Smarter log analysis

Modern computer systems generate an enormous number of logs. IBM Mining Effectively Large Output Data Yield (MELODY) is a unique and innovative solution for handling these logs and filtering out the anomalies and failures. MELODY can detect system errors early on and avoid subsequent crashes by identifying the root causes of such errors. By analyzing the logs leading up to a problem, MELODY can pinpoint when and where things went wrong and visually present them to the user, ensuring that corrections are accurately and effectively done. We present the MELODY solution and describe its architecture, algorithmic components, functions, and benefits. After being trained on a large portion of relevant data, MELODY provides alerts of abnormalities in newly arriving log files or in streams of logs. The solution is being used by IBM services groups that support IBM xSeries® servers on a regular basis. MELODY was recently tested with ten large IBM customers who use zSeries® machines and was found to be extremely useful for the information technology experts in those companies. They found that the solution’s ability to reduce extensively large log data to manageable sets of highlighted messages saved them time and helped them make better use of the data.

Introduction

Monitoring and managing computer systems is increasingly difficult as grid and cloud computing become more prominent than ever. Reading and analyzing system logs have thus become very important for a large number of network and system management tasks. One such important task is online failure and anomaly detection, that is, identifying system failures and anomalies during runtime, based on continuous system observation. In this task, system logs are often used as a means to detect system failures and to predict such failures in the near future as the log is being written. In fact, more than 20% of the works covered in a 2010 survey on online failure prediction [1] use system logs as their main source of information.

An emerging trend termed “self-star” (self*) vision [2] is related to the self-maintenance of large computer systems; it seeks to reduce the manual labor in tasks such as system repair and configuration. The system log is one of the major data sources for this, and the ability to automatically analyze those logs is a crucial step toward achieving better awareness of problems.

The challenge increases as the complexity of computers rises, and with it, the size of the system log also increases, which follows Moore’s Law as the number of processing units per system exponentially grows. Many current solutions that deal with various types of system logs offer tools for navigation, search, filter, and sorting, along with basic statistics and alerts based on human-made rules [3–5]. However, they do not exempt the user from reviewing a vast amount of data until pinpointing the source of a failure. Other related works, such as [6], often perform failure and anomaly detection based on a labeled training set, but obtaining a large-enough manually labeled data set is often unfeasible. Moreover, such methods will not detect unseen events since they have neither occurred nor were labeled. In a study of the system logs of five supercomputers [7], the authors put forth a number of recommendations that navigation tools do not meet, citing the chaotic nature of system logs, the evolution of systems over time, and the variety of log signatures as corresponding to different subsystem failures. The inevitable conclusion is that only an unsupervised and adaptive automated analysis can cope with the size of the logs and their nature.

This paper describes IBM Mining Effectively Large Output Data Yield (MELODY), which is a unique and innovative machine-learning solution for unsupervised...
anomaly and failure detection in system logs. MELODY enables a quick and easy way to summarize and navigate these logs visually while highlighting important data. In the last few years, MELODY was tested in two scenarios, as a guide for service teams who provide service on IBM xSeries* servers and as software that assists companies in managing their information technology (IT) on IBM zSeries* servers. In both settings, the tool was found to be extremely useful. The evaluation of such a system can and has been carried out via tedious work of manually inspecting the highlighted data and repeatedly examining the usefulness and correctness of the system.

Typically, we assume that a system log contains information in the format of message IDs (or message type) and their corresponding timestamps. If this is not the case, prior clever parsing can extract message IDs from the log lines, as done in [8] for error logs. (Developing standards for log representation such as the common base event [9] can also solve this problem.)

In this paper, we introduce two analytic components that make up MELODY—clustering and analysis of the system dynamics—and the system as a whole. Figure 1 shows the operational workflow of MELODY, following the machine-learning paradigm, i.e., a one-time adaptation of the system to new log types, followed by training on a large data drop, and then using the learned model to analyze newly arriving log files and periodically retraining the models on the most recent accumulated data.

