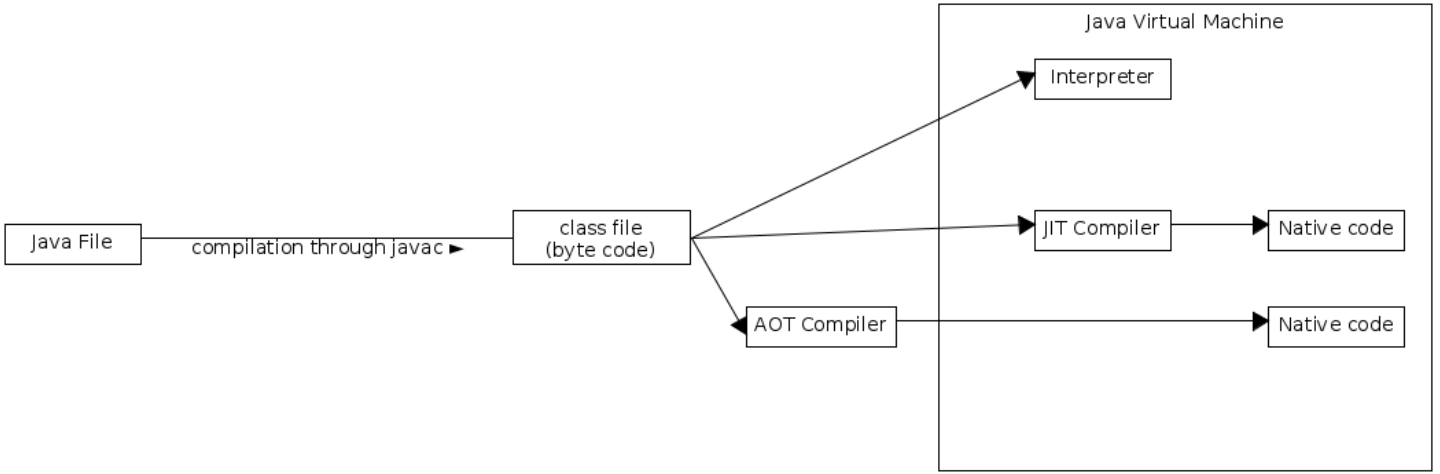
Performance Evaluation of a Middleware Framework for CPS/IoT Ecosystem

Can Deliorman

Background
- Java Virtual Machine

Java Virtual Machine

A virtual machine that enables a computer to run Java programs that are compiled to Java bytecode.



Java Virtual Machine - Class data sharing

The class data that should be loaded into a JVM is loaded first to a cache. During subsequent JVM invocations, the shared cache is memory-mapped to allow sharing of class data for these classes among multiple JVM processes.

