SysGen: System State Corpus Generator

Ben Lenard
DePaul University
blenard@anl.gov

James Wagner
University of New Orleans
jay.wagner88@gmail.com

Alexander Rasin
DePaul University
arasin@cdm.depaul.edu

Jonathan Grier
Grier Forensics
jgrier@grierforensics.com

ABSTRACT

Security investigations often rely on forensic tools to deliver the necessary supporting evidence. It is therefore imperative that forensic tools are scientifically tested in both their accuracy and capabilities. The primary means to develop and validate forensic tools is by evaluating them against a set of known answers (i.e., a data corpus). While researchers have long recognized the need for standardized forensic corpora, there are few such tools or datasets available, particularly for database management systems (DBMS). In fact, there are currently no publicly available tools that can generate a DBMS dataset for forensic testing. In this paper, we share SysGen, a customizeable data generator and a pre-built corpus that offers a reference for most major relational DBMSes. The pre-built corpus includes individual DBMS files, the full disk snapshot, the RAM snapshot, and network packets taken from a set of clean virtual machines. SysGen can be easily adapted to execute a custom workload scenario, capturing a new data corpus; it can also create other variations of full system snapshots, even beyond DBMS testing.

KEYWORDS

Dataset generator, forensic benchmark, forensic tool testing

ACM Reference Format:
Ben Lenard, James Wagner, Alexander Rasin, and Jonathan Grier. 2020. SysGen: System State Corpus Generator. In *The 15th International Conference on Availability, Reliability and Security (ARES 2020), August 25–28, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3407023.3409202

1 INTRODUCTION

Forensic and security software suites seek to prevent and detect cyberattacks by employing methods such as file carving, malware signature identification, and intrusion detection. These tools require extensive testing to assess the accuracy of their reports for mission-critical environments (e.g., a criminal trial), provide insight into deployment configurations, and evaluate their capabilities to detect and interpret cutting-edge threats. Current efforts to test these tools are disconnected and disorganized; testing is based on procedures defined by individual organizations (or researchers), and performed internally on the basis of in-house datasets.

Universal and standardized testing protocols are hindered by the lack of data corpora. A major requirement for such standardized data corpora is that the *entire system state* must be captured; copying individual files is often insufficient for comprehensive testing. Specifically, this paper considers database management systems (DBMS). A corpus consisting of DBMS data files is not complete because a DBMS is a complex system consisting of many data files, system metadata describing the data files, transaction logs, audit logs, and backup files. Furthermore, DBMS system state elements also reside in RAM (e.g., modified data not yet written to disk), on the disk image outside of DBMS storage (e.g., deleted DBMS storage released back to the operating system), and in network traffic data.

In this paper, we created and published SysGen, a data corpus generator that easily adapts to incorporate a wide variety of datasets and simulates user-specified workloads. The initial version of SysGen focuses on a representative data corpus for DBMS forensic and security analysis, but the overall SysGen framework can easily be customized to create other datasets. For a fully controlled environment where tools are instrumented easily, SysGen uses VMware virtual machines for each data generation sequence. VMware instances allowed us to accurately capture the entire system state (i.e., RAM, disk image, and network traffic). Moreover, blank, dedicated VMware instances ensure that the captured snapshots are not contaminated by external data (our data loading process is tailored to exclude input data files from the captured VM).

SysGen includes a representative dataset and models a workload scenario for several major and widely used DBMSes. We created and executed the scenario on the basis of a recognized benchmark from the database community – Star Schema Benchmark or SSBM [10]. The data tables were further extended to provide a better coverage of different data types (e.g., `TIMESTAMP`, `VARCHAR(4000)`). Figure 1 summarizes the process. SysGen loads data into a DBMS using a remote client VM, to ensure that the input file does not contaminate the snapshots. Our query workload was modeled based on what [6] found to be representative of real-world DBMS workloads. We created random query predicates and composed the workload from read (`SELECT`) and write (`INSERT`, `DELETE`, `UPDATE`) SQL queries based on the distribution in [6]. Although testing should use both real and synthetic data, the advantage of synthetic data is that each forensic artifact was generated by our workload. In fact, real data has its own challenges because it may be difficult to know precisely what the dataset contains (i.e., the ground truth is unknown).

