Assessing Operational Impact in Enterprise Systems by Mining Usage Patterns

Mark Moss and Calton Pu

CERCS, Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA 30332
{markmoss,calton}@cc.gatech.edu

Abstract. Performing impact analysis involves determining which users are affected by system resource failures. Understanding when users are actually using certain resources allows system administrators to better assess the impact on enterprise operations. This is critical to prioritizing system repair and restoration actions, and allowing users to modify their plans proactively. We present an approach that combines traditional dependency analysis with resource usage information to improve the operational relevance of these assessments. Our approach collects data from end-user systems using common operating system commands, and uses this data to generate dependency and usage pattern information. We tested our approach in a computer lab running applications at various levels of complexity, and demonstrate how our framework can be used to assist system administrators in providing clear and concise impact assessments to executive managers.

Keywords: operational impact analysis, system management, data mining.

1 Introduction

An important question in system management is determining which users are affected by system resource failures [1]. System administrators are often tasked to assess the impact of system resource failures on business operations, which we refer to as *assessing the operational impact*, and to present this assessment to management executives. Administrators must present this information in a manner that is clear and operationally relevant to the executives. An example assessment is: "The purchasing department users will not be able to access their invoice applications for the next two hours because our application server has failed", which connects the effects of the technical event clearly to one or more business operations. It also captures the concept that an *operational* impact occurs only if the purchasing department needs to access the invoice application during the outage period. Taking this usage information into account makes the impact assessment significantly more effective.

Consequently, producing effective operational impact assessments requires: (1) connecting lower-level technical events to their impacts on higher-level, user-relevant resources; and, (2) integrating an understanding of when the higher-level resources

A. Clemm, L.Z. Granville, and R. Stadler (Eds.): DSOM 2007, LNCS 4785, pp. 159–170, 2007.

are actually needed by the enterprise's users. Most users are aware of the applications that they use to manage their local and remote data. We categorize these resources – programs, data files, and web sites – as higher-level because they are generally well understood by users in the context of allowing them to achieve the enterprise objectives. These resources can vary in importance based on timing considerations. As one example, having access to certain financial files may be important to the enterprise only near the end of a fiscal quarter or year. In this case, we say that there is a *schedule-based usage pattern* for the users accessing those files. As a different example, consider a special application that is accessed only when a certain combination of other resources are accessed – perhaps in response to a rare customer request. These requests might not occur at regularly scheduled times, as in the schedule-based example. However, we may still be able to determine that the application is always accessed within a certain amount of time after the customer request is placed, and we say that there is a *demand-based usage pattern* for the users accessing this special application. We believe that this information can be collected, processed and integrated to assist system administrators in producing effective operational impact assessments.

The main contribution of the paper is an operational impact assessment system that integrates events from all system and application components. These events are clustered through simple data mining and statistical techniques to infer usage patterns needed for assessment. By integrating events from all relevant components, our system is able to translate a low level event (e.g., failure of a device or router) into user level impact assessments meaningful to system administrators and managers. We demonstrate our approach by collecting and analyzing operational data at Georgia Tech for 35 days. Our experimental results show promising results in providing quantitative statistical support for operational impact analysis.

The rest of the paper is organized as follows. We define our key goals and definitions in Section 2. We highlight how our approach is distinguished from previous work in Section 3. We cover our approach in Section 4, and examine how we collect, process and integrate the topology and usage pattern data. We provide our experimental data in Section 5, including an example scenario of how we would produce the impact assessment from a specific technical event. Finally, we consider some possible extensions of our work in Section 6.