Notice that MELODY includes statistical components that model message appearance and also deal with low occurrences of categorical data. These were described in previous works [10, 11] and are thus not repeated here. MELODY also contains a rule-based engine that allows the combination of experts’ knowledge with the analysis results and alerts on known cases of errors or failures.

The motivation for doing clustering of the data and for analyzing system dynamics lies in several observations about the data. These observations are based on mining the data, as well as on user feedback during a testing phase of early versions of MELODY.

Log messages provide the most detailed documentation available of low-level events. A higher point of view is given by lower resolution and could discover a system-level event that possibly triggered a set of messages to be written to the log. Opening a database, for example, could involve the initialization of several modules and interprocess communications, resulting in an initial burst of log activity as each initialization takes place. This burst would be followed by a change in pattern of consequent logging activity as each initialization takes place. This burst would be followed by a change in pattern of consequent logging activity as these modules perform their coordinated activities. A set of messages that were caused by a single system event will thus have a high correlation in their appearance patterns. In many occurrences in the log, we expect similarity between patterns of logs each time the same event appears. These hidden patterns will be discovered by clustering the data and without the need for a continuous sequence of
events. For example, collecting logs from a variety of independent faulty machines shows that event patterns repeat across machines.

On the other hand, if such data is available, for example, a collection of logs from a single machine over a large period of time, we can take advantage of the continuity of the events and model the dynamics in the system. The repetitive nature of the system is an inherent feature in large computerized systems, particularly those that handle massive amounts of human interactions, such as banks, heavy traffic web sites, and large e-mail providers. Such systems enjoy a very predictable activity schedule that emanates from large-number statistics of human activities [12]. In addition, nonhuman-generated automated scheduling of system tasks is often also set to occur at specific timed intervals. The result is that various components of the system operate in constant cycles that can be learned and integrated into models of system dynamics. This gives a basis for forecasting failures when a system deviates from its normal pattern, namely, failure prediction.

IT experts who work with machine logs often have their own sets of rules regarding how the logs should be maintained or managed. Such rules typically address the most common problems or the most crucial ones, and they typically follow simple formats such as if-then. An example might be as follows: If this particular message occurs in the log, then the problem is such and such and requires the following action. The rules may have a slightly more complex structure, in which the if part contains advanced rules. In many such cases, the rules identify complex situations that are problematic and must be addressed. In MELODY, we utilize the IBM ILOG [13] rule engine for this purpose. It reviews the log data and discovers where rules apply, passing them to MELODY, which alerts the user.

In the training phase, the above two components and the previously published statistical components [10, 11] are combined to build the model for each message ID. The choice of which algorithmic components to use depends on the nature of the data. The architecture of MELODY, described in detail later, allows modularity of the components, and the user can easily turn components on and off. Similarly, additional components can be easily added as the research progresses. The model built for each component serves in the online detection to provide an anomaly score for each newly arriving time window in the log, based on the message IDs appearing in it. In turn, a collection of these scores is presented to the user in a user-friendly way.

Following this modularity, there are two main versions of MELODY, varying in their data sources and in the components they combine to train a model by. The first version learns from a large drop of logs of failed systems. This version includes an unsupervised method that analyzes the match between the messages’ histograms and the expected ones. This method was described by Sabato et al. [10], and the particular challenge of evaluating abnormalities where there are few occurrences of categorical data was studied in reference [11]. In this version of MELODY, we also employ a rule engine to analyze the configuration of the system and alert on abnormal configurations. The second version of MELODY analyzes many messages arriving from the same machine, where the number of log messages can be millions and typically spans a few months. Both versions employ clustering algorithms, and the latter version also employs algorithms specifically tailored for dynamic systems.

The first version is used in production on logs from thousands of similar machines, assisting engineers who analyze problems with IBM xSeries servers (Intel central processing unit (CPU)-based blade servers). The second version uses many logs from the same machine over a period of time, supporting maintenance of IBM zSeries (IBM mainframe computer) machines.

In the next sections, we first describe the MELODY system architecture and then we introduce the two analytical components of the system, i.e., the clustering component and the component that addresses the analysis of a dynamic system. We also discuss the evolution of these features based on user feedback and their evaluation process. Finally, we propose future directions and summarize.

