002971223–Need to revoke admin access,*
  4. *Domain Drive M:\ setting\ application data\SAP access denied.*

  It can be observed that issues are often described in plain English dictionary words, while all other details are non-English words. For example, in description 1 above, issue is described by

  reset the Unix password for login id.

All the other words such as *A-IM002930432*, *Pl*, and *bchho02 @ 156.5.238.69* are all non-English words.

- *Some domain-specific words related to the issue are non-English:* Consider a description such as

  Portal Alert on appax040 at 2012-07-11 11:44:40.000 SwapPercentUsed is CRITICAL.

  Ideally, the issue described in above description is

  Portal alert for SwapPercentUsed is critical.

  But, the approach of retaining only English words identifies

  Portal alert for is critical

  as the issue. Here, *SwapPercentUsed* is a non-dictionary word without which the issue is meaningless.

- *Some domain-specific words are not related to the issue but are English:* Some English words need to be excluded since they are domain-specific names of system-of-origin. Consider a description such as

  Portal alert on d-6x9ttk1: /export/home FileSystem is full.

  The approach of retaining only English words identifies

  Portal alert on export home filesystem is full

  as issue. In Unix, name of the *filesystem* often contains folder names such as *home*, *root*, *export*, etc. Hence, we exclude such words while extracting an issue.

Based on above observations, we provide following solution.

- We use English dictionary and mark each word as English or non-English. We separate non-English words and retain only the English dictionary words related to the issue.
- We prepare an *Include list* that contains domain-specific non-dictionary words and captures knowledge about a particular domain. The words in the *Include list* are retained in the description since they describe the issue. This list is populated by subject matter experts (SME) of respective domains. Some example words in *Include list* for Unix domain are *FSCapacity_Alarm*, *SwapPercentUsed*, etc.
- We prepare an *Exclude list* that contains domain-specific English words. Some examples of words in *Exclude list* are *export*, *home*, etc. for the domain of Unix. We make use of this *Exclude list* to remove such words.
- We also provide a lever to detect patterns in the form of regular expressions. For example, in case of Oracle, *Include list* contains patterns such as *ora** that denotes Oracle errors and *memname = *.sh* that denotes the shell script.

The pseudo code for the data preparation step is as follows.

1. Input and Output
   - Input: D_r = List of description, D = List of dictionary words, I = List of include words, E = List of exclude words, S = List of special characters
   - Output: D_c = List of cleaned description
2. Prepare a list of words, W, by tokenizing all descriptions in D_r by space.
3. For each word in list of words, W,
   - Check if word is a subset of domain include list, I
   - If no, tokenize word by special characters, S and add it to W
4. Prepare a list of non-dictionary words, $ND = W - (D + I)$
5. Add list of exclude words to non-dictionary words list, $ND = ND + E$
6. Remove non-dictionary words in ND from descriptions D_r to make a list of cleaned descriptions, $D_c = D_r - ND$

3.2 Service catalog mapping

Service catalog of a domain is a predefined set of issues in a domain for which resolution steps are known. It contains factual knowledge of the issues in a domain. The service catalogs are based on various technologies in IT infrastructure such as Unix, Linux, Oracle, Windows, etc. While extracting issues, we make use of this prior knowledge of known issues, by mapping cleaned descriptions, obtained by data preparation step, to the service catalog items. The service catalog makes the search guided and allows customization for each domain. Hence, many issues can be easily extracted that are otherwise difficult to extract. For example, consider a description as

D-8Y6TTK1 - cluster failover issue.

Here this issue can be easily mapped to service catalog item

Cluster FailOver/FallBack

in Linux service catalog.

Service catalog is structured, well defined, and contains known and finite set of issues. On the other hand, ticket descriptions are unstructured, unknown and can occur in various ways. The challenge is to map ticket descriptions with service catalog items.

Each service catalog item is defined as a two-tuple <*action object*>. For example, action = *create*, and object = *table*. However, ticket descriptions may refer to action in different ways. For example, *create table* can be referred as *make table*, *construct table*, etc. Hence, to best map service catalog item to descriptions, we construct multiple synonyms of action. Thus, for a service catalog item, if both object and action are present in the description, then we map the description to the corresponding service catalog item. The action and object keywords are either domain sensitive words or English words. For domain sensitive words, domain experts are required whereas for English words, an English thesaurus can be used for synonym creation. For example, in service *diskdetectionissue*, domain sensitive words can be *disk*, *drive*, *tape* and English words are *detect*, *recognize*, *identify*, etc. While mapping descriptions to service catalog items, following cases may occur.

