CS486C – Senior Capstone Design in Computer Science

Project Description

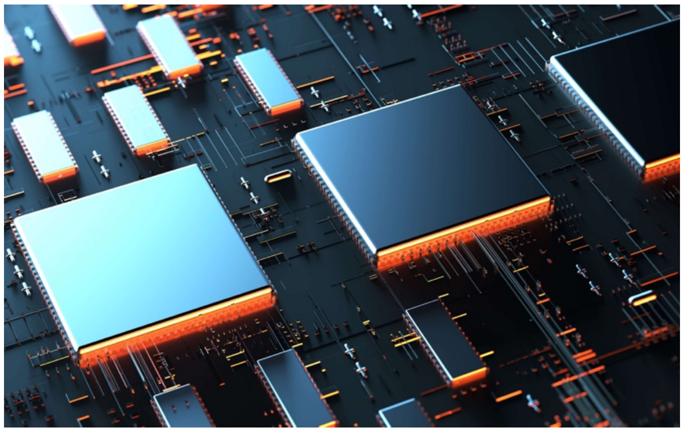
Project Title: VICE (Validation Infrastructure in Cloud Environment)	
Sponsor Information: FPG	**Chris Ortiz**, Senior Technologist Western Digital - Chris.Ortiz@wdc.com **Scott Hancock**, Director of Operations Western Digital - Scott.Hancock@wdc.com

Project Overview:

Fast reliable storage subsystems are at the heart of all computing architectures, in forms ranging from cache, to primary RAM, to hard drive secondary storage. One of the major advances in computer hardware over the last decade has been a fundamental shift away from traditional spinning platter hard drives to lighter, faster, more reliable, and more power-efficient Solid State Drives (SSD). WDC (Western Digital Corporation) is one of the top producers of SSDs worldwide; we supply high-performance, reliable SSD products to a wide range of private and corporate customers around the globe for products ranging from clusters, to servers to consumer machines.

Problem background: testing and validation of SSD firmware as it evolves

As with most complex hardware devices, SSDs have a firmware kernel associated with them that manages and optimizes the low level device hardware and provides an interface to the host operating system. SSD firmware is continually being improved, modified, and extended to function on both existing and new SSD devices. It should be obvious that any software bug or failure at this firmware level could have catastrophic consequences for a deployed SSD system, e.g., loss or corruption of stored data. Thus, all new firmware versions must undergo an intensive and rigorous testing process to ensure that no bugs have been introduced, and that any SSD devices using that firmware function perfectly and reliably across a broad range of conditions. This testing and quality control process is called *validation*.



In principle, validation of SSD firmware is similar to software testing at the application software level, i.e., centered around development and repeated execution of test cases based on various criteria related to compatibility, compliance, interoperability, and characterization. Validation must be much more thorough for SSD firmware, however: failure of application level software might lead to an irritating bug, possibly even a crash of the software; failure of SSD firmware could result in permanent data loss or corruption. A further complicating factor is that SSD devices are incorporated into many hardware platforms, and the detailed differences in these platforms may affect SSD performance. For example, a given SSD subsystem could appear in multiple brands of laptops, various desktops, and perhaps other computing devices; these are called "hardware platforms". Thus, new firmware versions must be carefully tested on a full range of platforms that are carefully chosen to represent and cover all possible systems that the SSD will be deployed in to ensure that the SSD firmware works perfectly on all of them. Finally, on top of all this, a given hardware platform could have any of several operating systems installed, so testing must not only cover all hardware platforms, but also a maximum variety of operating systems on those platforms. Let's call the combination of a particular hardware platform and a host operating system running on it a *testing configuration".*

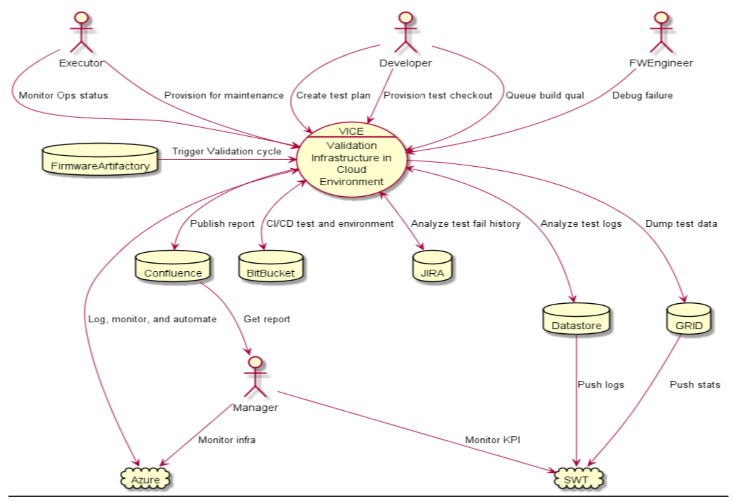
In general, validation of a new firmware candidate is mapped out in a “test plan” that specifies all of the testing configurations (again: hardware device plus an OS) that the new firmware version has to be tested on, meaning they will be part of validation cycles every time our product team releases new SSD firmware. We normally run our validation cycles every two weeks and, during any given validation cycle, there will be tests that fail; when that happens, the platform resources involved in that test will be flagged and the release put on hold until our firmware engineers can debug the problem.

