Chapter 7

# On-Board High-Performance Computing For Multi-Robot Aerial Systems

Leonardo Camargo Forero, Pablo Royo and Xavier Prats

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.69443

**Abstract**

With advancements in low-energy-consumption multi/many core embedded-computing devices, a logical transition for robotic systems is *Supercomputing*, formally known as high performance computing (HPC), a tool currently used for solving the most complex problems for humankind such as the origin of the universe, the finding of deceases' cures, etc. As such, HPC has always been focused on scientific inquires. However, its scope can be widening up to include missions carried out with robots. Since a robot could be embedded with computing devices, a set of robots could be set as a cluster of computers, the most reliable HPC infrastructure. The advantages of setting up such an infrastructure are many, from speeding up on-board computation up to providing a multi-robot system with robustness, scalability, user transparency, etc., all key features in supercomputing. This chapter presents a middleware technology for the enabling of high performance computing in multi-robot systems, in particular for aerial robots. The technology can be used for the automatic deployment of cluster computing in multi-robot systems, the utilization of standard HPC technologies, and the development of HPC applications in multiple fields such as precision agriculture, military, civilian, search and rescue, etc.

**Keywords:** high performance computing (HPC), cluster of computers, multi-robot systems, HPC-ROS, ubiquitous supercomputing, unmanned aerial system (UAS), unmanned aerial vehicle (UAV), HPC cluster of robots

## 1. Introduction

High performance computing (HPC) can be thought as a tool for allowing or speeding up the solving of problems that either cannot be solved in a regular desktop computer or it would take too much time to do it. Within the scope of robotics, for example in missions carried

with unmanned aerial systems (UASs), HPC approaches and technologies could be used for distributed on-board computing, in order to maximize efficiency.

Among many existing HPC infrastructures, a cluster of computers is perhaps the most common. The idea consists on integrating multiple computing units (e.g., CPUs, GPUs, desktop computers, servers, etc.) into a single platform where software can use the entire resources acting as one, therefore, facilitating a timely execution. To do so, the software must be written with specific parallel or distributed software libraries in order to exploit the resources among which it is being executed. For example, parallel software on-board robots could exploit embedded CPU cards with multiple cores, etc., even among independent robots acting as a cluster of computers.

To sustain the objective of speeding up computation, given the fact that a cluster of computers is build up out of multiple independent computing units, certain features must be provided by the platform itself. Such features are scalability, failure tolerance, robustness, reliability, and user transparency. For the rest of this document, these features are referred as HPC features.

Scalability refers to the idea of adding or subtracting computing units to the cluster (nodes), without compromising the entire system. In the context of robotic systems performing real-world missions, scalability can be used to adapt the mission objectives to the available robots, providing the capability of adding or subtracting new robots if needed.

Failure tolerance refers to the capacity of recovering in case of failures such as those occurring in nodes, software, etc. Given the changing environment conditions, in which real-world robotic missions are carried out, different sources of failures are expected. Tolerance to such failures, as in load redistribution, for example, is a much desired feature for robotic systems.

Robustness refers to the capacity of adaptation to possible errors that might occur during software execution, where the probability of such occurrence is higher, in comparison with sequential (in a single computing unit) execution.

Reliability relates to the results obtained by a software execution. This is, software output, obtained through distributed/parallel approaches must be the same as it would be in a sequential approach. Both robustness and reliability are much desired for software embedded on robotic systems.

User transparency refers to the fact that the user interacting with the HPC infrastructure does not need to understand the complexity necessary to set a single computing facility built up of distributed independent computing units. This offers a whole set of options to robotic systems. For example, a parallel application could be developed to pilot all robots at once.

Furthermore, time efficiency and task distribution can also be considered as HPC features.

The HPC features are provided by the platform itself and by parallel software libraries such as message passing interface (MPI) [1]. The platform is depicted as a set of specific software layers with particular functions and multiple open-source and privative software solutions (for each layer)–HPC software layers. These layers are operating system, user system/file system, batch system, and applications.

http://dx.doi.org/10.5772/intechopen.69443

**Figure 1** depicts the organization of the HPC software layers in a traditional cluster of computers (a) and the proposal for replication of such layers in a cluster of computers built with robots (b), in this case UASs, as presented in the following sections.

