Smarter disk space management for Silicon workflows

Jigar K. Savla
Juniper Networks, Inc
Sunnyvale, USA
jks@ieee.org

***Abstract*—As projects approach their deadlines, multiple teams - design and verification - are running several simulations and regressions. Many times, the test logs and waveforms across several TestBenches lie around way past their usefulness. This causes project delays as space reduces to such an extent that no simulations or work can proceed or at a glacial pace. We've experienced the same across several projects and have come to a better way to manage and administrate these disks by using the ageing mechanism effectively by giving users and disk administrators relevant information and controls.**

Keywords—disk space management; data engineering; Verification; Design; Methodology

I. Introduction

Disk space per volume is valuable and limited. Derived files (build directories, tests, logs, waveforms, etc.) occupy a significant portion of this space; they also get snapshotted, which increases the disk usage. To add to this, users do not clean up their files and directories until the volume's disk space is nearly full.

In this paper, we'll go through existing solutions, their shortcomings and present our smarter way to manage and administrate these disks, tuned for silicon workflows.

II. Problem at hand

Anyone who has had to manage disk volumes during crunch time has experienced the pains of disk running out of space and then not able to proceed. It's a common issue.

A. Types of Disks and areas of debug

Figure 1 shows how you would generally have a separate nightly regression disk and a local disk for day to day usage. During regular active debug use, you'll end up using only part of the space under both disks, shown in blue.

Our task is now to isolate that active use area, while we retain only relevant info from the remaining areas.

B. Concept of 'derived' and 'non-derived' and Workarea

Workarea – Current sandbox or work environment.

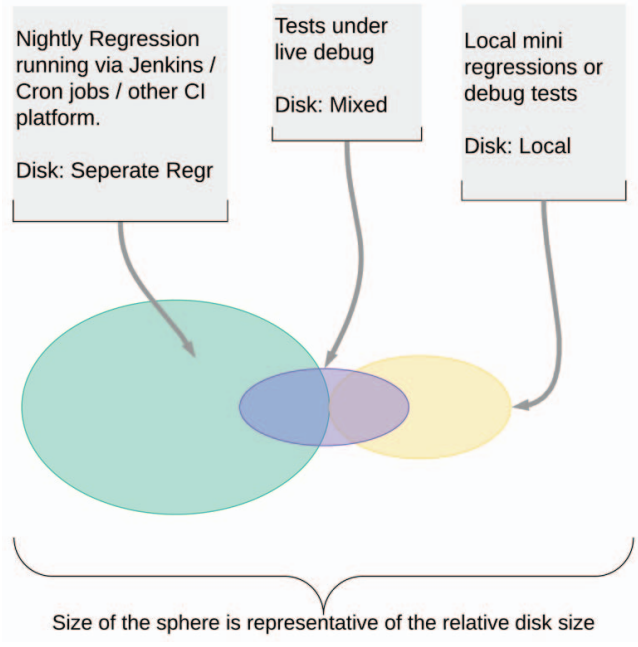

Figure 1: Disk types based on usage

Regarding file types, we start with categorizing between:

- "derived" : generated files like waveforms, test logs, build directories, emulation builds [1][3][4][8][9] which can be re-created and backup copies, like the ones highlighted in Figure 3.
- "non-derived" : like source code

C. How the disk gets filled up

Figure 2 shows how at the start of the project we have very little derived files sitting around, though by the end of the project our disk is almost choking with derived files

III. Existing solutions

There are several ways to solve this problem. We'll take you through the most common ones.

