Automation of Data Consumption by Pluggable ModuleSoftware

Vinay Balamuraliand **Prof. Venkatesh S**

Department of Electronics and Instrumentation Engineering,
RV College of Engineering, Bangalore
vinaybalamurali.ei17@rvce.edu.in, venkateshs@rvce.edu.in

Abstract*. Servers are required to monitor the health of the various I/O cards connected to it to alert the required personnel to service these cards. The Data Collection Unit (DCU) is responsible for detecting the I/O cards, sending their inventory as well as monitoring their health. Currently the keys required to detect these I/O cards are manually coded into the source code. Such a task is highly laborious and time consuming. To eliminate this manual work, a Software Pluggable Module was devised which would read the I/O card related information from the I/O component list. This software design aims at using Data Science and OOPS concepts to automate certain tasks on server systems. The proposed methodology is implemented on a Linux system. The software design is modular in nature and extensible to accommodate future requirements. Such an automation framework can be used to track information maintained in Excel Spreadsheets and access them using an Application Programming Interface (API).*

Keywords*: Data Collection Unit, Pluggable Module, I/O Matrix Module*

1 Introduction

In today’s ever-changing world, automating manual work has become the need of the hour. This not only saves corporations billions of dollars in wages but also makes mundane tasks more efficient. It can also drive employees to pursue tasks which are productive and helpful for the corporation. According to a McKinsey report, 45% of current paid activities can be automated by today’s technology, an equivalent of $ trillion in annual wages. Automation can only be accelerated by latest technologies like data analytics and machine learning. 65% of workers view Artificial Intelligence (AI) as something that would free employees from menial tasks.

The Data Collection Unit (DCU) is an OS-level service that gathers inventory of Operating System (OS) - visible hardware components on server systems. The hardware components include I/O cards such as Graphics Processing Unit (GPU), Network Interface Card (NIC), Non-Volatile Memory Express Controllers (NVMe), etc. It also proactively monitors the health of those components and reports this information to the Rack Management Controller (RMC) via the in-band Intelligent Platform Management Interface (IPMI) system interface. It is an agentless service that gathers health status of the various I/O cards connected to the server and is essential in providing Mission Critical capabilities to these servers. DCU then reports these errors to the firmware. These errors are projected onto a Graphical User Interface (GUI) and could show potentially crucial information. This can include data such as when a card has failed and when it requires replacement.

Currently, enablement of DCU to monitor health of new IO cards on server platforms is a repetitive and tedious task. In the current implementation, we need to carefully examine and investigate the supported I/O Component list to transform the same into C++ header files. A key is required to uniquely identify an IO card from the I/O list, which could be a combination of PCI-ID’s (vendor Id, sub vendor Id, device Id, sub device Id) or Model Number, Vendor Number. Each I/O subsystem maintains different key, hence separate header files must be maintained per subsystem. Current approach has the following shortcomings:

(a) DCU needs to manually collate, correlate the data from various sources (IO Component list, https://pci-ids.ucw.cz/v2.2/pci.ids, inputs from partners) to generate the complete I/O Component list.
(b) For each IO update on the platform, we need to parse the entire I/O list, fetch the key required attributes, and update the header files. This amounts to significant development and test effort.
(c) Due to manual parsing, the code is prone to human errors.
(d) Lastly, it doesn’t align with C++ code reusability objectives.

DCU supplements attributes like Model Name, Spare Part Number, etc. which are significant for monitoring the I/O cards on server platforms and enabling faster troubleshooting for client applications. This is done through the RMC which exposes health events for client applications (GUI). These tools receive service events from the RMC, which are actionable events and are automatically forwarded to Support Personnel, enabling the quick service response.

Thus, there is a need to automate the identification of each modification to the server I/O Sub-system components. The intent of this paper is to provide a pluggable module to enable DCU or any other application to integrate the IO subsystem components data seamlessly. This helps to improve the overall supportability of the server platform. Figure 1 shows a typical server in a datacentre.

2 Literature Survey

Most of the work done so far has been related to monitoring of networks or IoT devices. Hassan Jamil Syed et al. [1] proposes a system to monitor data through the host Operating System without interfering with its functioning. This is done by linking the data with the cloud controller. The data was gotten through the Proofs file in Linux and this was relayed to a dashboard on the node of the cloud controller. Therefore, an elegant, efficient, and light-weight framework for monitoring was devised. This is scalable and has almost negligible overhead involved.