2 Problem Goals and Definitions

Our goal is to present a framework that helps system administrators assess the operational impact by determining the users affected by a component failure. This framework supports assessments in the current time period, and also provides a predictive capability by leveraging the information generated from usage pattern mining to infer the likelihood of impacts during future time periods. We don't expect this approach to assess the operational impact perfectly; the intent is that it will provide clear, operationally focused, and timely feedback that assists system administrators in assessing the operational impact for the executive users of the system. Our approach is based on collecting operating system data from selected end-systems to construct a model of the intra-system and inter-system resource dependencies. This information is then

aggregated to construct a dependency model for the overall enterprise system. The data is also time-stamped, and data mining techniques are applied to detect usage patterns. The dependency topology and usage pattern information is then used to assess operational impacts.

We define an *Enterprise System* as a distributed system of components that are used in combination in pursuit of one or more functional objectives. We model an enterprise system as a directed graph of its' distributed resources, where the nodes represent the system's resources, and the edges represent the functional dependencies between resources. An edge from a source node to a sink node implies that the failure of the sink node would likely prevent the source node from completing its tasks successfully. Fig. 1 presents our terminology and definitions.

Enterprise System = {*Resources, Dependencies*}
where *Dependencies* = $\{\mathbf{d}_{i,j} \mid \mathbf{r}_i, \mathbf{r}_j \in$ *Resources* and $\mathbf{r}_i \rightarrow \mathbf{r}_j\}$, *source*(*dependency* $\mathbf{d}_{i,j}$) = $\mathbf{r}_i$
sink(*dependency* $\mathbf{d}_{i,j}$) = $\mathbf{r}_j$, and *RealUsers* ⊂ *Resources*

Technical Event = <*Failed*, $t_{failure}$, *duration*, *Status*>
where *Failed* ⊂ *Resources*, *Status* = { $\mathbf{d}$:*active*($\mathbf{d}$, $\mathbf{t}_{failure}$) | $\mathbf{d} \in$ *Dependencies* }
and *active*(*dependency* $\mathbf{d}$, *time* $\mathbf{t}$) = { *1* if $\mathbf{d}$ occurs at $\mathbf{t}$; *0* otherwise }

Impact Assessment = { <*Path*, t_{start}, t_{stop}, p_{impact})> }
where *Path* = { $\mathbf{d}_{(1)}$, $\mathbf{d}_{(2)}$, …, $\mathbf{d}_{(k)}$ }, *source*($\mathbf{d}_{(1)}$) ∈ *RealUsers*, *sink*($\mathbf{d}_{(k)}$) ∈ *Failed*,
$\forall$ i *sink*($\mathbf{d}_{(i)}$) = *source*($\mathbf{d}_{(i+1)}$), and $t_{failure} \leqslant t_{start} \leqslant t_{stop} \leqslant (t_{failure} + duration)$

Fig. 1. Operational Impact Terminology and Definitions

We define a *Technical Event* as a 4-tuple which represents the instance where a certain set of resources have *Failed* at time $t_{failure}$, and will not be repaired or restored until ($t_{failure}$ + *duration*). In most cases, the average repair time (i.e. MTTR) can be used as an approximate duration value. *Status* captures the operational status of the system resources at the time of failure. Capturing all system status data might not be possible in some environments, but even partial status data can be useful in assessing impact. We then define an operational *Impact Assessment* as a set of 4-tuples. Each tuple represents how one user will be affected by one of the failed resources along a given *Path*, during the period from t_{start} to t_{stop}, with a likelihood of p_{impact}. The path information is generated from the topology data, while the t_{start}, t_{stop} and p_{impact} values are generated from the usage pattern data.

3 Related Research

There has been significant research in the areas of impact analysis, dependency discovery and data mining. Two main factors distinguish our approach from previous research: (1) a focus on the subset of the topology with a directly traceable impact to one or more users; and (2) close integration of the resource dependency and resource usage data. Both of these factors help increase the operational relevance of the resulting assessments.

