Noah L. Schrick
Professor: Dr. Abdulrashid
Final Project Proposal

Topic: Data Mining Techniques on Algorithm Analysis for Compliance Graph Generation

Introduction

Compliance graphs are generated graphs (or networks) that represent systems' compliance or regulation standings at present, with expected changes, or both. These graphs are generated as directed acyclic graphs (DAGs), and can be used to identify possible correction or mitigation schemes for environments necessitating compliance to mandates or regulations.

Compliance graphs are an appealing approach since they are often designed to be exhaustive: all system properties are represented at its initial state, all violation options are fully enumerated, all permutations are examined, and all changes to a system are encoded into their own independent states, where these states are then individually analyzed through the process. Despite their advantages, compliance graphs do suffer from their exhaustiveness as well. As the authors of [1] examine, even very small networks with only 10 hosts and 5 vulnerabilities yield graphs with 10 million edges. When scaling compliance graphs to analyze the modern, interconnected state of large networks comprising of a multitude of hosts, and utilizing the entries located in the National Vulnerability Database and any custom vulnerability or regulation testing, graph generation quickly becomes infeasible. Similar difficulties arise in related fields, where social networks, bioinformatics, and neural network representations result in graphs with millions of states [2]. This state space explosion is a natural by-product of the graph generation process, and removing or avoiding it entirely undermines the overall goal of compliance graphs.

Related research works such as those seen by the authors of [2], [4], [5], [6], and [7] extend the generation process to function on distributed computing environments to take advantage of the increased computing power using message-passing. The approach utilized for this work is novel, and makes use of a task parallelism approach for the generation process, and examines the effect of various parameter tuning and algorithm design choices and their effects on speedup and efficiency.

Objective

The goal of this work is to analyze the timing data obtained through the deployment of the new MPI tasking algorithm. By analyzing this data, an answer will be sought out for the following questions:

1. How does each parameter contribute to the overall runtime?
2. Which parameters have the largest effect on overall runtime?
3. Can an optimal set or subset of parameters with corresponding values be identified?
4. What is the best way to visualize and represent the data and findings?

Analyzing this data will provide further insight into the effectiveness of the algorithm, and allow future contributions to be made for algorithmic improvements based on the successes identified with this approach.

Data
The algorithmic approach is to break the generation process into six main tasks. Dependent variables of the data are time per task and overall runtime. To measure the speedup and efficiency, data is collected for each task based on parameter tuning, as well as data for the overall process. Reducing the dependent variables, it can be shown that there is one dependent variable (overall runtime) that is comprised of 6 sub-dependent variables (time per task).

Across the six main tasks and throughout the generation process, a total of 4 additional independent variables are used. These independent variables are: the number of compute nodes used, the number of exploits, the number of applicable exploits, and the memory load before database operations are performed. Each variable is changed individually across a given range to capture a full-spectrum image of their effects on time.

The number of nodes ranges from 2 through 12, incrementing by 1.
The number of exploits ranges from 6 to 49152, incrementing by a factor of x2.
The number of applicable exploits ranges from 0% to 100%, incrementing by 25%.
The database load ranges from 0% to 100%, incrementing by 25%.

The dataset therefore has 2,464 entries, with each entry spread across 11 columns, leading to a total of 27,104 individual pieces of data. The data will be collected in an automated fashion. The experiment will be deployed on the University's supercomputer cluster, with custom scripting to automatically run each subsequent test, and collect the data into a raw, unfiltered CSV.

Since this approach is novel, this data is not public, and has not been analyzed by any other individuals or groups.

Methodology
The methodology will utilize multivariate analysis, ridge regression, covariance, sums of squares, general regression techniques, heatmaps, and will leave room to explore methods such as PCA or LASSO for dimensionality reduction.

Since data is to be collected only for the effect of the parameters on overall runtime, additional care is needed to ensure that noise is filtered as best as possible. Timings for the algorithm need to be carefully inserted so as to not bias or detract from the goal of the research. Substantial discussion will need to be included to justify the regression approaches.

Expected Outcome
Ideal results will include definitive answers to analysis questions 1, 2, and 3 through figures and various discussion. Results for analysis question 4 will include figures along with a discussion on whether the figures successfully capture all findings.

The expected outcome will, at minimum, provide insight into the effectiveness of the algorithm. The outcome should also highlight the effectiveness of each task within the algorithm, and determine if individual components are or are not effective.

Ideal conclusions will have definitive answers to analysis questions 1 through 3, with substantive figures and discussion for analysis question 4.

Limitations of Analysis/Data and Future Works

A discussion and justification will be required for the regression techniques and PCA and/or LASSO. PCA and LASSO will both work toward dimensionality reduction, but achieves the goal in separate means. A discussion of limitation will be required to best answer which approach best fits the data, and shortcomings of each approach as it relates to the data.

Based on the results, future work could include discussion of future algorithm improvements, suggestions for corrections or alternate approaches on algorithm design or analysis approaches, or highlight novel or unseen effects of parameter tuning as they relate to speedup and efficiency of compliance graph generation.

References

[1] X. Ou, W. F. Boyer, and M. A. Mcqueen, "A Scalable Approach to Attack Graph Generation," CCS '06: Proceedings of the 13th ACM conference on Computer and communications security, pp. 336–345, 2006.

[2] J. Zhang, S. Khoram, and J. Li, "Boosting the performance of FPGA-based graph processor using hybrid memory cube: A case for breadth first search," FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 207–216, 2017

[3] S. Arifuzzaman and M. Khan, "Fast parallel conversion of edge list to adjacency list for large-scale graphs," in HPC '15: Proceedings of the Symposium on High Performance Computing, pp. 17–24, Apr. 2015.

[4] X. Yu, W. Chen, J. Miao, J. Chen, H. Mao, Q. Luo, and L. Gu, "The Construction of Large Graph Data Structures in a Scalable Distributed Message System," in HPCCT 2018: Proceedings of the 2018 2nd High Performance Computing and Cluster Technologies Conference, pp. 6–10, June 2018.

[5] P. Liakos, K. Papakonstantinopoulou, and A. Delis, "Memory-Optimized Distributed Graph Processing through Novel Compression Techniques," in CIKM '16: Proceedings of the 25th ACM International Conference on Information and Knowledge Management, pp. 2317–2322, Oct. 2016.

[6] J. Balaji and R. Sunderraman, "Graph Topology Abstraction for Distributed Path Queries," in HPGP '16: Proceedings of the ACM Workshop on High Performance Graph Processing, pp. 27–34, May 2016.

[7] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," IEEE Transactions on Dependable and Secure Computing, vol. 13, no. 5, pp. 519–532, 2016.