**System architecture**

The MELODY architecture is intended to create a flexible tool, easily customized to support diverse use cases in the domain of log summarization, anomaly detection, and failure prediction. We also considered portability and efficiency in the design to allow deployment on diverse systems without burdening their resources.

The system is written entirely in Java**. It can be accessed using either a Java application programming interface (API) or command-line Java executables. A configuration file allows the system administrator to customize MELODY’s operation according to the input format, type of algorithms used, and various parameters of the output.

The MELODY architecture contains the following basic modules:

*Log parsing*—MELODY contains parsers that convert log files into lists of MELODY message instance objects. While MELODY comes with an array of predefined log parsers, system administrators can add to them by using the Java API, without recompiling the system. For the purpose of executing MELODY’s algorithm, the system iterates over these message instances sequentially, without loading entire logs into memory. It also contains parsers for reading system configuration parameters.
Summary and analysis—This group of modules contains an array of algorithm building blocks that may be combined in different workflows to achieve diverse tasks. The functions of these building blocks include summarizing logs using various parameters, detecting unusual configuration parameters, training a number of prediction models, and performing various types of analysis. Thus, the system may be easily customized to accommodate different needs.

Database—The database serves several purposes, including long-term storage of log summaries and configuration parameters for research, short-term storage for periodic retraining, storage of prediction models used for analysis, and analysis results.

Rule engine—Some versions of MELODY also contain IBM rules engine software called IBM ILOG [13], where experts’ rules can be viewed, created, maintained, and used to set alerts for important problematic configurations.

Output—The output of MELODY is produced in extensible markup language (XML) files that can be parsed by other systems. Built-in MELODY extensible stylesheet language transformations (XSLT) scripts allow a user-friendly view of this output. Figure 2 shows a snapshot of the high-level system view output by MELODY. This view allows the user to easily focus on the problematic areas in the log and drill down to see a summary of the messages in the log by clicking the problematic intervals.

Users can construct workflows composed of the modules described above, based on their system needs. For example, to allow a continuous online operation of MELODY, we save in the internal database the information required for subsequent training. During the online analysis of arriving logs, MELODY stores a summary of the logs that contains the essence of the logs required for training. The training process is (automatically or manually as the user chooses) periodically activated on the most recent logs collected by MELODY (for example, every month on the previous three months), and the new model is then safely loaded to memory to continue the smooth analysis. When starting the process, we assume that a training model is already present; this can be done in the configuration step (see Figure 1) by uploading a batch of logs and creating the initial training model. Alternatively, the first period will not be analyzed until enough data is accumulated and a training model is produced.

Analytics components
The algorithmic components in the core of MELODY aim at learning from the data the appearance pattern of messages in the system and alert on discrepancies from the learned model during runtime. Each component captures a different aspect of the system. This section is devoted to the two main algorithmic components that make up MELODY. The first is the clustering module, and the second is the one that exploits the system repetitive dynamics to find
Two observations are evident when exploring the log data from machines. First, different system-level events have different characteristic footprints, in terms of which message IDs appear in the log as the events occur. If we know which message IDs appear as a result of a system event, we could detect the occurrence of this event by watching for spikes in the number of message IDs belonging to this group. Second, messages that have common causes (i.e., result from the same event or are the product of the dynamics of interaction of a specific group of processes) would have a joint distribution of appearances, characterized by relatively high mutual information. The appearance or nonappearance of a specific message ID in the log during a certain time interval would have a significant effect on the probability of seeing other messages that arise from the same dynamics. At the same time, it would be rather uninformative in predicting the appearance of unrelated messages. To catch these patterns, clustering is often used; for example, Salfner and Malek [14] use fuzzy c-means clustering to identify attack patterns, as well as new kinds of attacks in intrusion detection systems. Salfner and Tschirpke [8] devise a new distance measure (based on previous work on the hidden semi-Markov model [15]) to group similar error sequences. Here, we take an information-based approach and make use of these observations by running a clustering algorithm called iClust [16].