1. *One service catalog item maps to multiple descriptions:* The descriptions contain many variations of the action and the object. These descriptions are mapped to a single service catalog item. For example, consider descriptions as follows:

   - *Unlock account,*
   - *Unlock password of my account,*
   - *Requesting unlocking of account.*

   All these cleaned descriptions contain <*action object*> pair <*unlock account*>. As a result, they all map to service catalog item *UnlockAccount*.

2. *One description maps to many service catalog items:* This can happen in following cases.

   - Issue is composite: Consider a description as

     After installing Windows 7, system is very very slow. For unlocking machine, system takes more than 20 min. Please resolve this ASAP.

     We observe that the actual issue might be

     (a) *windows installation* problem, or
     (b) *account related unlocking* problem.

     Here, the description contains multiple <*action object*> pairs such as <*install windows*>, or <*unlock system*> which maps to multiples items in service catalog.

   - Service catalog is too detailed: Consider a description as

     backup failure.

     This gets mapped to multiple service catalog items as

     (a) *Full backup failure*,
     (b) *Incremental backup failure* ,
     (c) *Differential backup failure* ,
     (d) *Cumulative backup failure* ,
     (e) *Archive log backup failure* .

Following is the pseudo code for mapping descriptions to service catalog item.

1. Input and Output

   - Input: D_c = List of cleaned description SC = Service catalog
   - Output: Mapping $< d_i, SC_j >$

2. For each cleaned description d_i in D_c

   - For each item SC_j in service catalog SC with object O_j and action A_j
     - If object, O_j and action A_j keywords are present, assign the corresponding service catalog item to cleaned description d_i

3.3 Issue extraction in system-generated tickets

In this section, we present the approach of extracting issues from remaining descriptions that do not map to service catalog.

3.3.1 Clustering

The information unrelated to issue is separated from the descriptions in the data preparation step. The cleaned descriptions still contain variations based on the configuration of the monitoring tools for different applications in the system. We address these variations by clustering the descriptions based on similarity. Clustering helps grouping similar descriptions together and assign dissimilar descriptions to separate groups.

Consider two descriptions as

- *BMC portal alert on d-hw6ttk1-lvmh: /var for filesystem full*, and
- *BMC portal critical alert on d-hw6ttk1-lvmh/var for filesystem full* .

After cleaning, the descriptions are

- *BMC portal alert on filesystem full*, and
- *BMC portal critical alert on filesystem full.*

These two descriptions are same except the word *critical* present in the second description. We group such similar descriptions into one cluster by computing similarity between two descriptions.

Some of the approaches to compute similarity [8] between two descriptions are Jaccard coefficient, Dice coefficient, etc. Dice coefficient gives twice the weight to common elements. Since we emphasize on commonality, we use Dice coefficient to compute similarity between two descriptions. Let A and B be sets of words in two descriptions. Dice similarity, D, between A and B is defined as follows:

$$D = \frac{2 * |A \cap B|}{|A| + |B|} \tag{1}$$

For example, if

- A = *BMC portal alert on filesystem full*, and
- B = *BMC portal critical alert on filesystem full*,

then $|A| = 6$, $|B| = 7$, $|A \cap B| = 6$ and $D = \frac{2*6}{7+6} = 0.923$.

We compute Dice similarity between every pair of clean description. We construct a similarity graph of clean descriptions in which nodes are clean descriptions. There is an edge between two clean descriptions if they are similar. We consider two clean descriptions similar if the similarity coefficient between them is greater than a predefined threshold *threshold_similarity*. We cluster clean descriptions by applying graph clustering on the similarity graph of clean descriptions. Various graph clustering techniques can be used for clustering such as cliques [4], connected components [5], graph partitioning [9], graph cuts [7], etc.. We have used cliques to identify clusters of clean descriptions. The similarity threshold can be set automatically using certain heuristics such as *median* or *mean + (k * standard deviation)* of the distribution of similarity threshold values.

