Assessing Invariant Mining Techniques for Cloud-based Utility Computing Systems

Antonio Pecchia, Member, IEEE, Stefano Russo, Senior Member, IEEE, and Santonu Sarkar, Member, IEEE

Abstract—Likely system invariants model properties that hold in operating conditions of a computing system. Invariants may be mined offline from training datasets, or inferred during execution. Scientific work has shown that invariants’ mining techniques support several activities, including capacity planning and detection of failures, anomalies and violations of Service Level Agreements. However, their practical application by operation engineers is still a challenge. We aim to fill this gap through an empirical analysis of three major techniques for mining invariants in cloud-based utility computing systems: clustering, association rules, and decision list. The experiments use independent datasets from real-world systems: a Google cluster, whose traces are publicly available, and a Software-as-a-Service platform used by various companies worldwide. We assess the techniques in two invariants’ applications, namely executions characterization and anomaly detection, using the metrics of coverage, recall and precision. A sensitivity analysis is performed. Experimental results allow inferring practical usage implications, showing that relatively few invariants characterize the majority of operating conditions, that precision and recall may drop significantly when trying to achieve a large coverage, and that techniques exhibit similar precision, though the supervised one a higher recall. Finally, we propose a general heuristic for selecting likely invariants from a dataset.

Index Terms—Invariants, Cloud, SaaS, Workload characterization, Anomaly detection.

1 INTRODUCTION

Dynamic invariants are properties of a program or a system expected or observed to hold during executions. Dynamic program invariants can be inferred from execution traces as likely invariants [1], a relaxed form modeling properties which hold during one or more executions, though not necessarily over all possible executions. Program likely invariants have been shown to support several software engineering activities [2] [3] [4] [5].

Likely system invariants [6] are attractive for modeling runtime behavior of data centers and cloud-based utility computing systems from a service operation viewpoint. They are operational abstractions of their dynamics.

Due to the size and complexity of such systems, it is very hard for human operators to detect application problems in real time. Especially transient or silent errors occur rarely - e.g. in case of overload, timing issues and exceptions - and often do not cause an immediately observable failure such as a crash or hang, hence are hard to detect. Typically, likely system invariants hold in normal operating conditions; as such, their violations are considered symptoms of execution malfunctions. By monitoring execution and checking for broken invariants, it is possible to automatically detect failures [7] and to request actions to the operations personnel (e.g. jobs re-execution).

Defining invariants is pretty natural for cluster computing or Software-as-a-Service (SaaS) platforms, and generally for systems performing batch work, providing services to applications often consisting of jobs, in turn comprising tasks. These systems include monitoring and logging facilities1 collecting metrics - e.g., job/task completion time, resource usage and status codes - which can be used to establish invariants. Indeed, likely system invariants have been shown to be effective for modeling execution dynamics in a variety of service computing systems [8], and for supporting a range of operational activities, including capacity planning, detecting anomalous behaviors [9], silent failures [10], and violations of service level agreements [11].

While previous scientific work has shown that invariant mining techniques may be beneficial for the above goals, practitioners face several problems, including (i) how to select a proper technique for their analysis goals, (ii) how many invariants are needed, and (iii) what accuracy they can expect. We cope with the challenge of filling the gap between past studies and the concrete usage of likely system invariants by operations engineers of cloud-based utility computing systems. By empirically analyzing and comparing techniques to mine invariants, we contribute to gain quantitative insights into advantages and limits of such techniques, providing operation engineers with practical usage implications and a heuristic to select a set of invariants from a dataset.

The study focuses on three techniques: two unsupervised, namely clustering and association rules, and one supervised, decision list. They are applied to two independent datasets collected in real-world systems: a cluster operated by Google, whose traces from about 12,500 machines are publicly available, and a SaaS platform in use by various medium- to large-scale consumer packaged goods (CPG) companies worldwide. The datasets comprise 679,984 executions (correct and anomalous) of batch units of work, namely jobs and transactions.

We explore the use of the techniques for two typical applications of invariant-based analysis, namely executions characterization.

1 E.g., Nagios (www.nagios.org) and Ganglia (http://ganglia.info).
System invariants have been shown by several authors to be effective for modeling system dynamics and for detecting anomalous behaviors. Jiang et al. [6] introduced the concept of flow intensity in transactional systems, whose behavior depends on user requests. They presented an approach for modeling the relationships between the flow intensities, and demonstrated experimentally that flow intensity invariants do exist for distributed transaction system. In [12] they used the technique for detecting faults like memory leaks, missing files and null calls.

Sharma et al. [8] described positive experiences in a variety of IT systems with the SIAT product built around the flow intensity mining algorithms, used 24x7 for detecting invariants and locating faults as well as for capacity planning; they reported that the violation detection can be performed in seconds, after a training in the order of minutes.

Lou et al. [9] discovered invariants from console logs of Hadoop and CloudDB, revealing constant linear characteristics of program work flows, used for detecting execution anomalies.

Miao et al. [10] described an approach for silent failure detection in wireless sensor networks by finding correlation patterns.

In [11] we have described a framework to discover dynamic invariants from application logs, and supporting the online detection of violations of Service Level Agreements in SaaS systems.

For all these applications, invariants were mined from various types of data sources, mainly with techniques based on time series analysis, possibly refined with graph-based techniques.

In their work on flow intensity invariants in transactional systems, Jiang et al. [6] mine time series in monitoring data using AutoRegressive models with eXogenous inputs (ARX) [13] to learn linear relationships between pairs of flow intensities (called local invariants). In the subsequent work [14] the computational complexity of the invariant search algorithm is overcome with two further algorithms which are approximate but more efficient.

The mining of flow intensity invariants suffers from the combinatorial explosion of the search space when applied to global invariants, i.e. to higher order correlations among system attributes. Hence, in [15] a Bayesian regression technique was proposed for global invariants: first a regression model is built, whose solution is such that only spatially correlated attributes have non-zero coefficients; then temporal dependencies of attributes are considered. The application to a wireless UMTS system shows that the technique achieves the detection rate 0.848; just a very rough estimate is provided of the false positive rate (as low as 0.08).

In their work on the SIAT tool [8], Sharma et al. discussed also two approaches to reduce noise in broken invariants so as to support faults localization in a distributed sensors system: a spatial approach exploiting a graph of the extracted invariants, and a temporal approach which marks an invariant as actually broken only if it is broken for three consecutive samples of a time series.

In the previous work [11] we too used an ARX model of time series from 9 months of log events of a SaaS platform. With respect to [8], we mined invariants with much higher goodness of fit using a Recursive Least Square algorithm adapted from [13], yielding a relatively small number of invariants whose violation is directly indicative of the fault location.

In [16] we discussed accuracy and completeness of invariant-based anomaly detection, showing that (i) a well tuned approach can reach good completeness at an accuracy in the range 50-74%; (ii) the sampling time can be set to find a tradeoff between the mining time and the violation detection time, respectively in the order of minutes and seconds, confirming what reported in [8].
As for the datasets used in this study, it is worth considering various analyses of the publicly available Google cluster dataset [17]. This is a trace log of one of Google cloud data centers; it contains data of jobs running on 12,500 servers for a period of 29 days, accounting for 25 millions submitted tasks.