We use iClust to cluster message IDs based on their appearance in 10-minute intervals across a period of several days. Specifically, each message is an indicator vector with 1s in the intervals at which it appeared and 0 otherwise. Notice that we treat the intervals as a bag of intervals with no regard for their order. iClust attempts to partition the message IDs into a desired number of clusters such that the average within-cluster mutual information is high. Thus, groups of messages that usually have a common cause (stemming from the same events) would tend to be clustered together. Once we partition the messages into clusters, we treat these clusters as the footprints that correspond to system events. A sudden rise in the number of different messages belonging to a cluster (which we term cluster spike) is an indication of the occurrence of this cluster’s underlying system event. Figure 3 shows patterns of appearances of message IDs during a two-day period for one system and their clustering by iClust. Only clusters that show spiking activity are shown (9 out of 30). The graph summarizes the appearance of 447,176 messages with 2,408 unique message IDs. As shown, a system reboot is characterized by a simultaneous occurrence of several cluster spikes corresponding to the initializations of several modules. A preceding system shutdown corresponds to a different set of cluster spikes as several modules are closed. The cluster spikes are marked. Some clusters also indicate a periodic appearance of some message IDs. For example, cluster 7 is spiking every hour.

An attractive feature of our clustering algorithm is that it uses a “leftovers” cluster to group together those messages with sporadic appearance patterns that do not tend to consistently align with other messages. We can then take the messages from the leftovers cluster, along with rare messages that were kept out of the clustering procedure, and measure the likelihood of spotting them based on a Poisson model, which assumes that the messages’ appearances each come from a Poisson distribution whose parameters we learn from the data [10], and messages are independent (i.e., our probability is calculated as a product of Poisson probabilities). The negative log likelihood serves as an anomaly score. By removing those messages that were clustered and did not end up in the leftovers cluster, we are able to eliminate the logging of nonproblematic system events while keeping unclustered or out-of-context messages that appear outside their cluster scope.

A natural extension of this feature is to cluster the messages and the 10-minute time frames simultaneously. In this approach, which is called biclustering, a subset of messages are clustered together if they are correlated to a subset of time frames.

We observed that a message may appear in more than one system event and that there is no reason for two messages to correlate over an entire timeline. Using the one-way clustering approach, such clusters of messages would be missed, as their mutual information will be low over the entire timeline. However, inspecting only those times in which a specific system event occurred may reveal the desired cluster of messages in its correct context. To do so, we first cluster the 10-minute intervals based on the message IDs. We then repeat clustering of the message IDs in each group of intervals that were found as clusters in the first stage. This provides a set of clusters of message IDs. This simple approach yields groups of message IDs that have high mutual information in a subset of the intervals. Detecting anomalies in the cluster pattern proceeds as before, based on the leftover messages that do not align with any cluster.

Modeling system dynamics

Obtaining a good model of the system dynamics is a crucial aspect of predicting future values of a system’s features.
Many options exist for modeling time series, and reviewing them is beyond the scope of this paper. Instead, we describe what we did within the scope of the MELODY project and our reasoning behind it. Our main premise is that system activity and, specifically, its reflection in system logs contains multiple periodicities that are often phase locked. Therefore, models for predicting future values of system features should take into consideration previous system observations, at intervals for which evidence shows periodicity in the data. One specific setting in which we explored this premise was in estimating the number of unique message IDs in a time interval using records of previous such counts. We assumed that if we observe an unexpectedly high number of message IDs, then some unexpected event or events occurred. Moreover, if an unexpectedly low count is observed, then some expected event or events failed to occur.