The main contributions of this paper are:

ARES 2020, August 25–28, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-8833-7/20/08…$15.00
https://doi.org/10.1145/3407023.3409202

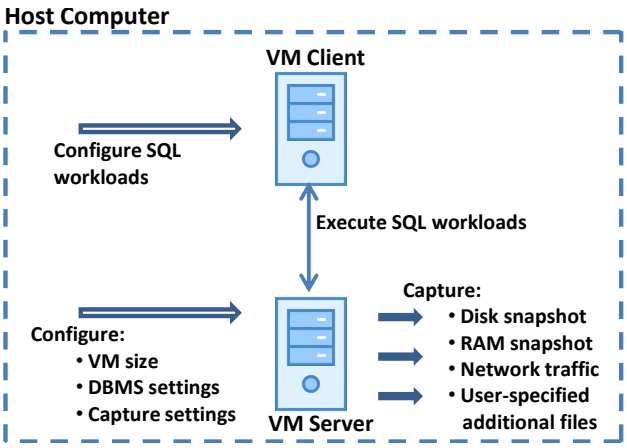

Figure 1: SysGen architecture.

- A data corpus generator, SysGen, that initializes a VM server with a pre-installed DBMS and 1) cleanly (i.e., from a second independent VM) executes a series of database queries, and 2) captures the entire resulting state of the virtual machine (RAM, network traffic, disk snapshot, and custom individual files) before and after each query sequence.
- A corresponding dataset based on a well-known database benchmark that was extended to include additional data types for better coverage (e.g., Character Large Object or CLOB) and a real-world representative SQL query workload.
- A resulting data corpus for several DBMSes (MySQL, Oracle, SQLite, DB2, and PostgreSQL) generated using our workload.

We share the data corpus as well as the scripts, data, VM setup files, SQL statements, and all generated artifacts for the community to utilize. These artifacts are available at http://dbgroup.cdm.depaul.edu/SysGen/. The data corpora will be maintained and extended as future work for this project. In fact, we envision other uses for SysGen, including non-DBMS corpus generation and timeline database corpus (i.e., series of snapshots representing the evolution of a single VM through a series of workloads). Currently, all artifacts are generated on a Linux platform; however, we will expand SysGen to support Windows and Macintosh in our subsequent releases (see Section 6).

The rest of the paper is organized as follows: Section 2 provides work related to SysGen and data corpora; Section 3 presents the SysGen and the data corpus requirements; Section 4 describes the SysGen generation process and output; Section 5 presents detailed steps for automating and scaling to multi-machine and multi-platform deployment. Finally, Section 6 describes future directions for SysGen and our data corpora maintenance goals.

2 RELATED WORK

Data corpus properties. Data corpora and generators research focused specifically on the desired properties, modalities, expectations, and comparative advantages of different data generation approaches. Garfinkel [3] lists seven requirements to define the quality of a digital corpus. **Representative** of the real-world data to be encountered. **Complex** with information from many sources. A wide variety of data should be included. **Heterogeneous** using a range of computer systems and usage patterns. **Annotated** allowing for new tools to be validated against existing ones. **Distributed** in open file formats. Tools should be able to easily access the data. **Available** to researchers in an unclassified environment. **Maintained** to reflect constantly changing and evolving computer systems and cybersecurity threats.

Garfinkel et al. [5] defined the major data corpus modalities: 1) disk images, 2) memory images, 3) network traffic, and 4) files. Additionally, time sequence multi-modality data are important to observe for the given subject during its operations. In other words, it is useful for researchers and tools to have the stages of the subject's operational lifespan. Garfinkel et al. [5] emphasized the need for digital forensic corpus so that the scientific process can be used to evaluate different tools and algorithms against a known reference set so for reproduceable results. Yannikos [19] also cites the need for varied corpora that are multi-modal since the current corpora are domain and area specific such as files, network or memory only.

One of the key characteristics highlighted in [3] and [4] is the maintenance of the digital corpus since digital forensic tools need to be continuously evaluated due to the rapidly evolving technology. Similarly, the reference set corpus needs to be updated to support the maintenance of the tool or techniques. SysGen ensures that our process can be used to iteratively rebuild the dataset.