2332-5674/19/$31.00 ©2019 IEEE
DOI 10.1109/MTV48867.2019.00015

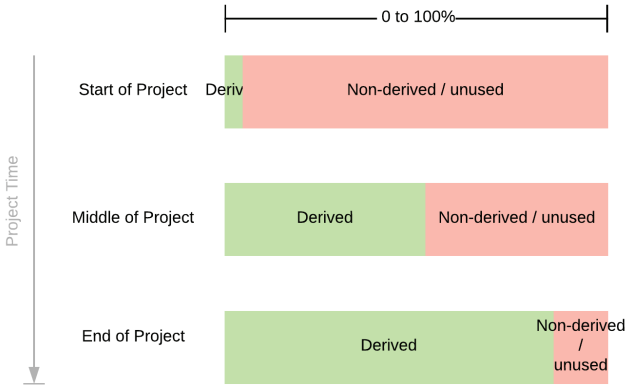

Figure 2. Share of disk space usage as the project evolves

```
../run/results/latest_run/ % ls -rtl
total 471008K
-rw-rw-r-- 1 jigar        238 Jul 26 21:52 cliCmds
-rwxrwxr-x 1 jigar       3915 Jul 26 21:52 runScript.csh
-rwxrwxrwx 1 jigar        358 Jul 26 21:52 dvewaves
-rw-r--r-- 1 jigar        953 Jul 26 21:52 runTestArgs
-rw-rw-r-- 1 jigar      90229 Jul 26 21:52 knobs.log
-rw-rw-r-- 1 jigar   52961736 Jul 26 22:10 verilog-1.trn
-rw-rw-r-- 1 jigar  425514752 Jul 26 22:10 verilog.dsn
-rw-rw-r-- 1 jigar    1771599 Jul 26 22:10 test.log.gz
-rw-rw-r-- 1 jigar         28 Jul 26 22:10 diskUsageBeforeClean
-rw-rw-r-- 1 jigar      12192 Jul 26 22:10 sig.log
-rw-rw-r-- 1 jigar         15 Jul 26 22:10 passOrFail
```

Figure 3. A typical run directory with large file size highlighted

A. *Continuous Integration, Continuous Delivery (CICD)*

- CICD is a platform for Automated Regression infrastructure to handle running and cleanup of tests, like Jenkins [11]
- When tests pass, just store the arguments to rerun the test if needed and summary files.
- Remove any passing test's logs and waveforms. They can be regenerated as needed. Passing testcases, in our experience, have rarely been regenerated.
- All tests run in CICD would have the default overriding flag to the equivalence of "-nowaveform", so that waveforms are not automatically generated.
- Single click options to be provided to end user to remove a regression area right from their email or a web dashboard.
- Keep only the last N number of regression / builds. Remove all older ones, unless user has specifically asked it to be kept alive. Even then, keep a reducing day count where the user has to refresh their keepalive request at regular intervals.

B. *Backup snapshot of the disks*

- Unix offers native support for creating snapshots of disks for backup and redundancy.
- Turn on for disks which are under active debug, for rest, turn off snapshot

C. *Use compression for the log files*

- *GZip* or *tarball* any text files that you can.
- *Emacs, VIM, less, more,* all support reading gzipped files.
- If you wish to *grep* through this file, just use *zgrep v/s grep* and *zegrep v/s egrep*.
- This can help you save lot of space, as silicon workflows are still very text heavy.

D. *Using the native Disk quotas in inodes of Unix filesystem.*

- Helps in allocating disk storage per user to identify any outliers in disk usage.

E. *Running 'du' once the disk reaches capacity is also widely used.*

- *Du* is a linux utility for figuring out disk usage
- Figure 4 shows what a typical email message to users would like, notice how it lacks any clear actionable information.

F. *Use the tool's capabilities*

- If the tool has knobs to turn off storage of transient files / compressing output files. Use it.

```
From: automated_email_server
Sent: today
To: high_consumption_users@coolcompany.com
Subject: Disk Usage Report

/vol3 is 88% Full
8.2T / 7.8T Used, 400G Available
-----------------
Size    Directory
467G    /vol3/users/user0
362G    /vol3/users/user1
355G    /vol3/users/user2
...
171G    /vol3/users/userN
```

Figure 4: Typical excess disk usage email

G. *Root problem with all of these solutions*

1) How does the user know which directory is the biggest one, how old is it?

2) When was it last accessed?

3) Command to quickly clean it up?

IV. Our Approach

A. *Idea*

Idea is to separate out derived files from the source code files. Do it in an automated fashion with the least disruption to the users. Then give insight to the managers and admins to ensure automated disk usage controls. Figure 5 shows the old workflow's weaknesses:

1) ineffective email notifications

2) build and test directories being created in the same place as the local work volumes.

Figure 6 shows the new workflow which leads to build and test directories being created in a separate disk than using the local work volumes.

