Robot Operating System - An Introduction

Maruthi S. Inukonda

18st Apr 2019

License

Agenda

- Robot Operating System (ROS)
- Architecture
- Installation, Packages
- Nodes, Messages, Topics
- Publisher and Subscriber
- Server and Client
- Parameter Server and Parameters
- Bag file, Recording, Playback
- Launch files, Launching
- Demo

Introduction

What is ROS? (1/2)

- Robot Operating System (ROS).
- Inspired by PR1 Robot project at STanford AI Robot (STAIR).
- Started by Willow Garage, a Robotics Research Lab, in 2007.
- Used in PR2 Robot by Willow Garage.
- Open sourced under BSD License for wider adoption, contribution.
- Now maintained by Open Source Robotics Foundation (OSRF).
- Used in Personal and Industrial Robots, Drones, Autonomous Cars.

PR1 at Stanford

ADV at Univ of Texas

ADV at Georgia

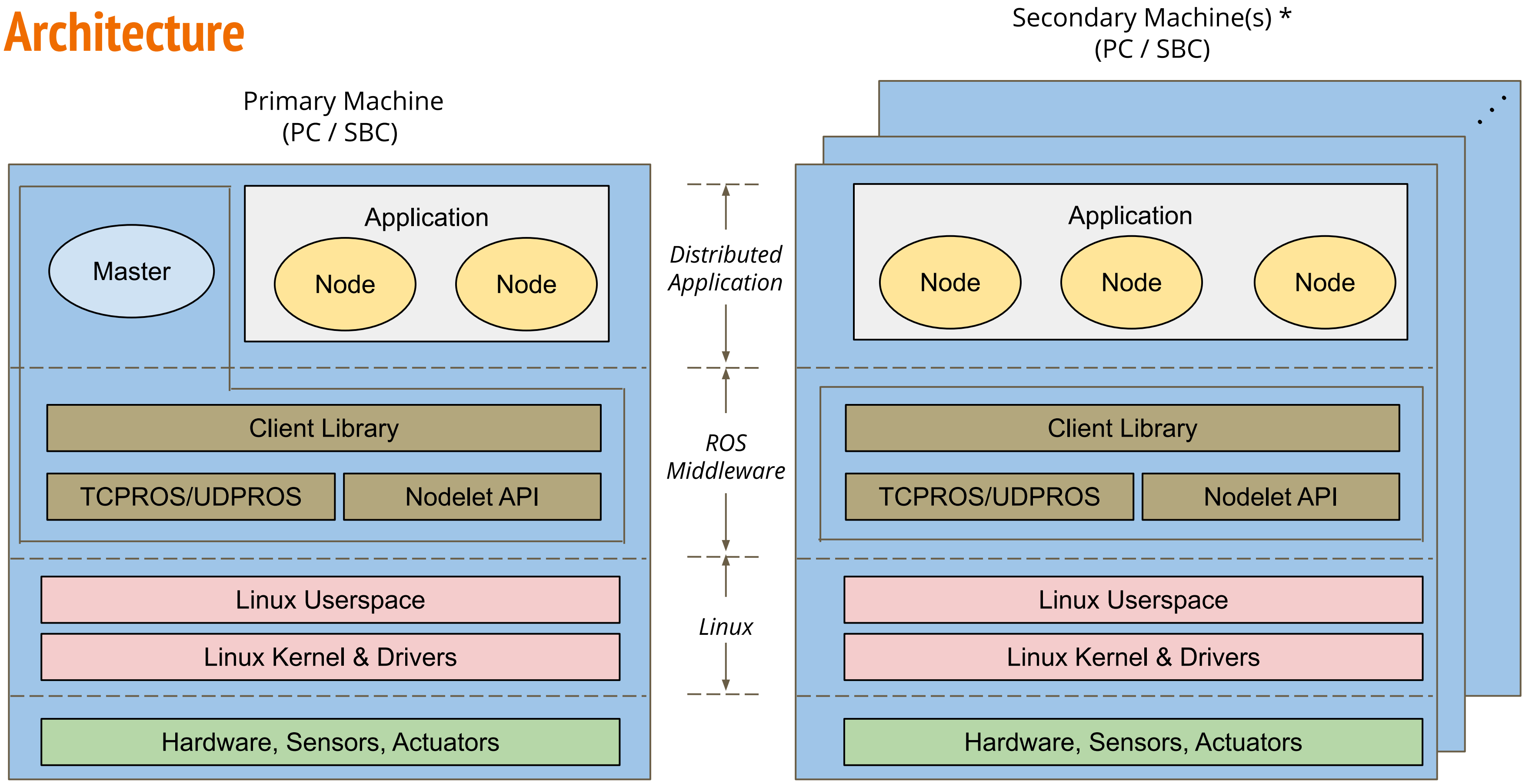
ADV at Stanford

What is ROS? (2/2)

- Not a traditional operating system.
- Is a middleware that runs on Linux/Windows/MacOS.
  - Sits between OS and Application
- Is a heterogeneous distributed computer cluster.
  - Sensors, Micro-controllers, System-on-Chip, Workstations, Android devices.
- Versions
  - ROS 1.x : Stable well maintained.
  - ROS 2.0 : Under heavy development.