3.3.2 Cluster label

A single cluster contains many variations of descriptions. The next step is to provide a label to each cluster that best represents all the members within a cluster. The set of common words from all descriptions within a cluster are probable candidate for a cluster label. If we arrange these common words in any order, then the cluster label cannot be easily understood and is not meaningful. For example, a label,

host connect unable to

does not make sense. While we have narrowed down to the words to be used for the label, it is also important to place them correctly to form meaningful phrase. For example, the correct order of words for the previous example is

unable to connect host.

One of the criteria to compute the position of a word is based on its position in individual description. For instance, the word that occurs most frequently on 1st position is placed in the 1st position in the label.

Following is the pseudo code of clustering and labeling algorithm we have implemented.

1. Input and Output
   - Input: D_c = List of cleaned description, $threshold\_similarity$ = similarity threshold
   - Output: Mapping $< d_i$, Issue label $>$
2. Compute dice similarity for each pair of cleaned description $< d_i, d_j >$ in D_c,
   - IssueAdjacencyMatrix[i,j] = Dice similarity coefficient
3. Build adjacency matrix for the issue-similarity graph
   - If (IssueAdjacencyMatrix[i,j] $\geq$ $threshold\_similarity$), IssueAdjacencyMatrix[i,j] = 1
   - else IssueAdjacencyMatrix[i,j] = 0
4. Identify a maximum clique C_k in IssueAdjacencyMatrix
5. Remove issues identified in clique C_k from IssueAdjacencyMatrix
6. Repeat step 4 till all issues are covered
7. For each clique C_k, identify a label
   - Identify set of common words C_w from the set of cleaned description belonging to the clique
     - For each word w in C_w,
       * Compute its position in set of cleaned description belonging to clique, p = Mode of the position of the word w
       * Label of the clique = Arrangment of the words in C_w according to p

3.4 Issue extraction in user-generated tickets

In this section, we present the approach of extraction of issue from user-generated tickets.

3.4.1 Preprocessing

User description is free-form text primarily written by users who face issues. Such descriptions contain lot of ambiguity, grammatical errors, spelling mistakes, different form of same words, etc. This results in variations in describing the same issue. For example, consider descriptions

- *job running late*,
- *job execution delayed*, and
- *abnormal delay observed on job exec.*

All these descriptions, mean the same issue, that is,

job running late,

but are written differently.

To overcome this problem, we use following different levers.

1. Stemming: Users write same words in different forms such as *lock*, *locked*, *freeze*, *freezing*, *connecting*, *connect*, etc. according to their position in description. We make use of Porter stemmer algorithm [16] and replace these words with their root such as *lock*, *freez*, *connect*, etc.
2. Spelling correction: Users often make spelling mistakes while describing issues. We identify such words and perform spelling correction [15]. For example, users often write *password* as, *passwd*, *pasword*, etc. By using spelling correction, we correct these words.
3. Synonyms: Consider examples such as (1) *remove*, and *delete*, (2) *modify*, and *change*, (3)*big* and *large*, etc. These words are often used in place of each other by the users while describing a problem. We make use of WordNet [10], which is a lexical database for English language, to automatically detect synonym words and make them consistent.

We apply these levers, to ensure that all variations in the descriptions are made consistent. By applying levers of Stemming, Spelling correction and Synonyms on above descriptions, we obtain

- *job exec delay*,
- *job exec delay*, and
- *abnormal delay observed on job exec.*

3.4.2 Keyword extraction

The next step is to extract issues from the processed set of descriptions. We first extract keywords from descriptions which occur frequently, such as *job*, *memory*, *filesystem*, *swap*, etc. While extracting the keywords we consider only nouns, adjectives, verbs, and adverbs. This makes the word search space limited. We consider top frequently occurring keywords to further form clusters. Another option could be to use TF-IDF for selecting top keywords. Larger the number of keywords considered, larger is the number of clean descriptions for which we can extract issue and hence a larger coverage.

3.4.3 N-gram construction

We next construct n-gram out of these extracted keywords. For this, we tag each of these keywords to the descriptions where they occur. For each keyword, we identify descriptions to which they are tagged and extend this keyword to form n-grams by identifying other words which appear in the descriptions. Hence, we make n-grams of these keywords such as

- *job failure*,
- *high memory utilization*,
- *filesystem backup*,

- *create swap space*, etc.

which describe the issue in a better way. Each of these n-grams represent issues and we map each of these n-grams to descriptions. The set of descriptions corresponding to each n-gram represents a cluster. Further, the n-gram is considered as the label of the cluster as well. max_n is the maximum length of n-gram considered for issue extraction and label. A longer n-gram enables a better explanation of the issue and larger correctness.