In [4], Garfinkel cited that over the years they encountered various policy issues when working with sensitive data, ranging from privacy issues to copyright issues to illegal content. Since SysGen code is fully available by design, we will avoid such issues. Our generated data corpus is partially available for download (for open-source DBMSes that have no licensing constraints) and the rest can be recreated using SysGen code and downloadable software (for closed-source DBMSes such as Oracle).

Barse et al. [2] observed that data generation methodology should be representative of the desired behavior observed in a real-world scenario. [2] used small amounts of authentic data to seed the generation of large amounts of data for testing. Similarly, we used a sample of the SSBM queries to generate a large workload to be executed against the database. Barse et al. used synthetic data instead of real data due to benefits such as repeatably and control. Hsu et al. [6] analyzed industry-specific workloads to obtain metrics for their usage patterns. We followed the findings in [6] to create a representative generic workload observed across different industries. The average logical read ratio across all industries was 93.9%, with only 6.1% of the workload was performing `INSERT`, `DELETE`, and `UPDATE`.

Data corpora. Database community used TPC-H and SSBM [1] for a wide variety of DBMS performance research projects and papers. SSBM [10] is a star schema database that has addressed certain issues within the TPC-H schema; the SSBM schema was built specifically to represent a typical data warehousing database.

SQLite is a popular DBMS for embedded systems and in recent years it has gained popularity for internal use within applications. Nemetz et al. [9] developed a forensic corpus for SQLite that encompasses seventy-seven different SQLite database instances. The researchers followed [3, 5] principles to create the SQLite corpus in terms of characteristics and data sensitivity. The SQLite forensic corpus only focused on the SQLite DBMS, and only captured the SQLite files. Alternatively, we also captured the RAM, entire disk, and network packets, providing a multi-modal corpus.

While SQLite is a common DBMS for many applications, there are many other widely used DBMS vendors in the field (e.g., Oracle or PostgreSQL) that are also subject to forensic investigation. For example, Wagner et al. [16] studied forensic investigation of data in eight different DBMS platforms (including SQLite).

Besides [9], we are not aware of any other DBMS corpora. Our digital corpus falls into the category of a multi-modal corpus since we preserve the memory, harddisk and network contents over time. In the following section, we will discuss the corpus in relation to the seven characteristics outlined by Garfinkel as well as the sensitivity of the data in the generated corpus.

3 REQUIREMENTS

This section discusses the goal requirements for SysGen and our data corpus, based on requirements in [3]:

Representative. Our workload and dataset was designed to represent a typical data warehouse environment. The workload was generated based on queries from the SSBM benchmark; we generated an expanded workload consistent with values from the SSBM dataset. As discussed earlier, we used a read, or `SELECT`, ratio of 93.9% in the workload generation based on observations in [6]. The dataset was based on SSBM Scale Factor 4 which contains over two million records (or ~2.5GB). SysGen can be easily configured to execute any user-specified workload to create additional corpora representative of other types of settings (e.g., a high-frequency transaction or a single-user environments).

Complex. We augmented the complexity of the SSBM benchmark to include data types not represented in the existing schema. The SSBM schema primarily used `VARCHAR` and `INTEGER`; we modified it to include `CHAR`, `VARCHAR(4000)`, `CLOB`/`LONGTEXT`, `FLOAT`, `DOUBLE`, `TIME`, `DATETIME`, and `TIMESTAMP`. `Customer_Extended` is a new table created to represent these data types. Each column of the new table (with the exception of `c_custkey` which is an integer) represents a different data type not present in the original SSBM data. The DBMS-specific types chosen to represent column data type for each DBMS are summarized in Table 1. For the first corpus release, we excluded other exotic data types such as `BLOB`s and embedded code such as PL/SQL or SQL/PL. While the loaded data is synthetic, the comprehensive sample of available DBMS data types and sizes provides a wider coverage of DBMS forensic artifacts.

Heterogeneous. In addition to introducing complex data types and generating a query workload representative of a data warehouse, we used different DBMS platforms to make the corpus heterogeneous. SysGen automates evaluation of our benchmark against the following DBMSes: MySQL / MariaDB 5.5, IBM DB2 (Express-C) 11.1.4.4, Oracle 19c, PostgreSQL 11, and SQLite3. For this release, all client and server VMs used CentOS 7.6 x86_64 since it is a common Linux distro within the enterprise. We considered data collection on IBM Power in the first release, but we believe the results would be similar other than the endianness change. Section 5 discusses our plan to introduce IBM Power support in subsequent releases.