To verify that there are multiple periodicities in the data, we calculated autocorrelation coefficients of the number of unique message IDs in 1-minute time lags, ranging from zero to two weeks, in IBM z/OS* system logs from a production zSeries system that was collected over a 22-week period. Figure 4 shows autocorrelation coefficients for the number of unique message IDs in this system. The existence of multiple periodicities is clear, and they appear to include 10-, 20-, 30-, and 60-minute intervals, as well as 1- and
7-day intervals. We can guess that the short periodicities are due exclusively to automated job scheduling, whereas the daily and weekly cycles are due to a mix of automated scheduling and human activities. The clear dip in correlations for time lags that are a few days apart is mainly due to the effect of comparing data from weekdays to weekends, which are often quite dissimilar. Such periodicities can be also found in other system measurements such as CPU loads. With knowledge of the prevalent periodicities, we looked at averages of this parameter—of the number of unique message IDs in 10-minute time frames—over a period of 22 weeks. Specifically, we measured the mean values and their standard deviations for every time of the week, in 10-minute time frames over 22 weeks. Aligning the average over the activity of a week, we can identify deviations from the average (shown in Figure 5). It is clear from the figure that the pattern for weekdays is more active than that for weekends, and the repetitive patterns of the system allow identifying when abnormal events occur. The resulting analysis using this measure is shown in Figure 2; unexpected bursts of activities and unexpected low activity are identified during analysis of new logs and are output to the user in addition to the scoring provided by the other components. More details on specific examples where this measure provides added value are given in the evaluation section below.

These measurements can be also used for predicting the next time frame. The mean can be used as the prediction value for each time of week, and the standard deviation is used for confidence scoring. If the variance across the relevant time period tends to be constant, then a simpler model can be learned where only the mean is estimated per time in period and the variance from that mean is uniformly estimated across the whole period. This simpler model requires less data for training and is the method currently implemented within MELODY. Retraining over the latest batch of data provides some adaptivity to the changing dynamics of the system.

Future approaches may involve replacing means and standard deviations with medians (50th percentile) and extreme percentiles (the 5th and 95th percentiles, for

---

**Figure 4**

Correlation coefficients at lags for different time scales: (a) Lag period of up to 2 weeks; (b) lag of up to a single day; and (c) lag of up to 1 hour.
example). The percentile approach is more robust to outlier values. Other methods such as autoregressive moving average models [17] are more dynamically adaptive as they operate on a moving window. However, corrections may be needed in cases of missing values that often occur in operational systems due to problems in the logging or when the machine is off.

**MELODY evaluation**

Evaluation of MELODY is a challenge as the tool uses unsupervised methods to alert on untypical and what seems to be problematic configurations or log messages. Thus, the system evaluation was carried out in a straightforward way by obtaining human feedback on the clusters derived and alerts created. This was carried out in various ways, i.e., by sharing the results with IT experts and getting their feedback on the value of the alerts, by sharing the results with potential users, and by providing a working MELODY system to a few hundred service personnel and creating automatic counters to monitor the level of use of MELODY. Note that the last method for evaluation seems to directly monitor the value of MELODY as the service people are aiming at optimizing their time and will only consult software such as MELODY if they believe it will shorten their work time. The counters show a slow increase in usage during the first year and a steady level of subsequent usage.

Following the user feedback, we were able to identify scenarios where MELODY produces false and missing alarms, defines their cause, and adds components to directly deal with these scenarios. One such observation has led to introducing message clustering, and another has led to exploring system periodicity. A particular example is the case where we worked with a customer and were able to identify a problem (unexpected partition shutdown), but the initial version of MELODY could not assist in early identification of it or performing root-cause analysis. The problem occurred during the weekend, and since the basic MELODY version did not contain the periodicity component capturing time of week activity, the cause of the problem and the exact time at which the problem started was not

![Figure 5](image-url)

System periodicity captured over time of week. The bottom plot shows the average number of unique messages in each time of the week, averaged over 22 weeks. Each green bar is a 10-minute interval, and its heights is the average value. The top four plots show the number of unique messages over four different weeks, superimposed with the average. Abnormal overactivities (unexpected burst) and underactivities (“machine taking a day off”) are marked by magenta and red, respectively.
detected as it did not yet have a large impact on the system. However, the version with the periodicity component clearly detects a burst of (seemingly normal) activity prior to the shutdown event. Inspecting this period reveals the real problem (storage problem) that had an impact on other partitions as well. Early detection of it might have helped in preventing the failure.