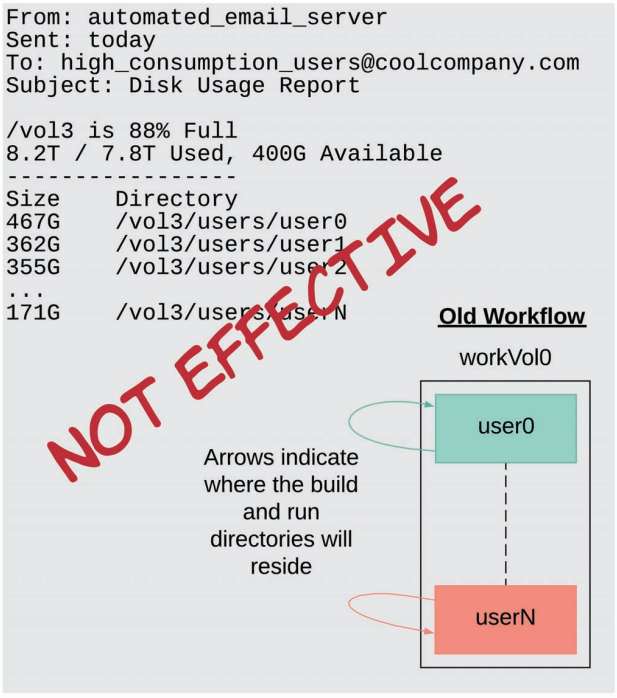

Figure 5: Old workflow

B. Why is this approach more effective?

- Disk space is filling up irrespective of the space thrown at users[2].
- We have seen that builds, test results i.e. derivative files occupy most of the disk space.
- This results in derivative files also being snapshott'ed. Leading to even more space wastage.
- With this, we are trying to find a way where we can automate build and test directory locations to a central one. Later, we can fine tune our algorithm to automatically clean up build folders once they cross certain age, help users easily manage their own disk usage.
- It also puts less stress on *'du'* since it doesn't have to traverse through the whole tree to figure out where the major disk consumer is.
- Good time to start: when new projects are about to begin.

C. How our solution addresses the root problem

1) We start with analyzing the current breakup of the files.

*2) Our data found the derived files started taking **upto 70% consumption** of disk space.*

3) Then we went about figuring out ways to handle this.

4) Split up the derived and non-derived files

5) Turn off backup for disk volumes with derived files or reduce the frequency / amount drastically

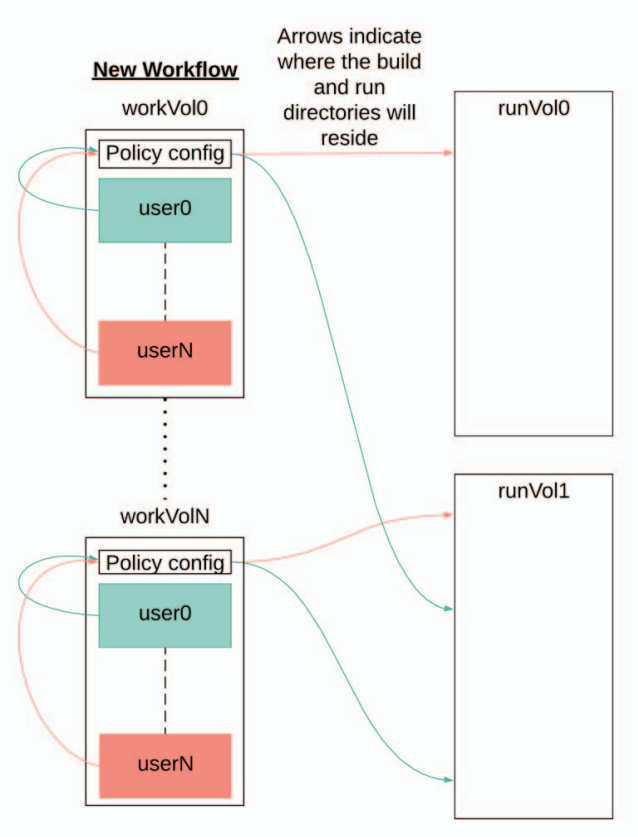

Figure 6: New workflow

6) Symlink: We've figured out ways to symlink out the biggest chunk of our derived files to a separate scratch area, where we run further statistical analysis to know which ones to keep and which ones to remove.