As it can be seen in **Figure 1**, nodes in an HPC cluster can be classified as master and slaves. The master node represents the machine in charge of managing the entire infrastructure, monitor it, control it, and serving as an access point for the users. The slaves nodes are the ones in charge of performing computation.

Theoretically, any type of computing unit could potentially be part of an HPC cluster of computers, provided that such software layers can be deployed over it. Such idea extends to robots.

While, nowadays, the computing capability of a common HPC node far exceeds those of a robot, the idea of splitting a software task and performing a parallel/distributed approach holds. Moreover, the tasks being performed by a robot are not as complex as it would be a task performed by a common HPC node.

And since an operating system can be installed in computing cards that can be implemented on robotic infrastructures, theoretically, an HPC cluster of computers could be deployed with multiple robots.

Attempts to do so have been done in the past. Holland et al. [2] proposed the UltraSwarm system, with the objective of providing an unmanned aerial vehicle (UAV) swarm with a single controlling intelligence based on HPC.

The UltraSwarm system is a combination of swarm intelligence and wireless cluster computing based on Linux and Bluetooth communications. However, the authors did not implement the HPC cluster computing software layers, which ultimately leads to not being able of offering user transparency, scalability, failure tolerance, robustness and reliability, in the sense of classical cluster computing.

Marjovi et al. [3] proposed the concept of robotic cluster and defined as: A robotic cluster is a group of individual robots, which are able to share their processing resources among the group in order to quickly solve computationally hard problems.

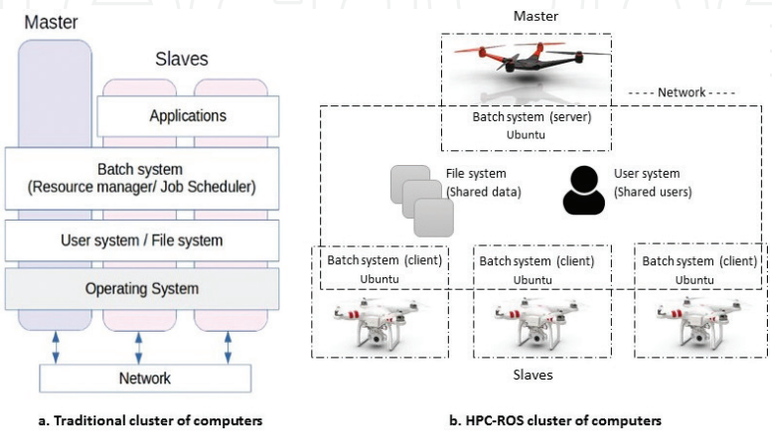


**Figure 1.** HPC cluster of computers software layers.

The authors implemented a software platform that resembles the one here presented, specifically to what they called the middleware layer. In their case, the HPC libraries MPI, which allows software to run in parallel and offers certain features to handle communication among distributed processes.

As for the concept of middleware, as a software layer, between two software layers (for this case: the operating system and the parallel applications layer) and with the objective of seeing the cluster's computing units as a single one, it is valid to consider MPI as such. However, it is perhaps more adequate to consider the batch system (**Figure 1**) as the HPC cluster middleware and allocate MPI libraries to the applications layer, as it is, in the traditional concept of an HPC cluster of computers. Moreover, the authors Marjovi et al. [3] did not implement the user system/file system layer, which is the one providing an HPC cluster with the ability of replicating data among the independent computing units and to set a multipurpose/multiuser infrastructure.

Moreover, a single robot (e.g., an UAS) could be considered as an HPC infrastructure in itself. Multiple and different types of computing boards (e.g., CPUs, GPUs, FPGAs) could be embedded on it, giving the robot the ability of locally processing data and even doing it using parallelism (e.g., open MP [4], open MPI [5] or compute unified device architecture–CUDA [6]). This could reduce the dependency on data links, favoring therefore mission real-time decisions.

