DRACA: Decision Support for Root Cause Analysis and Change Impact Analysis for CMDBs

Sarah Nadi[+], Ric Holt[+], Ian Davis[+], and Serge Mankovskii[*]

University of Waterloo[+], CA Labs Canada[*]

Abstract

As business services become increasingly dependent on information technology (IT), it also becomes increasingly important to maximize the decision support for managing IT. Configuration Management Data Bases (CMDBs) store fundamental information about IT systems, such as the system's hardware, software and services. This information can help provide decision support for root cause analysis and change impact analysis. We have worked with our industrial research partner, CA, and with CA customers to identify challenges to the use of CMDBs to semi-automatically solve these problems. In this paper we propose a framework called DRACA (Decision Support for Root Cause Analysis and Change Impact Analysis). This framework mines key facts from the CMDB and in a sequence of three steps combines these facts with incident reports, change reports and expert knowledge, along with temporal information, to construct a probabilistic causality graph. Root causes are predicted and ranked by probabilistically tracing causality edges backwards from incidents to likely causes. Conversely, change impacts can be predicted and ranked by tracing from a proposed change forward along causality edges to locate likely undesirable impacts.

1 Introduction

Many organizations today commonly rely on an underlying Information Technology (IT) system for their day-to-day operations and to deliver services to their customers. As organizations' functionalities become dependent on the operation of the underlying IT infrastructure, the management of that IT infrastructure becomes more crucial. Configuration Management Databases (CMDBs) have emerged to help in managing these systems.

A CMDB is used to store information about the various critical components in a system including hardware, software and services provided by the company (see Figure 3). It records information about these items, their change history, their incident history, as well as the relationships between them. Each item stored in the CMDB is referred to as a Configuration Item (CI). The data in the CMDB can be represented as a graph where each CI is a node and relationships between CIs are edges. Thus, the CMDB serves as the repository for important information about a system which can be used in decision making processes such as root cause analysis and change impact analysis [9]. CMDBs have been adopted by organizations as they realize the need to reduce financial losses caused by problems in their IT systems. CA is one vendor of the CMDB [1].

We have been investigating CMDBs from the perspective of CA CMDB experts, as well as from the perspective of three of CA customers who have been using CMDBs. From our discussions, we identified key usages of the CMDB as well as miss-

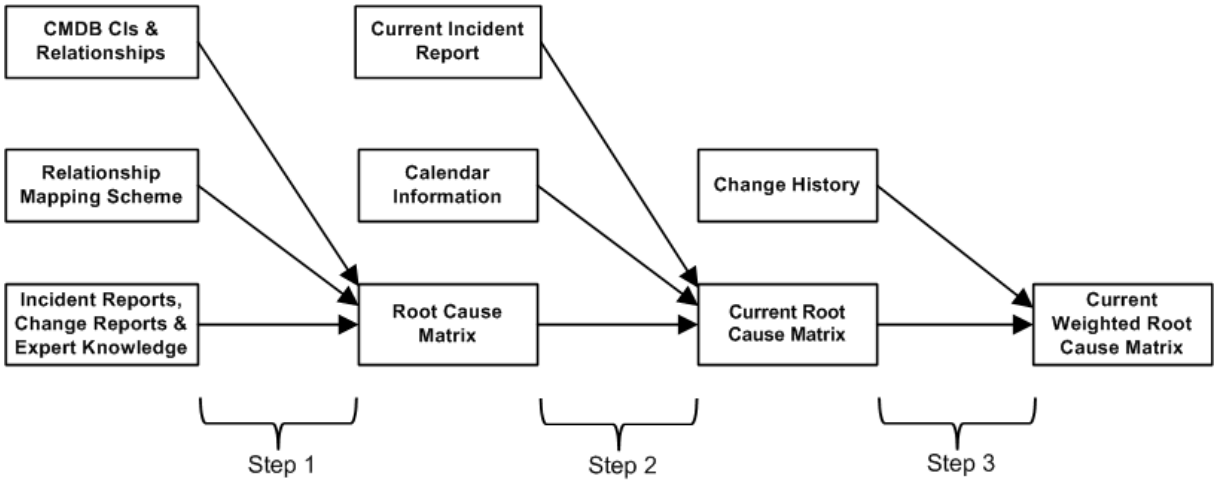

Figure 1: DRACA Framework

ing features from the customers' perspective.