Rivadulla Campello et al. [2] discusses Internet of Things platforms needing to deploy a Wireless Sensor Network's (WSN). To ensure smooth functioning of this

Fig.1. Server Room

paradigm, debugging, and monitoring tools are required. Monitoring Platforms gather about the working of WSN thereby detecting errors and evaluating its performance. This monitoring can be implemented via software or through hardware. Using Hybrid Platforms has many more advantages than limitations. Many tools used today have synchronization issues. A monitoring platform called HMP is presented through the means of this paper. It combines hardware, software, passive as well as active approaches. This enables the HMP to gather data from the sensor nodes directly (active) or through the WSN (passive). This type of system is reusable and causes very less interference to the network. It also contains an effective trace synchronization mechanism.

S. V. Sugared et al. [3] discusses the pros and cons of using a database for monitoring intellectual software and hardware in lingering objects. Certain pipeline related parameters are stored in the data base and this is the key component of the monitoring software. Status data of the pipeline such as pumping modes, technical documentation data, diagnostic logs, electrometric survey logs, etc., are present. This data is utilized to carry out the control and monitoring of the pipeline. The database contains structures for processing these data and helps limit mechanical deformation, corrosion, and damages.

Data Science methodologies were also studied to gain knowledge about the best practices in the industry. Pandas was found to be the most relevant and simplified method to read data from Excel sheets. Igor Stannic et al. [4] reviews various data mining and big data analysis libraries present in Python. All the advantages and disadvantages are discussed in detail. There are six groups to which they can be classified: core libraries, data preparation, visualization, machine learning, deep learning, and big data. Pandas was recommended as the best method for data preparation.

3 Methodology

Our proposal comprises of defining a nightly Jenkins job to download the latest IO list and picids file, combine various data sources and process them using Data Science libraries – pandas and NumPy. Data processing is required:

(a) To accommodate different IO subsystems with different keys
(b) To accommodate multi-platform support for a particular IO card
(c) To filter out the relevant attributes needed by pluggable module
(d) To transform the data frames into a ‘CSV’ format

The CSV file is ready to be consumed by the C++ Pluggable Module, which can then be integrated to the DCU or any other client application on the server systems. Thus, enabling client applications to fetch the IO card details based on keys as per the required specification. IO Component list continuously goes through changes, with either new subsystems are added, or existing card details are modified. Our approach will be able to handle all these modifications and eliminate the need for C++ code changes in the client application.

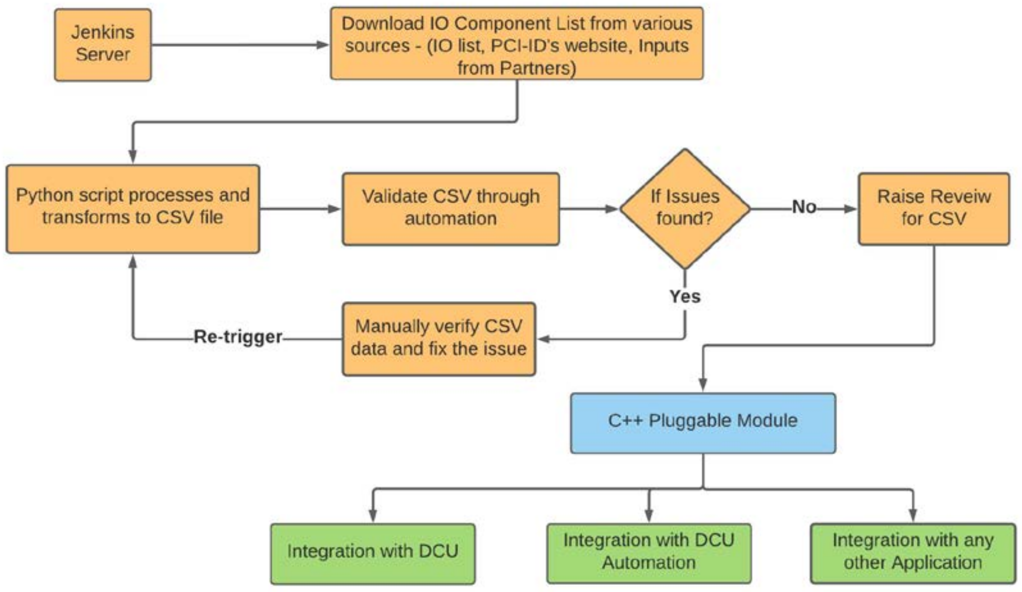

Jenkins Server
Download IO Component List from various sources - (IO list, PCI-ID's website, Inputs from Partners)
Python script processes and transforms to CSV file
Validate CSV through automation
If Issues found?
No
Raise Reveiw for CSV
Yes
Manually verify CSV data and fix the issue
Re-trigger
C++ Pluggable Module
Integration with DCU
Integration with DCU Automation
Integration with any other Application