Di et al. [18] used a K-means clustering algorithm to classify applications in an optimized number of sets based on task events and resource usage; they also found a correlation between task events and application types, with about 81.3% of fail events belonging to batch applications. Chen et al. used the dataset for analysis [19] and prediction [20] of job failures; Guan and Fu [21] identified anomalies through Principal Component Analysis of monitored system performance metrics. Rosà et al. [22] analyze unsuccessful tasks/jobs executions and propose Neural Networks-based prediction models. While these studies do not specifically address invariants, some of their results about workload characterization and failures identification are in line with the ones we present based on the three mining techniques.

In summary, the literature shows that invariants can be mined and used effectively for a wide range of computing systems - data centers, cloud systems, web hosting infrastructures, wireless networks and sensors-based distributed systems. We are not aware of any work comparing mining techniques on different datasets, so as to learn practical usage implications and general heuristics useful for practitioners. This is the goal of the present study.

3 DATASETS

3.1 Google cluster

We provide a short description of the dataset, whose details can be found in [17]. The workload consists of tasks, each running on a single machine. Every task belongs to one job; a job may have multiple tasks (e.g., mappers and reducers). There are six tables in the dataset: Machine_events, Machine_attributes, Job_events, Task_events, Task_constraints and the Resource_Usage. Every job and every machine is assigned a unique 64-bit identifier. Tasks are identified by means of the ID of their job and an index; most resource utilization measurements are normalized.

Machines are described by two tables. Machine_events reports addition, removal or update of a machine to the cluster, along with its CPU and memory capacity. Machine_attribute lists key-value pairs of attributes representing properties such as kernel version, clock speed, and presence of an external IP address. Attributes are integers if there is a small number of values for the attribute; obscured strings based on a technique proposed in [23], otherwise. The Job_events and Task_events tables describe jobs/tasks and their lifecycle. They indicate transitions between the states shown in Fig. 1. The Task_constraints table lists task placement constraints that restrict the machines onto which tasks can be scheduled. Usually all tasks in a job execute exactly the same binary with the same options and resource request. The Resource_Usage table reports resource usage of the tasks.

3.2 SaaS platform

The SaaS platform we consider provides cloud-based data processing and analysis capability to several consumer packaged good (CPG) companies. The platform accepts and transforms data files provided by customers through FTP servers or email attachments.

The data files, referred to as workitems, go through processing stages, such as format validation, verification, data extraction and transformations. Once a data file is accepted by the SaaS platform, a transaction is started: the processing stages of a workitem are run in the context of the transaction. Based on the outcome of the latest processing stage, the transaction is either continued or aborted; it is completed if all the processing stages are accomplished successfully. At any time through all the elaboration, the status of a transaction can assume one out of four values, i.e., in_process, user_error, exception, and processed. User_error and exception denote the failure of the transaction.

A processing stage within a transaction can result in a success or a failure: if successful, the transaction moves to the next stage. Upon the failure of a stage (i) the platform generates an exception, (ii) the transaction is aborted, and (iii) the customer is notified about the problem for future resubmission or correction. Further details of the platform can be found in [11]; a reliability analysis of the operational failures is presented in [24].

Fig. 2 shows the high-level architecture of the SaaS platform. The processing modules implement the above-mentioned stages; management modules are responsible for handling the transactions workitems pertain to, and monitoring the progression of the stages. The platform relies on databases containing the configuration and business rules (e.g., management of customers and data files); the staging database maintains intermediate results/transformations of the workitems and internal audit logs containing execution informations and error events. Logs are stored as database tables. Applications and DBs server are based on VMware ESX VMs running on Intel Xeon processors.

In this study we use the Details Log table. Each line (examples are shown by Table 1) lists details and outcome of a processing stage, such as the id of the workitem and start/end times. Important fields are Stage (the processing stage), Status (the outcome of the stage), and Fail Reason (a short textual description of the outcome). The log has been collected during operation from April to October 2012, covering a time frame of seven months.

![Fig. 1: State transitions of the jobs in the Google cluster dataset.](image)

![Fig. 2: High-level architecture of the SaaS platform.](image)

<table>
<thead>
<tr>
<th>Event</th>
<th>Workitem</th>
<th>Stage</th>
<th>START time</th>
<th>END time</th>
<th>Fail Reason</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>4346482</td>
<td>308145</td>
<td>IT3</td>
<td>2012/05/02 01:57:54</td>
<td>2012/05/02 01:57:54</td>
<td>Corrupt_File</td>
<td>L1_REJ</td>
</tr>
<tr>
<td>4346810</td>
<td>309135</td>
<td>IT3_PVLD</td>
<td>2012/05/10 10:24:32</td>
<td>2012/05/10 10:25:03</td>
<td>Invalid_File</td>
<td>L1_REJ</td>
</tr>
<tr>
<td>4347484</td>
<td>309467</td>
<td>IT4_TRNF</td>
<td>2012/05/12 04:13:42</td>
<td>2012/05/12 04:13:45</td>
<td>System_Error</td>
<td>SE</td>
</tr>
</tbody>
</table>

TABLE 1: SaaS dataset: structure of the Details log.
4 Invariant Mining

A workload unit \( W \) (i.e., a job in the datacenter or a processing stage of a transaction in the SaaS platform) is abstracted by a set of \( N \) attributes \( A_1, A_2, \ldots, A_N \). These attributes represent the computing resources used or parameters such as duration, priority and return codes, being collected during the execution of \( W \). The attributes that characterize the execution of a workload unit assume a value in the Cartesian product \( \{V_{A_1} \times V_{A_2} \times \cdots \times V_{A_N}\} \), where \( V_{A_j} \) denotes the set of the possible values of \( A_j (1 \leq j \leq N) \). The values of the attributes are extracted from the input dataset to form an \( M \times N \) attributes matrix, where \( M \) denotes the total workload units \( W_i \) (1 \( \leq i \leq M \)).

Fig. 3 shows the framework and steps that underlie invariant mining. Given the input monitoring data at a given time \( t_i \), (i) workload abstraction infers the \( M \) workload units \( W_i \) and the values of the attributes for each \( W_i \); (ii) invariant mining infers the set of recurring relationships among the values of the attributes from the data collected until \( t_i \), i.e., invariants \( I_{t_i} \) in Fig. 3. At \( t_i = t_0 \) (where \( t_0 \) denotes the time of the first ever mining), the set of invariants available to operations engineers is \( I_t = \emptyset \), which is mined from the data at \( t_0 \). Moreover, engineers will select a subset of invariants in \( I \), i.e., actionable invariants in Fig. 3, that will be used for a specific application, e.g., anomaly detection.

Invariants might undergo reviews/changes upon feedback by operations engineers. From a methodological standpoint the framework encompasses the scenarios shown by Fig. 3: (i) refinement of the actionable invariants based on application results, field experience or domain knowledge (e.g., to fix a rule that runs in too many missed detections); (ii) entire update of \( I \) when new incoming data become available at \( t_i > t_0 \) and, for example, engineers deem the system has been subjected to major workload changes or hardware/software upgrades. In the latter case \( I \) is merged/replaced with the new set of invariants \( I_{t_i} \) found at \( t_i \) (dotted box in Fig. 3); new actionable invariants are selected from \( I \) for subsequent application.

In this study invariants are mined offline. Data available at \( t_0 \) consist of the entire datasets described in Section 3; no more data are fed to the framework beyond the datasets at \( t_0 \). It should be noted that invariant mining and application do not interfere with the system operations because they work on attributes extracted from the monitoring data.

We classify a workload unit to be correct, when it is correctly executed by the system, anomalous otherwise. The Google dataset indicates explicitly whether a job execution was correct, and contains information about how many times tasks were resubmitted. In the SaaS dataset, processing stages of transactions have an exit code and an explanation; the assignment of the ground truth (correct or anomalous) was done by the operation engineers.