All our interviewees agreed that root cause analysis and change impact analysis are important processes in their respective organizations. Although the CMDB provides a visualization of the system which allows IT personnel to understand which CIs are related to the CI having an incident, it was agreed that having semi-automated root cause analysis which provides a list of the CIs to check would be valuable. Accurate change impact analysis was also considered valuable because a change that is not thoroughly studied before its implementation can lead to costly outages.

Another interesting point that came up in our discussions is that of temporal constraints on the relationships in an IT system. That is, some relationships only exist at certain times. These are commonly scheduled events such as build processes or backup processes that only occur at specified times in the calendar of a system. The fact that these relationships do not exist at all times should be taken into consideration while performing both root cause analysis and change impact analysis.

Based on these field studies, we developed the DRACA framework that addresses concerns and problems CMDB users face. We present DRACA as a decision support framework for semi-automated root cause analysis and change impact analysis. This paper focuses mainly on root cause analysis and only briefly discusses change impact analysis. More details on change impact analysis will be provided in our future work.

DRACA consists of three steps that are shown in Figure 1. Step 1 produces a root cause matrix based on the CIs and their relationships in the CMDB and a defined relationship mapping scheme. The probabilities used in this matrix are mined from incident reports, change reports and expert knowledge. Step 2 examines the calendar information of the system and correlates it with the time of the current incident to produce a root cause matrix that matches this time. Step 3 considers the change history of CIs to weigh each one according to its last time of change. Changes closer to the time of the incident will get higher weights. This step produces the current weighted root cause matrix which shows the probability of each CI being a root cause of the current incident. Based on these probabilities, a ranked list of suspect CIs can be produced.

The rest of this paper is organized as follows: section 2 defines terminology used in this paper. Section 3 provides background information. Section 4 presents the DRACA framework in details, explaining each of its three steps. Section 5 discusses challenges involved in root cause analysis. Section 6 presents related work. Section 7 suggests ideas on how this work could progress. Section 8 concludes this paper.

Figure 2: Faults, Failures and Incidents

2 Terminology: Faults, Failures and Incidents

The terminology used in this paper follows the standards of the IT Infrastructure Library (ITIL) [8]. A *fault* is a design flaw or malfunction that causes a *failure* of one or more CIs or IT Services. A *failure*, generally caused by a fault, is the loss of ability to operate to specification, or to deliver the required output. A failure may cascade to cause more failures in other CIs. A failure, or a failure cascade, may eventually cause an *incident*. An *incident* is an event that is not part of the standard operation of a service and that causes, or may cause, an interruption to, or a reduction in, the quality of that service. An incident is externally observable and is usually recorded in an incident report. Figure 2 illustrates these definitions.

Accordingly, a *root cause* is the underlying original fault leading to a particular incident. *Root cause analysis* tries to map an incident to its underlying fault. A *change* is the addition, modification, or removal of anything that could have an effect on IT services. A poorly planned change may lead to a fault in the system. *Change impact analysis* tries to predict if a change will cause a fault which may lead to an incident.

For example, the inability to view a certain web page is an incident. A failure associated with this incident may be the inability to access the web server hosting the web page. The root cause of the problem may be that someone changed the settings in the Domain Name Server (DNS), and thus the web server is no longer reachable by that name.

3 Background Information

3.1 CMDBs

ITIL [9] is a set of standards concerned with providing best practices for business effectiveness and efficiency in the use of information systems. ITIL defines configuration management as "the process of identifying and defining configuration items in a system, recording and reporting the status of configuration items and requests for change, and verifying the completeness and correctness of configuration items". A configuration item (CI) is a component of an IT infrastructure which is put under the control of the configuration management process. The goal of configuration management, apart from accounting for all IT assets and their configurations, is to provide decision support for change management, incident management, problem management, and release management [9]. A CMDB can either be a unified or federated database [6]. This means that internally, it can be a single database or a collection of databases which appear to the user collectively as one database through a unified interface.

A CMDB, usually, provides visualization capabilities to view the different CIs in the system and the relationship between them. Figure 3 shows an example CMDB to illustrate the concepts of CIs and relationships. We will use this small example throughout the paper to apply the concepts in our framework. The different CIs in the graph have been numbered in order to simplify reference to them in the rest of the paper.