- Recent releases
  - Lunar for Ubuntu 14
  - Kinetic for Ubuntu 16
  - Melodic for Ubuntu 18, Debian, MacOS

Architecture

Architecture



* In a simple All-In-One setup, secondary machines do not exist. All nodes run on the primary machine.

ROS Features & Concepts

- Features
  - Distributed or All-In-One.
  - Asynchronous multicast simplex communication (Publisher - Subscriber model)
  - Synchronous unicast full duplex communication (Client - Server model)
  - APIs in C++, Python, Lisp. (Java on Android).
  - Hard real-time system from version 2.0.
- Primary concepts
  - Machines
  - Packages
  - Master
  - Nodes (Publisher, Subscriber, Server, Client)
  - Topics
  - Messages
  - Parameter Server
  - Bags

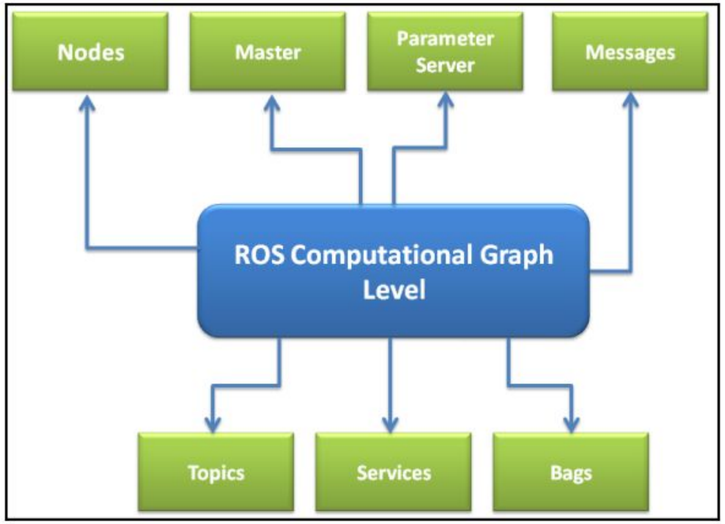


Image: Mastering ROS, Lentin Joseph.

http://wiki.ros.org/ROS/Concepts

Installation, Packages

Installation

- Depending on your Ubuntu Release, choose ROS release
  - ROS Kinetic for Ubuntu 16 (Xenial)
  - ROS Melodic for Ubuntu 18 (Bionic)
- Follow the instructions given in the Install Guide. *
- Verify the installation

```
$ apt list --installed | grep ros
```

- In case of multi-machine setup, install required packages on primary and secondary machines.

