Distributed ROS platform

1. Goal

In this work we are trying to clarify the best way to run ROS-base program and packages over a distributed hardware(HW) platform.
The first step is to know the ROS system and architecture by itself. Then go through the ROS solution by itself. Because every system and well developed system already faced with these problems and they have developed techniques for that. Then later we will see if that applies to our need and what more should be added. If you already know the ROS and its working structure, you can start from solution section.

2. What is ROS?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.[http://wiki.ros.org/ROS/Introduction]
In simple way, it has a variety of low and high level software package that full fill most of your necessities regarding to your HW platform to run the package and also vast instrument and electronic devices to develop a robot or anything likely.
According to our goal, message passing between processes and the package management would be more important. ROS has its own ways to communicate between different ROS working packages.

3. ROS modular structure

Ros has a good modular structure. ROS implements several different styles of communication, including synchronous RPC-style communication over

services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. [http://wiki.ros.org/ROS/Introduction]
Briefly, a running ROS system is formed by roscore and other ros-nodes as executables. Roscore is the main package manager of whole system. Therefore it should be run first. Then any other service or program can be run as a node under the same OS or any other OS that is working under the connected network to OS with running roscore. In other words, roscore functions as a main manager and broker for rosnodes. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. they can also provide or use a Service. [http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes]

As it was mentioned above, nodes can interact in two ways with each other. First with publish a topic to provide a data or subscribe a topic to receive it.
Secondly they can collaborate via services. In the next sections, those will be explained in more detail. But first it explain how to set up the roscore as main.

4. Setting up the roscore
It assumes that you have already installed the ROS according to ros.org .Every OS that runs will get an IP, even though it wouldn't be connected to any network. Since the ROS works more under Unix_like OS, the instructions will be accordingly. To set up the roscore first the IP of system should be discovered by

$ifconfig

You should use the IP of network HW that you will likely use it as your ROS network. Then set the following parameters of your bash

$export ROS_HOSTNAME=<system-IP>

$export ROS_MASTER_URI=http://<system-IP>

$source /opt/ros/kinetick/setup

Since the kinetic ros package is installed, the source goes through “kinetick” name. else you are using other packages, you should type its name.

Then the roscore can be run simply as below

$roscore

If it runs without any errors, you can step forward to run any existing package nodes in your system. Every packages that publish something will create a topic. *rqt_graph* can show a graphical view of publishers, subscribers, and topics between. *rostopic* is also a good application to find out useful data about topics in a ROS network. You can find more information in ROS tutorial of “understanding ROS Topics”.

[http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics]

5. ROS-nodes Interaction methods

5.1. Services:

As it was mentioned before, the service is kind of Remote Call Procedure pattern. Nodes can send request to existing services and receive the response. Nodes can also send some parameters along with request and receive regarding response from services.

Rosservice has different commands and services should be attached to call.

5.2. Parameters:

rosparam allows you to store and manipulate data on the ROS Parameter Server. The Parameter Server can store integers, floats, boolean, dictionaries, and lists. rosparam uses the YAML markup language for syntax.[http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams]

To create ros message and services please go to ros tutorials as below link

http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv

5.3. Topic:

Another way that Ros nodes can communicate is publishing/subscribing to one or more topics. Every ros node use client libraries to enable this. It is an asynchronous communication. It means the data sender or receiver never block each others progress.

If you need more detailed information please refer to the link below:

http://wiki.ros.org/Topics

6. solution

To address our main goal ros topic will help very well. The main idea is that different running packages can be distributed over different hardware platform to work remotely or reducing the computation load in main computer. To run any ros package over a system, first of all the roscore should be run. Then it will manage all other packages that shall work as a united system. According to **roscore set up** section, the roscore should be run in desired HW part. Then to run a package in another HW or OS, you should setup the bash that is going to run the ROS node.

$export ROS_HOSTNAME=<system-IP that remote ROS node will run in>

$export ROS_MASTER_URI=http://<system-IP that roscore is running in>

$source /opt/ros/kinetick/setup (catkin_ws/devel/setup.bash) // it depends where your package exists

This will set the master's URI according to the IP that roscore is running. Then automatically the published or subscribed topic will be sent and requested from master URI. Indeed when a node uses the topics, it doesn't matter that who is going to use it and where it is. The node just need to know where the master(roscore) there is. Also the subscriber just concern where the master is.

7. Practical Test

To prove this we run the navigation package that was developed by other team over an **Odroid XU4**. the roscore was running in the main computer of robot. The Odroid was connected to that, through a private network(LAN). the ROS_MASTER_URI was the IP of main computer in private network and the ROS_HOSTNAME was the IP of Odroid in this network.

Firstly, the roscore was run in the main PC, then it runs the navigation package in the host OS in Odroid. We check different computing load as well to see how far the navigation package can works. We change some parameters to increase the computation load ad it works fine.

If other packages and nodes run in remote machines, it is important to know that they are healthy. The first parameter to check would be the CPU load. The next section will explain how it is done by this work.

Consider a remote OS is running one or more ros nodes. The performance of code execution and response time of nodes is highly depended on host resources. Also the nodes computation load may vary according to function statuses. It's needed to monitor the remote host OS to discover the critical time. Then the developer can define its own scenario in case of any remote resource restriction.

8. Remote Host CPU load Monitoring

To watch our remote hardware resources there are some ROS packages.

rqt_top can display the resource usage of the local mochain. It needs graphical interface too.

The *arni* package is developed for remote monitoring. It has a good structure and good information. In the below link you can find more information about

http://wiki.ros.org/arni

These packages working fine. I didn't find the alarm and warning level setting so user friendly to set alarm level and publish it. Also it should set statistics and do more effort to get alarm message when it happens. Indeed, the current alarm and error architecture has its own strength. It works well but here I preferred to develop a lightweight CPU load publisher package.

There are clean notes to edit which parameter in code to set alarm level and time hysteresis loop. The CPU load that published here, is based on Odroid CPU architecture. Odroid x4u CPU has 8 core. We get the average

CPU usage from /proc/stat file and publish it.Alarm level is according to CPU load percentage.

If this average exceeds the ALARMLEVEL for defined number(ALARMNSET) of cyclic check, it will publish the alarm message. If the CPU average load goes under the alarm level and stay for a defined number(ALARMNRSET) of cyclic check, it will stop publishing alarm message.

The *cpu_load* package that is developed by this work, is a minimal work to monitor the host simply.

Repository to find package source.

https://github.com/katigari/cpu_load.git,

9. Future work

Add codes to monitor used and free RAM space.

Extend the CPU usage to show which nodes are running in which core and publish the core load within message.