3.2 Causality Graph

A causality graph, as its name indicates, is a graph that records the cause and effect relations among different components of a system. The graph consists of nodes (for the components), and edges (for the relations). A causality edge from x to y shows that a problem in x could cause a problem in y. Thus, a path on this graph shows a cause-effect chain. Causality graphs should have some way of modeling the cause and effect strength of the relationships, usually through probabilities. We attach probabilities to the edges of the graph to accomplish that. Figure 4 shows an example causality graph that is mapped from the CMDB in Figure 3.

For example, in Figure 4 the probability on the edge from node 1 to node 2, $K_{12} = 0.4$ shows that if there is a fault or failure in node 1, there is a 40% chance that it will cause a failure in node 2. We as-

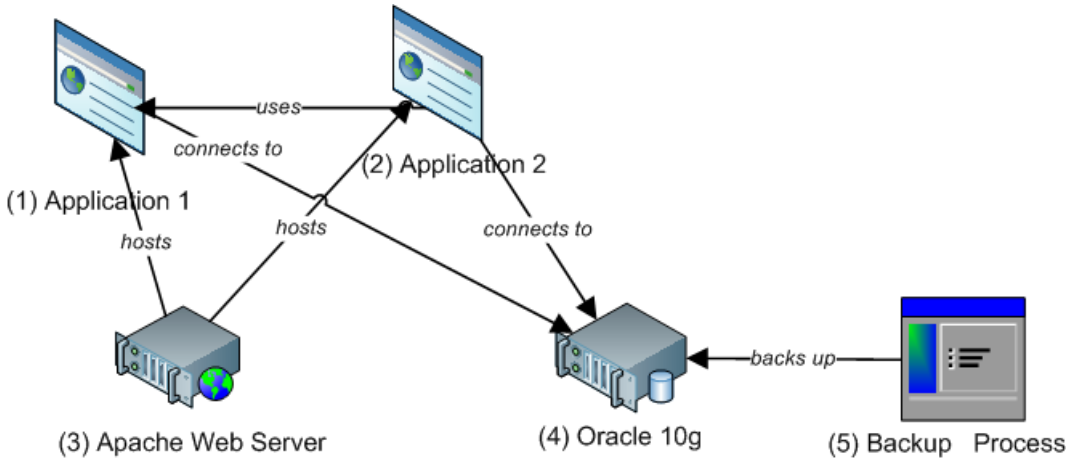

Figure 3: Example CMDB

sume that the propagation of errors along a node's outgoing edges are independent events, and are not mutually exclusive. This means that an error in one node can simultaneously propagate along more than one edge that leaves from that node. Accordingly, the probabilities on the edges coming out of a node do not have to add up to 1.0. For example, probabilities on the edges propagating out from node 4 (0.6 and 0.9) add up to 1.5, and not to 1.0.

4 DRACA: A Decision Support Framework for Root Cause Analysis and Impact Analysis

As explained in the introduction, in order to produce a ranking for suspect root causes of an incident, many factors must be taken into consideration. DRACA goes through three steps to accomplish this (see Figure 1). Step 1 produces the root cause matrix which maps any CI to its possible root causes (section 4.1). Once the root cause matrix is produced from step 1, step 2 checks the time of the incident and includes the system's calendar to produce a root cause matrix that corresponds to the time the incident was reported (section 4.2). Finally, step 3 considers the changes and events that occurred before the incident to weigh the CIs according to their time of change (section 4.3). After this step, we will have a final root cause matrix that effectively gives us a ranking of the suspect root causes of the current incident. This completes our overview of the three step DRACA framework; we will now discuss these steps in detail.

4.1 Step 1: Producing the Root Cause Matrix

The root cause matrix is produced in three substeps. Step 1a produces the causality graph by mapping the CIs in the CMDB into nodes, and mapping the CMDB relationships into causality edges according to a defined mapping scheme. Step 1b attaches probabilities to the causality edges by mining incident reports, change reports, and expert knowledge. Step 1c stores these probabilities in a causality matrix K and calculates the probability that each CI could be the root cause of another CI. This probability will be stored in the root cause matrix R.

Step 1a: Mapping CMDB Relationships into Causality Edges

The data available in a CMDB records how the different CIs interact together as shown in Figure 3. Based on these relationship meanings, we develop a mapping into causality edges. For example, in Figure 3, Application 2 connects to the Oracle database, so if the Oracle database has any problem, Application 2 will malfunction because it can no longer access the database. In other words, a problem in Oracle can cause a problem in Application 2. In a causality graph, this would map to an edge going from Oracle (node 4) to Applica-

tion 2 (node 2) as shown in Figure 4. To achieve this, we develop a mapping scheme to convert the different relationships in the CMDB into causality relationships. Table 1 shows the scheme we use to map the CMDB in Figure 3 to the causality graph in Figure 4.