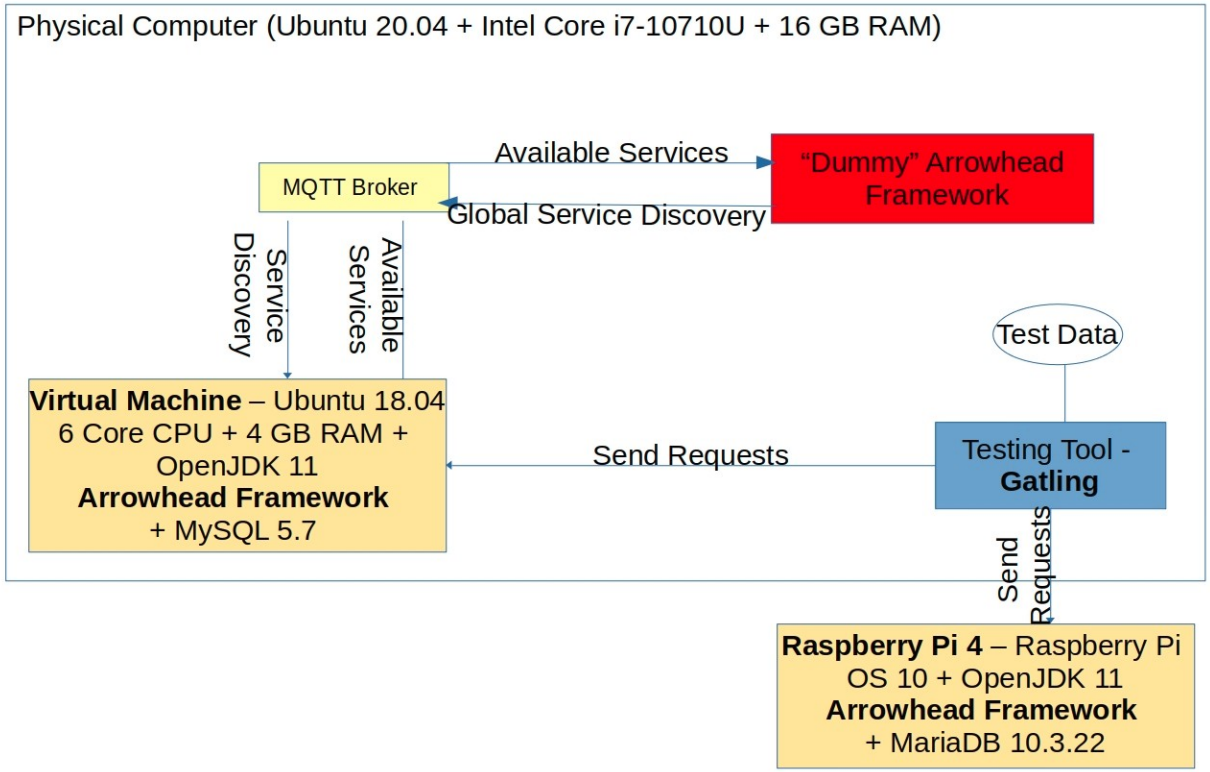
- With JDK 10, Oracle has introduced Application Class-Data Sharing (ApsCDS) to include selected classes from the application classpath.
- OpenJ9 combines the class data sharing with AOT compiling to increase startup times and reduce memory consumption.

Methodologies & Implementation

Objectives of our Performance Evaluation

- Find the limits, where we have an acceptable response time under 10 seconds/request on Arrowhead Framework
- Find out the impact of devices with limited resources on these limits.
- Find potential errors
- Find the impact of alternative JVMs and different configurations such as shared classes cache (SCC) on performance and resource consumption.
- Find the optimal hardware and software configurations for different use cases in real world.

Testing Environment



Generating Test Data

- Gatling as testing tool
- Our test data is fed to Gatling through the CSV files
- 10,000 services with variables of a service were randomly generated: serviceDefinition, systemName, port, authenticationInfo (only in secure mode), serviceUri, version, interfaces.
- 10,000 intra-cloud and inter-cloud rules with the following parameters: consumerId, (only for intra-cloud), cloudId (only for inter-cloud), providerId, interfaceId, serviceDefinitionId.
- Single CSV file for Orchestrator tests: consumerName, port, and serviceDefinition (of the requested service)
- 10,000 Arrowhead compliant X.509 identity certificates as authentication information were also generated

JSON Data for Service Registration

{"serviceDefinition": "**${serviceDefinition}**",
"providerSystem": {
"systemName": "**${systemName}**",
"address": "localhost",
"port": **${port}**,
"authenticationInfo": "**${AuthInfo}**" // existent only in secure mode},
"serviceUri": "**${serviceUri}**",
"secure": // in tests with secure mode "TOKEN" ,
//in tests with insecure mode "NOT_SECURE"
"metadata": {
},
"version": "**${version}**",
"interfaces": ["**${interfaces}**"] }