7) We chose the 'aging' approach: We once again found through gut intuition and backed up by statistical analysis that anything **more than two months old is 97.4% guaranteed to not be used again**.

*8) Guidelines:*Turn off waveform generations in regression tests. Add it as a guideline to your Design Verification (DV) engineers and then provide options to your regression which can override any "-waveform" with a "-nowaveform". Or re-run only failing tests to produce waveforms.

V. Overall Workflow

A. Figure 7 explains how a user would start the process:

1) User0 launches regression of tests

2) The wrapper polls the policy, causing in turn a chain of actions. 'policy' is described further in section VI.A

3) The launch volume is being read to figure out where to set the target volume for the derived files, for this user.

4) Does the symlink-ing setup from source to target volumes.

5) Central server makes sure policy files are kept up-to-date.

6) The derived disks's usage is monitored by the central server.

7) All of this information is stored in the Central Database whose structure is explained in detail in section VI.B.

8) Emails are sent out to users based on pre-configured settings.

VI. Code Structure

A. Policy based approach – for easier administration

'Policy' files are config files per disk volume to indicate where a user's build directories should get routed to, the target volume. It is not necessary to have all the redirection from a single volume to go to another volume, you can have potentially multiple target volumes shared across multiple source volumes. To begin with, each user per volume will only get one redirection volume.

Process till now?

*1) We have created a policy directory. If it's not present, it'll default to the status quo, which is to create the directory in the <block>/run/<..>/**

2) If policy is present, then it will create a symlink to the target directory, so that all tools work as expected.

3) Test it in a couple of different environements and after revisions of cleanups, check in to see if things work normally before making further changes.

B. Database design

The script running off of the policy database checks all users' run directories for aging and notifies via email. Operating on a per-volume basis.

1) Policy databases are currently on a per-volume basis; they are stored in 'policy.db' file.

2) The policy database is made up of 3 relational tables: Owner, RunDir, and RunOwner table: each row is a user on policy with 2 columns:

a) name = username; primary key meaning it is table's go-to reference and unique in table

b) email_freq = compound time string representing interval of time between policy notices; format #m#w#d

c) del_time = compound time string representing interval of time before auto-deletion

3) RunDir table: each row is a top run directory under policy with 10 columns

a) policy_dir = symlink path from policy directory; primary key meaning it is table's go-to reference and unique in table

b) run_dir = path from workarea/run directory

c) real_dir = resolved path of run directory

d) run_type = type of run directory (build or results)

e) workarea = workarea linked to run directory

f) total_size = total size of run directory and its contents

g) total_ctime = latest ctime (File change time) among run directory and its contents

h) email_freq = list of compound time strings for when to notify owner about untouched run directory

i) email_index = current index of email_freq

j) last_email = timestamp of last email on which RunDir was

k) del_time = compound time string representing interval of time before auto-deletion

l) owner = Owner referenced name of who owns run directory

4) Run table: each row is a run file/directory under a top run directory with 6 columns

a) policy_path = symlink path from policy directory

b) run_path = path from workarea/run directory

c) real_path = resolved path of file/directory

d) size = size of file/directory

e) ctime = ctime of file/directory

f) run_dir = RunDir referenced path; based on run_path

This relational design allows for easy querying and enables extensibility. For sending email notices, RunDirs are grouped by Owner, making it easy to compile RunDirs per Owner. Runs are grouped by RunDir, which makes it easy to update RunDir's contents and calculate total size and ctime.

C. Automated email notification procedure

It is designed to run once per day per disk volume, so Owners get notified on the correct day based on their policy email preferences. The current system in place for automatically scheduling daily runs uses Python's Event Scheduler (sched module); it uses a recursive calling method to schedule for the next day. This procedure initiates the top recursive call, and the rest is taken care of by the scheduler.

1) Daily routine

a) Iterate through every RunDir in DB; delete RunDir if path does not exist.

b) Walk through volume's policy links and add any new RunDirs and Owners.

c) Check all Owners' RunDirs for compiling and scheduling email notices.