CMDB Relationship	Causality Mapping
A hosts B	A causes B
A connects to B	B causes A
A backs up B	A causes B
A uses B	B causes A

Table 1: Relation Mapping Scheme

In a real system, there would be more relationships defined in a CMDB. Additionally, as time passes, CMDB managers may add new types of relationships to fit their needs. In such a case, they should enhance the mapping scheme to include the new relationships.

Step 1b: Estimating the Probabilities

Given the causality edges calculated in step 1a, step 1b estimates the probabilities on the edges. Estimating the strength of the causality edges, which we represent as probabilities, is one of the most challenging parts of root cause analysis. This is done by mining the incident reports that were prepared for previous incidents. The reports will contain information about the root cause of the incident, and how it was solved. Additionally, some incident reports may contain information about other related failures or symptoms which occurred at the same time of this incident. This is useful in estimating the probability on the edges. Simple counting techniques such as how many times CI x had a reported incident (count(x)) and how many times CI y was the cause of this incident (count($y \rightarrow x$)) can give us an estimate for the strength of the relation between y and x in the form of count($y \rightarrow x$)/count(x).

Change reports are another source of relevant information. When a change needs to be implemented, related CIs that need to be checked and other changes that need to implemented as a result are usually documented. The same counting technique can be used to estimate the strength of the relationship between two CIs. If two CIs commonly change together then there is a greater chance that these CIs are related and that a fault in one can cause a failure in the other.

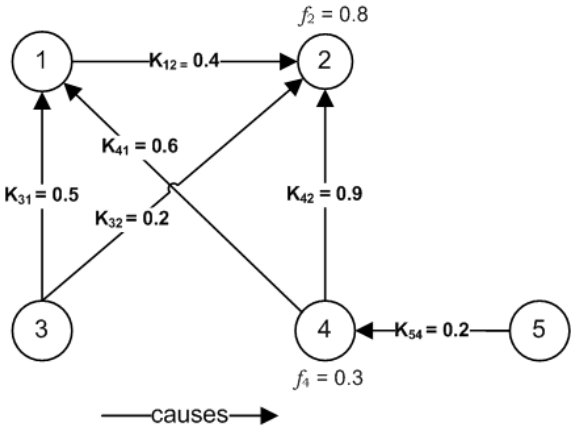

Figure 4: Example Causality Graph

Finally, expert knowledge is an important source of information. If the above information is not well documented, experts who have been working with the system for years could provide it as they have extensive knowledge about the things that go wrong, and what are their most likely root causes. We use these sources of information to estimate the probabilities shown in Figure 4. Ideally, incident reports and change reports would be used to estimate the probabilities, and experts would verify this information or provide additional information when proper documentation is missing. However, the amount of reliance on expert knowledge greatly depends on the quality of historical information present.

Step 1c: Calculating Root Cause

Steps 1a and 1b produce the causality graph that we will now use to produce the root cause matrix. In order to do this, we store this causality graph in a matrix K. Given a causality graph with n nodes, the edges in the graph which represent a direct causal relation between two nodes will be stored in the n by n matrix K. The probability that CI i might cause a problem in CI j is found in the entry K_{ij}. The following causality matrix K stores the probabilities shown in the graph in Figure 4.

$$K = \begin{pmatrix} 0.0 & \mathbf{0.4} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.5} & \mathbf{0.2} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.6} & \mathbf{0.9} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & \mathbf{0.2} & 0.0 \end{pmatrix}$$

Representing this information in a matrix provides a convenient way of calculating indirect cause propagation. An entry K_{ij} is the probability of an error propagating from i to j through a direct edge (i.e a path of length one). Now, consider K^2, which is the square of K, i.e., K multiplied by K. Entry K^2_{ij} is the probability of propagating from i to j through any path of length two. In order to calculate this probability, we must consider all paths of length two from i to j and assume we take any, some or all of them. This probability is calculated by taking taking the "or" of all possible two step paths from i to j according to the probability rule: $P(A \cup B) = P(A) + P(B) - P(A) * P(B)$ where A and B are independent events. We use this probability rule when multiplying the matrices and when adding them to compute the transitive closure matrix.