The above example is a representative one, and we were able to find many similar cases analyzing an internal testing environment. This is shown in Figure 2: partition sys1 is showing abnormal behavior starting at around 11:30, as the red bars indicate, followed by a shutdown of the system (no logging activity). Prior to that, at 11:00, the green intervals do not show abnormal behavior, but the bottom squares are red and show a deviation from normal logging load for that time of day, allowing an early detection of the problem. Inspecting the messages within this time period indeed reveals a storage problem in partition sys1 already starting at 4:00 a.m., manifesting a little before 11:00, and resulting in its abnormal shutdown at around 11:30. The shutdown also manifested as connection problems to device at the rest of the partitions.

Finally, in the past 3 years, we have accumulated examples where experts believe there is a problem, particularly a problematic message that needs to be addressed and thus needs to be picked up by MELODY if the problematic event is in effect. The creation of such a set is a tedious work, but we hope to soon create a testing environment that compares the performance of MELODY with the experts’ expectations and that facilitates quantitative analysis of performance.

**Summary and conclusion**

IBM Mining Effectively Large Output Data Yield (MELODY) is a natural, yet unique, branch of the problem determination toolbox. In an analogy to the work of a physician, the goal is to detect the problem as it occurs, foresee the upcoming symptoms, and provide some remedy before problems escalate. By building on machine-learning technology, MELODY provides the means to automate the process of detection–prediction–action. Through integration with IBM ILOG rules engine, it also provides a means to encode and incorporate the knowledge from subject matter experts.

From a knowledge management perspective, our approach to information summarization is lossless, in which “noninteresting” data is preserved, although it will get low relevance scores. We successfully address the nontrivial challenge of devising an information extraction mechanism that solely utilizes data stemming from faulty machines. Our analysis includes the incorporation of clustering techniques that enable analysis not only at the level of individual log messages but also automatic construction of metastates, which carry information at a higher abstraction level. From an operational point of view, we moved from an offline learning and an interactive usage scenario to an online, self-adapting system that captures system dynamics, provides alerts, and makes predictions in real time. All of this brings us closer to the vision of a self-healing system.

Looking forward, it seems to us that the next major challenges will stem from the migration of the MELODY framework to the system level, which contains many components that yield multiple logs that are not necessarily synchronized. From an information and knowledge management perspective, defining an event at this higher abstraction level, and detecting and synchronizing the information produced by the various components, are all quite challenging. From an operational standpoint, our aim is to build a coherent and concise picture of the system’s state. This picture should allow the detection of events at the system level as they start to emerge. We intend to alert and propose actions, both at the system and the system component levels (e.g., tackle a domino syndrome), maintaining the system in a healthy operational state.

**Acknowledgments**

The authors would like to thank the IBM System x, BladeCenter, & Modular (xBCM) Support team for many years of cooperation. Special mention goes to Christopher M. McCann, who leads the Multitool Tools team; Christian Queen, who is the Dynamic System Analysis Team Lead; and Chris Dombrowski, who was the Systems and Technology Group xBCM Serviceability Lead.

The authors would also like to thank the IBM System z resiliency team for years of cooperation, in particular, Geoffrey E. Miller, who is the System z* Resiliency Lead; James M. Caffrey; and the System z artificial intelligence development team, including Erin Farr, David M. Thornley, and Garth Godfrey for their continuous cooperation and support. Finally, the authors would like to thank Sivan Sabato, Ohad Rodeh, and Ya’akov Fernandas, Research Staff Members who contributed to these efforts.

*Trademark, service mark, or registered trademark of International Business Machines Corporation in the United States, other countries, or both.

**Trademark, service mark, or registered trademark of Sun Microsystems in the United States, other countries, or both.

**References**


Received April 28, 2011; accepted for publication May 19, 2011

Ehud Aharoni  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (eahud@il.ibm.com). Mr. Aharoni is a Research Staff Member in the Machine Learning Technologies Group at IBM Research, Haifa. He received his B.Sc. and M.A. degrees in computer science from the Technion, Haifa, in 1995 and 1997, respectively. He joined IBM Research in 2003 and, since then, has worked on various machine-learning-related projects in the domains of hardware verification, bioinformatics, and anomaly detection.