Following pseudo code describes our approach to extract issues from user-generated tickets.

1. Input and Output
   - Input: D_c = List of cleaned description
   - Output: Mapping $< d_i$, Issue label $>$
2. Compute all one words and their frequency of occurrence in cleaned descriptions.
3. Select top k one words as keywords.
4. For each word, construct n-gram until the issue is explained by n-gram
5. These set of n-grams represent issues. Assign all n-grams to cleaned descriptions.

4. CASE STUDIES

We have applied the approaches proposed in this paper on various real-world case-studies. In this section, we present two real-world case studies, one demonstrating the approach used for system-generated tickets and other for user-generated tickets.

4.1 Evaluation of the approach for system-generated tickets

In the first case-study, we applied the proposed approach on the Unix domain of a major retailer in the US. We analyzed ticket history of 6 months where 3854 tickets were produced. Out of total 3854 tickets, we were able to extract issues for 90.32% of the tickets. We validated the correctness of extracted issues by manual verification by domain expert. For each ticket, the domain expert compared ticket description with extracted issues and tagged the ticket as correct or incorrect extraction. We correctly extracted issues for 84.37 % of tickets. An example of correct issue extraction is as follows : Consider a description as

> *Patrol alert on udbax2021: STAGE N3 Filesystem udbdataetlsi01data995_0 is not mounted on UDBAX202*

and correctly extracted issue is *Patrol alert on stage filesystem is not mounted* An example of incorrect issue extraction is as follows : Consider a description as

> *Portal Alert on bllaplx104e at 2012-11-28 23:30:58.000 the Swap Space Percent Available is CRITICAL*

and extracted issue is *Portal Alert for number of processes*. We were not able to extract issues for the 9.68 % of the tickets primarily due to insufficient ticket descriptions. These ticket descriptions did not contain any details of the ticket and only referred to the affected system-of-origin or the time of occurrence.

1. *Evaluation of data preparation:* We next present evaluation of data preparation step. As we described earlier in Section III A , we make use of domain-specific *Include list* and *Exclude list* of words.
   - *Include list:* We constructed a dictionary of frequently occurring non-English words in Unix domain such as *bmc*, *portal*, *ip*, *filesystem*, etc. We also made regular expressions such as *fscapacity_**, *vcluster_**, etc. We observed that 2970 tickets (77.06%) contained domain words and regular expressions which otherwise would not have been captured.
   - *Exclude list:* The exclude list contained English dictionary words that have specific meaning in Unix environment and should not be considered for issue extraction. For example, *home*, *export*, *root*, *etc* are folder names in filesystem. Hence these words need to be removed. 45.92% of tickets contained such exclude words.

   We translated each description to cleaned description. We generated cleaned descriptions for 90.22% of tickets. Remaining 9.68% of tickets correspond to 372 tickets. These remaining descriptions contained only non-English words and hence, no issues were generated for these tickets. For example, consider description

   > texcemdbin051d - texcmedbin052d

   After cleaning, many otherwise different looking descriptions became consistent. After cleaning 3854 descriptions, we generated 47 unique cleaned descriptions.
2. *Evaluation of service catalog mapping:* We next demonstrate the impact of using service catalog mapping. The service catalog of Unix domain contained 110 items of frequently occurring issues in Unix environment. By using proposed algorithm on cleaned descriptions, we observed a match in 21.01% of tickets. One of the most dominant match was for the service *"Address Space-Crunch"*. Note that this service catalog item matched to 5 different cleaned descriptions such as

   > BMC portal alert on dell filesystem is at x percent full,
   > BMC portal alert on filesystem is critical at x percent full.