Impact Analysis. There are still a significant number of administrators and executive users who perform impact analysis manually, based on best practices and rules of thumb [2][3]. Other automated, dynamic approaches have been proposed [4][5][6][7], but many require external expert knowledge in the form of SLAs, QoS metrics, user-defined use cases, and weighing the importance of various resources. Our approach is automated, and infers the importance of system resources during different time frames by monitoring usage directly. Thereska et al [8] address the need to consider impact analysis in a proactive manner. They propose a "what-if" approach that supports interactive exploration of the results of system changes. Their analysis is focused on determining the technical impact of configuration changes on the system's performance, where our approach focuses more directly on assessing the operational impact for enterprise users.

Dependency Discovery. There has also been significant research on the importance of dependency analysis in determining the impact of a resource failure. The most similar approach to ours is a forensic analysis tool [9] to help system administrators identify entry points when investigating security intrusions. They load modules into the Linux kernel, and use the information gathered to detect dependency relationships between objects by tracking events in which one object affects the state of another object. They focus on three types of dependencies based on the objects being monitored: process/process, process/file and process/filename. Similarly, our approach uses the structure of the collected data records, including relationships between key fields, to determine dependencies. This differs from more sophisticated statistical approaches as used in [10][11]. In addition, the forensic analysis [9] focuses on files and programs, our approach includes a wider array of objects such as users, devices, network ports, remotes sites and routers.

Also, our approach collects data from the end-user workstations only, and doesn't require any modification of the system hardware or software resources. This is similar to the philosophies and techniques in [12][13]. Some approaches capture data using methods such as offline perturbation, and modification of system components [14][15][16]. Though these approaches might yield richer and more comprehensive topology results, the intrusiveness of these techniques could make implementation and management more difficult in production environments.

Data Mining. Data mining has been used for system management [17] and application management [18]. Our approach goes beyond previous work by abstracting and integrating system level events and application level events. As an example, in the mining of usage data to detect business workflow patterns, Aalst et al [18] mention the exploitation of timing data as an open problem in workflow mining. We use timing as the underlying fabric on which we integrate events from all system and application components.

4 Overview of Our Approach

Our approach is divided into four basic phases: Collection, Discovery, Mining and Assessment (Fig. 2). During the *Collection Phase*, we collect the operating system command output on various end-user workstations. The data is also time stamped to

ensure consistency when linking records from different commands during the Discovery Phase, and pattern mining during the Mining Phase. In the *Discovery Phase*, we construct a single, enterprise system-wide dependency topology using the collected data. The topology is then used to compute transitive dependencies between resources and users during the Assessment Phase. The *Mining Phase* detects usage patterns based on when dependencies occur (schedule-based), and how different dependencies are related in terms of their activity status (demand-based). Finally, the *Assessment Phase* integrates the system topology and usage patterns to produce an impact assessment for a given technical event.

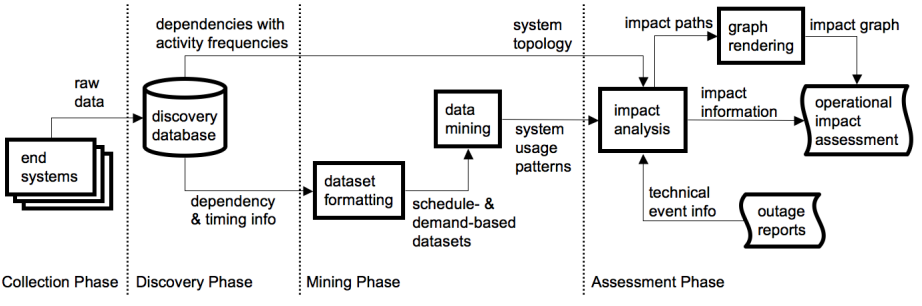

Fig. 2. Impact Assessment Dataflow

Collection Phase. We use cron-activated batch files to capture data about the current state of the workstation being monitored. The batch files execute common Linux operating systems (OS) commands like w(), ps(), lsof(), and traceroute(), to collect data about users, programs and processes, open files, and remote sites. The batch files also format the output for further processing during the Discovery Phase.