Fig.2. Workflow of Implementation

4 Implementation

The proposed methodology is implemented on a Linux system as the support is extensive and it is open source. The Pluggable module was converted to a Shared Library, that is, a Shared Object (SO). This allows dynamic linking of DCU code with the Pluggable Module during run-time. It also ensures that the Pluggable Module code remains hidden and only the essential objects are seen. The Data Collection Unit retrieves the IO card data in the form of a map structure where the key is unique. The values are later used to identify the I/O cards through interaction with the firmware. Shared Libraries creates a dependency between the DCU and Pluggable Module. Every time DCU needs to identify an I/O card; the Pluggable Module .so file must be included.

Figure 2 depicts the overall workflow to implement the proposed methodology.

4.1 IO Matrix Module

The IO Matrix module is essentially involved in reading the IO component list which is present in an Excel spreadsheet and processing the data appropriately. It is developed using popular Data Science libraries, that is, NumPy and Pandas. The module was designed in a modular structure so that future versions of the IO component list wouldn’t require much code changes. The IO component list to be processed by the IO Matrix module was uploaded onto the web-based interface of Jupyter.

4.2 Pluggable Module

This module is developed using the C++ 14 standard. It is tasked with reading the IO Matrix csv file using file pointers. The Pluggable Module can be integrated with DCU using a library file - ‘.dll’ or a shared object. The class declarations were coded in header files. The class definitions were coded in their respective ‘.cop’. files.

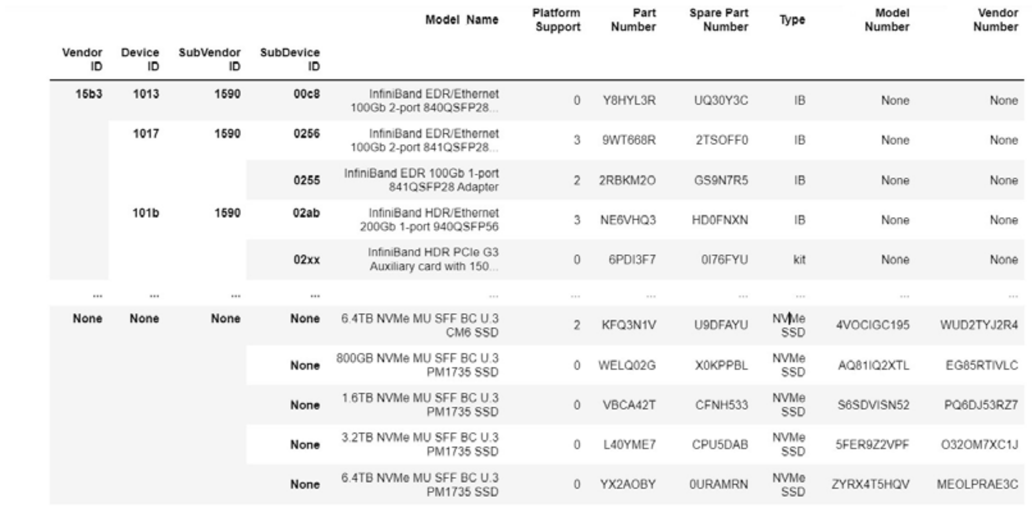
Visual Studio provided a simplified experience to build the design. The debugger helped set breakpoints to debug errors. The project file contained the entire solution and could be exported with the necessary source files. The program I/O was through a console.

5 Results

The results or outputs obtained from the execution of the programs, developed in the previous sections, are discussed here. Both outputs of the Pluggable Module and IO Matrix Module are discussed. Some test scripts were written to generate these as no console output is really obtained in the implementation.

5.1 Testing IO Matrix Module

The IO Matrix Module output was printed onto the screen before the generation to CSV file was conducted. This output had to be manually verified for randomly chosen rows to ensure data integrity. This output is shown in Figure 3. It can be noted that the PCIID’s field is empty for SSD and HDD. A unique combination of Model Number and Vendor Number is used to retrieve the value from the map.