Shai Fine  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (fshai@il.ibm.com). Dr. Fine is a Research Staff Member and Senior Manager of the Analytics department at IBM Research, Haifa. He received his Ph.D. degree in computer science from the Hebrew University of Jerusalem in 1999 and conducted his postdoctoral research at the Human Language Technologies department in the IBM Thomas J. Watson Research Center. In 2002 he joined IBM Research, Haifa, and served as the lead for machine-learning activities. In 2005, the Machine Learning and Constraint Satisfaction (MLCS) Group was formed and Dr. Fine was appointed its first manager. Since 2008, Dr. Fine has been the senior manager of the Analytics department. Dr. Fine has authored and coauthored approximately 30 papers and is a co-inventor in ten patents. He has served on program committees in machine-learning conferences and most recently served as the local chair for ICMIL 2010 and COLT 2010.

Ya’ara Goldschmidt  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (yaarag@il.ibm.com). Dr. Goldschmidt is a Research Staff Member in the Machine Learning Technologies Group at IBM Research, Haifa. She received her B.Sc. degree in bioinformatics from Ben Gurion University in 2001, and her M.S. and Ph.D. degrees in bioinformatics from the Weizmann Institute of Science in 2002 and 2007, respectively. She subsequently joined IBM at the Haifa Research Lab and has worked on several projects involving text mining, anomaly detection, and bioinformatics.

Ofer Lavi  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (oferl@il.ibm.com). Mr. Lavi is a Research Staff Member in the Machine Learning Technologies Group at IBM Research, Haifa. He received his B.Sc. degree in computer science from the Hebrew University in Jerusalem in 2001, and his M.Sc. degree in computer science from Tel Aviv University in 2010. He joined IBM Research in 2010, where he works on machine-learning-related projects focusing on anomaly detection.

Oded Margalit  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (odedm@il.ibm.com). Dr. Margalit is a Research Staff Member in the Data Mining and Machine Learning Group at IBM Research, Haifa. He received his B.Sc. degree in pure mathematics from Tel-Aviv University in 1982, and his M.A. and Ph.D. degrees in computer science from Tel-Aviv University in 1988 and 1992, respectively. He joined IBM Research in 2006 and worked on several projects with large amounts of data: anomaly detection, data clustering, and more. In 2008, he received an IBM On Demand Community (ODW) Excellence Award, and in 2009 he received the IEEE Membership Geographic Activities (MGA) award. He is the author or coauthor of four patents and eleven technical papers. Dr. Margalit is the editor of IBM Research monthly challenge corner Ponder-This.

Michal Rosen-Zvi  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (rosen@il.ibm.com). Dr. Rosen-Zvi is a Research Staff Member in the Data Mining and Machine Learning Group at IBM Research, Haifa. She received her B.Sc., M.Sc., and Ph.D. degrees in physics from Bar-Ilan University in 1994, 1996, and 2002, respectively. She subsequently continued her postdoctoral studies in machine learning at University of California, Berkeley, University of California, Irvine, and at the Hebrew University from 2003 to 2005. She joined IBM Research in 2006 and, for the last two years, has served as manager of the Data Mining and Machine Learning Group. Dr. Rosen-Zvi has authored and coauthored around 30 peer-reviewed papers, served on the program committees of a dozen machine-learning conferences, and co-chaired three international workshops.

Lavi Shpigelman  IBM Research Division, Haifa Research Lab, Haifa University Campus, Mount Carmel, Haifa 31905, Israel (lavi@il.ibm.com). Dr. Shpigelman is a Research Staff Member in the Data Mining and Machine Learning Group at IBM Research, Haifa. He received his B.Sc. degree in computer science from the Technion, Israel Institute of Technology 2000 and his M.S. and Ph.D. degrees in computational neuroscience from the Hebrew University in 2003 and 2010, respectively. He subsequently joined IBM Research in Haifa and has worked on several projects involving fraud detection, anomaly detection, healthcare, and neuroscience.