- The procedure goes through each Owner, compiles a list of their aged RunDirs, compiles a list of their RunDirs without an existing workarea (sandbox deleted), and schedules email to be sent later that day with all this information. For RunDirs with special email preferences, a separate email is sent with information of all aged that day.

- An auto-delete feature is in place, need to delete override in RunDir.needsDel() and update delete notice email message.

d) Run email scheduler to send out notices.

e) Schedule next day's procedure

D. Open Source code to leverage

Using Go programming language's concurrency support [10] and Python for server side management.

E. Quick Summary

We provide the ability to redirect storage of derived files to a different volume while keeping the work area in the same volume; the sandbox structure and the work flow both remain the same. Just the location of derived files has changed.

Use this structure to automate cleaning of derived files; after not touching the files for a specified amount of time, notify the user of their largest sized run directories and provide remove commands to copy and paste into their terminal.

Simplifying this user experience lead to larger adoption.

VII. LEARNINGS ON THE WAY

A large portion of the time was spent in educating users who were used to a certain way of operations. Some had to rewrite their custom scripts. Some had to completely abandon their ways and move to the new usage models. All of these folks were convinced to change only because they saw the benefits first hand.

Educating and clear elaboration of benefits yielded the largest benefits.

It led to standardization of flows including introducing few new ones[5][6], moving to industry standards, which was a unknowingly good outcome.

VIII. FUTURE WORK

Disk space management for silicon workflows is a really large topic, we've only scratched the surface here with one technique which we found was scalable, sustainable and easy to administrate. We are always on the lookout to improve our workflows.

Next up for us, is to have different front-ends tapping into the common dB to perform different operations with varying levels of information provided.

Better UX design to control the information overload as the project scales up exponentially with SoC like environments.

Integrate with EDA vendor's regression test ranking and online dashboard to further prioritize which test results to have higher shelf life.

IX. CONCLUSIONS

This helps us better manager and efficiently utilize our existing resources. Any time lost due to disk space exhaustion means delayed time to market, which means lost revenue.

ACKNOWLEDGMENT

Sincere gratitude goes out to our team at Juniper Networks for supporting us in prototyping, testing and deploying these tools at scale in production workflows.

The author would like to thank his current and former colleagues : Calvin Leung, Sanjeev Kumar, Pradeep Joginipally for their valuable inputs and help in reviewing the manuscript. The author would like to specifically call out Joe Aronson for assistance in developing the wrapper, dB management and email notification codebase mentioned in section VI.B, VI.C.

This work was previously presented at DAC 2019 [7].

REFERENCES

[1] J. Savla, "Let's Get Started on Co-Emulation: Transition Your Design and UVM Testbench to an Emulator," SNUG Silicon Valley, 2019.

[2] J. Savla, "Disk Space Management Techniques for Silicon Engineers," SNUG Austin, 2018.

[3] J. Savla, "Getting Started on Co-Emulation: Primer on Why and How to Transition Your Design and UVM Testbench to an Emulator," SNUG Austin, 2018.

[4] J. Savla, "Why and How to Transition your Design and UVM Testbench to an Emulator: A Primer," SNUG Boston, 2018.

[5] J. Scott, J. Sadowsky, J. Savla: "RamGen: Moving Memories from Physical to Logical Domain," Presented at the IEEE/ACM Design Automation Conference 2019 (DAC 2019), Las Vegas, USA, July, 2019

[6] J. Scott, J. Sadowsky, J. Savla, "RamGen: Moving Memory from Physical to the logical domain," Microprocessor Test and Verification (MTV), 2019 20th International Workshop, December 2019, in press.