Let us consider the graph in Figure 4, and consider the ways a problem in node 5 can propagate to affect node 2. To start with, $K_{52} = 0$ since there is no path of length one from node 5 to 2. Now, to calculate K^2_{52}, we need to consider going from node 5 to node 2 in any path of length two. Only one such path exists in our graph: $5 \rightarrow 4 \rightarrow 2$. Therefore, K^2_{52} is calculated as follows:

$$\begin{aligned} K^2_{52} &= 0.2 * 0.9 \\ &= 0.18 \end{aligned}$$

Similarly, K^3_{52} is calculated through the path $5 \rightarrow 4 \rightarrow 1 \rightarrow 2$ as follows:

$$\begin{aligned} K^3_{52} &= 0.2 * 0.6 * 0.4 \\ &= 0.048 \end{aligned}$$

When we add the probabilities from all paths leading from i to j in one step (K_{ij}), two steps (K^2_{ij}) ... ∞ steps (K^∞_{ij}), we will get the overall probability that an error or failure in i could propagate to j through a path of any length. In other words, we are computing the transitive closure of matrix K which we shall call T:

$$T_{ij} = \sum_{l=1}^{\infty} K^l_{ij} \tag{1}$$

T_{ij} gives the probability that an error or failure in i causes a failure in j through a path of any length. Following equation 1, the matrix T below, rounded to two decimal places, is the transitive closure of the given matrix K.

$$T = \begin{pmatrix} 0.0 & \mathbf{0.4} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.5} & \mathbf{0.36} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.6} & \mathbf{0.92} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.12} & \mathbf{0.22} & 0.0 & \mathbf{0.2} & 0.0 \end{pmatrix}$$

An entry T_{ij} shows the probability that a failure in i could propagate to j. For example, $T_{52} = 0.92$ means that if node 5 has a failure, there is a 92% chance that this failure will propagate to node 2.

We define the *fault proneness* f_i of a node i as the probability that i has a fault of its own accord. For example, $f_1 = 0.4$ means that there is a 40% chance that node 1 will initiate a fault. Using f_i and T_{ij}, we can compute R_{ij}, the probability that node i could be the root cause of a failure in node j, as follows:

$$R_{ij} = f_i T_{ij} \tag{2}$$

This calculates the probability that node i had a fault that propagated, directly or indirectly to cause a failure in j. Therefore, the term R_{ij} gives us the probability that a failure in j has i as its root cause. For illustration, assume f_i is the following vector:

$$f = \begin{pmatrix} 0.4 \\ 0.8 \\ 0.1 \\ 0.3 \\ 0.2 \end{pmatrix}$$

The fault proneness of each node is also shown on the causality graph in Figure 4. According to Equation 2, R would have the following probabilities rounded to two decimal places:

$$R = \begin{pmatrix} 0.0 & \mathbf{0.16} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.05} & \mathbf{0.04} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.18} & \mathbf{0.28} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.02} & \mathbf{0.04} & 0.0 & \mathbf{0.04} & 0.0 \end{pmatrix}$$

From the above matrix, $R_{42} = 0.28$ means there is a 28% chance that node 4 is the root cause of an incident in node 2.

4.2 Step 2: Producing the Current Root Cause Matrix

In step 1, we produced a root cause matrix based on the relations in the CMDB and other information such as incident reports. However, one such static view of the system may not always yield accurate results if certain aspects change from one point in time to another. Therefore, we take the calendar information of the system into consideration while performing root cause analysis. For example, if a backup process runs every night at 8:00 pm to backup a database server, then the relationship between the backup process and the database server only exists at that time. Such information should be considered to produce more accurate analysis.

Depending on the time of the incident being examined, we will adjust our root cause matrix according to the calendar information available. Using the example in Figure 4, if the backup process (node 5) only takes place on Wednesdays from 7:00 - 8:00 pm, then the edge from node 5 to node 4 will only exist at that time. Accordingly, if we receive an incident on Sunday at 2:00 pm, we look at our calendar information and check if there are any causality edges we can exclude to improve the analysis. In this case, we can exclude the edge from node 5 to node 4 since this relationship does not exist on Sunday at 2:00 pm.