   By using service catalog mapping algorithm, we were able to map 5 out of 47 cleaned descriptions to service catalog items. After service catalog mapping, 42 cleaned descriptions remained corresponding to 3013 tickets.
3. *Evaluation of clustering algorithm for system-generated tickets:* Many cleaned descriptions were similar and had only slight variations. We applied the proposed clustering approach to group these cleaned descriptions. We were able to group 42 cleaned descriptions into 19 groups. Figure 2 (c) shows the size of these 19 groups where size of a group refers to the number of cleaned descriptions in the group. Note that, the number of groups depends on the threshold *threshold_similarity*. We later present experiments for varying threshold values used for clustering.

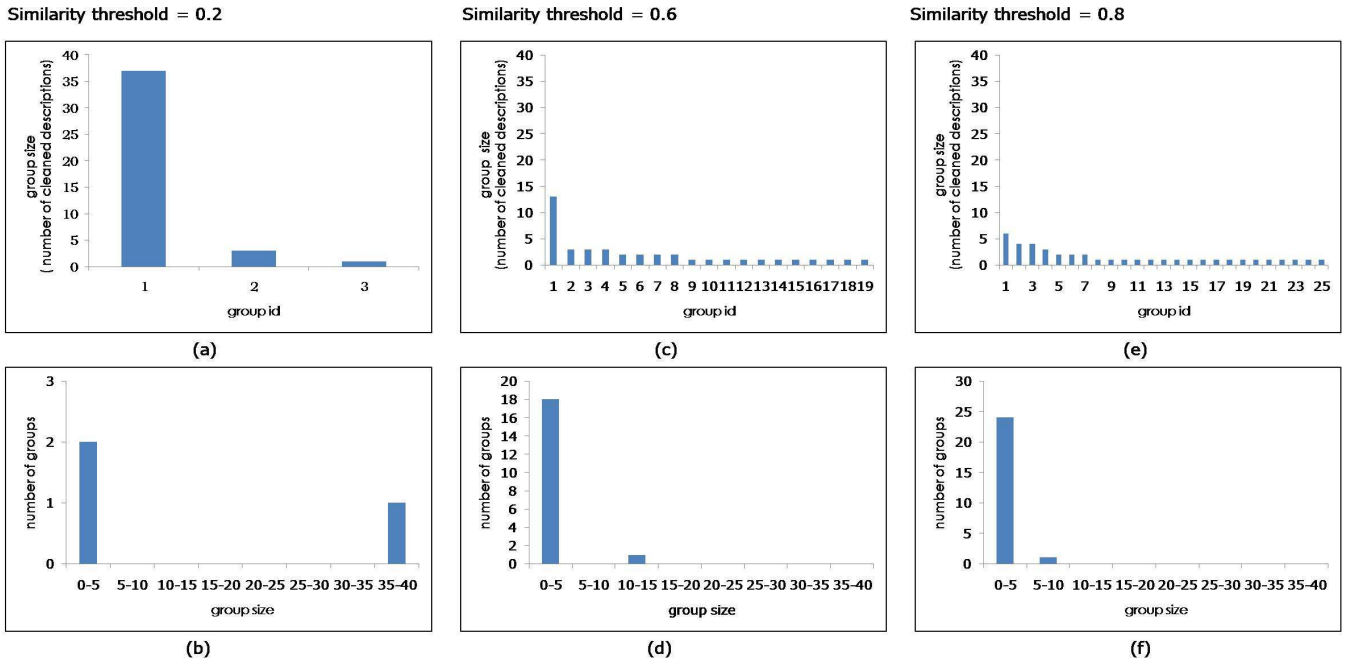

Figure 2: System-generated tickets: group id vs size of each group

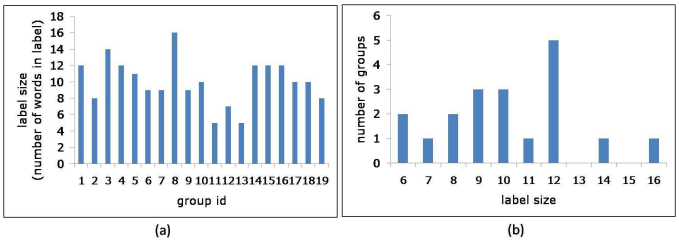

Figure 3: System generated tickets: (a) group id vs. label size. (b) label size vs. number of groups

4. *Evaluation of labels of clusters:* The next step is to assign label to these 19 clusters. As explained in Section 3, the set of common words from all descriptions within a cluster are probable candidate for a cluster label. We were able to assign long labels to most groups. Figure 3 (a) shows the label size for each of the 19 clusters. Average label size for each cluster was 10.05 words with maximum being 15 and minimum being 5 words. There is a structure and commonality in system-generated tickets. Hence we could generate long labels. Figure 3 (b) shows label size plotted against number of groups. We observe that there are 5 groups having label size of 12 words.