Discovery Phase. We use PostgreSQL views and scripts to load the OS output into a database, and extract dependencies in accordance with our dependency topology model in Fig. 3. The nodes correspond to fields in the command output, and each edge is labeled with the commands used to generate that specific type of dependency. As an example, the *w*() command gives the identifier of users logged onto the system, and is primarily used to distinguish real user accounts from accounts used to manage system services. Consider this sample *w*() output from the workstation *athena*:

```
USER   TTY      FROM                    IDLE   WHAT
adams  pts/0    achilles.cc.gt.atl.edu  4days  -bash
```

We derive the following three dependencies from this output:

- user | global | *adams* → site | global | *achilles.cc.gt.atl.edu*
- user | global | *adams* → program | *athena* | *–bash*
- program | *athena* | *–bash* → site | global | *achilles.cc.gt.atl.edu*

These dependencies correspond to the three edges in the topology model between the users, programs, and sites nodes.

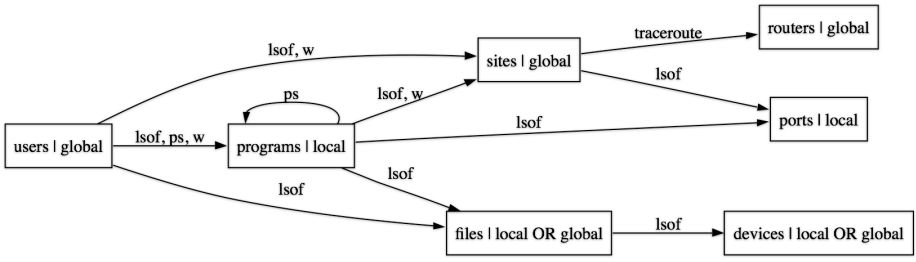

Fig. 3. Impact Topology Dependency Model

Each resource is represented by three values: *type* | *zone* | *identifier*. The resource types are defined in a relatively straightforward and intuitive manner. *Identifier* is the actual resource name, and *type* and *zone* are used to avoid machine-wide and system-wide name conflicts, respectively. Resources located exclusively on a particular end-system are assigned to that computer's *local zone*. Resources accessible by two or more resources located in different local zones are assigned to the system-wide *global zone*. As an example, a device can represent a (local) hard drive, or a (global) network attached storage system. A *system topology* view consists of the combined dependencies from one specific collection period. We use Graphviz to render the system topology and impact topology graphs.

Mining Phase. We mine the activity data for each dependency to find usage patterns. For each dependency, we extract the activity data from the database into the formats in Fig. 4. A schedule-based dataset captures the specific times a dependency is active, where the candidate attributes are the components of the timestamp t_i. The nominal attribute (i.e. class label) is 1 if dependency d_j is active at time t_i, and 0 otherwise. In contrast, the demand-based dataset captures when a dependency is active relative to the activity status of other dependencies. The candidate attributes are the set of dependencies (not including d_j) that are active at time t_i, referred to as the *system status* at t_i. The nominal attribute is 1 if the dependency d_j is active at any time during the t_i to (t_i + *duration*) time period, and 0 otherwise. The exact duration value is normally not known before the technical event occurs, but we can use common duration values for the advanced calculations. Schedule-based mining is similar to partial period pattern searches in time-series data, and demand-based mining is similar to autocorrelation analysis [19]. We use J48 and PART tools in WEKA [20] to generate the decision trees and rules, respectively, for the schedule-based and demand-based datasets. The trees and rules are stored as the *system usage patterns*.