After examining the calendar information, we exclude irrelevant edges according to the time of the incident which means that our causality matrix K will change. We then calculate a new current root cause matrix according to the new edges being considered. To calculate the current root cause matrix, we follow the calculations done in step 1c. If no calendar information is available for the system, then we simply use the root cause matrix calculated in step 1 as our main matrix for all times. In practice, we can skip generating the root cause matrix in step 1c, and simply generate it as needed in step 2.

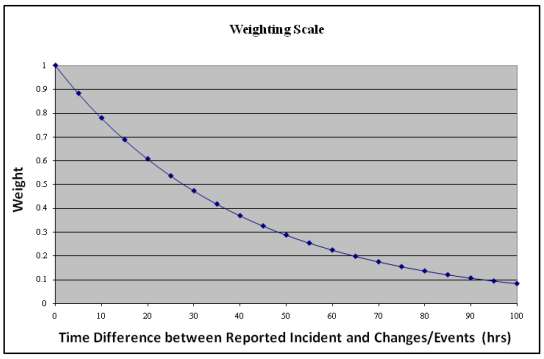

Figure 5: Weighting Scale

4.3 Step 3: Producing the Current Weighted Root Cause Matrix

After calculating the current root cause matrix in step 2 by considering the time of the incident and the available calendar information, we still need to consider the events and changes that actually happened in the system before the incident. This follows from the idea that if X causes some incident in Y, X has to occur before Y. Additionally, the fault or failure in X should occur at a time that is reasonably close to the time of the incident in Y. For example, it is more likely that a change or event that happened 2 hours before the incident occurred could be its root cause versus some other event that happened two days ago.

To do this, we compare the time of the incident and the time of the last change that occurred to the other CIs such that a higher weighting is given to CIs that had events or changes which took place closer to the time of the incident. A weighting scheme is needed to accomplish this. The choice of the weighting scheme does not affect the rest of the calculations we do. The main idea is to give decreasing weights to changes or events as they occur further back from the time of the incident. We choose to use an exponentially decaying weighting scheme from 1.0 to 0.0. The rate of decay, however, would depend on the nature of the system. The exponential weighting scheme we use in this example is shown in Figure 5.

According to when the last time a CI changed or an event associated with it occurred, its row in the root cause matrix, R, would be multiplied by its temporal weight. Accordingly, CIs which have not changed in a long time will have a very low weight

CI	Last Change	Weight
1	January 1, 2009 10:00 am	0.00001
3	May 4, 2009 10:00 pm	0.94
4	April 30, 2009 2:00 pm	0.09

Table 2: Temporal Weights

factor (almost 0) which will greatly decrease their probability to be a suspect root cause of the current incident.

Assume we have received an incident for CI 2 on Monday May 4, 2009 at 2:00 pm. Table 2 shows the different change times of the CIs, and their temporal weight according to the weighting scale shown in Figure 5. For example, since the last change in CI 4 happened on April 30, 2009 at 2 pm (i.e., 96 hours from the time of the current incident), its weight will be 0.09 according to the graph in Figure 5.

We then multiply the temporal weight of each CI by its corresponding row in the root cause matrix R to get the new probabilities. Accordingly, the probabilities that were shown in R in step 1c (See page 6) will now change to those in $R_{weighted}$. These are shown below, rounded to three decimal places. Note that since the incident occurred on a Monday, the edge from node 5 to node 4 is not considered as this edge only exists on Wednesdays from 7:00 - 8:00 pm as shown in step 2. That is why all of row 5 is zeros in $R_{weighted}$, and we do not need to worry about the changes done to CI 5.

$$R_{weighted} = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ \mathbf{0.047} & \mathbf{0.034} & 0.0 & 0.0 & 0.0 \\ \mathbf{0.016} & \mathbf{0.025} & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Since the incident is in CI 2, we look at column 2 in the root cause matrix to determine the ranking of the suspect CIs. Without temporal information, our most likely suspect for the incident in CI 2 was CI 4 (with a 28% chance) followed by CI 1 (with a 16% chance) as shown in matrix R from step 1c (See page 6). With temporal information, our primary suspect now becomes CI 3 (with a 3.4% chance) followed by CI 4 (with a 2.5% chance) as shown in matrix $R_{weighted}$ above. This makes sense since CI 3 had a change very close to the reported time of the incident.

