Assessing Invariant Mining Techniques For Cloud-based Utility Computing Systems

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.

EXISTING SYSTEM

- 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

DISADVANTAGES

- It is very hard for human operators to detect application problems in real time.

PROPOSED SYSTEM

- The study focuses on three techniques: two unsupervised, namely clustering and association rules, and one supervised, de-cision 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 pack-aged 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 applica-tions of invariant-based analysis, namely executions

ADVANTAGES

- Accuracy and completeness of invariant-based anomaly detection.
- A well tuned approach can reach good completeness at an accuracy

HARDWARE REQUIREMENTS

- System : Pentium IV 2.4 GHz.
- Hard Disk : 40 GB.
- Floppy Drive : 1.44 Mb.
- Monitor : 15 VGA Colour.
- Mouse : Logitech.
- Ram : 512 Mb.

SOFTWARE REQUIREMENTS

- Operating system : Windows XP/7.
- Coding Language : ASP.net, C#.net /java

CONCLUSION

- 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 sys-tems. 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.

REFERENCE

- [1] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin, “Dynamically Discovering Likely Program Invariants to Support Program Evolution,” IEEE Trans. on Software Engineering, vol. 27, pp. 99–123, 2001.
- [2] C. Pacheco and M. D. Ernst, “Eclat: Automatic Generation and Classi-fication of Test Inputs,” in Proc. 19th European Conference on Object-Oriented Programming (ECOOP), pp. 504–527, Springer, 2005.
- [3] C. Csallner and Y. Smaragdakis, “Dynamically discovering likely inter-face specifications,” in Proc. 28th Int. Conference on Software Engineer-ing (ICSE), pp. 861–864, ACM, 2006.