Salamí et al. [7] implemented two applications: a parallel hot spot detection algorithm and a jellyfish detection algorithm on board unmanned aerial system (UAS) for fast response using and comparing three different computing cards (EPIA N700-15 VIA C7-1.5, Pandaboard with OMAP 4430 integrated CPU, INTEL T7500 on GENE-9655 motherboard and TilenCore-Gx board with TILE-Gx36). The results showed that performing on-board parallel computing provides faster response time for critical missions.

Multiple companies provide nowadays multi-core light-weight low-energy-consumption computing cards for embedded robotic systems (e.g., Intel, NVIDIA, Texas Instruments, Raspberry PI Foundation, Qualcomm, Parallela, etc.), ranging from dual core CPU cards like the Intel Edison architecture [8] up to the 256 NVIDIA CUDA cores Jetson TX1 [9].

With advancements in energy consumption decreasing, GPUs represent an important technology for robotic-embedded parallel computing. Jeon et al. [10] presented a real-time vision-guided control algorithm (detection and tracking) for an UAV using CUDA-enabled GPU cards. Cocchioni et al. [11] also showed the implementation of visual-based landing for an unmanned quadrotor using a NVIDIA Tegra Jetson K1.

Furthermore, a cluster of computers can be set up in a single robot as well, as Ribeiro et al. [12] proposed by designing a robotic flying crane with an embedded cluster of computers composed of ODROID-x2 processors [13], Linux Ubuntu and sun grid engine (SGE) [14] as batch system.

Considering multiple robots, an HPC cluster of computers could be set up among them. However, most studies on multi-robot systems are based on swarming approaches focusing

http://dx.doi.org/10.5772/intechopen.69443

mainly on collision avoidance, motion coordination, flying formation, etc., all applications that could be allocated to the applications layer of the HPC cluster of computers platform.

Multiple research has been done using robot operating system (ROS) [15] for robot swarming [16–22] but not setting up an HPC cluster of computers (HPC software layers). It is our belief that by setting up a cluster of computers with a multi-robot (e.g., UAS) system (*HPC cluster of robots*), several advantages appear in comparison with using a single UAS. The following lists such advantages, relating them with the HPC features (S = scalability, FT = failure tolerance, RO = robustness, RE = reliability, UT = user transparency, TE = time efficiency, TD = task distribution) and the HPC software layers:

1. Cooperative/distributed on-board mission execution (TE, TD) (e.g., mission on-board software). Some mission examples are tracking and target recognition, vegetation health analysis, hot-spot detection and extinction, surveillance and intelligence, etc. This is managed by the applications and the batch system layers.
2. Distribution of mission area to multiple UASs (i.e., each UAS is assigned with a mission subarea) (TE, TD). This is managed by the applications and the batch system layers.
3. Single access point (UT). The master node, in a cluster of computers, acts as user frontend. In this sense, once a user is logged in, the master can access the entire infrastructure, as it would be a single computing unit made up of multiple computing units. For example, a parallel multi-UAS autopilot could be developed or even a pilot could control the entire system by controlling only the master.
4. Multipurpose/multiuser systems. A cluster of computers is designed to allow multiple users execute multiple purposes software and doing it so efficiently. This is one of the specific objectives of the batch system and its interaction with the user and file system.
5. Lower system implementation costs (S, FT). Deploying a single UAS with high computing power will require higher costs that several UASs with little computing power acting as a cluster of computers.
6. No single point of failure (FT). In case of a single UAS, a failure could compromise the entire system.
7. Node addition in case of need (S). This is managed by the batch system.
8. Workload reassignment in case of software failure, node failure, or even robot destruction (intentionally or unintentionally) (S, FT, RO). This is managed by the batch system.

High performance computing has been used, traditionally, as a tool for solving the most complex problems known to humankind. However, recently, the emergence of the ideas behind big data expands HPC scope from sciences to commercial applications. The infrastructure underneath big data software is in fact a supercomputer with similar layers as the HPC software layers. The same applies for cloud computing, grid computing, and opportunistic computing. In this sense, supercomputing is everywhere, except in robots yet.