4.1 Workload Abstraction

A well-known concern in training and validating models used by engineers is that of invariants available to operations engineers is \( I \) due to lack of feedback and change in the system operations because they work on attributes extracted from the input dataset to form an \( M \times N \) attributes matrix, where \( M \) denotes the total workload units \( W_i \) (1 \( \leq i \leq M \)).

The workload units consist of jobs, made of tasks. After filtering out a small number of inconsistencies (e.g. malformed records and blank attributes), we count 649,959 jobs. Of these, 372,688 are finished, 267,464 are killed, and 9,807 are failed jobs. We consider finished jobs as the correct class; killed and failed jobs build the anomalous class. We characterize jobs by means of attributes - a common practice in job-level analysis [26]. Attributes are:

- Tasks (T): number of tasks pertaining the job;
- Priority (P): priority of the job;
- Resubmissions (R): number of resubmissions of the tasks until the completion of the job;
- Duration (D): total duration of the job;
- CPU usage (C): average CPU usage of the job;
- Server (S): the type of server(s) the tasks run on.

The values of the attributes of a job are extracted from the tables described in Section 3.1. For practical mining purposes, the huge space of numerical values taken by the attributes is mapped to categorical values, beforehand. Categorization is usually done in empirical assessments for summarizing workload parameters through a small number of actionable classes (e.g., low, moderate, medium, high), which can be easily applied/understood by practitioners [27]. In order to establish the categories we analyze the distribution of the attributes, beforehand.

The frequency of tasks, resubmissions, CPU usage and duration follows a strong power-law distribution. This finding has been also noted for the same dataset by Di et al. [18] in their jobs characterization, and in the failure analysis in [19]. Fig. 4a-4c show the cumulative distribution function (CDF) of tasks, resubmissions and CPU usage. For example, Fig. 4a plots the probability \( p \) a job consists of no more than \( f \) tasks. It is worth noting that the majority of jobs contains a relatively small number of tasks (e.g., the probability a job has no more than 100 tasks is 0.9457). Similarly, Fig. 4b indicates that the probability a job experiences no resubmissions is 0.9187.

![Fig. 3: Framework to mine invariants and feedback mechanisms.](image-url)

![Fig. 4: Cumulative distribution function (CDF) of tasks, resubmissions and CPU usage; number (#) of jobs by priority value.](image-url)
We use four categories (X0 to X3) to discretize T, D, R and C. Table 2 shows the mapping between the categorical and the original values of the attributes (P and S will be discussed later on in this Section). For instance, T assumes the categorical values \( V_T = \{ T0, T1, T2, T3 \} \), whose ranges are shown in the second column: a job consisting of 8 tasks resubmitted 10 times until completion is assigned the values T0 and R1 for T and R, respectively. Attributes have been discretized in a way to preserve the inherent power-law variability of the data as follows. Figs. 4a-4c suggest that the CDFs (i) increase sharply when the value of the attribute is small and (ii) start flattening until they converge to 1 over a very wide range of values. For each attribute T, D, R and C, the category X0 catches the initial increase of the CDF, while X1 and X2 cover the range of values before and after the knee-point, which characterizes the transition of the CDF from the sharp initial increase to the flat part.

Priorities (P) have been grouped in low (0-3), medium (4-7) and high (8-11). Fig. 4d shows the number of jobs by priority, which suggests our discretization suits well the data. The type of server (S) is categorical. There exists three types of servers - A, B and C: the tasks of a job are allocated to one or more of these servers. The servers are listed in the Machine_attribute table; Garraghan et al. [28] describe how the types can be mapped to physical machines. Values of \( V_S \) in Table 2 were inferred based on the server(s) the job was allocated to for execution.

It is worth investigating whether potential relationships among jobs influence the data precision. To this aim we measured the information content of the attributes through the notion of information gain [29], used here to quantify the usefulness of the attributes at predicting the label of the jobs (finished, killed, failed). The gain (IG) per attribute is in Table 2. We believe that no small subset of attributes predominates at predicting the label, and we decided to keep all attributes in the mining step.

**Sensitivity analysis.** The percentage of jobs falling into the categories of a given attribute (referred to as percentage cardinality in the following) depends on the ranges. For example, with the ranges in Table 2, T0, T1, T2 and T3 contain the 76.64, 22.38, 0.93 and 0.05% of the total jobs, respectively. Although we arranged the categories in a way to preserve the original power-law variability of the attributes, a different selection of the ranges might impact the cardinality and, in turn, invariants-related inferences made on the categories. We assess the sensitivity of the cardinality with respect to variations of the ranges shown in Table 2.

The assessment has been done as follows. It should be noted that the ranges are uniquely determined by three bounds, i.e., \( b_1 \), \( b_2 \), and \( b_3 \). For example, the bounds of T and C are 10, 1K, 5K and 0.05, 0.1, 0.2, respectively. Given an attribute and the bounds \( b_i (i=1,2,3) \), we compute the cardinality of the categories for each \( b_i \pm \frac{b_i}{100} \cdot \Delta \) (with \( \Delta=[5;50] \) by step 5). In other words, for each percentage variation \( \Delta \) we collect 6 observations per category. Fig. 5 summarizes the results of the sensitivity analysis (y-axis is in log-scale to better visualize X2 and X3). The dotted lines represent the percentage cardinality of each category, obtained using the bounds in Table 2; any other data point is the percentage cardinality of a given category subject to the percentage variation of the bounds reported by the x-axis. For example, when the bounds of tasks vary by 50% (i.e., \( b_1=15 \), \( b_2=[500;1500] \) and \( b_3=[2500;7500] \)) the average cardinality of T0, T1, T2 and T3 is 77.66, 21.26, 1.01 and 0.07% (Fig. 5a), which is close to the above-mentioned values obtained through the ranges in Table 2. Similarly, the cardinality of C2 and C3 (Fig. 5c) - which would seem the most variable categories due to the log scale - is 0.25 and 0.012 with our ranges selection, and, on average, 0.34 and 0.052 under a 50% variation of the bounds. The results suggest that the variation of the bounds does not significantly impact the cardinality of the categories. In this respect, it can be reasonably stated that the results presented hereafter for the Google dataset hold under different selections of the bounds.

### 4.1.2 SaaS dataset

The workload unit in the SaaS dataset is the processing stage of a transaction. We abstracted the processing stage through the following attributes, available in the Details audit log (Table 1):
- Stage (S): the name of the processing stage;
- Exit code (E): the outcome returned at the end completion of the stage (i.e., Status column);
- Reason (R): the reason leading to a given exit code (i.e., Fail Reason column).