Annotated. The input data and control scripts included with SysGen were named based on the function of the script and the platform of the pertinent DBMS (please see Section 4.2). Similarly, the output of SysGen capture and our pre-generated corpus are annotated based the underlying script phase and DBMS platform (please see Sections 4.3 and 4.4). We included all SQL statements, data, and Bash scripts used to build the corpus as well as inline documentation on how to configure these Bash scripts.

Available. SysGen code and the pre-built corpus are available at http://dbgroup.cdm.depaul.edu/SysGen/. We utilized free or trial versions of the DBMS software and CentOS. For SQLite, PostgreSQL, and MySQL, we included the entire collection of output files generated by SysGen. For Oracle and DB2, we included detailed instructions to install and configure these (freely downloadable) DBMSes on a clean VM and reproduce our results. Most importantly, everything in our data corpus can be easily recreated by running included scripts. Naturally, that may result in some small variations in the collected snapshots because data loading and query caching in RAM are not deterministic processes.

Distributed. Since this is a multi-modal corpus, there are several different files for each DBMS platform. We include the RAM snapshots (.vmss), harddisk snapshots (.vmdk) in their raw form, network packet capture in standard PCAP format, and additional DBMS or SQL files in their respective native formats.

Maintained. The corpus is subject to versioning since we plan on maintaining the corpus releases as the new software versions are released. We also plan to expand data types (e.g., embedded code such as PL/SQL), incorporate new workloads based on additional benchmarks, and other DBMSes and operating systems. Our maintenance plan is briefly described in Section 6.

4 DESIGN

4.1 Virtual Machine Setup

Each VM server (see Figure 1) consisted of 8GB of RAM, 4 vCPUs, 1 vNIC, and a 25GB VMDK file; the VMDK file was partitioned into a 350 MB boot partition, a 2GB swap partition, with the remaining space allocated to the root partition. Standard partitioning was used, not LVM, to simplify mounting the partitions out of the VMDK file for subsequent file extraction. The boot and root partitions were formatted using EXT4. Since the Oracle install required approximately 6GB, the Oracle VM had a 35GB VMDK file instead of 25GB. All VM servers had consistent kernel settings; FirewallD and SELinux were disabled to avoid any complications. Each VM began with the same default software packages; additional prerequisite packages for each particular DBMS were installed for at the time of DBMS installation. Where applicable, we enabled archivelogging, or the write ahead log (WAL), in line with best practices (e.g., [11]).

Each VM client (see Figure 1) houses all of the DBMS client libraries, SQL scripts, raw table data (i.e., CSV files), and DDL commands to run against the VM server. Similar to the VM server, the client runs CentOS 7.6 x86_64 with 2GB of RAM, 2 vCPUs, and a similar hard disk configuration. The host machine (see Figure 1) had 32GB of RAM – enough RAM for both client and server VMs and to prevent any type of swapping condition on the host. The host machine runs the VMware hypervisor software, which executed our VM control scripts (see Section 4.2), and captured network traffic. N-Tier architecture and security often place the database server on a host server so that if the web application were compromised the DBMS data has a less of a chance of being compromised as well. Thus it is very likely that a typical production DBMS will also be deployed on top of VMware hypervisor software.

Column Name	MySQL	DB2	Oracle	PostgreSQL	SQLite
C_Custkey	Integer				
C_Notes	VARCHAR(4000)				
C_Notes2	LongText	CLOB		Text	
C_Discount	Float		Float(126)	Float	
C_Discount2	Double		Float(126)	Double Precision	
C_Time	Time		VARCHAR(12)	Time	
C_Datetime	Datetime	Timestamp			
C_TS	Timestamp				

Table 1: Customer_Extended table summarizing the approximate equivalent of each data type across different DBMSes.