5 Discussion

In this paper, we have proposed DRACA as a framework for root cause analysis and change impact analysis. We have not discussed change impact analysis in this paper, but the main information needed to perform impact analysis is present in our framework. As with any model or framework, the accuracy of the produced results is a function of the quality of the data available. The more detailed the information stored about incidents, their root causes, and side effects, the more accurate the estimated probabilities are, and thus the more accurate the root cause analysis is. Since ITIL standards are becoming increasingly popular and are being adopted by organizations, we believe that more companies would adhere to these standards and have detailed documentation of their incidents. Additionally, as the need arises, the structure of the reports may eventually change to include more information that will be useful to future root cause and change impact analysis.

In order to have an effective framework, learning from the results previously done analysis is crucial. When an incident is reported, and the framework reports a suspected root cause, this suspect CI should be fixed, and then the problem should be re-evaluated to determine if it is still there or not. Whether the problem was fixed or not, and accordingly, whether the identified root cause was accurate or not should be stored in the incident reports. Every certain amount of time, the causality graph would be updated with the new information mined from these incident reports.

As incidents occur and get fixed, the totality of the picture may change and the strength of the causality edges may change accordingly. The frequency with which the causality graph would be updated would depend on the nature of the system at hand. If the system being managed is a highly dynamic system with many incidents occurring, then a higher frequency of updating would be needed. On the other hand, if the rate of incidents is low, then the initial causality graph would be more or less stable, and a lower frequency of updating would be needed.

One of the challenges with using a causality graph is how to deal with cycles in the causality graph. Practically, a cycle implies that an error in a particular CI could cause a ripple of failures which loop around and cause another failure in the orig-

inal CI, repeatedly. This implies that the loop can be traversed many times. Although we chose not to address loops yet, as they are not very common in real systems, and most work in the field has chosen to ignore them, our model does, however, allow for them. Using matrix multiplication to calculate the transitive closure can approximate the impact of loops.

6 Related Work

Most of the work done on root cause analysis (commonly referred to as fault localization) has focused on communication networks and network related events (e.g [5]). However, the data in the CMDB provides much more detailed information than is available in networks. As discussed in [3], IT systems have become service oriented and incidents are dealt with at the service level instead of the network level. Root cause analysis in IT systems should, therefore, be done at a service-oriented level and not a device-oriented level.

To do this, Hanemann [4] proposes a hybrid rule-based/case-based reasoning approach to identify faults. Through a set of rules that map symptoms to root causes, the root cause of an incident is identified by searching the set of rules for matching symptoms. If no match can be found, this incident is treated on a case based approach by manually resolving it the first time it is encountered, and then storing it to the set of available cases for future incidents. The downside to this approach is that it is difficult to identify all the needed rules, and manually resolving each case might be very costly. Our approach replaces the manual work needed in the case-based reasoning part as it saves time by automatically producing a list of ranked results. Additionally, identifying root causes based on the changes and events that actually took place in the system rather than static rules should yield more accurate results.

Previous work on fault localization includes using Bayesian Belief Networks to model end-to-end services in a system to identify root causes of problems [10]. The technique is based on analyzing the symptoms in the system, and using a belief network to compute the Most Probable Explanation set. This work, however, only works on a high-level to pinpoint the service which has the problem according to the present symptoms, but does not go further down to identify the problem on a lower level. In a CMDB, however, relationships between services and other underlying CIs are present which makes discovering the actual root cause of a problem more accurate.

Other previous work [2] uses decision trees to identify faults in the system. This technique works by classifying failed and successful requests that occurred during the faulty period and maps this information to the paths on the tree to predict the possible sources of the error. As the authors themselves point out, decision trees are not very competitive in terms of correct prediction. The main advantage of their technique is that they yield human interpretable results. Causal graphs which take temporal aspects into consideration have the potential to yield better predictability rates, and would still provide human understandable results.

Recently, Natu and Sethi [7] proposed a technique for fault localization for wireless adhoc systems that incorporates temporal aspects in the analysis. Both our technique and theirs use a matrix to store the probabilities tied to edges representing causal implications between problems and their possible reasons. Natu and Sethi refer to them as faults and symptoms. The first difference is that in our matrix, the rows and columns are the same set of CIs as any CI may have an incident and any CI could be a possible root cause. We do not divide them into separate sets of faults and symptoms.

Additionally, the issues their work is addressing is a somehow different from ours. They are addressing wireless adhoc systems where nodes are continuously changing. In our work, the nodes themselves rarely change, and is thus not a big concern. Our concern is, however, the change in the edges between the nodes i.e if they exist at certain times or not. A common concern both techniques have, however, is the time of analysis, and thus the best model (stored matrix) is used according to the time of the incident.