Mining datasets for *dependency* $\boldsymbol{d_j}$ over all collection period *timestamps* $\boldsymbol{t_i}$:
schedule-based: $< \{ day(\boldsymbol{t_i}), month(\boldsymbol{t_i}), date(\boldsymbol{t_i}), hour(\boldsymbol{t_i}) \}, active(\boldsymbol{d_j}, \boldsymbol{t_i}) >$
demand-based: $< \{ active(\boldsymbol{d_k}, \boldsymbol{t_i}) \mid \boldsymbol{d_k} \in Dependencies \text{ and } \boldsymbol{k \neq j} \}, \bigvee_{\Delta t=0..(duration-1)} active(\boldsymbol{d_j}, \boldsymbol{t_i} + \Delta \boldsymbol{t}) >$

Fig. 4. Schedule- & Demand-Based Dataset Mining Formats

We also use the *activity frequency* and *correlation* values to reduce the number of dependencies to be considered during the Mining Phase. Dependencies with a very

low activity frequency will be unlikely to cause an operational impact, and will also be likely to yield trivial patterns during the mining process. Dependencies with a very high activity frequency will, on the other hand, almost certainly cause an impact; however, they will also be likely to yield trivial patterns. Consequently, *dependencies with frequencies lower or higher than our established thresholds (e.g. 10% and 90%) are removed from mining consideration*. We calculate the correlation value for dependency pairs that have equivalent activity frequencies, or where the difference of their activity frequencies is smaller than an established tolerance (e.g. 2%). *If a pair of dependencies is strongly correlated (e.g. > 96.9%), then we can remove one of the dependencies from mining consideration.*

Assessment Phase. First, we use the *system topology* to calculate each path from a *failed resource* to a user who may be impacted by the given *technical event*. We then analyze the dependencies along each potentially impacted path. For each dependency, we use the *system usage patterns*, *time of failure*, *duration*, and *system status* information to determine the maximum likelihood that the dependency will be active during the outage period. For each path, we use the minimum likelihood of the dependencies on the path to determine the overall likelihood that the user will be operationally impacted by the failed resource. We remove any paths where the likelihood is less than a certain threshold, and return the remaining paths as the *operational impact assessment*.

5 Experimental Results

We tested our approach on a computer lab with six Linux–based end-user workstations, all of which are connected to a significantly larger campus infrastructure. The collector program was implemented as a Linux batch file on each workstation, and configured to collect data at roughly 5-minute intervals, which was then consolidated to one-hour groupings. We collected data from these systems over 35 days, and then aggregated the data on a central server to support the Discovery, Mining and Assessment Phases. We gathered more than 5000 distinct groups of data from the six end-systems, distributed over approximately 700 distinct collection times. The steps taken during the Discovery, Mining and Assessment Phases allowed us to significantly reduce this potentially overwhelming amount of data, making it much more manageable and operationally relevant. There are two significant motivations in reducing the size of the system and impact topologies: to reduce the amount of information processing needed to produce an impact assessment; and, to improve the clarity of the results for the system administrators and executive users, as shown in Table 1.

Table 1. Dependency Topology Sizes (Measured in Number of Dependencies/Edges)

	system-wide		**per technical event**			
	all	*real-users*	*all*	*freq < 0.1*	*0.1 ≤ freq ≤ 0.9*	*freq > 0.9*
Mean	3461	844	81	64	14	3
St. Dev.	1269	334	233	189	70	10
Skew	1.3	0.7	4.2	5.3	7.2	3.9

The system topology data values were distributed fairly evenly around the mean. The impact topology values, however, were skewed significantly towards positive values. This was caused when certain technical events impacted an unusually large number of resources. As an example, most port or device failures only affected 4 to 12 resources. In contrast, technical events involving the *http* port on dionysos (*port | dionysos | http*), and a local device on hera (*device | hera | 8-1*), impacted 405 and 1,554 resources, respectively.

The initial topology, using all of the data gathered from one collection period, has an average of 3,461 dependencies. We reduce size of the system topology by 75% by identifying the subset of this topology that has a potential impact on one or more real users. Similarly, the initial impact topology for a given technical event has an average of 81 dependencies. We reduce the number of dependencies to be evaluated for the impact assessment by 79% by eliminating those dependencies with a frequency lower than our established threshold of 10%. Finally, an average of 14 dependencies needed to be evaluated with the system usage patterns for a given technical event. We determined that 1,893 of the dependencies collected during our testing had a frequency between 10% and 90%, inclusively. Further testing showed that 1,775 of these dependencies were strongly correlated (97% or more), such that we needed to perform usage pattern mining on only 118 distinct dependencies. Our practice results so far confirm these percentages: we've had to perform usage mining on an average of 2 of the 14 dependencies, and the usage patterns for the remaining 12 dependencies were strongly correlated to these results.