5. *Evaluation of change in threshold threshold_similarity:* Note that, the threshold used to compute similarity between cleaned descriptions impacts how many number of clusters are constructed. We experimented with threshold values 0.2, 0.6 and 0.8. Figures 2 (a) (c) (e) show the group ids and their size (number of cleaned descriptions) for thresholds 0.2, 0.6 and 0.8 respectively. Smaller threshold places loose constraint on similarity and hence groups many irrelevant descriptions together. Hence, the algorithm constructed only 3 clusters having average size of 13.66 cleaned descriptions with 0.2 threshold. On the other hand, a larger threshold of 0.8 places a very strict constraint on similarity and does not even group different variations of same issue. Hence, the algorithm constructed 25 clusters having average size of 1.68 cleaned descriptions with 0.8 threshold. Similarity threshold of 0.6 balances this trade-off making 19 groups with an average size of 2.16 cleaned descriptions. Figures 2 (b) (d) (f) show the group size plotted against the number of groups. We observe that as threshold increases, smaller groups are formed.

6. *Evaluation of similarity threshold against accuracy:* For measuring the impact of similarity threshold against accuracy, we selected similarity thresholds as 0.2, 0.6 and 0.8 for clustering and computed accuracy as 41.12%, 76.18% and 84.37% respectively. Accuracy increases with increase in similarity threshold because high threshold enforces more similarity and hence less accidental matching.

7. *Comparison with state of the art:* The most common approach to classify a ticket is based on CTI as discussed in [12] and [13] and explained in Section 1. We compare the accuracy of proposed approach with that of CTI. It can be seen that the accuracy of CTI is 55.34 % while that of proposed approach is 84.37 %.

4.2 Evaluation of the approach for user-generated tickets

In this case-study, we applied the proposed approach for user-generated tickets on Windows domain of a major retailer in the US. We analyzed ticket history of 8 months

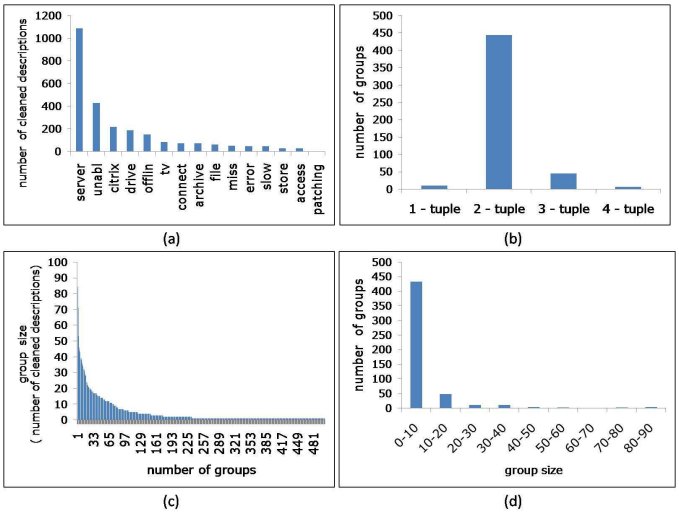

Figure 4: User generated tickets: (a) keywords Vs. number of cleaned descriptions (b) n-gram Vs. number of groups (c) number of groups Vs. group size (d) group size Vs. number of groups

where 6909 tickets were produced. Out of total 6909 tickets, we were able to extract issues for 89.34% of tickets. We verified the correction of extracted issues from domain experts. 81.63% of these issues were correctly extracted.

1. *Evaluation of data preparation:* We added 51 domain-specific words to the *Include list* such as *citrix*, *putty*, *sharepoint*, and *ip*. We observed that 4496 tickets (65.07%) contained domain words present in the *Include list*. We excluded some words such as *user*, *warning*, *etc*. We observed that 1629 tickets (23.58%) contained the words present in the *Exclude list*. We received total 6909 ticket descriptions. After data preparation step, these ticket descriptions got converted into 3460 cleaned descriptions.