Table 3 lists the top-8 occurring values of the stage (S) attribute, the four possible exit codes (E), and typical reasons (R) encountered in the dataset. Differently from the previous dataset, all the attributes are categorical in the SaaS log. The label of each

<table>
<thead>
<tr>
<th>Tasks (T)</th>
<th>Resub. (R)</th>
<th>Priority (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IG: 0.2071</td>
<td>IG: 0.115</td>
<td>IG: 0.119</td>
</tr>
<tr>
<td>V_T range</td>
<td>V_R range</td>
<td>V_P range</td>
</tr>
<tr>
<td>T0 [0;10]</td>
<td>R0 0</td>
<td>low [0;3]</td>
</tr>
<tr>
<td>T1 [10;1K]</td>
<td>R1 [0;100]</td>
<td>medium [4;7]</td>
</tr>
<tr>
<td>T2 [1K;5K]</td>
<td>R2 [500;1K]</td>
<td>high [8;11]</td>
</tr>
<tr>
<td>T3 [5K;+∞]</td>
<td>R3 [1K;+∞]</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 2: Google dataset: Attributes and ranges of values.**

**Fig. 5:** Sensitivity of the cardinality (%) with respect to the percentage variations of the bounds of the categories.
processing stage (either correct or anomalous), which denotes the correctness of its execution, has been provided by the experts of the IT company operating the SaaS platform. The execution is anomalous (i.e., the stage belongs to the anomalous class) if (i) reason is NULL when the stage exits with code L1_REJ or L2_REJ; (ii) the exit code is not L1_REJ when the processing stage is IT4_L1 and reason is File\_Validation\_Failure. Out of total 30,025 processing stages executions, 3,299 were anomalous.

### 4.2 Mining Techniques

The **invariant mining** step shown in Fig. 3 aims to infer recurring patterns among the attributes of the workload units. Likely patterns represent invariants, i.e., properties holding across different executions of batch work.

Let us clarify the notion of invariants through examples. In the Google dataset we noted that 54,976 jobs assume the values R0, low and D0 for attributes R, P, and D, respectively, meaning that a significant number of jobs experiencing no task resubmissions have low priority and small duration. Similarly, in the SaaS dataset, 10,701 processing stages assume the value IT3, L1_REJ, Invalid\_File (for S, E and R, respectively), indicating that the stage IT3 exiting with code L1_REJ fail because of an invalid file.

There are a number of considerations underlying the choice of the **clustering**, **association rules** and **decision list** mining techniques. First, production systems might generate clustering the and **reason** is **null**. **Invalid File** (for **S**, **E** and **R**, respectively), indicating that the stage **IT** 3 exiting with code **L1 REJ** fail because of an **invalid file**. **IT4 L1** and reason is **File Validation Failure**. Out of total 30,025 processing stages executions, 3,299 were anomalous.

#### 4.2.1 Clustering

The values of the attributes of each workload unit identify a point in a \( N \)-dimensional space. Fig. 6 shows the 3D scatterplot of all processing stages available in the SaaS dataset. It can be noted that the 30,025 stage concentrate around a few tens data points. A similar consideration can be done in the Google dataset. This technique identifies clusters (also known as groups) of data points. More important, as pointed out in [1], invariants should be comprehensible and useful to practitioners. Alternative invariant-based classifiers can be applied, e.g. neural or Bayesian networks; however, their output, e.g., probabilities and/or weights, have small explicative power for practical purposes.

#### 4.2.2 Association Rules

The second technique is **frequent itemset mining**, which extracts frequently observed patterns in a database in the form of **itemsets** or **association rules**. This technique is well known in the field of market basket analysis, where it is used to find out sets of products that are frequently bought together [32]. We apply the association concept to values of attributes.

Let \( B = \{i_1, \ldots, i_m\} \) be a set of items, any \( S \subseteq B \) an itemset, and \( T \) the bag of transactions under consideration (a transaction is a set of items). The absolute support (the relative support) of \( S \) is the number of transactions in \( T \) (the percentage of transactions in \( T \) that contain \( S \)). More formally, let \( U = \{X \in T \mid S \subseteq T\} \) be the set of transactions in \( T \) that have \( S \) as a subset (i.e., contain all the items in \( S \) and possibly some others). Then \( \text{suppabs}(S) = \frac{|U|}{|T|} \times 100\% \) is the absolute support of \( S \), and \( \text{supprel}(S) = \frac{|U|}{|T|} \times 100\% \) is the relative support of \( S \). Here \( |U| \) and \( |T| \) are the number of elements in \( U \) and \( T \), respectively.

In this study, Apriori [33] and Generalized Sequential Patterns (GSP) [34] are used to mine association rules. The attributes matrix constructed through the **workload abstraction** step in Fig. 3 is assumed to be the above-mentioned bag of transactions \( T \) (i.e., each row of the matrix is a transaction).

We use Borgelt’s implementation of Apriori [35], which returns the **maximal itemsets** in \( T \) whose relative support is larger than a threshold. The **support threshold** \( s \) is an input of the algorithm: the smaller it is, the larger the number of association rules that will be returned by the algorithm. Differently from Apriori, GSP performs different scans over \( T \). At any scan \( k \) (with \( k>1 \)) GSP generates a set of candidate \( k \)-sequences from frequent \((k-1)\)-sequences; candidate \( k \)-sequences undergo a pruning step. GSP stops when no more frequent sequences are found.

Association rules returned by either Apriori or GSP are assumed to represent an invariant. Rules are sorted by decreasing values of the support, i.e. by decreasing likelihood.

#### 4.2.3 Decision List

A decision list is an ordered set of classification rules. Given a workload unit abstracted by the value of the attributes, the list is scanned until a rule is found that matches the attributes: the label...
of the unit is assumed to be the one indicated by the rule. We use two different algorithms to obtain a decision list. The former is partial-decision-trees-based (PART) [36], combining the best of C4.5 and RIPPER, which represent the primary approaches to rule-based learning. The latter is the Decision Table / Naïve Bayes (DTNB) [37] algorithm, which combines a naïve Bayes approach with induction of decision tables.

Fig. 7 lists some of the 91 classification rules obtained for the Google dataset with PART. For instance, a job where \( T=\text{T2} \) and \( R=\text{R0} \) is classified as \text{KILLED} regardless the value of the remaining attributes because it matches the rule at line 2; similarly, by looking at line 4 and 6 it can be noted that a job where \( T=\text{T0} \) and \( R=\text{R0} \) and \( P=\text{High} \) and \( D=\text{D2} \) is classified as \text{FINISHED} if (i) it has been run on the server type \( B \) (regardless the CPU usage) or (ii) its CPU usage has been \( \text{CO} \) in the case the server type is \( C \).

Differently from clustering and association rules, decision list is a supervised technique because the model is learned from a labeled dataset (i.e., beside the attributes matrix, the construction of the tree requires the knowledge of the label of each workload unit). In this study, the rules in the list that aim to catch the \text{correct} workload units are deemed to be invariants; they are sorted by decreasing number of correct units they detect.

1 \( \text{if} \ (T=\text{T2} \text{ and } R=\text{R1}) \text{ then } \text{KILLED} \)
2 \( \text{else if} \ (T=\text{T2} \text{ and } R=\text{R0}) \text{ then } \text{KILLED} \)
3 \( \text{//omitted} \)
4 \( \text{else if} \ (T=\text{T0} \text{ and } R=\text{R0} \text{ and } P=\text{High} \text{ and } D=\text{D2} \text{ and } S=\text{B}) \text{ then } \text{FINISHED} \)
5 \( \text{//omitted} \)
6 \( \text{else if} \ (T=\text{T0} \text{ and } R=\text{R0} \text{ and } P=\text{High} \text{ and } D=\text{D2} \text{ and } C=\text{C0}) \text{ then } \text{FINISHED} \)
7 \( \text{//omitted} \)
8 \( \text{else if} \ (P=\text{SMEDIUM}) \text{ then } \text{FINISHED} \)
9 \( \text{default } \text{FINISHED} \)

Fig. 7: Examples of decision list rules for the Google dataset.