* http://wiki.ros.org/ROS/Installation

Packages

- Package is a reusable software module.
  - Packages can contain
    - Nodes
    - ROS-independent library
    - Configuration files
    - Third-party software
  - Each package’s name is in <string> convention.

- Load ROS environment into your shell.

```
$ source /opt/ros/<ros-release>/setup.bash
```

- List packages installed

```
$ rospack list
```

http://wiki.ros.org/Packages

Core Framework

Core Framework

- ROS Core is the middleware running in the primary machine.
    - Uses TCP/UDP for communication on a specific/given port no.
    - Monitors health of nodes on all machines
- It contains
    - A ROS master
    - A Parameter server
    - A Logging node
    - Client Library on all machines

http://wiki.ros.org/roscore

Bringing up ROS Core Framework

- Load ROS environment into your shell on the primary machine

```
$ source /opt/ros/<ros-release>/setup.bash
```

- Set Master node’s URI [only required for distributed setup]

```
$ export ROS_MASTER_URI=http://<master-ipaddress>:<portno>
```

  11311 is default port number.

- Start ROS core framework in one terminal on the primary machine.

```
$ roscore [ -p <portno> ]
or
$ roslaunch --core [ -p <portno> ]
```

  This prints logging path and brings up ROS Master, Parameter Server, Logging Node (`/rosout`).

- Logging done by all nodes is captured in log files under `/home/<username>/.ros/log/<run_id>/`

Nodes, Topics, Messages

Node

- Node is the smallest runnable unit of robotics software.
    - Helps in plug-and-play of application software.
    - Can seamlessly work on distributed or all-in-one setups.
    - Each node's name is in <string> convention.
- Types of Nodes
    - Publisher
    - Subscriber
    - Server
    - Client
    - Parameter Server

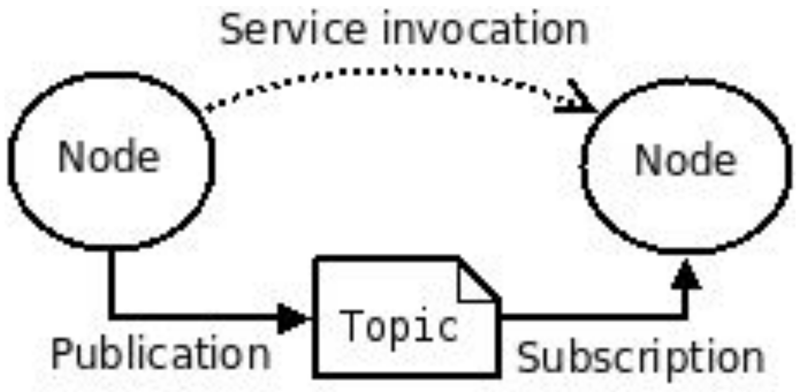


Image: Mastering ROS, Lentin Joseph.

http://wiki.ros.org/Nodes

Bringing up a node

- Load ROS environment into your shell

```
$ source /opt/ros/<ros-release>/setup.bash
```

- Set Master node’s URI [only required for distributed setup]

```
$ export ROS_MASTER_URI=http://<master-ipaddress>:<portno>
```

  11311 is default port number.

- Start a ROS node.

```
$ rosrun <packagename> <nodename> [ <args> ]
```

- List the nodes

```
$ rosnode list
```

Topic

- Topic is named channel in which nodes communicate messages.
    - Topics have anonymous publish/subscribe semantics.
    - Topics are intended for unidirectional, asynchronous streaming communication.
    - Each topic's name is in /<string> convention.
- Topics could be nested to prevent name clashes.
    - When more than one node need to publish topics from same package.
    - Eg More than one camera sending feed.
    - Each topic's name is in /<namespace>/<string> convention.