We will now demonstrate these principles with a practical example. Consider the technical event caused when the *mysql* port on the six end-systems used in our test environment are closed unintentionally by a faulty host firewall configuration. The comprehensive system topology for the entire testing period included over 92,000 distinct dependencies. Manually analyzing a topology of this size would be cumbersome and error-prone. We can use automated techniques to calculate more specifically which users are likely to be affected for this event, as shown in Fig. 5.

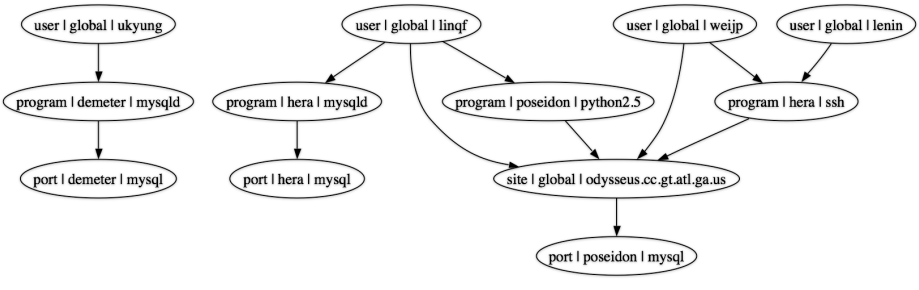

Fig. 5. Impact Topology Without Activity Frequencies

Using the impact topology results alone allows us to infer that the closed *mysql* port could potentially affect 4 of the 17 total users. We can leverage the system usage patterns to more specifically determine the impact. Fig. 6 gives an improved impact topology for this technical event, where each edge label represents the activity frequency for that dependency.

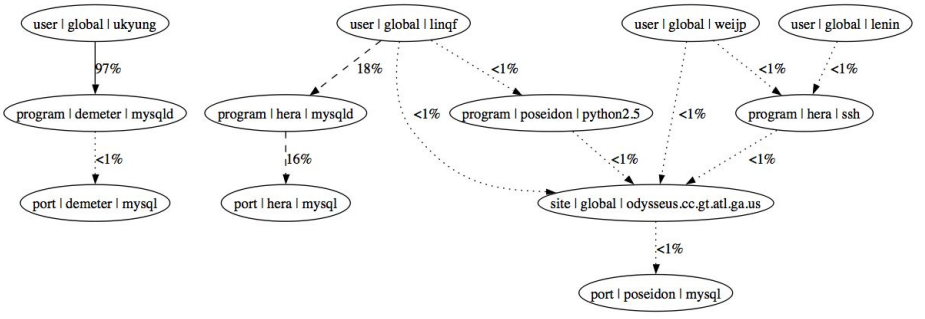

Fig. 6. Impact Topology with Activity Frequencies

We don't have enough information on the dependencies with a frequency < 10% to determine if they will be active during the outage period with any significant likelihood. Consequently, we remove the paths using these dependencies from consideration. The only path remaining for consideration is from *user | global | linqf* through *program | hera | mysqld* to *port | hera | mysql*. The next step is to use the timing and system status information from the technical event, along with the *system usage patterns*, to determine if there will be an impact on *user | global | linqf*.

