

1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers

Problem, context, and related work

Robot Operating System 2 (ROS) is a popular and widely adopted robotics engineering framework. Its main goal is to provide robotics engineers a platform that allows for the development of robotics applications. Framework's infrastructure, which is open source, is based on numerous distributed packages. ROS supports a wide range of devices, starting from small embedded devices up to autonomous industry robots. ROS was developed with real-time capabilities in mind. Since this technology is in its infancy, the ROS community encourages industry and academic research. Academic research in ROS has covered many aspects of robotics development and has achieved quantifiable results regarding soft and hard real-time constraints, lowering latency, and in general, improving performance. However, because of its complicated vertical stack of technologies, there is still room where significant performance benefits could be extracted. The default way to extract performance with aim to lower execution time is by utilizing multi-cores. However, ROS with its complex infrastructure and requirement from robotics engineers to learn parallelization from C/C++ perspective, make it hard to utilize multi-cores properly. Therefore, utilizing multi-cores to lower execution time is one of the main motivators that guides research in ROS. By lowering execution time we gain performance benefits and allow other parts of ROS infrastructure, which are influenced by latency and real-time constraints, to be positively impacted as well. Robots are very heterogeneous devices. Therefore, robots may come with a vast array of different system requirements and system constraints. Some of the system constraints are critical when it comes to the correct functionality of the robots. These critical constraints deal with the overall performance of the robotics application. In the scope of overall performance of the robotics application, the main goal of robotics engineers is to lower execution time or make a particular function execute faster. To facilitate this, we refer to utilization of multi-core processors. However, required specialized knowledge regarding concurrency and expertise, which robotics engineers do not have, slows them down in building their robotics application and decreases efficiency. Robotics engineers are required to understand the architecture of the CPU and how it interacts with main memory. On top of that robotics engineers are required to understand the intricacies regarding caches and data locality. Complemented with this is a software stack that is utilizing the CPU. They need to learn how to use multi-threading libraries, which is not simple. Functions for creating and managing threads require thinking about multiple data-flows in software stack. As a consequence, robotics engineers are required to understand in low-level detail how multi-threading works and how prioritization, scheduling, and affinity combine to solve the problem of lowering execution time. To conclude, the main gap is that robotics engineers do not have an intuitive way of utilizing multi-cores to be able to lower execution time. They are required to know low-level details regarding parallelization and have to develop multi-threading applications. Additionally, robotics engineers do not have accessible interfaces to control Operating System mechanisms regarding prioritization, scheduler type, and affinity

Requirements

- SR.01: Framework to facilitate development of multithreaded applications in ROS2
- SR.02: Framework to assist robotics engineers in developing software that takes advantage of multi-cores in order to reduce average execution time, integrated in ROS2.
- SR.02: Pre-locking heap memory - The framework enables pre-locking of heap memory during initialization of the application in order to avoid failed memory allocations, which can occur if application requires significant amount of memory.
- SR.03: Process priority configuration - The framework provides interface for setting priorities of threads in order to avoid priority inversion.
- SR.04: Measurement of heap allocations - The framework offers a visualization on how many memory allocations happened, and it should be easily accessible to the robotics engineers
- SR.05: Measurement of latency. - The framework has a mechanism that