4.2 SysGen inputs

Variations in data types across DBMS platforms presented its own difficulties when loading data (even for simple data types). Therefore, we provided a customized DDL of the SSBM schema, including our extended table with the data types outlined in Table 1. We also included the database creation scripts and, in Oracle's case, the response file where we create the tablespaces prior to loading the SSBM schema. For DBMSes such as DB2, Oracle, and PostgreSQL, we created a tablespace specifically for data and indexes; this is a common practice (see [7, 12]).

Input data. The raw CSV data was loaded into each DBMS using the native remote clients with the the exception of SQLite. Since SQLite does not have a remote client we wrote our own and loaded data via `INSERT` statements. On each VM server we deployed (and included in our distribution) the following:

- **ifcfg-ens33** – Static IP address.
- **.ssh** – folder with ssh keys.
- **install_dbms** – a script that installs each DBMS platform. For DB2, the script creates a DB2 user; MySQL, Oracle, PostgreSQL automatically add a user with the rpm installation.
- **cache_settings** – a folder with recommended DBMS cache settings for each platform. In general, MySQL and PostgreSQL require more manual configuration than Oracle and DB2 which automatically manage memory allocation.
- **Start[DBMS NAME].bsh** – starts the DBMS. For SQLite, it starts our custom python server application. For Oracle, it sets required environment variables. (e.g., ORACLE_SID).
- **Reboot.bsh** – reboot the VM from the host machine.

On the VM client we deployed (and included in our distribution) a folder **[DBMS]-client-setup** containing:

- **SSBM_schema.sql** – DBMS-specific DDL file since each DBMS has slight differences in their SQL semantics.
- **DropCreateDB.bsh** and **DropCreateDB.sql** – written per DBMS; for example, in MySQL one just has to issue "Drop database bench; create database bench," but in DB2 the script contains 20+ lines for creating the database (e.g., DB2 requires creating a backup after you create a database).
- **raw_data** – folder with SSBM and `Customer_Extended` data in CSV format. `INSERT` statements were used for SQLite.
- **control_files** – only Oracle requires CTL files for SQLLDR.
- **ssbmsql** – a folder containing pairs of files, e.g., **ssbmsql0.bsh** and **ssbmsql0.sql**. The first file is a wrapper bash script to execute the SQL workload with the DBMS-specific client. The second file is the SQL query workload. Although our current release executes and captures one workload, `SysGen` already automatically executes multiple files in this folder to capture a time series of snapshots.

For each DBMS we provide three host scripts, along with the SSH keys to the host computer. The host computer executed the following scripts via SSH remote commands:

- **runTests-[DBMS]-clean.sh** – initializes the DBMS.
- **runTests-[DBMS]-load.sh** – a script that runs a sequence of steps for a DBMS load process.
- **runTests-[DBMS]-run.sh** – a script that runs a sequence of steps to execute the workload against a DBMS. A set of individual files and folders to be extracted from the root partition of the harddisk can be specified in this script.

These scripts were also responsible for suspending and unsuspending the VM, and copying the files while the VM is suspended. See Sections 4.3 and 4.4 for a detailed description.

SSBM Workload Generator. In order to have a workload generated to our specifications, we expanded on the [10] query set. We include a python script to gather the unique values for SSBM data. Our workload generator reads these CSV files and loads statistics about unique values to generate the corresponding SQL queries. Our python script is configured with the number of iterations, or runs, to produce, as well as the number of select, insert, update, and delete statements. The SSBM query set is limited to `SELECT` statements so we created other types of operations from scratch. Insert generating function increments last c_custkey by one, and generates a random date inline with the other fields. For the update statements, we update either `Customer` table or our new `Customer_extended` table, with a random selection of 5 different update statements. For deletes, we pick a random c_custkey value and delete customer's records from `Lineorder`, `Customer`, and `Customer_extended` tables.

4.3 Data corpus generation

`SysGen` has three components: clean, load, and run phases.

Clean phase. During the clean phase, **runTests-[DBMS]-clean.sh** builds and automatically deploys client and server VMs with our configuration settings. These settings include making the VMs network interface a static IP, turning off FirewallD and SELinux, adding our ssh keys for remote access, installing the DBMS software, and creating the DBMS instance where applicable. This script also stores snapshots of the clean phase to verify that disk and RAM contents where clean prior to the load phase, and as a checkpoint in case we need to rerun our data generation.