[7] J. Savla: "Get Higher Productivity and Smoother Tape-out by Smarter Disk Space Management," Presented at the IEEE/ACM Design Automation Conference 2019 (DAC 2019), Las Vegas, USA, July, 2019

[8] J. Savla, "Methodology for co-emulation: Transition your design and UVM Testbench to an Emulator," EDN.com

[9] J. Savla, "Getting started on Co-Emulation: Transition your Design and Testbench to an Emulator," Microprocessor Test and Verification (MTV), 2018 19th International Workshop, December 2018, pp. 46-51 doi: 10.1109/MTV.2018.00019

[10] https://github.com/dixonwille/skywalker

[11] Jenkins: https://jenkins.io

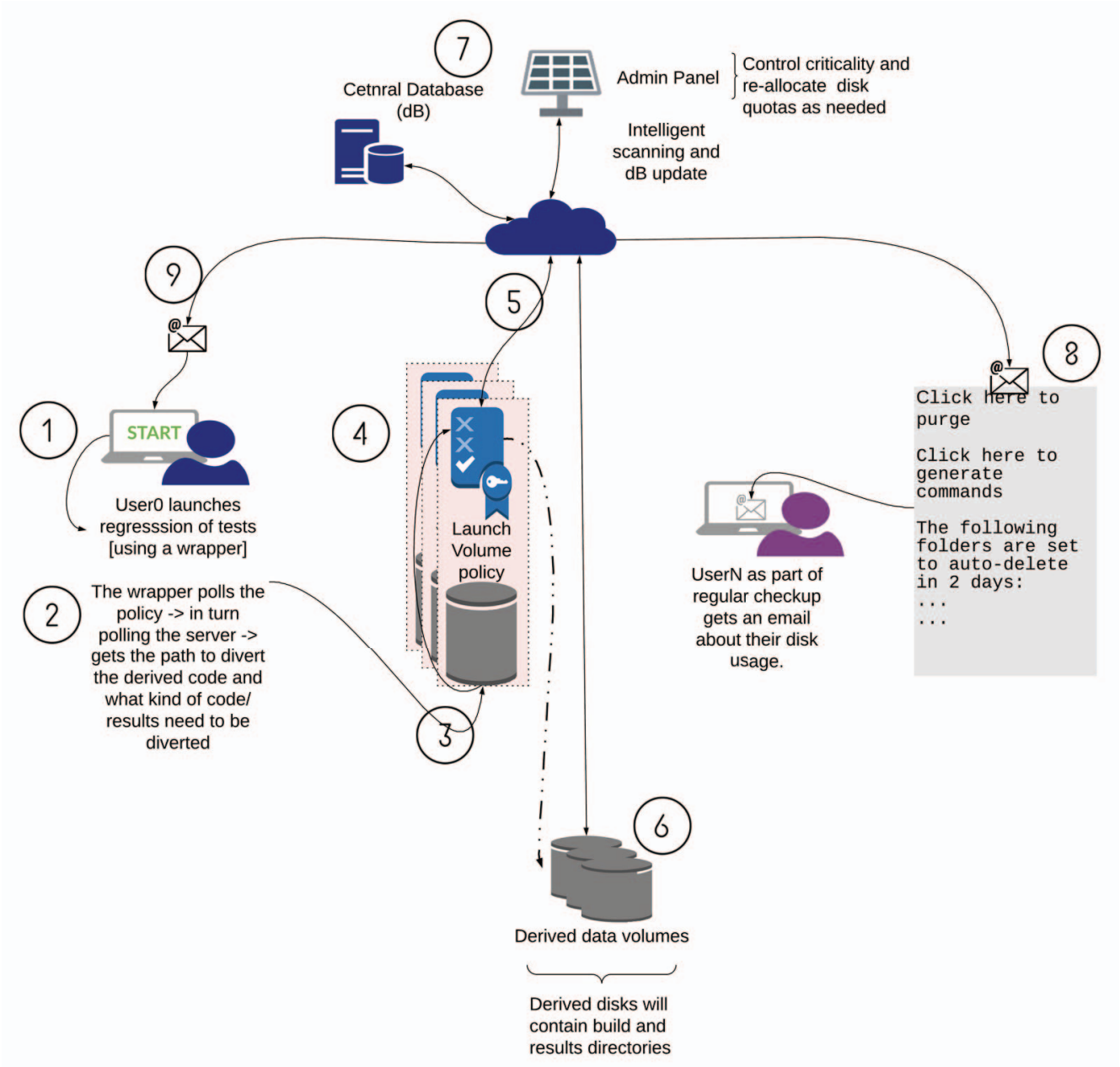

Figure 7. Overall workflow