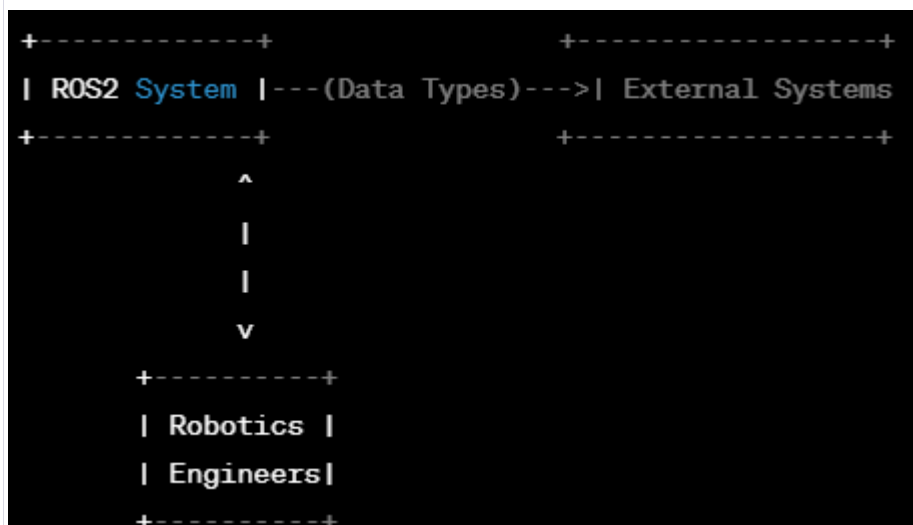
measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: Measurement of memory fetches. - The framework has a mechanism which measures memory fetches. • SR.09: Executor type - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing executor type configuration containing nodes. & ROS executor • SR.10: Node to process assignment - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing configuration. & node assignment, thread, process • SR.11: Allocator type - The framework accepts the message data types that are used throughout the application and it advises robotics engineers on which allocators to use. • SR.12: Easy to use user interface - All information and capabilities of the framework should be gathered in easy-to-use user interface. • SR.13: Guideline - Robotics engineers will be able to express their intentions of parallelizing the code in the framework. The framework will output guidelines for robotics engineers. • SR.14: Parallel Configuration - The framework will accept non parallelized code and will turn it into parallel configuration. & Parallel Configuration Quality requirements • SR.01: Increased CPU Utilization - The framework facilitates the development of multi-threaded applications, and as a result, increased CPU utilization. If the number of cores are n , (and $n > 1$), then utilization is at least $0.7 * n$. • SR.02: Decreased Heap Allocation - The framework facilitates the mechanism with pre-locking memory during the start of the execution of the application. At least 50 percent decrease in heap allocations regarding memory is expected. • SR.03: Increased general performance of the application - The framework facilitates the mechanism for process priority configuration. We expect an increase in general performance in a region of 10 percent. • SR.04: Decreased average execution time - The framework facilitates the mechanism for decreasing average execution time by utilizing multi-threading. • SR.05: Increased CPU Cache utilization - We expect an increase in CPU Cache utilization by 30 percent. • SR.06: Decreased Memory Fetches - We expect to have a decreased memory fetches by 20 percent as a result of increased CPU Cache utilization. • SR.07: Increased productivity of robotics engineers - The facilities of the framework will output parallelized configurations. Compared with by-hand optimization, this process will be expected to take 30 percent less time. & increased productivity • SR.08: Usability - The framework will include an easy to use user interface. We expect robotics engineers to quickly understand the framework and use it efficiently. • SR.09: Modularity - The framework's design will closely follow ROS design philosophy. We expect the framework to be at least modular as ROS and to fit on top of ROS. & modularity Use cases • UC.01: Direct Inverse Kinematics - Sensors, Actuators, Moveable joints, Rotating Joints & Inverse kinematics is a field which concerns itself with correct positioning of the joints in order for end-actuator to reach desired position. As the name indicates, the positioning is worked out from the end-actuator backwards to the first-actuator. • UC.02: Detecting human faces and torso orientation - People in a scene, robot scanning the scene - Image processing is a very important field in robotics engineering. This use case tries to identify the way of doing image processing in robotics, so we can learn how to parallelize common functions. • UC.03: Kalman Filter & A specific robot does sensor fusion and uses kalman filter for prediction - Prediction of the measurements. Kalman filtering is a mathematical approach on how to improve our measures that come from the sensors. • UC.04: Assessing execution time of real-time loop - Real-Time Loops, Function, Time & Since ROS was built with real-time capabilities in mind, measuring real-time loop execution time can give us an insight on performance improvements.

The problem is that robotics engineers lack an intuitive way to utilize multi-cores to lower execution time in the Robot Operating System 2 (ROS) framework. This is due to the complex infrastructure of ROS and the requirement for robotics engineers to learn parallelization from a low-level perspective. The solution is to develop a software system architecture that facilitates the

development of multithreaded applications in ROS2. This framework should assist robotics engineers in developing software that takes advantage of multi-cores to reduce average execution time, provide pre-locking heap memory during initialization, offer a process priority configuration interface, measure heap allocations, latency, jitter, CPU cache usage, and memory fetches, output the best performing executor type configuration and node assignment, advise on which allocators to use, have an easy-to-use user interface, output guidelines for robotics engineers, and accept non-parallelized code and turn it into parallel configuration. The quality requirements include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. The use cases include Direct Inverse Kinematics, detecting human faces and torso orientation, Kalman Filter, and assessing execution time of real-time loop.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (<https://c4model.com/#SystemContextDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Here's a C4 system context diagram for the conceptual solution:



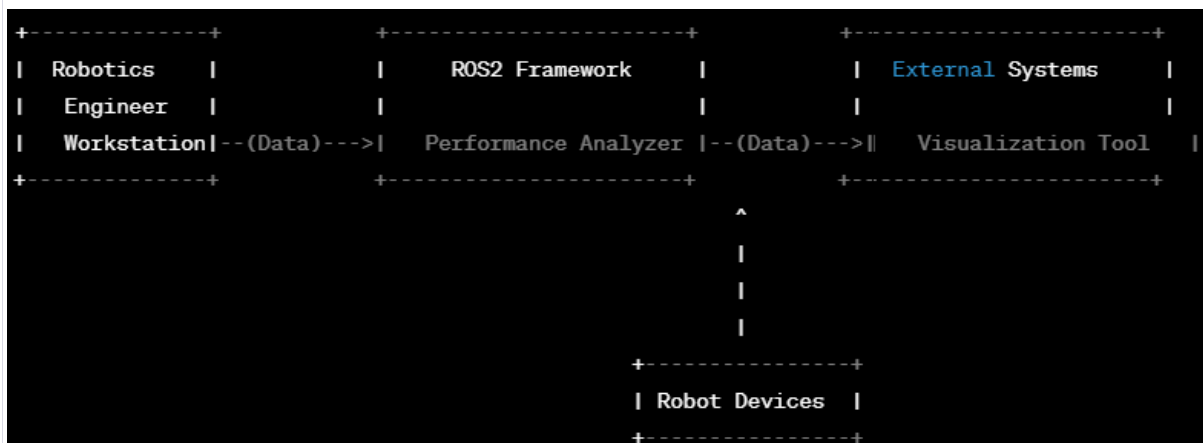
The diagram shows a ROS2 system, which is the focus of the conceptual solution. It communicates with external systems through various data types that are transferred between components. The relevant external stakeholders are Robotics Engineers who use the ROS2 system to develop robotics applications.

The data types transmitted between components include information about the best performing executor type configuration and node assignment, advice on which allocators to use, guidelines for parallelizing code, and measurements of various metrics such as heap allocations, latency, jitter, CPU cache usage, and memory fetches. These data types are communicated between the ROS2 system and external systems, as well as between the ROS2 system and the Robotics Engineers.

Overall, this conceptual solution aims to provide a framework that assists Robotics Engineers in developing software that takes advantage of multi-cores to reduce average execution time in the ROS2 system. It also provides various measurement tools and advice to optimize the performance of the system.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (<https://c4model.com/#ContainerDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Here's a C4 container diagram for the suggested architecture:



The diagram shows the main stakeholder, Robotics Engineers, who use a workstation to interact with the ROS2 Framework. The Framework includes a Performance Analyzer that analyzes the performance of the ROS2 system, and a visualization tool that allows the Robotics Engineers to view the performance metrics. The system communicates with external systems through various data types that are transferred between components.

The quality properties that are important for this system include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. To address these quality properties, the following architectural solutions are suggested:

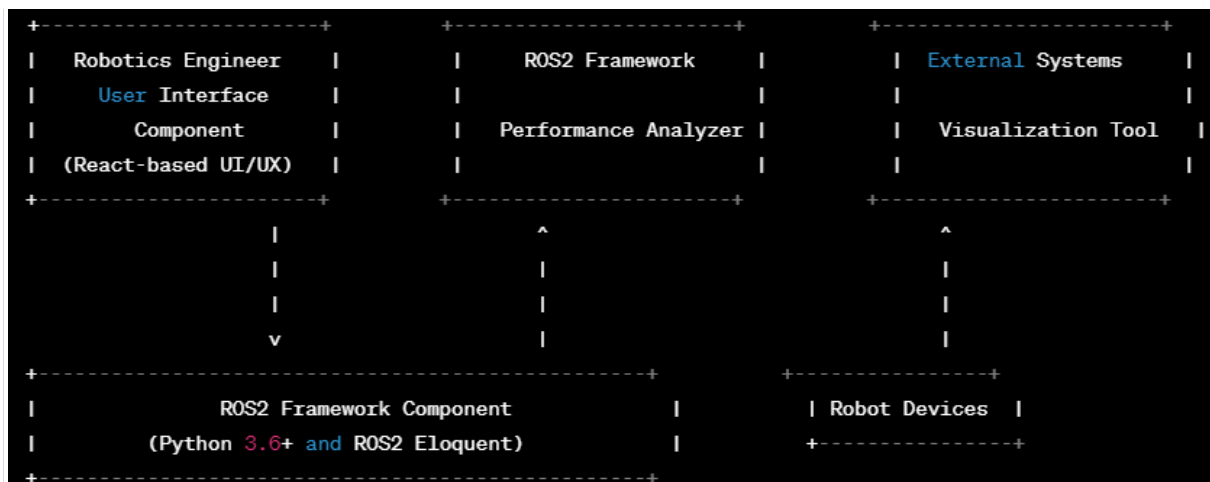
- Architectural style: The system can be designed using a microservices architecture to provide modularity and flexibility.
- Design patterns: The system can utilize the Observer pattern to notify components of changes in the system's performance, and the Adapter pattern to provide a uniform interface for the visualization tool to interact with the ROS2 Framework.
- Design tactics: The system can use tactics such as pre-locking heap memory, process priority configuration, and thread pooling to address the quality properties.

<ul style="list-style-type: none"> Components: The system can be divided into the following components: <ul style="list-style-type: none"> ROS2 Framework: This component includes the ROS2 infrastructure and packages. Performance Analyzer: This component analyzes the performance of the ROS2 system. Visualization Tool: This component provides an easy-to-use interface for Robotics Engineers to view the performance metrics. Robot Devices: These components represent the physical devices that the ROS2 system interacts with.
<ul style="list-style-type: none"> Data types: The data types transferred between components include information about the best performing executor type configuration and node assignment, advice on which allocators to use, guidelines for parallelizing code, and measurements of various metrics such as heap allocations, latency, jitter, CPU cache usage, and memory fetches. Interfaces: The following interfaces are required and provided by the components: <ul style="list-style-type: none"> ROS2 Framework: <ul style="list-style-type: none"> Required interface: Data Types Provided interface: Data Types Performance Analyzer: <ul style="list-style-type: none"> Required interface: Data Types Provided interface: Data Types Visualization Tool: <ul style="list-style-type: none"> Required interface: Data Types Provided interface: Data Types Robot Devices: <ul style="list-style-type: none"> Required interface: Data Types Provided interface: Data Types
<ul style="list-style-type: none"> Anti-patterns to avoid: <ul style="list-style-type: none"> God Object: To prevent the ROS2 Framework from becoming a God Object, it should be divided into smaller, more manageable components. Tight Coupling: To prevent tight coupling between components, the system should utilize loose coupling and messaging protocols to communicate between components.

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (<https://c4model.com/#ComponentDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider following:

- Define how the main stakeholder will use the solution. Add it to the diagram.
- Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it.
- Define data types transferred between components.
- Summarise data types transmitted between components. Add this information to the connections and interfaces between components.
- Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types.
- Document which anti-patterns we need to avoid when implementing this solution.

Here's a C4 component diagram for the suggested architecture:



The diagram shows the main stakeholder, Robotics Engineers, who interact with the system through a User Interface (UI) component that is built using React. The UI component communicates with the ROS2 Framework component, which includes the ROS2 infrastructure and packages, and the Performance Analyzer component, which analyzes the performance of the ROS2 system. The system communicates with external systems through a Visualization Tool component that allows Robotics Engineers to view the performance metrics. The Robot Devices component represents the physical devices that the ROS2 system interacts with.