Load phase. Data loading is controlled by the host computer via **runTests-[DBMS]-load.sh**. The only exception to this is starting

the network sniffer, vmnet-sniffer, for packet capture; all network traffic is captured on the virtual network between the client, server, and host during the load phase. Similar to other network sniffing tools, vmnet-sniffer utilizes the standard PCAP format so that common network analysis tools can read these files. To minimize complexity, we did not enable SSL traffic between the client and VM so that network packets did not require decryption. After the network sniffer was started, our script powers on the VM server and waits 30 seconds before rebooting the VM. Rebooting the VM ensures that its memory contents are clean. Once the VM is rebooted, the automation script starts the DBMS, where applicable, then drops and re-creates the database for loading. In Oracle, the script drops and re-creates the tablespaces and users to re-create the database. Our script loads the data using the corresponding DBMS client tool: SQLLDR for Oracle, Import for DB2, Infile for Mysql, Copy for Postgress, and a custom process for SQLite.

Since SQLite does not offer a remote client, or library, for SQL execution over TCP/IP, we built a simple one allowing us to stream data from the client VM to the server VM and receive responses. We use our client to execute `INSERT` statements during the load phase and query workload during the run phase. This prevents the VM from being contaminated with the raw data or queries.

Once the load process is completed, our script suspends the VM; suspending the VM forces its RAM into a file on disk and forces the current filesystem state to disk. Once the VM is suspended and the contents are written to disk, our script copies the contents of the VM directory to another directory with the date and phase appended to the directory name (see Section 4.4 for details). At this point, we manually stopped the packet capture since we know the VM is suspended.

Run phase. In run phase, **runTests-[DBMS]-run.sh** executes a workload of 1,000 SQL statements against the DBMS; of the 1,000 SQL statements there are 939 `SELECT` statements, 20 `UPDATE` statements, 20 `INSERT` statements, and 21 `DELETE` statements. A 93.9% ratio was used based on the average of workloads across various industries; we will develop other representative workloads in subsequent releases (running and capturing other workloads is trivial using SysGen). Similar to the load phase, we manually started the network sniffer on the host machine. Following that we run the automation script from the host machine; the script resumes the VM server, and runs the SQL statements from the client machine. Following the completion of the SQL statements, our script suspend the server VM and captures the contents of its memory and disk (see Section 4.4).

4.4 SysGen output

The following lists the files copied when the VM is suspended [15]:

- **Virtual Disk.vmdk** – virtual hard disk.
- **[VM NAME].vmx** – VMware configuration for the given VM; a .lck file appears when the VM is powered on.
- **[VM NAME].vmss** – a relatively small file that contains information about the suspended state.
- **[VM NAME]-<UUID>.vmss** – This contains the RAM contents of the VM at the time of suspension.

For each DBMS, SysGen currently creates two folders, **output-[DBMS]-Time0-[TimeStamp]** and **output-[DBMS]-Time1-[TimeStamp]**. Each folder contains the full state of the server VM, before and after the workload execution, respectively. The folder also contains the files specified in **runTests-[DBMS]-run.sh** script. Additional SQL workload files create subsequent **Time** snapshots.

The copy of the disk file represents the image of the entire hard drive. For non-freely-available DBMSes, our data corpus includes DBMS data files instead of the complete disk and RAM snapshots (although the entire snapshot can be easily recreated using SysGen). To extract files from a VMDK file, we use a VMDK mounting utility. We created VM disks using standard partitioning and not LVM to aid in this file extraction process. Our script then copies any individual files or folders configured by the user. We use this process to extract Oracle and DB2 data files and distribute it with our data corpus (as full snapshots of disk or RAM with Oracle or DB2 cannot be freely distributed due to their inclusion of copyrighted code). We expect that all database files will be converted into DB3F format designed to store database forensic artifacts while remaining carver-agnostic. Both the format description and a corresponding viewer tool are freely available for download [17].

5 RUNNING SYSGEN AT SCALE

In our initial SysGen release, we performed the corpus generation on a desktop computer with 32GB of RAM. Thus, the current release does not fully achieve our vision of scaling to a big data environment and generating a corpus that encompasses multiple nodes and multi-architecture VMs. This section describes our detailed planned steps to introduce further automation and scalability features into the future releases of SysGen. Our envisioned approach will allow SysGen to scale beyond one desktop to multiple hyper-visors with testing similar to a continuous integration (CI) pipeline.