				Model Name	Platform Support	Part Number	Spare Part Number	Type	Model Number	Vendor Number
Vendor ID	**Device ID**	**SubVendor ID**	**SubDevice ID**							
15b3	**1013**	**1590**	**00c8**	InfiniBand EDR/Ethernet 100Gb 2-port 840QSFP28...	0	Y8HYL3R	UQ30Y3C	IB	None	None
	1017	**1590**	**0256**	InfiniBand EDR/Ethernet 100Gb 2-port 841QSFP28...	3	9WT668R	2TSOFF0	IB	None	None
			0255	InfiniBand EDR 100Gb 1-port 841QSFP28 Adapter	2	2RBKM2O	GS9N7R5	IB	None	None
	101b	**1590**	**02ab**	InfiniBand HDR/Ethernet 200Gb 1-port 940QSFP56	3	NE6VHQ3	HD0FNXN	IB	None	None
			02xx	InfiniBand HDR PCIe G3 Auxiliary card with 150...	0	6PDI3F7	0I76FYU	kit	None	None
...	...	...	...	...	...	...	...	...	...	...
None	**None**	**None**	**None**	6.4TB NVMe MU SFF BC U.3 CM6 SSD	2	KFQ3N1V	U9DFAYU	NVMe SSD	4VOCIGC195	WUD2TYJ2R4
			None	800GB NVMe MU SFF BC U.3 PM1735 SSD	0	WELQ02G	X0KPPBL	NVMe SSD	AQ81IQ2XTL	EG85RTIVLC
			None	1.6TB NVMe MU SFF BC U.3 PM1735 SSD	0	VBCA42T	CFNH533	NVMe SSD	S6SDVISN52	PQ6DJ53RZ7
			None	3.2TB NVMe MU SFF BC U.3 PM1735 SSD	0	L40YME7	CPU5DAB	NVMe SSD	5FER9Z2VPF	O32OM7XC1J
			None	6.4TB NVMe MU SFF BC U.3 PM1735 SSD	0	YX2AOBY	0URAMRN	NVMe SSD	ZYRX4T5HQV	MEOLPRAE3C

Fig.3. Pandas Data-frame Output

5.2 Testing Pluggable Module

The information in the Pluggable Module is stored in a Map structure. The unique key combination for different types of cards maybe different. Also, the value to be retrieved for these cards may vary. This can be seen in Figure 4. The key is queried from 'main' function to print the values. The Figure 4 shows the command prompt output on Windows when the executable file of the Pluggable Module is run.

6 Interpretation of Results

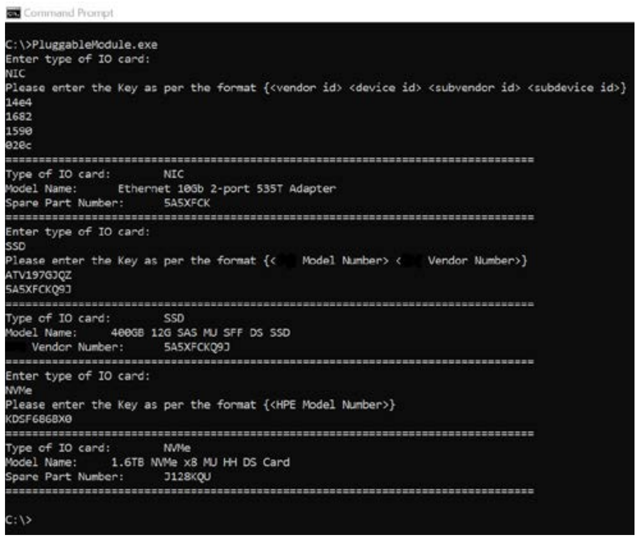
(a) The primary objective of the design was to provide a Pluggable Module which can be integrated with DCU. This Pluggable Module would remove the necessity to hard code or manually code certain data into the header files of DCU source code. This data includes PCI-ID's, Model Number etc. The other objectives included reading the IO component list and providing support for all three OS.

(b) The IO Matrix Module was designed to process the IO component list. Directly reading the IO component list through C++ code was impractical as the data was vast. Therefore, the data had to be processed first. The Pluggable Module was then designed to consume this CSV file using

OOPS paradigms as it ensures code reusability and modularity. To start of providing support for all three OS, Linux OS support was provided first.