Gatling Code

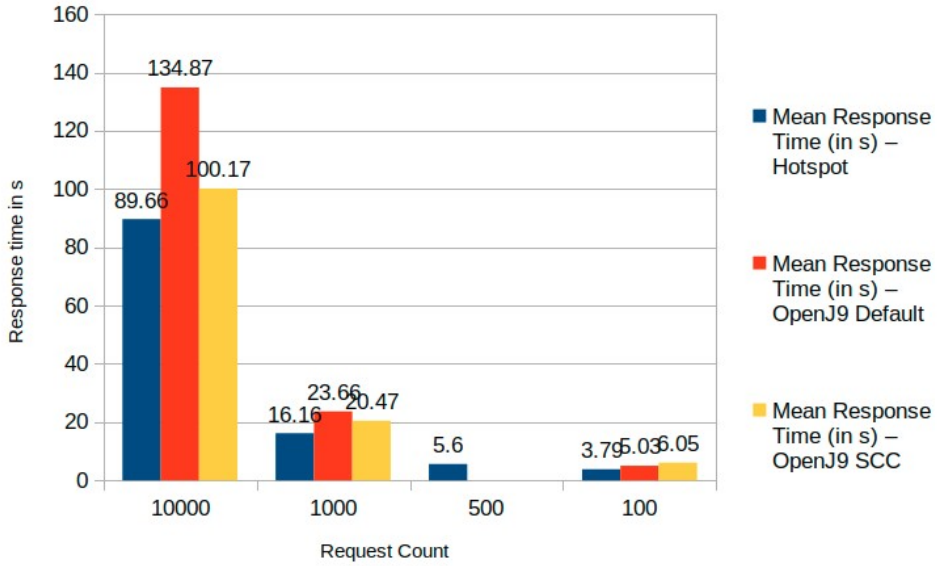
```
val feeder = csv("Name_of_the_csv_file.csv").queue
val stringBody = """JSON data to be sent"""
val httpProtocol = http
.baseUrl("Base_URL_of_testing_object")
.acceptHeader("text/html,application/xhtml+xml,
application/xml;q=0.9,*/*;q=0.8")
.doNotTrackHeader("1")
.acceptLanguageHeader("en-US,en;q=0.5")
.acceptEncodingHeader("gzip, deflate")
.userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0)
Gecko/20100101 Firefox/31.0")
val scn = scenario("SimulationName").feed(feeder).exec(http("request_1").
post("URL_of_object").body(StringBody(stringBody)).asJson)
SetUp( scn.inject(atOnceUsers(Count_of_simultaneous_requests))).protocols(httpProtocol)
```

Tested Components and Functionalities

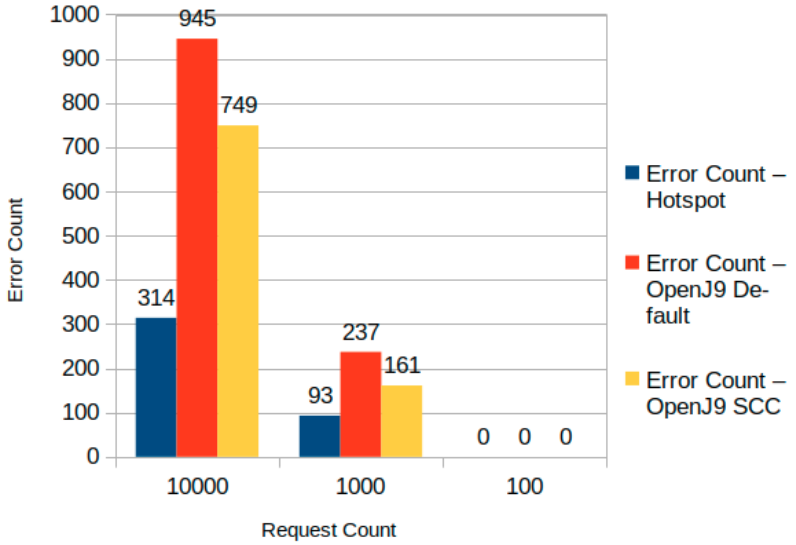
- Service Registry: Service Registration process
- Authorization: Inserting intra-cloud authorization rules.
- Orchestrator: Intra-cloud and inter-cloud service discovery process.
- Authorization and Orchestrator tests are also integration tests
- Up to 10000 simultaneous requests were sent to the components.
- Tests were conducted on an x86-based processor with
  - HotSpot JVM with default configurations
  - OpenJ9 JVM with default configurations
  - OpenJ9 with enabled shared classes cache
- Tests were also repeated on Raspberry Pi 4 with Hotspot JVM with default configurations