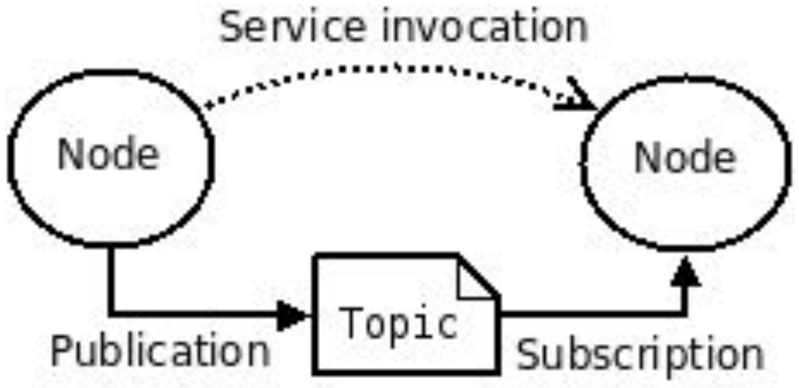


Image: Mastering ROS, Lentin Joseph.

http://wiki.ros.org/Topics

Message

- Nodes communicate with each other by publishing messages to topics.
    - A simple data structure, comprising typed fields.
    - Supported types
        - Integer, Floating point, Boolean, Strings.
        - Arrays, Structures.
    - Each message's name is in <string> convention.
    - A message need to be fully qualified with its package name. Eg <package>/<string>

http://wiki.ros.org/Messages

Demo - Bringing up USB Camera nodes

- ROS has ready-made publisher/subscriber nodes for UVC standard camera.
- Load ROS environment into your shell

```
$ source /opt/ros/<ros-release>/setup.bash
```

- Start publisher node `uvc_camera_node`

```
$ rosrun uvc_camera uvc_camera_node
```

- List the nodes, topics & messages

```
$ rosnode list
$ rostopic list
$ rosmsg list | grep -i image
```

- Start subscriber node `image_view`.

```
$ rosrun image_view image_view image:=/image_raw
```

- List the nodes

```
$ rosnode list
```

Demo - Bringing up OpenCV nodes

- ROS has ready-made OpenCV publisher/subscriber nodes for all cameras.
- Load ROS environment into your shell

```
$ source /opt/ros/<ros-release>/setup.bash
```

- Start publisher node `cv_camera_node*`

```
$ rosrun cv_camera cv_camera_node
```

- List the nodes, topics & messages

```
$ rosnode list
$ rostopic list
$ rosmsg list | grep -i image
```

- Start subscriber node `image_view`

```
$ rosrun image_view image_view image:=/cv_camera/image_raw
```

- List the nodes

```
$ rosnode list
```

* Custom subscriber nodes using OpenCV can easily interoperate with the cv_camera_node publisher.

Parameter Server & Parameters

Parameter Server

- Parameter server is a shared, multivariate dictionary that is accessible via network APIs.
  - Used for maintaining small amounts of state.
  - Uses XMLRPC format for communication
  - Implemented inside the ROS master.
- Nodes use this server to store and retrieve parameters at runtime.
  - Uses YAML format for set/get.
- Parameter
  - Each parameter is a key-value pair.
  - Each parameter's name <paramname> is in <string> convention
  - Named using ROS naming hierarchy to avoid name clashes
    /<topicname>/<paramname>

http://wiki.ros.org/Parameter%20Server

Retrieving and Storing Parameters

- Load ROS environment & Set Master node's URI
- List all parameters stored in parameter server.

```
$ rosparam list
```

- Get a parameter.

```
$ rosparam get { <parametername> | <node> }
```

- Set a parameter value.

```
$ rosparam set { <parametername> | <node> } <value>
```

- Get all parameters and values.

```
$ rosparam get /
```

- Dump all parameters and values.

```
$ rosparam dump
```

Bag file, Recording & Playback

Bag file

- A file capturing time ordered messages from all/interested topics.
  - Used for recording messages from many publishers (sensors) in file(s).
  - The bag file(s) can be played back later with same time synchronization without having actual publishers.