In a data center environment, automation is key for building a physical server, or a VM, and automation of provisioning is key to scalabilty as it then requires less manual interaction. While we used VMware Desktop for this SysGen version, we would eventually switch to a VMware Server on the host machine so that we can scale beyond a single desktop. VMware Server offers scability features that cannot be easily obtained with the desktop version; for example, it provides an API to communicate and control N hyper-visors and a shared file system without the performance issues associated with using NFS. VMware currently only supports x86 architecture virtual machines. Kernel Virtual Machine (KVM) offers similar functionality and supports POWER architecture as PowerKVM; therefore we are looking at supporting KVM in addition to VMware. Regardless of hyper-visor software, since an API or command line interface (CLI) is available to create scripts, we can provision new VMs for corpus generation in an automated fashion. In our current SysGen version, the initial VM is created manually, since there was no easy way to create a new VM from the command line using VMware Desktop.

For the OS building process, SysGen can use either a VM template or, alternatively, PXE booting a VM through a kickstart file. The advantage to using VM templates (as we currently do) is that you do not incur the cost of an OS build; the disadvantage is that the template has to be rebuilt with an OS update or a package update. Thus, while template is a simpler, it is associated with long-term maintenance costs. Future versions of SysGen will allow for a consistent VM creation to be automated. Once a VM built, the database

management software (or other software needed) has to be installed on the VM and necessary kernel parameter modifications have to be applied. We intend to host a configuration management service such as Salt [14] or Ansible [13]. These two projects allow for controlled and automated configuration setting and deployment of the DBMS software itself. The templating process does not support data corpus evolution where the OS build is frequently updated, such as beta releases. Therefore we would prefer to transition to automated rebuilding of the OS every time changes are made. We envision that DBMS configuration, desired benchmark workloads, and configurations will be executed by this process based on the definitions stored in a repo.

In order to generate database query workloads, we would incorporate Jenkins [8] into our future SysGen releases. The Jenkins project has been developed to support automation, CI, and continuous deployment (CD) which is similar to what we are doing. The user of SysGen would kick off a Jenkins job to execute a given workload (i.e., a data corpus scenario) against a given database type. Jenkins would provision the virtual machine(s) in an automated fashion, then initialize and build the current OS. After the OS build, software installations would be executed by tools such as Ansible or Salt. Once the VM is at a ready state, the query workload would be to executed from the client VM by Jenkins. Within Jenkins itself, one job would be executed by the user but in-turn multiple jobs would run executing various tasks. Jenkins would automate all of the tasks, including starting and suspending the VM, starting the network sniffer, and copying all relevant artifacts as configured for each step of the benchmark scenario. We will copy the artifacts to another file system and identify collected data with the a unique identifier capturing Jenkins logs for that run.

6 CONCLUSION

This paper lays the groundwork for a much larger project; we plan to advance this effort following Garfinkel's corpus requirements (Sections 2 and 3). Specifically, we will focus on maintenance, large-scale corpus generation, data complexity, additional representative workloads, and heterogeneity.

For the pre-built corpus in this paper, we will update VM images (or data files for non-free DBMSes) as new major DBMS versions are released. Major DBMS versions are released every several years for both Linux and Windows. We believe the frequency of DBMS releases makes this feasible. For example, Oracle 12c was released in 2013, followed by Oracle 18c in 2018 and Oracle 19c in 2019. PostgreSQL releases a new major version no more than once a year. While DBMSes release service packs several times a year, these product updates mostly contain bug fixes to existing functionality; significant new features are deployed in major releases. Using automated software deployment tools (discussed in Section 5) will serve to automate minor DBMS updates and changes to the OS or VM configuration.

While our corpus includes most DBMS data types, we intend to expand SysGen by adding exotic data types such as PL/SQL, SQL/PL, XML, Small Int, Big Int. Adding to complexity of the corpus (i.e., data types and sizes) will be best achieved by designing new representative workloads and incorporating more DBMS platforms (SQL Server is the first on our list).