### 5 Evaluation Metrics

The set of invariants \( I = \{i_1, i_2, \ldots, i_j\} \) returned by a mining technique is assessed through widely established information retrieval metrics, in order to quantify to what extent the invariants are able to (i) abstract recurring properties of the executions of workload units (i.e., jobs or processing stages), and (ii) discriminate correct/anomalous executions. The invariants in \( I \) are sorted by likelihood, beforehand (as discussed in Section 4.2). We assess how the metrics vary by using a progressively increasing number of less-likely invariants from \( I \); this strategy allows to establish the number of invariants that properly characterize a dataset.

Let \( s_i \) \((1 \leq i \leq 10)\) denote the subset of the top-\(i\) invariants. For example, in the case \( I = \{i_1, i_2, i_3, i_4\} \) four sets are obtained, i.e., \( s_1 = \{i_1\} \), \( s_2 = \{i_1, i_2\} \), \( s_3 = \{i_1, i_2, i_3\} \), \( s_4 = \{i_1, i_2, i_3, i_4\} \). It can be noted that \( s_3 \) ranges between the sets \( \{i_1\} \) and \( I \), denoting the most likely invariant and all the invariants in \( I \), respectively.

Each subset \( s_i \) is run against the input dataset. A workload unit in the dataset is assigned to one out of four disjoint classes based on the result of the comparison between the (i) label and (ii) outcome of the invariant-based checking. The sets (Fig. 8) are:

- **true negative** (TN): the workload units (\( W \)) with label correct and matching at least one invariant in \( s_i \);
- **false negative** (FN): set of \( W \) with label anomalous and matching at least one invariant in \( s_i \);
- **false positive** (FP): set of \( W \) with label correct and matching no invariant in \( s_i \);
- **true positive** (TP): set of \( W \) with label anomalous and matching no invariant in \( s_i \).

![Fig. 8: Confusion matrix.](image)

Given \( s_i \), we compute the coverage \( (C) \) as the ratio between the number of workload units matching at least one invariant in \( s_i \) and the total number of workload units, and specificity \( (S) \), recall \( (R) \), and precision \( (P) \) as follows:

\[
S = \frac{|TP|}{|TP| + |FP|}, \quad R = \frac{|TP|}{|TP| + |FN|}, \quad P = \frac{|TP|}{|TP| + |FP|}
\]

*Specificity* is the ratio between the number of correct workload units detected by \( s_i \) and the total number of correct workload units. *Recall* is the probability that an anomalous workload unit is detected by \( s_i \). *Precision* is the probability that a workload unit, which matches no invariant in \( s_i \), is actually anomalous. *Coverage* can be computed also for unlabeled datasets.

Let us present an example. The use of clustering with \( K=10 \) for the Google dataset returns the invariants listed in Table 4 by decreasing likelihood.

![TABLE 4: Google dataset: invariants mined by clustering (K=10).](image)

Based on these, it is possible to build 10 subsets \( s_i \), whose \( C, S, R, P \) are listed in the four rightmost columns of Table 5. Since invariants in \( I \) are ordered by decreasing likelihood, the cardinality of \( s_i \) indicates the invariants of \( I \) being assessed. For instance, \( s_3 \) consists of the three most likely invariants \( \{\text{Low, D2, T0, C0, R0}\}, \{\text{Med, D2, T1, C0, bc, R0}\}, \{\text{Med, D2, T1, C0, bc, R0}\} \), namely \( i_1, i_2 \) and \( i_3 \) in Table 4. They match a total of 160,560 jobs (\( \text{TN} + \text{FN} \)), with a coverage of 0.25; the specificity \( S \) equals 0.15 because \( s_3 \) detects 57,117 TNs out of total 372,688 negatives. The use of two further invariants, i.e. \( s_5 \), would increase the coverage from 0.25 to 0.38.

![TABLE 5: Google dataset: values of the evaluation metrics for invariants of Table 4.](image)
of executions. Fig. 9b shows how precision/recall vary with the specificity: the plot allows appreciating the tradeoff between the number of correct executions detected by $I$ and the goodness of the detection. Fig. 9 summarizes the values in Table 5: for example, for the subset $s_3$ discussed above, in Fig. 9a we see $C=0.25$ and $S=0.15$; for $s=0.15$ (x-axis) in Fig. 9b, we obtain $R$ and $P$ of the three invariants in $s_3$ (0.63 and 0.36, respectively).

6 APPLICATIONS

The techniques are investigated in the context of two typical applications of invariant-based analysis, namely executions characterization and anomaly detection. We establish the best set of invariants returned by K-medoids and Apriori through a sensitivity analysis, beforehand; the comparison with the remaining algorithms is presented in Section 6.2 and 6.3.

6.1 Sensitivity Analysis

We assess the sensitivity of K-medoids and Apriori with respect to the input parameters $K$ (number of clusters) and $s$ (relative support threshold), respectively. For example, in the Google dataset the Apriori algorithm set with $s=38\%$ returns the invariants $\{(T0,C0,c,R0), (Low,T0,C0), (Low,C0,R0), (D2,T0,C0,R0), (D2,C0,c,R0), (Low,C0,c)\}$ if $s=35\%$. Lowering the support by 3\% results into different sets of invariants.

For the Google dataset, Fig. 10a and 10e show how recall and precision vary with respect to the specificity for invariants obtained under different values of $K$. Analysis has been done with $2 \leq K \leq 20$ by step 2; however, Fig. 10a and 10e show the results for $K=8,12,14,18$ for better visualization. For example, the top-3 invariants returned by means of K-medoids with $K=12$ (i.e., full blue, △-marked, series - third point from the left) have $S=0.25$, $R=0.65$, $P=0.39$; on the other hand, the top-3 invariants in $K=8$ (i.e., dotted grey, □-marked, series - third point from the left) have $S=0.22$, $R=0.71$, $P=0.40$. An optimal set of invariants - the one resulting into the best recall/precision with respect to specificity - is obtained with $K=14$ (i.e., dotted black, ×-marked, series).

For any value of $K$, Fig. 10a indicates that recall ($R$) is a monotonic decreasing function of the specificity. As shown by Eq. 1, while the denominator of $R$ is constant regardless the number of invariants ($|TP|+|FN|$ is the total number of anomalous workload units), the numerator of $R$, i.e., $|TP|$, decreases as specificity decreases. Differently from recall, precision ($P$) might either increase or decrease with the specificity.

Fig. 10b and 10f show recall and precision of the invariants obtained through Apriori with different values of the support threshold. Again, we explore values of the support generating from 2 to 20 invariants by step 2; however, a smaller subset is shown for the sake of clarity. As a result, we choose the invariants obtained with $s=6\%$ (dotted black, ×-marked, series).

For the SaaS dataset, Fig. 10c and 10g show recall and precision of the invariants mined by K-medoids with $K=8,10,12$. The best set is obtained with $K=10$ (dotted black, ×-marked, series). On the other hand, $s=12\%$ is the value of the support that allows obtaining the best set of invariants through Apriori, as it can be noted from Fig. 10d and 10h (dotted grey, □-marked, series).

3. The larger the number of invariants (and the coverage), the smaller the number of workload units that will be classified as anomalous (i.e., positive).
6.2 Executions Characterization

We assess to what extent the invariants returned by the techniques hold across the executions of different workload units. To this aim, coverage and specificity are measured. Table 6 and 7 report the 6 (for Google) and 3 (for SaaS) most likely invariants returned by the algorithms, respectively. The invariants mined by K-medoids, DBSCAN and DTNB indicate the value of all the attributes; Apriori, GSP and PART might specify only some attributes.