With regard to the proposed definition of robotic cluster [3], it is clear that the driver of implementing HPC infrastructures remains as the capacity of quickly processing computationally hard problems (computing efficiency), according to the authors. In fact, it is fair to say that HPC has always been targeted as a tool for such purpose.

However, in this work, it is proposed to put such main driver aside and focus on the HPC features used to provide it. By exploiting such features, the potential of developing and creating complex applications is enormous. Applications running over multiple independent computing units, acting as one (an abstraction of user transparency) and enjoying high computing efficiency.

To do so, it is proposed the development of an HPC enabler, a platform middleware technology to bring HPC features everywhere—*Ubiquitous supercomputing*. Not only to common HPC cluster nodes or robots but also to any device capable of supporting it.

Such enabler should allow (HPC enabler features):

1. The automatic deployment (installation and configuration) of the HPC software layers.
2. The use of standard HPC software applications for each layer.
3. Serve as a middleware where users and researchers can develop software applications to be deployed on the HPC applications layer (uppermost) relying on scalable subjacent software layers, and;
4. Adaptation to the specific conditions of the devices and the missions carried out by such devices. For example, the nodes in an HPC cluster of computers are normally connected through high-speed high-bandwidth low-latency communication technologies. Such conditions are not available for robotic applications nowadays.

In this work, it is presented as the high-performance computing/robot operating system (HPC-ROS) package, an enabler technology for ubiquitous supercomputing.

Robot operating system (ROS) is a middleware to facilitate robotics systems development, easing technology reutilization, and research cooperation via packages. As far as we can tell, there is neither a single available package in the current ROS version (KINETIC) nor the previous versions package repositories, that provides the features proposed by the HPC-ROS package presented here.

Following, Section 2 introduces a novel conceptual description to implement HPC robotic applications in the future. In Section 3, the HPC-ROS package is presented and Section 4 depicts current conclusions.

## 2. Conceptual description

In this section, it introduced a conceptual description (ontology) to facilitate the design of missions carried out with HPC cluster of robots.

http://dx.doi.org/10.5772/intechopen.69443

Considering the idea of a robot as a software container, a mission can be understood as a set of software executed in the robots and in ground stations. This software could be of two types: *control* and *mission*. Control refers to the software used for the management of the robot, and it is classified in three types, as:

- Operation: e.g., flying (telemetry, autopilot, etc.).
- Payload: Sensor, actuators management software.
- Awareness: Software for the interaction with the environment and other actors e.g., Air traffic management (ATM).

Mission refers to the embedded software used for the carrying of the mission.

Under these ideas, the concept of *HPC job* is presented. An HPC job is a set of software that requires a set of hardware resources for its execution (e.g., quantity of CPUs, RAM, GPUs, disk space, etc.) during a time window known as walltime.

The conceptual description, here presented, serves two purposes:

1. Facilitate the design of scalable missions carried out by HPC cluster of robots
2. Mapping of a mission into HPC jobs.

The following terms are introduced within the scope of this conceptual description.

A *system* is a set of independent heterogeneous or homogeneous nodes (e.g., robots, ground stations, computers, etc.) performing a set of missions distributively and/or cooperatively.

A *mission* is a set of HPC jobs running on an infrastructure.

An *infrastructure* is the system's subset of nodes assigned to a mission.

The following features are part of this conceptual description:

1. A system is composed of 1 up to N missions.
2. A mission is composed of 1 up to M Jobs (HPC Jobs) and an infrastructure.
3. A job is composed of 1 up to X software applications.
4. Software is classified as payload or operation.
5. Each software has a required amount of resources (e.g., CPU cores, GPU cores, RAM, etc.) and a walltime (estimated time of completion).
6. An infrastructure is an HPC cluster, and therefore, it implements all the HPC layers and at least a master node and a set of slave nodes.
7. Each system, mission, job, software, infrastructure, cluster, node has a cost, that can be used for inter (multi-system) or intra (mission-wise, cluster-wise, job-wise, etc.) system negotiation.