- Two methods to record/playback bag files
  - "`rosbag record`" and "`rosbag play`" commands
  - Rosbag APIs available in C++, Python.

http://wiki.ros.org/rosbag

Recording to a bag file

- Load ROS environment & Set Master node’s URI.
- Start all the required publisher nodes.
- Start recording messages from all/interested topics.

```
$ rosbag record { [TOPIC] … } <bagfile>
```

- List topics in a bag files

```
$ rosbag info <bagfile>
```

Playing back from a bag file

- Load ROS environment & Set Master node's URI
- Start all the required subscriber nodes
- Start playback of messages to all recorded topics.

```
$ rosbag play <bagfile1> [<bagfile2> …]
```

- Start playback of messages to interested topics.

```
$ rosbag play [ { --topic <topic> } ] … <bagfile> [<bagfile> …]
```

- List topics in a bag file

```
$ rosbag info <bagfile>
```

Demo - Recording/Playing multiple sensors

- Load ROS environment & Set Master node's URI
- Recording
  - Start publisher nodes for camera 0 and 1 *
    ```
    $ rosrun cv_camera cv_camera_node _device_id:=0 __name:=cam0
    $ rosrun cv_camera cv_camera_node _device_id:=1 __name:=cam1
    ```
  - Start recording messages from all topics.
    ```
    $ rosbag record -O cvcam2.bag --dur 2 /cam0/image_raw /cam1/image_raw
    ```
- Playback
  - Start subscriber nodes `image_view`
    ```
    $ rosrun image_view image_view image:=/cam0/image_raw
    $ rosrun image_view image_view image:=/cam1/image_raw
    ```
  - Start playback of messages from the bag file.
    ```
    $ rosbag play <bagfile>
    ```

* Each camera device on Linux has unique name and minor (device) no. /dev/video0, /dev/video1

Launching, Launch files

Launching

- A method to easily launch/stop master and batch of nodes.
- All nodes and the master launched using launcher could be stopped as a batch.

- Nodes could be local or remove machine (via ssh).
- A launch file enlisting nodes, is used for specifying all inputs required for the nodes.
- Parameters required for calibrating the sensors could be saved in the file.
  - These parameters are set in the Parameter Server
- Re-spawning of nodes could be done by the launcher.

Launch file

- A configuration file enlisting nodes, used for starting/stopping as batch.
  - A file in XML format
    ```
    <launch>
        <node attr=val >
            <param attr=val />
        </node>
    </launch>
    ```
  - Can respawn the nodes if they quit. Use `respawn` attribute of `<node>` tag.
  - Parameters required for each node could be saved in `<param>` tag.

- For remote nodes
  - Create <machine> tag as a sibling to <node>
    ```
    <machine name=”mac_name” address=”ip_or_fqdn”
    env-loader="path_to_ros_env" user="someone"/>
    ```
  - Use `machine=”mac_name"` attribute of `<node>` tag.

http://wiki.ros.org/roslaunch/XML/node http://wiki.ros.org/roslaunch/XML/machine

roslaunch

- Load ROS environment & Set Master node’s URI
- Create launch file with machines, nodes, and parameters for each nodes.
- To launch a batch of nodes

```
$ roslaunch <lauchfile>
```

http://wiki.ros.org/roslaunch

Demo - Launching nodes

- Load ROS environment & Set Master node's URI
- Create launch files once.
  - Create one file with publisher nodes for camera 0 and 1 *

    ```
    $ vi cvcam2_pub.launch
    ```
  - Create one file with subscriber nodes `image_view`

    ```
    $ vi cvcam2_sub.launch
    ```

- Launch the nodes
  - Start publisher nodes for camera 0 and 1

    ```
    $ roslaunch cvcam2_pub.launch
    ```
  - Start subscriber nodes `image_view`

    ```
    $ roslaunch cvcam2_sub.launch
    ```

References

References

- ROS Installation
- ROS Concepts
- Mastering ROS, Lentin Joseph, Pact Publishing.

Q & A