FIG. 11: Coverage of the techniques: Google (a-c), SaaS (d-f).

### TABLE 6: Google dataset: top-6 invariants mined by algorithms.

<table>
<thead>
<tr>
<th>K-medoids (K=14)</th>
<th>DBSCAN</th>
<th>Apriori (s=6%)</th>
<th>GSP</th>
<th>PART</th>
<th>DTNB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low, D2, T0, C0, c, R0</td>
<td>Low, D0, T0, C0, c, R0</td>
<td>Med, D2, T1, C0, bc, R0</td>
<td>D2, T0, C0, c, R0</td>
<td>Low, D0, T0, C0, c, R0</td>
<td>Low, D2, T0, C0, c, R0</td>
</tr>
<tr>
<td>High, D2, T0, C0, c, R0</td>
<td>High, D3, T0, C0, c, R0</td>
<td>High, D2, T0, C0, c, R0</td>
<td>D2, T0, C0, c, R0</td>
<td>Low, D0, T0, C0, c, R0</td>
<td></td>
</tr>
<tr>
<td>Med, D2, T0, C0, c, R0</td>
<td>Med, D2, T0, C0, c, R0</td>
<td>High, D3, T0, C0, c, R0</td>
<td>D2, T0, C0, c, R0</td>
<td>Low, D0, T0, C0, c, R0</td>
<td>Low, D2, T0, C0, c, R0</td>
</tr>
</tbody>
</table>

### TABLE 7: SaaS dataset: top-3 invariants mined by algorithms (IF: Invalid_File; FVF: File_Validation_Failure).

<table>
<thead>
<tr>
<th>K-medoids (K=10)</th>
<th>DBSCAN</th>
<th>Apriori (s=12%)</th>
<th>GSP</th>
<th>PART</th>
<th>DTNB</th>
</tr>
</thead>
<tbody>
<tr>
<td>IT3, IF, L1_REJ</td>
<td>IT3, IF, L1_REJ</td>
<td>IT3, IF, L1_REJ</td>
<td>IF, L1_REJ</td>
<td>L2_REJ</td>
<td>IF, L1_REJ</td>
</tr>
<tr>
<td>IT3_PVL, IF, L1_REJ</td>
<td>IT3_PVL, IF, L1_REJ</td>
<td>IT3, IF, L1_REJ</td>
<td>L2_REJ</td>
<td>NULL, L2_REJ</td>
<td>IT4, L2_REJ</td>
</tr>
<tr>
<td>IT4, T2, FVF, L2_REJ</td>
<td>IT4, NULL, L2_REJ</td>
<td>IT4, NULL, L2_REJ</td>
<td>FVF, L2_REJ</td>
<td>NULL, L2_REJ</td>
<td>IT4, T2, L2_REJ</td>
</tr>
</tbody>
</table>

Fig. 11 shows how the coverage varies with respect the number of invariants by technique and dataset. For each n along the x-axis, the y-axis is the value of the coverage achieved by the n most likely invariants. For example, the most likely 5 invariants returned by Apriori obtain a coverage of 0.60 in Google as it can be noted from Fig. 11b (∆-marked series), i.e., they hold in 389,029 out of total 649,959 job executions; similarly, the most likely 3 invariants returned by K-medoids in SaaS hold in 22,912 out of 30,025 processing stages, i.e., coverage is 0.76, as shown by Fig. 11d (∆-marked series).

The plots indicate that, for any technique/dataset, coverage flattens sharply after a relatively small number of invariants (i.e., roughly 8-9 in Google and 2-3 in SaaS): the use of an arbitrary large number of invariants does not help improving coverage significantly. For example, K-medoids starts flattening at the 9th invariant in Google, where coverage is 0.66: 6 more invariants contribute to increase the coverage by only 0.03, i.e., data point (14, 0.69) - K-medoids in Fig. 11a; similarly, the coverage of the top 4 GSP invariants in Google is 0.79, while 10 more invariants contribute to increase the coverage by roughly 0.1, i.e., Fig. 11b. The techniques achieve a rather different maximum coverage in Google, which ranges from 0.50 (DTNB) to 0.87 (Apriori).

In spite of the different coverage, the techniques converge to similar specificity (coverage of the correct executions), roughly 0.85 for Google and 0.9 for SaaS (Fig. 12). For instance, the coverage of PART is about 0.53 for Google, and its specificity is 0.8, i.e. 80% of correct executions match some invariant. The maximum specificity (0.97) is observed with GSP. No few-fits-all invariants can reasonably be mined for all the correct executions.

### 6.3 Anomaly Detection

Likely invariants have been used in past studies for anomaly detection. Properties that are likely invariants hold in regular operating conditions; their violation is considered symptom of execution malfunctions. This is a common practice in unsupervised learning, where the assumption is made that correct instances are far more likely than anomalies [25].
they match the same number of correct executions. Recall/precision vs specificity allows to compare techniques when they exhibit different specificity rather than the number of invariants. This is because the chance of holding in more and more executions; however, it also increases as specificity increases; however, it stops, or even drops, sharply after having reached a maximum value. This can be clearly noted both in Fig. 14. Again, the threshold is represented by the value 0.8-0.9 of the specificity. Surprisingly, precision of supervised techniques, such as PART and DTNB, does not outperform unsupervised algorithms, such as the ones used for clustering or association rules mining. For example, in Google the maximum value of precision obtained by PART/DTNB is around 0.75, while precision of Apriori and K-medoids is 0.72 and 0.70. Although GSP might seem to achieve up to 0.89 precision, it must be noted that the recall corresponding to such precision is only 0.3; in practice the precision of GSP cannot be pushed beyond 0.62 without compromising the recall.

The workload units have been closely investigated to gain insights into recall and precision. We count the number of workload units that assume a given combination of values of attributes, i.e., pattern, in the following. Due to the larger variability of the data, we present the results from Google, for which we found out 565 distinct patterns. Fig. 15 shows the 100 most frequent patterns by decreasing number of workload units. A point along the x-axis denotes a pattern; the y-axis is the number of workload units assuming the pattern broken down by correct (i.e., finished), and anomalous (i.e., killed or failed) classes, in logarithmic scale. For example, total 53,710 workload units assume the values (Low, D0, T0, C0, c, R0), being represented by pattern #3 (third column from the left). The top-10 patterns (i.e., the first ten columns at the left side of the dotted vertical line in Fig. 15) sum up to 465,665 jobs. Although the most frequent patterns consist of almost all correct workload units (i.e., finished in Fig. 15), they might still contain a certain number of anomalous executions: invariants supposed to catch the frequent patterns will inadvertently misclassify anomalous executions, which affects the value of the recall. More important, a very long tail of patterns encompasses a significant number of correct executions. The remaining 555 patterns in Fig. 15 (i.e., right side of the dotted vertical line) account for total 47,053 correct executions. In order to correctly classify these executions, invariants should be able to depict such a large number of patterns. However, unfrequent patterns, although correct, would not result into likely invariants: as a result, precision is impacted.

