CSCI 656

Assignment 1

Matthew Collins

The architecture described in the paper **"Koziolek et al. - 2021 - Dynamic Updates of Virtual Programmable Logic Controllers (PLCs) Deployed as Kubernetes Microservices"**, introduces a dynamic updating strategy for Virtual PLCs in K8s environments, leveraging a specialized K8s Operator. Their method permits seamless application updates without disrupting production. Notably, their tests indicate that this mechanism can transfer internal states of sizable control applications (with 100,000 state variables) utilizing just 15% of the available cycle slack time, opening doors for real-time updates and cloud-native migrations.

The internal state transfer is a critical aspect of the architecture. For instance, for a control algorithm that requires 10% of the cycle time for computation (i.e., 10 ms), a 400 KByte internal state needs to be transferred in less than 90 ms. This demands a minimum transfer rate of about 4.5 MByte/sec. While typical RAM, bus, and network bandwidths can handle this, the main bottleneck might be the CPU and possibly the network latency. The controller, which is responsible for the state transfer, is implemented as a separate POSIX thread with high real-time priorities. This ensures deterministic execution of the state transfer, especially when deployed in a real-time capable operating system.

This paper doesn't dive deeply into security, but it's implied that the architecture relies on established technologies and standards like OPC UA, which has inherent security features. Additionally, the use of Kubernetes and containerization also brings in security practices inherent to these technologies.

The architecture is designed to handle large industrial control applications. However, there are some limitations. For instance, the approach is not meant for fast machine control applications in discrete automation that require sub-millisecond cycle times. The system was tested on comparably fast server hardware, which might not be available in many smaller production plants. However, such hardware is essential for running non-trivial container workloads in an orchestration system. Overall, the scalability of this solution for plants with the capability of hosting a sizable server room, is better than the alternative of having remote PLCs throughout the plant/manufacturing facility.

The architecture detailed in this paper is not suitable for applications requiring extremely fast cycle times (sub-millisecond). The experiments were conducted on fast server hardware, which might not be universally available. The experiments included only simulated IO devices and not real devices communicating with the OpenPLC runtime. The prototypical implementation relied solely on open-source components, meaning results might vary with different software components, whether open-source or commercial. The ADRs were done using adr-tools and

merged with pandoc. See the attached ADRs at the end of this paper. This design is still in a very early prototype phase. All tests were performed in simulation. However, given the design constraints of 100 ms detailed by a standard IEC 61131 PID controller, the architecture presented in this paper performed well within the constraints. The worst case test runs presented in this paper were a state size of 500,000 variables, and the longest run took less than 70ms.

1. Transfer Mechanism

Date: 2023-09-13

Status

Accepted

Context

A transfer mechanism i sneeded to enable reliable, flexible, and fast state transfer..

Decision

Open Platform Communications Unified Architecture (OPC UA) will be used to provide a middleware for transfering state quickly and reliably. OPC UA is designed for monitoring industrial devices from workstations, but was lately extended to also support fast, deterministic controller-to-field device communication. OPC UA includes a client/server protocol on top of TCP/IP, as well as a publish/subscribe mechanism on top of UDP. OPC UA address spaces may hold both configuration and sensor data. The Open Process Automation Forum (OPCAF) has identified OPC UA as the core communication mechanism in future open and interoperable industrial control systems. Controllers and certain field devices shall be equipped with OPC UA clients and servers, while legacy field buses shall be integrated via OPC UA gateways.

Consequences

The only downside is OPC UA is niche protocol. Less open source libraries will support OPC UA. Fewer developers and engineers will know the protocol. More custom code will need to be written and less off the shelf support will be expected.

2. Serialization Mechanism

Date: 2023-09-13

Status

Accepted

Context

A way to store the state and memory structure before the state is transfered is needed.

Decision

Binary large objects (BLOB) will be used to transfer the application state. A BLOB allows to abstract the internal application memory structure, which is important as the internal memory state structure might be vendor-specific. Hence, using a BLOB instead of structured data, allows to keep the interface of the state transfer service stable for different virtual PLCs. Additionally, it gives the ability to use data compression techniques or encryption

Consequences

Storing data as BLOBs may result in reduced data integrity because there are no constraints or validations applied to the binary data. Structured data can have rules and constraints to maintain data consistency.

3. State Transfer Service as K8s Extension

Date: 2023-09-13

Status

Accepted

Context

A state transfer service is needed to implement the OPC UA client and Virtual PLC Controller.

Decision

A K8 operator (Virtual PLC Operator) running in a pod will be the state transfer service.

Consequences

There is not out of the box K8 operator for this task. A custom pod will need to be developed with OPC UA, open62541, and a webserver.

4. Container Virtualization

Date: 2023-09-13

Status

Accepted

Context

A way to deploy these virtual PLCs is needed.

Decision

We will use containers in the form of Kubernetes pods to orchestrate instances and components which handle the compute and networking needed for the virtual PLC engines and state transfer service.

Consequences

An decrease in perforance in terms of jitter is possible with Docker/Kubernetes pods. SWAP limitations are a concern in embedded environments.