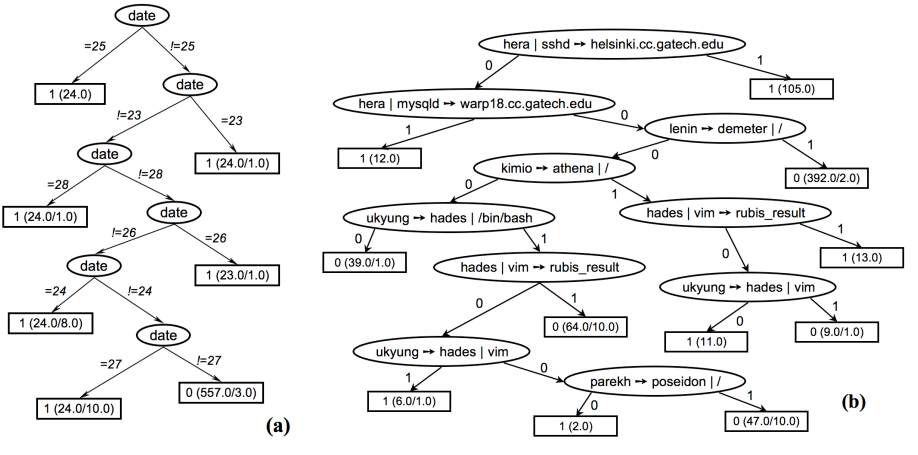

Fig. 7. Schedule-Based (a) and Demand-Based (b) Decision Trees

The two dependencies are strongly correlated, so we can use the same system usage pattern results for both dependencies. Fig. 7 shows the relevant decision tree results for these relationships. The scheduled-based decision tree has a correctly classified instances value of 96.57%, and we can use this as our measure of the likelihood of an impact. If the outage occurs between the 23rd and 28th of the month, then we would assess that user *linqf* has a 96.57% likelihood of being impacted during the outage period. Similarly, if the event occurs on the 22nd at 9pm, with an expected duration of 6 hours,

then we would adjust our assessment such that user *linqf* has a 96.57% likelihood of being impacted between the hours of midnight and 3am on the 23rd.

Now, suppose the event occurs on the 15th at 4pm, and lasts 6 hours. The schedule-based patterns do not indicate activity during this period, but the demand-based patterns might still indicate activity based on the status of other resources. Our approach will assess an impact if either set of patterns – schedule-based or demand-based – indicates that the dependency is likely to be active during the outage period. The demand-based decision tree has a correctly classified instances value of 95.57%, and was generated based on the designated outage period of 6 hours. As an example, if the *sshd* program on the computer named *hera* has an active connection to the *helsinki.cc.gatech.edu* site at the time of failure, then we can infer that the dependencies *user* | *global* | *linqf* → *program* | *hera* | *mysqld* and *program* | *hera* | *mysqld* → *port* | *hera* | *mysql* will also be active at some time during the 6-hour outage period. Consequently, we would assess that user *linqf* has a 95.57% likelihood of being impacted during the outage period.

This example demonstrates how the using the combination of system topology and system usage pattern information has allowed us to improve the clarity and operational relevance of our impact assessments. In the given scenario, the impact topology indicates that the closed *mysql* port might impact four different users. Incorporating the usage patterns allowed us to further determine which specific users had a significant likelihood of being affected during the outage period for the failed resource. This is precisely the kind of information that many system administrators need to make their impact assessments more operationally relevant for management executives.

6 Conclusion and Future Research

We described an operational impact assessment model and system (Section 4) that integrates events from all system and application components. By clustering events through simple data mining and statistical techniques, our system translates a low level event (e.g., failure of a device or router) into a probabilistic user level impact assessment meaningful to system administrators and managers. We demonstrate our approach by collecting and analyzing operational data at Georgia Tech for 35 days. Our experimental results (Section 5) show the promise of our approach in providing quantitative statistical support for operational impact analysis.