4. Fig. 13 is very close to the receiver operating characteristic (ROC) curve; however, the x-axis is specificity rather than (1 – specificity) in order to visualize the effect of increasing number of invariants from left to right.
7 Practical Implications

Experimental results reveal pros and cons of widely-established techniques in invariant-based analysis. We leverage the results to infer practical implications that allow dealing with the concrete selection of likely invariants by operations engineers. We address the problems of (i) setting the input parameters of the mining algorithms (K-medoids and Apriori) before the analysis and (ii) selecting the optimal number of invariants out of the output produced by a mining algorithm. We propose a heuristic that can be used to select likely invariants. It is worth noting that our heuristic does not require labeled data: in this respect, it can be generally applied also to unlabeled datasets, where practitioners are expected to make decisions without the knowledge of metrics, such as recall and precision.

7.1 Invariants Selection

Invariants are inferred in two steps, i.e., (i) construction of $I$, i.e., the set of recurring properties (ordered by decreasing likelihood) among the attributes in a given dataset, and (ii) selection of a subset of invariants in $I$.

7.1.1 Setting of the mining algorithm

Algorithms, such as K-medoids and Apriori in this study, might require the setting of an input parameter to mine invariants. We measure how the invariants returned by an algorithm vary with the metrics. It can be noted that, differently from a number of invariants taken where coverage is saturated, the knee-points (percentages denote supports; number of clusters, otherwise).

It can be noted that $J_c$ decreases as the number of invariants increases, which indicates that the sets $I$ start stabilizing (i.e., the values of the attributes that characterize the invariants become similar - small $J_c$). In 3 out of 4 cases, the best set of invariants selected according to the sensitivity analysis presented in Section 6.1, i.e., grey, o-marked points in Fig. 16, correspond to small values of $J_c$, if not even 0, such as the case of K-medoids. As a rule of thumb, we suggest to run a mining algorithm under different values of the input parameters; values should be selected in a way to allow the algorithm to return a progressively increasing number of invariants. The computation of $J_c$ between consecutive sets $I$ highlights the value of the input parameter where invariants stabilize: according to our data, a good set of invariants is returned by a configuration of the input causing $J_c$ to be small.

7.1.2 Number of invariants

Beside the setting problem, which occurs for those algorithms relying on an input parameter, practitioners are required to select a subset of likely invariants out of the output of any mining algorithm. Selecting the proper subset of invariants is critical: Fig. 11 and 12 indicate that few likely invariants contribute to increase coverage/specificity sharply; Fig. 13 and 14 suggest that recall and precision are negatively impacted by increasing values of the specificity. The problem of selecting invariants is even exacerbated in unlabeled dataset, where specificity/recall/precision cannot be computed. We address the problem as follows.

Experimental results provide reasonable evidence of the following relationship among number of invariants, coverage and classification-related metrics: selecting a number of invariants where coverage is saturated, results in a value of specificity that corresponds to the threshold where recall and precision start decreasing sharply. This can be noted across different techniques and datasets. We discuss a data point for the sake of clarity: the coverage of K-medoids in Google enters saturation at the 9th invariant, i.e., Fig. 11a. A number of invariants bigger than 9, e.g., 12, obtains a specificity of 0.83, i.e. Fig. 12a: the value 0.83 denotes the point of the x-axis in Fig. 13a and 14a where recall and precision of K-medoids have decreased/stopped sharply.

Let us denote knee-point of the coverage the number of invariants where coverage stops increasing significantly: the knee-point represents the beginning of the coverage saturation. For each dataset/technique, Table 8 shows the knee-points (bold character) and the points immediately before/after the knee, as it can be inferred from Fig. 11; moreover, the table shows the value of the metrics. It can be noted that, differently from a number of invariants taken where coverage is saturated, the knee-points represent a reasonably good tradeoff between recall and precision.
Accordingly, we propose the following heuristic: a good number of invariants is indicated by the knee-point of the coverage. The heuristic is generally applicable in practice because the computation of the coverage does not require the knowledge of the label.

We note that association rules are not the best suited for anomaly detection. Table 8 indicates that Apriori and GSP achieve the minimum recall for the datasets. Nevertheless, given the high coverage, association rules are still useful to characterize the system execution (e.g., to support workload characterization, usage profiling and capacity planning). On the other hand, decision list is strongly recommended for anomaly detection; however, it needs the label in order to be applied. Clustering might be used in place of decision list to deal with unlabeled datasets.

TABLE 8: Number of invariants selected through the knee-point heuristic, and values of evaluation metrics.

<table>
<thead>
<tr>
<th>Google dataset</th>
<th>SaaS dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$C$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>K-medoids</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
</tr>
<tr>
<td>Apriori</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td>GSP</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>PART</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
<tr>
<td>DTNB</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

Fig. 17: Google dataset: frequency distribution of the attribute values of the anomalous workload units.

7.2 Results Validation

The invariants selected with the proposed heuristic are used in the Google dataset. Fig. 17 shows the frequency distribution of the attribute values of the anomalous class (killed and failed jobs), comparing the actual distribution (i.e., actually anomalous jobs) to those obtained through K-medoids, GSP and DTNB. For example, the CPU usage of 271,283 actual anomalous jobs is C0; this value is 214,655 in K-medoids, 129,155 in GSP and 333,404 in DTNB. We note that DTNB infers a very similar distribution when compared to the actual data series: in fact anomaly detection done by means of this technique results into the maximum recall and precision (0.89 and 0.73, respectively), as shown in Table 8. These figures are consistent with [20], which proposes a failure prediction study of the Google dataset. At the other end of the spectrum, GSP produces a rather different distribution. For example, although Low is the most frequent priority value of anomalous jobs, GSP would erroneously suggest that the frequency of the priority values of anomalous jobs is similar. K-medoids allows preserving many of the characteristics of anomalous jobs. For example, differently form GSP, it correctly infers that Low is the most likely priority of anomalous jobs.

Similarly, we analyze the anomalies detected through invariants in the SaaS dataset. Discussion focuses here on decision list; however, similar results have been noted in the other techniques. The three invariants in Table 7 - $i_1$, $i_2$, $i_3$ (PART) - detect 5,293 anomalous processing stages executions. We group the stages assuming the same attribute values into the same class and obtain total 24 classes. Table 9 shows the classes, whose cardinality is reported by the leftmost column of the table. According to the notion of invariant, none of the classes in Table 9 matches $i_1$, $i_2$, $i_3$. For instance, in no case Reason in Table 9 is Invalid_File.

Each class has been carefully reviewed by the SaaS service operation experts of the IT company with the aim of validating the results. For example, the top two classes, which account for 1,981 and 1,321 instances, respectively, are true anomalies because they match the domain rules in Section 4.1.2. False anomalies are denoted by $\times$ in Table 9: as also noted in Google, correct executions that generate infrequent combinations of attributes will likely cause false positives. For example, this is the case of the classes assuming the value File_already_exist as Reason.

The experts team acknowledged that 5 out of 24 classes needed further investigation: they are denoted by ‘!’ in Table 9. Beside the ones encompassed by the domain rules, invariant-based analysis detected total 461 stage executions reporting a NULL value for the Reason. Our analysis revealed that the stages exiting with the SE code might occasionally report no reason. Although this behavior of the SaaS platform was unnoticed until our study, the experts confirmed that it did not impact operations. On the contrary, 6 out of the above-mentioned 461 cases, i.e., the IT4, NULL, MIN_PP class in Table 9, were taken on high priority because they represented a silent data corruption leading to a system failure. The operations team confirmed that during the month of September 6 transactions were erroneously handled because of the missing Reason field from the processing stage. Overall, these findings were valuable to the service operation team.

8 Threats to Validity

As for any data-driven study, there may be concerns regarding the validity and generalizability of the results. We discuss them, based on the four aspects of validity listed in [39].