2. *Evaluation of service catalog mapping:* We next demonstrate the impact of using service catalog mapping. The service catalog of Windows domain consisted of 44 items of frequently occurring issues. We used proposed algorithm on the cleaned descriptions and observed a match of 2136 tickets (30.92%). 20 service catalog items mapped to 363 cleaned descriptions. We observed that there were two items which were dominant and covered 310 cleaned descriptions. Service catalog item of *Monitoring Disk Utilization* covered 198 different cleaned descriptions and other item of *Limited Permissions FileShare* covered 112 different cleaned descriptions. By using service catalog mapping algorithm, we were able to map 363 out of 3460 cleaned descriptions to 20 service catalog items. After service catalog mapping, 3097 cleaned descriptions remained corresponding to 4043 tickets.

3. *Evaluation of clustering algorithm for user-generated tickets:* Unlike system-generated tickets, we observe many variations in user-generated tickets. We selected 15 words as keywords from 883 words based on frequency and developed n-grams from them. Depending upon how the issue is written and what type of words are used to write the descriptions, the issues can be explained by a k-tuple. For this dataset, we considered maximum 4-tuples. We applied the proposed n-gram clustering approach to group the remaining 3097 cleaned descriptions and were able to form 505 groups. Figure 4 (a) shows 15 1-tuples with their corresponding number of tickets. 1089 (35.16%) of cleaned descriptions contained keyword *server*. We obtained 505 groups of n-grams from these 15 keywords. Figure 4 (c) shows all 505 n-grams plotted against their sizes where size of a n-gram denotes number of cleaned descriptions that are contained in this n-gram. The top 20 most frequent n-grams cover 856 out of 3097 tickets (27.64%). These 505 groups contain a mixture of 1-tuple, 2-tuple, 3-tuple and 4-tuple depending on how these tuples explain the issue. Figure 4 (b) shows n-grams plotted against number of groups. We observe that majority of the groups, that is 444 are explained by 2-tuple. Figure 4 (d) shows group size plotted against number of groups. We observed that many small groups were formed. We discovered 433 groups that have less than 10 cleaned descriptions.

4. *Effect of number of keywords on coverage of tickets:* The number of keywords used to form n-grams has a direct impact on the number of tickets for which issues are extracted by the n-grams. We now evaluate the effect of varying the number of keywords used for issue extraction. We evaluated the algorithms for varying number of keywords ranging from 5 to 100. Figure 5(a) shows the number of keywords plotted against the number of tickets for which issues were extracted. As expected, the number of tickets for which issues are extracted increases with increasing number of keywords. However, for the given data-set the increase is only marginal after 30 keywords. Our experience on various data-sets indicate that, in most cases, top 50 keywords are sufficient to extract issues of more than 95% of the tickets.

5. *Evaluation of size of n-gram on correctness of issues:* The size of n-gram impacts the level of details captured about an issue. A single-tuple often fails to give sufficient details of the issue, such as *server*, or *application*. Instead, a 2-tuple or a 3-tuple better captures an issue, such as, *server failure*, or *application slow*. We refer to the maximum value of n that is used for creating n-grams as *max_n*. For the given set of tickets, we fixed the number of keywords and extracted issues with changing value of *max_n*. We then computed the correctness of the extracted issues for each experiment. Figure 5(b) shows the effect of different values of *max_n* on the correctness of the extracted issues. The correctness of issues increases with increasing value of *max_n*. However, for the given dataset we were able to extract correct issues for most of the tickets with 2-tuples. Only few tickets required the 3-tuples or 4-tuples to completely explain the issue. Hence, increasing the value *max_n* to 3 and 4 provided only marginal increase in correctness.

6. *Comparison with state of the art:* The most common approach to classify a ticket is based on CTI as dis-

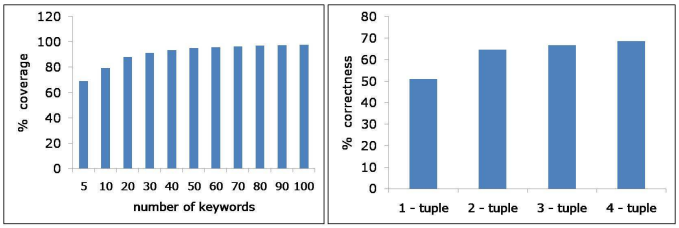

Figure 5: User generated tickets: (a) number of keywords vs % coverage (b) n-gram vs %correctness

cussed in [12] and [13] and explained in Section 1. We compare the accuracy of proposed approach with that of CTI. It can be seen that the accuracy of CTI is 28.56 % while that of proposed approach is 81.63 %.