To summarize, each SSD subsystem has a particular test plan developed for it that specifies (a) the (storage/retrieval) tests that need to be run and their expected values; and (b) an entire matrix of hardware platforms and OS’s combinations that need to be tested. The job of the Validation team is to locate each of the targeted hardware platforms, install the targeted OS(s) and validation software on it, and run the tests on each.

One further relevant practical detail is that not all testing platforms are available at all times. Developing the low level hardware testing platforms can be a complex and expensive process, as the hardware often has to be specially instrumented (think: adding sensors or “hooks”) to support the testing and debugging process. To save expense, not all of the hardware platforms being tested are instrumented in this way. Thus, when a test fails on an un-instrumented platform, the Validation Team must try to reproduce the same failure on another platform that is instrumented, or must take the time and effort to add debug capabilities to that platform…which knocks that platform out of use for a time. Another possibility is that a test platform is put on hold when our test developers are checking out their test code or test environment before they deploy it as part of our next test plan. Finally, a test platform can also be placed on hold when it requires maintenance.

The Problem: Organizing and Managing the Validation workflow

It should be evident that, while the firmware validation process is fairly straightforward in principle, it is quite complicated to execute consistently and efficiently in practice: the targeted hardware platforms have to be located, complicated by their current location, status, and level of instrumentation; appropriate software images have to be installed and tests run; and results communicated effectively between team members and the firmware engineers. Currently, we manage this with a lot of manual coordination, mainly through frequent meetings between the firmware team, test developers, functional managers, and test executors. This is very costly in both salary, time, and expended organizational effort, and has only been aggravated by remote work during COVID.



Graphical overview of potential elements in the Validation as a Service (VaaS) model.

Solution Vision: The VICE validation management tool.

What is needed is a powerful software tool to organize, manage, reduce costs, raise efficiency, improve validation process robustness, and generally streamline our entire firmware validation process. Specifically, we envision a powerful, secure web application hosted in a cloud-based architecture called VICE (Validation Infrastructure in Cloud Environment) that allows us to:

- Register all existing and new hardware testing platforms: model, brand, configuration, and provisioning with debugging instrumentation.
- Easily track the status of each platform: (checked out vs. available), location, expected availability, utilization history. This not only streamlines access to needed platforms, but will allow tracking the

cost/benefit performance of individual platforms, e.g., highlighting platforms that are heavily utilized, signaling us to scale up their number.

- Allow us to “virtualize” validation: regardless of physical location, a platform can be “checked out” and can have a software image (OS + testing harness) loaded onto it remotely. The testing suite is then launched and results collected and saved to a database for review.
- A validation manager tool: Allows registration of SSD subsystems, and one or more test plans (sequences of device+OS+test suite) developed for the SSD. When a new firmware candidate comes out, one can create a new “validation cycle” by selecting an SSD and one its test plans and then “launching” that cycle. The Validation team can easily monitor cycles in progress and their current status, assign management of various cycles to team members, and be notified when something goes wrong.

Building this tool based on the remote, virtualized concept outlined here would be revolutionary, essentially enabling a fundamental shift to a new, more efficient “Validation as a Service (VaaS) model.

There are many possible extensions and “stretch goals” for this project, e.g., more extensive automation of the testing plan execution process, or using analysis of past testing logs to predict which testing configurations are most likely to cause problems and dynamically adjusting the test plan to prioritize those tests. As a basic starting point, however, this project will focus on developing the basic validation testing framework, tracking the test platforms and various testing images (OS+test harness), and representing and facilitating monitoring of test plan execution status by team members.

Knowledge, skills, and expertise required for this project:

- Knowledge in UML (PlantUML), Python3, Node.js/TypeScript, Powershell, Kusto Query Language, Linux KVM via libvirt, Docker Containers/Kubernetes Orchestration and Windows Hyper-V.
- Knowledge of CI/CD (Continuous Integration/Continuous Deployment) concepts; this firmware validation problem is an example of such a process.
- Knowledge of REST API, Redfish, Virtual Network, Authentication/Authorization and Webhook.
- Experience in Azure Cloud or other virtualized compute environment (e.g. AWS)

- **Equipment Requirements:**
  - Access to private test platforms and other resources that exist in WDC will be provided as needed by the client.
  - Azure Cloud access. Should be able to use their free trial service for most of development, but other resources will be provided if needed.
  - All development tools and frameworks that will be used should be Open-Source and free as the infrastructure is also aiming to be public or community compliant.

Software and other Deliverables:

- A functioning web application implementing VICE, hosted and demonstrated on a final platform of clients choosing.
- All reports and deliverables should show deep understanding of our process and use the language of the VICE Validation Team Domain.
- A design concept expressed in UML using PlantUML diagrams. The script source of the PlantUML should be shared as this will be integrated into our Confluence pages via RenderMacro widget for PlantUML.
- Must include a complete and clear User Manual for configuring and operating the software in unlocked PDF files which we can freely modify.
- Complete professionally documented codebase delivered both as a repository BitBucket, and as a physical archive on a ZIP file that can be downloaded from Azure Blob Storage.