In this paper, we provided SysGen a workload and dataset that represents a data warehouse. We plan to build more corpora using other real-world scenarios. For example, the data warehouse scenario in this paper is not representative of typical activity for SQLite. A typical SQLite workload and dataset would model a cell phone or a web browser. We built SysGen so that running new workloads and datasets is trivial – however, we have not yet found an authoritative source summarising SQLite type workloads.

This paper used SysGen for major relational DBMSes on Linux, but it was designed to simplify adding new systems. The corpus in this paper will be expanded to include same DBMSes evaluated on Windows and other DBMSes such as SQL Server. New corpora will include different types of systems, most notably different types of databases. These include column-store DBMSes (e.g., C-Store and MonetDB) and NoSQL DBMSes (e.g., MongoDB and Riak). We started with row-store relational DBMSes because there is 1) a database-agnostic approach to forensic analysis [18], which enables comparing cross-DBMS behavior and 2) many different forensic tools for at least one row-store DBMS (SQLite) [9], which allows for cross-tool comparison. To our knowledge, neither column-store nor NoSQL DBMSes have progressed to that stage yet.

ACKNOWLEDGMENTS

This work was partially funded by the US National Science Foundation Grant CNS-1656268 and by the Argonne National Laboratory.

REFERENCES

[1] M. Barata, J. Bernardino, and P. Furtado. An overview of decision support benchmarks: Tpc-ds, tpc-h and ssb. In *New Contributions in Information Systems and Technologies*, pages 619–628. Springer, 2015.
[2] E. L. Barse, H. Kvarnstrom, and E. Jonsson. Synthesizing test data for fraud detection systems. In *ACSAC*, pages 384–394. IEEE, 2003.
[3] S. Garfinkel. Forensic corpora: a challenge for forensic research. *Electronic Evidence Information Center, April*, pages 1–10, 2007.
[4] S. Garfinkel. Lessons learned writing digital forensics tools and managing a 30tb digital evidence corpus. *Digital Investigation*, 9:S80–S89, 2012.
[5] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6:S2–S11, 2009.
[6] W. W. Hsu, A. J. Smith, and H. C. Young. Characteristics of production database workloads and the tpc benchmarks. *IBM Systems Journal*, 40(3):781–802, 2001.
[7] IBM. Assignment of table spaces to physical storage. https://www.ibm.com/support/knowledgecenter/en/SSEPEK_10.0.0/intro/src/tpc/db2z_tablespacestophysicalstorage.html, 2019.
[8] K. Kawaguchi. Jenkins automation server. https://www.jenkins.io/, 2020.
[9] S. Nemetz, S. Schmitt, and F. Freiling. A standardized corpus for sqlite database forensics. *Digital Investigation*, 24:S121–S130, 2018.
[10] P. O.Neil et al. The star schema benchmark and augmented fact table indexing. In *Performance evaluation and benchmarking*, pages 237–252. Springer, 2009.
[11] Oracle. Maa best practices - oracle database. https://www.oracle.com/database/technologies/high-availability/oracle-database-maa-best-practices.html.
[12] Oracle. Database vldb and partitioning guide. https://docs.oracle.com/database/121/VLDBG/toc.htm, 2018.
[13] RedHat. Ansible automation framework. https://www.ansible.com/, 2020.
[14] SaltStack. Saltstack infrastructure automation software. https://www.saltstack.com/, 2020.
[15] VMware. What files make up a virtual machine? https://www.vmware.com/support/ws55/doc/ws_learning_files_in_a_vm.html, 2019.
[16] J. Wagner, A. Rasin, and J. Grier. Database forensic analysis through internal structure carving. *Digital Investigation*, 14:S106–S115, 2015.
[17] J. Wagner, A. Rasin, K. Heart, R. Jacob, and J. Grier. Db3f & df-toolkit: The database forensic file format and the database forensic toolkit. *Digital Investigation*, 29:S42–S50, 2019.
[18] J. Wagner, A. Rasin, T. Malik, K. Heart, H. Jehle, and J. Grier. Database forensic analysis with dbcarver. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research*, 2017.
[19] Y. Yannikos et al. Data corpora for digital forensics education and research. In *IFIP International Conference on Digital Forensics*, pages 309–325. Springer, 2014.