The quality properties that are important for this system include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. To address these quality properties, the following architectural solutions are suggested:

- Architectural style: The system can be designed using a microservices architecture to provide modularity and flexibility.
- Design patterns: The system can utilize the Observer pattern to notify components of changes in the system's performance, and the Adapter pattern to provide a uniform interface for the Visualization Tool component to interact with the ROS2 Framework component.
- Design tactics: The system can use tactics such as pre-locking heap memory, process priority configuration, and thread pooling to address the quality properties.
- Components: The system can be divided into the following components:
 - Robotics Engineer User Interface: This component provides an easy-to-use interface for Robotics Engineers to interact with the system and view the performance metrics.
 - ROS2 Framework Component: This component includes the ROS2 infrastructure and packages.
 - Performance Analyzer Component: This component analyzes the performance of the ROS2 system.
 - Visualization Tool Component: This component provides an easy-to-use interface for Robotics Engineers to view the performance metrics.
 - Robot Devices Component: These components represent the physical devices that the ROS2 system interacts with.

- Data types: The data types transferred between components include information about the best performing executor type configuration and node assignment, advice on which allocators to use, guidelines for parallelizing code, and measurements of various metrics such as heap allocations, latency, jitter, CPU cache usage, and memory fetches.
- Interfaces: The following interfaces are required and provided by the components:
 - Robotics Engineer User Interface:
 - Required interface: Data Types
 - Provided interface: Data Types
 - ROS2 Framework Component:
 - Required interface: Data Types
 - Provided interface: Data Types
 - Performance Analyzer Component:
 - Required interface: Data Types
 - Provided interface: Data Types
 - Visualization Tool Component:
 - Required interface: Data Types
 - Provided interface: Data Types
 - Robot Devices Component:
 - Required interface: Data Types
 - Provided interface: Data Types
- Anti-patterns to avoid:
 - God Object: To prevent

5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

To implement the suggested solution, we can use a combination of existing and new tools. Here is a possible toolchain:

1. ROS2 Eloquent: We can use the latest version of ROS2 as the foundation of the system.
2. Python 3.6+: We can use Python as the primary programming language for the ROS2 Framework and Performance Analyzer components.
3. React: We can use React to build the Robotics Engineer User Interface component.
4. Docker: We can use Docker to containerize the components and simplify deployment.
5. Prometheus: We can use Prometheus as a monitoring and alerting tool for the system.

The data types transferred between components include information about the best performing executor type configuration and node assignment, advice on which allocators to use, guidelines for parallelizing code, and measurements of various metrics such as heap allocations, latency, jitter, CPU cache usage, and memory fetches.

The following summarizes the data types transmitted between components:

- Robotics Engineer User Interface:
 - Required data types: Executor type configuration, Node assignment, Guidelines, Performance metrics

	<ul style="list-style-type: none"> • Provided data types: Executor type configuration, Node assignment, Guidelines, Performance metrics
• ROS2 Framework Component:	<ul style="list-style-type: none"> • Required data types: Executor type configuration, Node assignment, Guidelines • Provided data types: Executor type configuration, Node assignment, Guidelines, Performance metrics
• Performance Analyzer Component:	<ul style="list-style-type: none"> • Required data types: Performance metrics • Provided data types: Performance metrics
• Visualization Tool Component:	<ul style="list-style-type: none"> • Required data types: Performance metrics • Provided data types: Performance metrics
• Robot Devices Component:	<ul style="list-style-type: none"> • Required data types: ROS2 messages • Provided data types: ROS2 messages

The interfaces between components are defined as follows:

• Robotics Engineer User Interface:	<ul style="list-style-type: none"> • Required interface: ROS2 Framework Component, Performance Analyzer Component, Visualization Tool Component • Provided interface: ROS2 Framework Component, Performance Analyzer Component, Visualization Tool Component
• ROS2 Framework Component:	<ul style="list-style-type: none"> • Required interface: Robotics Engineer User Interface, Performance Analyzer Component • Provided interface: Robotics Engineer User Interface, Performance Analyzer Component
• Performance Analyzer Component:	<ul style="list-style-type: none"> • Required interface: ROS2 Framework Component, Robotics Engineer User Interface • Provided interface: ROS2 Framework Component, Robotics Engineer User Interface, Visualization Tool Component
• Visualization Tool Component:	<ul style="list-style-type: none"> • Required interface: Performance Analyzer Component, Robotics Engineer User Interface • Provided interface: Performance Analyzer Component, Robotics Engineer User Interface
• Robot Devices Component:	<ul style="list-style-type: none"> • Required interface: ROS2 Framework Component • Provided interface: ROS2 Framework Component