Construct validity. The study is based on two independent, real-world datasets, representative of two important categories of service computing platforms. Both sets contain data collected in operation under the natural workload/faultload, and encompass a total of about 680,000 data points concerning jobs, tasks and processing stages. The study builds on experiments aiming to infer possibly general insights, useful towards putting invariant-based techniques into common practice. This is pursued by considering well-established algorithms available in widespread...
TABLE 9: Anomalies detected by decision list (PART) in SaaS with three likely invariants. Answer by the SaaS experts: true anomaly (√), false anomaly (×), needed further investigation (!).

<table>
<thead>
<tr>
<th>#</th>
<th>Stage</th>
<th>Reason</th>
<th>Exit code</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.981</td>
<td>IT4</td>
<td>NULL</td>
<td>L2_REJ</td>
<td>√</td>
</tr>
<tr>
<td>1.312</td>
<td>IT4</td>
<td>NULL</td>
<td>L1_REJ</td>
<td>√</td>
</tr>
<tr>
<td>355</td>
<td>IT3_PVL</td>
<td>Corrupt_File</td>
<td>L1_REJ</td>
<td>×</td>
</tr>
<tr>
<td>354</td>
<td>IT3</td>
<td>Corrupt_File</td>
<td>L1_REJ</td>
<td>×</td>
</tr>
<tr>
<td>350</td>
<td>IT4</td>
<td>NULL</td>
<td>SE</td>
<td>!</td>
</tr>
<tr>
<td>1.54</td>
<td>IT4_TKN</td>
<td>System_Error</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>92</td>
<td>IT4_L2</td>
<td>System_Error</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>79</td>
<td>IT4_STG</td>
<td>NULL</td>
<td>SE</td>
<td>!</td>
</tr>
<tr>
<td>35</td>
<td>IT5</td>
<td>Load_failed</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>28</td>
<td>IT3_P</td>
<td>File_is_not_movable</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>28</td>
<td>IT3</td>
<td>File_is_not_movable</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>24</td>
<td>IT5</td>
<td>System_Error</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>20</td>
<td>IT5</td>
<td>NULL</td>
<td>SE</td>
<td>!</td>
</tr>
<tr>
<td>19</td>
<td>IT4_STG</td>
<td>System_Error</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>12</td>
<td>IT3_P</td>
<td>Move_Failed</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>12</td>
<td>IT3</td>
<td>Move_Failed</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>7</td>
<td>IT3_PVL</td>
<td>Infected_File</td>
<td>L1_REJ</td>
<td>×</td>
</tr>
<tr>
<td>7</td>
<td>IT3</td>
<td>Infected_File</td>
<td>L1_REJ</td>
<td>×</td>
</tr>
<tr>
<td>6</td>
<td>IT4_PREP</td>
<td>NULL</td>
<td>SE</td>
<td>!</td>
</tr>
<tr>
<td>6</td>
<td>IT4_L1</td>
<td>File_Validation_Failure</td>
<td>MIN_PP</td>
<td>√</td>
</tr>
<tr>
<td>6</td>
<td>IT4</td>
<td>NULL</td>
<td>MIN_PP</td>
<td>!</td>
</tr>
<tr>
<td>2</td>
<td>IT6</td>
<td>System_Error</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>2</td>
<td>IT3_P</td>
<td>File_already_exist</td>
<td>SE</td>
<td>×</td>
</tr>
<tr>
<td>2</td>
<td>IT3</td>
<td>File_already_exist</td>
<td>SE</td>
<td>×</td>
</tr>
</tbody>
</table>

mining packages. More sophisticated techniques, such as feature engineering, may complement invariant-based analysis. However improving performance through ad hoc fine-tuning of algorithms for the specific dataset at hand is not within the perspective of this study, and it might have made the results tailored on the chosen datasets. We are confident that the details provided support the replication of our study by researchers and practitioners.

**Internal validity.** We used two different datasets and six mining algorithms to provide evidence of the actual relationships among the variables under assessment, such as number of invariants, coverage and information retrieval metrics. The use of a mixture of diverse datasets and techniques mitigates internal validity threats. The key findings of the study are consistent across the datasets and techniques, which provides a reasonable level of confidence on the analysis.

**External validity.** The steps of the analysis should be easily applicable to similar systems/datasets supporting the abstraction of workload units and attributes, such as systems performing batch work. Attributes, such as computing resources, duration, priority and return codes of jobs/tasks, are collectable by many established monitoring tools or available through systems/applications logs. For example, the simplest and most common way to extract attributes from logs is to `grep` the messages tracking performance and usage statistics [40]. Once mined, actionable invariants can be implemented through regular expressions to monitor production logs. Given the wide spread of log management tools, invariant mining and related applications are reasonably feasible in practice. The overhead entails the time required to establish the message types of the log that contain the metrics of interest; this is done once at the beginning of the analysis. Maybe more important, invariant-based analysis does not interfere with the system operations; features extraction, invariants mining and application can be entirely automated. Our findings, supported by measurements on real data, are useful to get an overall understanding of the characterization that can be performed through invariants and its practicability and limitations in real-world systems. We provided a number of practical suggestions in order to support the generalization of the analysis to both labelled and unlabeled datasets.

**Conclusion validity.** Conclusions have been inferred by assessing the sensitivity of the results with respect to the experimental choices. We assessed the sensitivity of the categories with respect to the selection of the ranges in the Google dataset: analysis indicates that the categories are not biased by the specific selection of the ranges adopted in the paper. We replicated the analysis under different configurations of key parameters, i.e., number of clusters and support. We assessed the sensitivity of the evaluation metrics with respect to the setting of the underlying mining algorithm: comparisons have been made across the optimal set of invariants in order to ensure the findings have been not biased by a particular configuration. Inferences made for the Google cluster are consistent with those of other cited studies on the same dataset. The validity of the invariant analysis for the SaaS dataset involved direct communication with the cloud operations team, which confirmed the anomalies went otherwise unnoticed.

9 Conclusions

Likely system invariants can be mined for a variety of service computing systems, including cloud systems, web service infrastructures, datacenters, enterprise systems, IT services and utility computing systems, network services, distributed systems. They represent operational abstractions of normal system dynamics. The identification and the analysis of their violations support a range of operational activities, such as runtime anomaly detection, post mortem troubleshooting, capacity planning. In this work we have used two real-world datasets - the publicly available Google datacenter dataset and a dataset of a commercial SaaS utility computing platform - for assessing and comparing three techniques for invariant mining. Analysis and comparison was based on the common metrics coverage, recall and precision. The results provide insights into advantages and limitations of each technique, and practical suggestions to practitioners to establish the configuration of the mining algorithms and to select the number of invariants. The high-level findings are the following. A relatively small number of invariants allows to reach a relatively high coverage, i.e., they characterize the majority of executions. A small increase of the coverage of correct executions may produce a significant drop of recall and precision. The techniques exhibit similar precision, but the decision list supervised technique outperforms the unsupervised ones in recall. Finally, we presented a general heuristic for selecting a set of likely invariants from a dataset. All these results aim to fill the gap between past scientific studies and the concrete usage of likely system invariants by operations engineers.

Acknowledgments

The work by A. Pecchia and S. Russo has been supported by the GAUSS national research project (CUP E52F16002700001), funded by MIUR under the PRIN 2015 program. The work by S. Sarkar has been initially carried out at Infosys Labs, India. His later work at BITS Pilani has been partially supported by a research grant from Accenture Technology Labs, USA.