Results & Discussion

Orchestrator – Intra-cloud service discovery

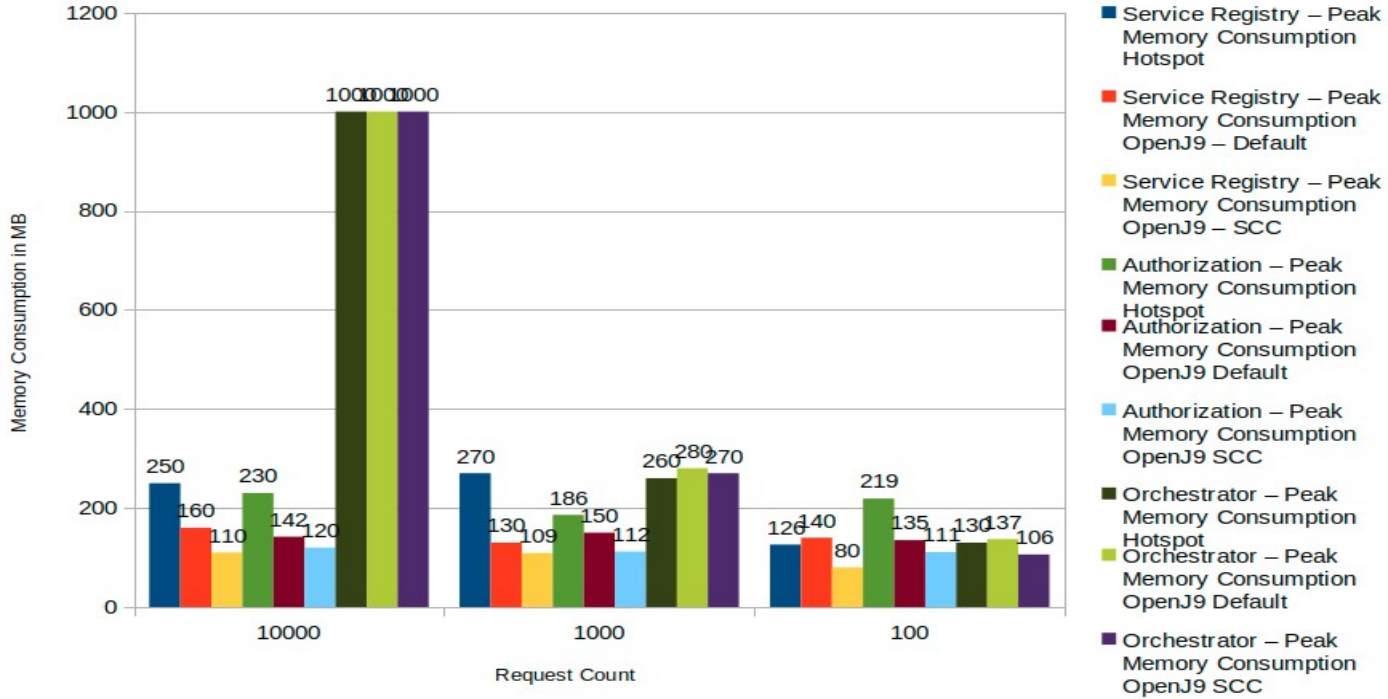


Mean Response Time on different JVMs



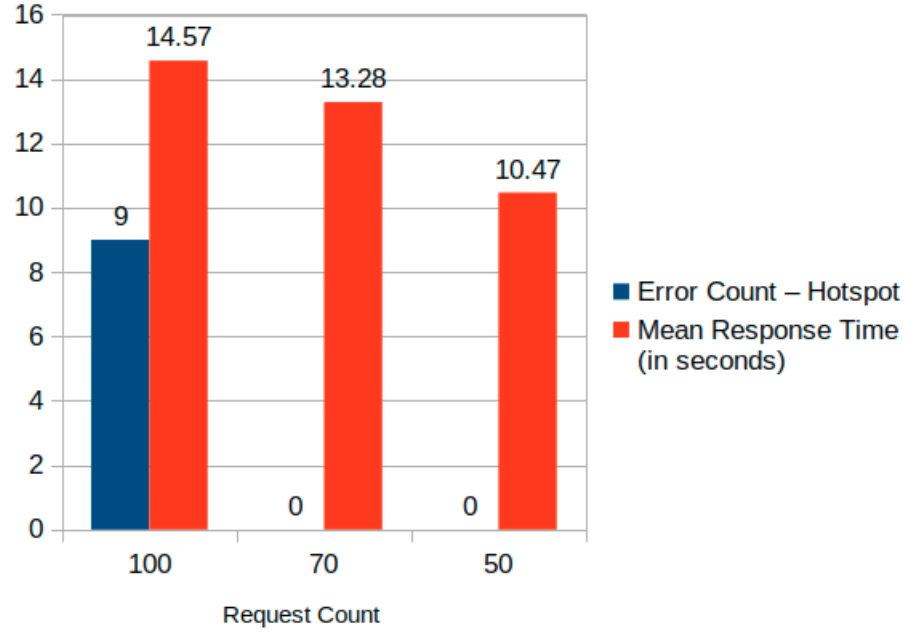
Error count in tests

Orchestrator – Intra-cloud service discovery



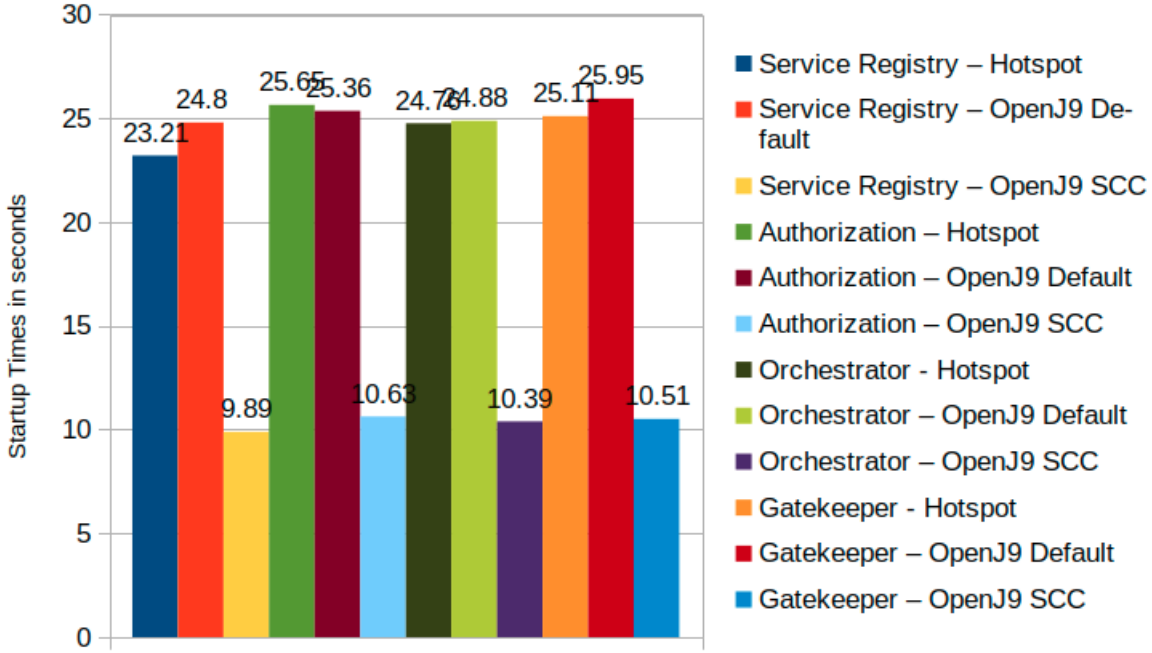
Memory Consumption of different components on different JVMs

Orchestrator – Intra-cloud service discovery



Mean response times and error counts on Raspberry Pi

Startup Times



Startup Times of different components on different JVM configurations

Discussion & Conclusion

- No crashes
- Exception Handling worked properly
- The intra-cloud and inter-cloud service discovery process is very resource-intensive
- Raspberry Pi 4 would be eligible for expected low traffic

Component	X86 Hotspot – Lower Limit (in req./sec.)	Raspberry – Lower Limit (in req./sec.)
Service Registry	<1000	<100
Authorization	<1000	<100
Orchestrator - Intracloud	<500	<50
Orchestrator - Intercloud 1	<40	<20
Orchestrator - Intercloud 2	<50	<10

Discussion & Conclusion

- For devices with expected low traffic, OpenJ9 SCC and hardware with limited resources may be used
- For devices with high traffic, high performance requirements and better availability, HotSpot is the optimal solution.
- The lower start-up times with OpenJ9 SCC allow us to better scale up applications e.g., in the event of higher demand.

Java Virtual Machine	Advantages	Disadvantages
HotSpot default configurations	Best Performance	Highest memory consumption
OpenJ9 default configurations		Worst performance, High memory consumption
OpenJ9 JVM enabled shared classes cache	Lowest memory consumption	Worse Performance than HotSpot

Thanks for your attention!