7 Conclusion

The software design of Pluggable Module aimed to solve the problem of manually coding the information related to I/O cards into DCU and it was achieved by using automation. A Pluggable module which is modular in nature and reusable was designedto consume the CSV output of the IO component list. An intricate combination of Data Science using pandas library and OOPS concepts achieved the processing and organization of data into a CSV file, and its later consumption by the Pluggable Module. Hence, this design uses the complete software automation loop and requires no human intervention, whatsoever.



```
C:\>PluggableModule.exe
Enter type of IO card:
NIC
Please enter the Key as per the format {<vendor id> <device id> <subvendor id> <subdevice id>}
14e4
1682
1590
020c
================================================================================
Type of IO card:        NIC
Model Name:      Ethernet 10Gb 2-port 535T Adapter
Spare Part Number:      5A5XFCK
================================================================================
Enter type of IO card:
SSD
Please enter the Key as per the format {<[redacted] Model Number> <[redacted] Vendor Number>}
ATV197GJQZ
5A5XFCKQ9J
================================================================================
Type of IO card:        SSD
Model Name:     400GB 12G SAS MU SFF DS SSD
[redacted] Vendor Number:      5A5XFCKQ9J
================================================================================
Enter type of IO card:
NVMe
Please enter the Key as per the format {<HPE Model Number>}
KDSF6868X0
================================================================================
Type of IO card:        NVMe
Model Name:     1.6TB NVMe x8 MU HH DS Card
Spare Part Number:      J128KQU
================================================================================

C:\>
```

Fig.4. Output of Pluggable Module

8 Future Scope

Automating the consumption of IO Component list has eased the need of adding support for new IO cards. This technique of reading information from excel sheets can automate processes were only parameters have to be changed. The change of parameters can trigger an automatic build and hence produce the new executable code.

References

1. H. J. Syed, A. Gani, F. H. Nasaruddin, A. Naveed, A. I. A. Ahmed and M. Khurram Khan,"CloudProcMon: A Non-Intrusive Cloud Monitoring Framework," in IEEE Access, vol. 6, pp. 44591-44606, 2018, doi: 10.1109/ACCESS.2018.2864573.
2. M. N. Mendoza, J. C. Campelo Rivadulla, A. M. Bonastre Pina, J. V. Capella Hernandez and´ R. Ors Carot, "HMP: A Hybrid Monitoring Platform for Wireless Sensor Networks Evaluation," in IEEE Access, vol. 7, pp. 87027-87041, 2019, doi: 10.1109/ACCESS.2019.2925299.
3. S. V. Susarev, N. G. Gubanov, D. A. Melnikova, Y. V. Sarbitova and A. A. Odintsova,"Use of the database during the operation of the intelligent hardware and software monitoring complex of lingering objects in real-time mode," 2017 XX IEEE International Conference on Soft Computing and Measurements (SCM), 2017, pp. 509-511, doi: 10.1109/SCM.2017.7970632.
4. I. Stancin and A. Joviˇ c, "An overview and comparison of free Python libraries for data min-´ ing and big data analysis," 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2019, pp. 977-982, doi: 10.23919/MIPRO.2019.8757088.
5. N. M. Babu and G. Murali, "Malware detection for multi cloud servers using intermediate monitoring server," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, pp. 3609-3612, doi: 10.1109/ICECDS.2017.8390135.
6. M. Seo and R. Lysecky," Work-in-Progress: Runtime Requirements Monitoring for Statebased Hardware," 2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2018, pp. 1-3, doi: 10.1109/CODESISSS.2018.8525882.
7. C. Guo, X. Li and J. Zhu," A Generic Model for Software Monitoring Techniques and Tools,"2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing, 2010, pp. 61-64, doe: 10.1109/NSWCTC.2010.22.

8. M. Beastrom and P. Morreale," Scalable Agentless Cloud Network Monitoring," 2017 IEEE4th International Conference on Cyber Security and Cloud Computing (CSCloud), 2017, pp.
171-176, doe: 10.1109/CSCloud.2017.11.
9. D. K. Peters and D. L. Parnas," Requirements-based monitors for real-time systems," inIEEE Transactions on Software Engineering, vol. 28, no. 2, pp. 146-158, Feb. 2002, doi: 10.1109/32.988496.
10. Liu Yanbin, Zhu Xiaodong, Wang Yigang, Feng Jing, and Qu Changzheng," Model and implementation for runtime software monitoring system," 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), 2010, pp. 761-765, doi:
10.1109/ICCAE.2010.5451251.