5. RELATED WORK

Many researchers in the past have worked on performing various types of analysis on alerts and tickets based on various objectives. Some of these objectives are

1) *Automated problem inference*: Authors in [17] have proposed a system that aims to do automated problem inference for network trouble tickets.

2) *Resolver analysis*: Authors in [14] provide statistics-based discovery algorithms to identify experts, specialists and highly experienced people for specific service tasks.

3) *Problem occurrence pattern*: Authors in [12] develop a subgroup discovery technique and demonstrate its use to identify expensive problems in an IT Infrastructure Support (ITIS) organization. Authors in [13] propose statistics-based discovery algorithms to answer specific business questions related to ITIS operations such as identifying heavy hitter problems, identifying problems facing unusually high SLA violations and identifying problems suitable for automation, etc.

All above work requires knowledge of issues. Our work falls in the category of extracting issues from text. Ticketing tools such as ServiceNow [1], Remedy [2], etc. provide drop-down menus for classification of issues in classes such as Category, Type, Item and Summary (CTIS). However, the classification is often coarse grained, as the quality of classification is dependent upon the customization by the organizations. Further, the resolvers and users make mistakes in categorizing the issue. As a result, the analysis based on this classification is often misleading.

6. CONCLUSION

In this paper, we address the problem to mine textual descriptions in tickets to extract issues. We propose use of informational retrieval techniques along with domain knowledge to extract issue. We capture the knowledge of IT and business domains in the form of include and exclude lists of words and regular expressions. We propose two different algorithms to extract issues for system and user-generated tickets. In system-generated tickets, the algorithm is based on clustering descriptions based on similarity, while for user-generated tickets, we propose keyword based approach to cluster similar descriptions.

We have applied the proposed ideas on several real-world case-studies. The proposed algorithms were able to derive issues that were otherwise difficult to extract. We demonstrate the proof-of-concept of proposed approach using two real-world case-studies. One for system-generated tickets and other for user-generated tickets. In system-generated tickets, by using the proposed approach we were able to correctly extract the issues for 76.16% of tickets. Whereas, in user-generated tickets, we were able to correctly extract the issues for 81.63% of tickets.

7. REFERENCES

[1] `http://www.servicenow.com/products/it-service-automation-applications.htm%l`.

[2] `http://www.bmc.com/it-solutions/it-service-management.html`.

[3] `http://www.itil-officialsite.com/aboutitil/whatisitil.aspx`.

[4] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[5] Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

[6] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[7] Daniel Freedman and Tao Zhang. Interactive graph cut based segmentation with shape priors. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 755–762. IEEE, 2005.

[8] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[9] George Karypis and Vipin Kumar. Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[10] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[11] Makoto Nagao and Shinsuke Mori. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of japanese. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 611–615. Association for Computational Linguistics, 1994.

[12] Maitreya Natu and Girish Keshav Palshikar. Interesting subset discovery and its application on service processes. In *Data Mining for Service*, pages 245–269. Springer, 2014.

[13] Girish Keshav Palshikar, Harrick M Vin, Mohammed Mudassar, and Maitreya Natu. Domain-driven data mining for it infrastructure support. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 959–966. IEEE, 2010.

[14] Girish Keshav Palshikar, Harrick M Vin, V Vijaya Saradhi, and Mohammed Mudassar. Discovering experts, experienced persons and specialists for it

infrastructure support. *Service Science*, 3(1):1–21, 2011.

[15] Joseph J Pollock and Antonio Zamora. Automatic spelling correction in scientific and scholarly text. *Communications of the ACM*, 27(4):358–368, 1984.

[16] Martin Porter. Snowball: A language for stemming algorithms, 2001.

[17] Rahul Potharaju, Navendu Jain, and Cristina Nita-Rotaru. Juggling the jigsaw: Towards automated problem inference from network trouble tickets. In *NSDI*, pages 127–141, 2013.