From the system management point of view, we consider the work described here as a concrete and significant first step. A natural next step is to apply our approach on systems of increasing scale and complexity. Through automated monitoring and analysis, we will collect a much larger event data set and build a more complete topology model of usage patterns. In addition to observed (real world) failures, we can also conduct controlled experiments by inducing faults in various resources to evaluate the accuracy and coverage of dependencies captured by our topology model. Trade-offs between the granularity/length of data collection and the accuracy/coverage of topology models (e.g., schedule-based and demand-based patterns) are another area of interesting research. Finally, a user-based assessment (e.g., feedback gathered from users via the help desk) matched to technical events (e.g., device failure) may provide a management-level validation of our approach and system.

From the technical point of view, we will investigate methods to deploy the collection systems more easily (e.g. Java applets). We will also examine ways to distribute the discovery and mining processes to the end systems, without sacrificing the accuracy of the results. This could improve the scalability of the overall system by reducing the bandwidth and processing costs when compared to our current, centralized approach. Future research will include incorporating our usage pattern mining techniques with more sophisticated dependency discovery systems. Our current approach allows us to integrate the dependency topology tightly with our usage mining techniques. Using more sophisticated dependency discovery systems might allow us to complement our current focus on the end-systems with improved visibility into the infrastructure.

References

[1] Kar, G., Keller, S., Calo, S.: Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. NOMS (2000)
[2] Singh, A., Koropolu, M., Voruganti, K.: Zodiac: Efficient Impact Analysis for Storage Area Networks. USENIX FAST (2005)
[3] Assistant Secretary of Defense, National Information Infrastructure (ASD-NII): Department of Defense Instruction (DoDI) 8580.1, Information Assurance (IA) in the Defense Acquisition System (2004)
[4] Jobst, D., Preissler, G.: Mapping Clouds of SOA- and Business-related Events for an Enterprise Cockpit in a Java-based Environment. Intl. Symp. JAVA Prog. (2006)
[5] Hanemann, A., Schmitz, D., Sailer, M.: A Framework for Failure Impact Analysis and Recovery with Respect to Service Level Agreements. IEEE SCC (2005)
[6] EMC2|SMARTS Business Impact Manager: http://www.emc.com/products/software/smarts/bim/
[7] IBM Tivoli Application Dependency Discovery Manager. http://www-306.ibm.com/software/tivoli/products/taddm/
[8] Thereska, E., Narayanan, D., Ganger, G.: Towards self-predicting systems: What if you could ask "what-if"? In: Workshop Database & Expert Systems Applications (2005)
[9] Sitaraman, S., Venkatesan, S.: Forensic Analysis of File System Intrusions using Improved Backtracking. In: IEEE Workshop on IWIA (2005)
[10] Brown, A., Kar, G., Keller, A.: An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Environment. IM (2001)
[11] Ensel, C.: A Scalable Approach to Automated Service Dependency Modeling in Heterogeneous Environments. IEEE EDOC (2001)
[12] Aguilera, M., Mogul, J., Wiener, J., Reynolds, P., Muthitacharoen, A.: Performance Debugging for Distributed Systems of Black Boxes. SOSP (2003)
[13] Mortier, R., Isaacs, R., Barham, P.: Anemone: using end-systems as a rich network management platform. Microsoft Technical Report, MSR-TR-2005-62 (2005)
[14] Chen, M., Kiciman, E., Fratkin, E., Fox, A., Brewer, E.: Pinpoint: Problem Determination in Large, Dynamic Internet Services. DSN (2002)
[15] Kiciman, E., Fox, A.: Detecting Application-Level Failures in Component-Based Internet Services. IEEE Trans. Neural Networks 16(5), 1027–1041 (2005)
[16] Hariri, S., et al.: Impact Analysis of Faults and Attacks in Large-Scale Networks. IEEE Sec. & Priv. Mag., 49–54 (September-October 2003)

[17] Cohen, I., et al.: Capturing, indexing, clustering, and retrieving system history. SOSP (2005)
[18] van der Aalst, W.M.P., et al.: Workflow Mining: A Survey of Issues and Approaches. ACM TKDE 47(2), 237–267 (2003)
[19] Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edn. Morgan-Kaufmann, San Francisco (2006)
[20] Witten, I., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)