7 Future Work

Our next step is to use simulation to test our model. That is, we will develop a small system and store its configuration in the CMDB. We will simulate faults causing a failure cascade which eventually causes an incident according to the probabilities stored in our model. We would then use the probabilities in

our model to identify the root causes of these incidents.

Additionally, it would be interesting to mix the hybrid technique [4] described above with our framework. Having a rule-based approach combined with a case-based approach that uses our model to automatically identify the root cause instead of manually finding it could produce an improvement to both techniques.

We are also currently analyzing sample data from CA's own CMDB that is used to manage their internal network. We are looking at the incident and change reports stored there in order to apply our technique. Once our technique is mature on this sample data, we hope to test it on different data sets.

8 Conclusion

This paper presented DRACA, a framework for performing root cause analysis and change impact analysis. We concentrated on root cause analysis in this paper, but the data needed for change impact analysis is also available in our framework. We demonstrated how the data in the CMDB can be mined to produce a causality graph, and how this graph can be represented in a matrix. Through manipulations on this matrix, we produced a root cause matrix which records the probability that one CI is the root cause of an incident in another CI. This is done through three steps: forming the basic root cause matrix from the data in the CMDB, considering the time of the reported incident along with the system's calendar information to adjust the current root cause matrix accordingly, and finally having different weights for the different CIs according to the last time they have changed to produce the final root cause matrix. The final root cause matrix effectively provides a ranked list of the suspect CIs for the current incident.

This work is unique in that it combines different aspects of root cause analysis into one framework. The problems we address are real world challenges which we have extracted from many discussions with CMDB users and CMDB experts. Working closely with CA Labs has given us an insight into the problems faced by CMDB users. DRACA lays the foundation for a framework that addresses these challenges.

Acknowledgments

This research is supported by a research grant from CA Canada Inc. This project is also partly funded by OCE and NSERC. We would also like to thank our participating CA customers for their contributions.

About the Authors

Sarah Nadi, BSc., The American University in Cairo, 2007, is currently pursuing her Master's degree at the University of Waterloo. Her research interests include software architecture and mining software repositories.

Richard C (Ric) Holt, PhD, Cornell University, 1971, is a Professor of Computer Science at the University of Waterloo. He has done basic work on deadlock theory, compilers, software architecture and mining software repositories. He is co-author of the Turing programming language.

Ian Davis, PhD, University of Waterloo, 1989 has been employed by the University of Waterloo since 1994 as a project manager and research associate. Research interests include software development, database, XML, visualization, and fact extraction.

Serge Mankovskii is a Research Staff Member with CA Labs. He has over 25 year of industry experience in operating systems, expert systems, machine learning, reasoning, telecommunication software, enterprise job scheduling and event-based enterprise integration.

References

[1] CA. `http://www.ca.com/us/`.

[2] M. Chen, A.X. Zheng, J. Lloyd, M.I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing*, 2004.

[3] A. Hanemann. *Automated IT Service Fault Diagnosis Based on Event Correlation Techniques*. PhD thesis, Universitat der Bundeswehr Munchen, 2007.

[4] A. Hanemann, M.N.M. Team, L.S. Center, and G. Munich. A hybrid rule-based/case-based reasoning approach for service fault di-

agnosis. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, 2006.

[5] S. Kandula, D. Katabi, and J.P. Vasseur. Shrink: a tool for failure diagnosis in IP networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 173–178. ACM New York, NY, USA, 2005.

[6] H. Maddurt, SB Shi, R. Baker, N. Ayachitula, L. Shwartz, M. Surendra, C. Corley, M. Benantar, and S. Patel. A configuration management database architecture in support of IBM Service Management. *IBM SYSTEMS JOURNAL*, 46(3):441, 2007.

[7] M. Natu and A.S. Sethi. Using temporal correlation for fault localization in dynamically changing networks. *International Journal of Network Management*, 18(4), 2008.

[8] Office of government commerce (ogc), ed.: Glossary of terms, definitions and acronyms v3. *IT Infrastructure Library (ITIL)*, 2007.

[9] Office of government commerce (ogc), ed.: Service support. *IT Infrastructure Library (ITIL)*, 2000.

[10] M. Steinder and AS Sethi. Probabilistic fault localization in communication systems using belief networks. *Networking, IEEE/ACM Transactions on*, 12(5):809–822, 2004.