Moreover, the nodes within a system are classified, according to the *robot role description* (*RRD*) concept introduced here, as:

1. Master: This is the HPC cluster (infrastructure) master, the machine in charge of implementing the server-side of each HPC software layer. The master node acts as the frontend, to which users interact via HPC jobs submitted to the server-side of the batch system. Normally, though not necessarily, the master node does not perform computation, and therefore, it does not require implementing powerful computing resources such as CPUs, etc.
2. Shadow master: In case of master failure, a shadow master configured as AP (Active/Passive) or AA (Active/Active–load balancing) is set in an HPC cluster.
3. Slave: The slave nodes are the ones in charge of performing computation. The client-side of each HPC software layer is set upon slave nodes.
4. Cluster operator: Considering the autonomy level of a robot, some nodes might require a human operator, pilot, etc. However, relying on the HPC software layers, software in the applications layer could be deployed to operate the entire cluster. This node is the pilot/cluster interface. This is, the pilot interacts with the cluster operator, which in turn commands the entire cluster. A complete scheme can be devised using the file system to handle multiple user profiles and node operators.
5. Mission controller: This is the node, which controls the mission. It is the node, where a software application, in the applications layer, is deployed to do so. In this sense, this node belongs to the infrastructure assigned to the mission. This is a node whose scope is intra-cluster.
6. System controller: This is the node, which controls the system. It is the node, where a software application, in the applications layer, is deployed to do so. In this sense, this node could belong to the infrastructure of a particular mission or be an independent node. This is a node whose scope is inter-cluster.
7. User-defined roles: To support all type of missions, a node can be assigned a user-defined role e.g., fire-detector, fire-extinguisher, etc.

The RRD allows for the combination of multiple roles in a single node. For example, the master node of mission infrastructure could act as mission or system controller. However, a system controller and a mission controller must exist in every system no matter the amount of nodes, missions, etc.

Moreover, an HPC cluster of computers can be set up in a single node, considering that multiple computing cards or even complete computers could be embedded in a robot. This way, using multiple embedded computers, multiple roles could be assigned to a single node.

The batch system is a very configurable software layer that can be used to set up and manage computing or noncomputing resources, with different computing queues (e.g., queue for nodes with GPUs, queue for nodes with FPGAs, etc.), priorities (per user-profile, per-group, per-user, per-job-type, etc.), simple (e.g., FIFO), or complex prioritization schemes such as allowing a second-priority job to run first if the required resources for the first-priority job are not available and only during the time such resources are busy. By combining the RRD

and the batch system and subjacent layers, it is possible to devise very complex and efficient systems, encouraging all type of imaginable missions.

To clarify these ideas, it is introduced the following example. Consider a system called *Mars rover/UAS cluster of robots*.

The *system* is a set of two UASs and two Mars Rovers, whose objective is to perform *in-situ* analysis of Martian ground. Both UASs are not necessarily homogeneous in regards of hardware (computing cards, sensors, actuators, etc.) or software (in the application layers). Same applies for the two rovers. However, all four robots have Ubuntu as operating system installed in their computing cards. The system is composed of two missions:

1. *Air*: This mission consists of one HPC job; aid the rovers to move efficiently and safely. To do so, on-board software for 3D mapping is executed. The *infrastructure* for this mission is the set of UASs.
2. *Ground*: This mission consists of HPC parallel jobs for analysis of Martian ground samples. The *infrastructure* for this mission is the set of rovers.

Following *RRD* for this system is presented. This is one of many possible configurations.

RRD per mission is

1. *Air mission*: There is an *UAS master* and two *UAS slaves*. The master acts as slave. There is no *cluster operator*. A human operator controls each UAS. The slave acts as *shadow master* in AP mode.
2. *Ground mission*: There is a *rover master* and two *rover slaves*. The master acts as slave. There is no *cluster operator*. A human operator controls each rover. The slave acts as *shadow master* in AP mode.

RRD per system is:

1. *System controller*: Earth ground station. A node is not necessarily a robot. These ideas are extensible to all sorts of devices.
2. *Mission controller* (Air and Ground): Mission masters.

Some features of this system are:

1. The system controller (Earth ground station) interacts with the missions' controllers
2. The missions' controllers (master UAS and master rover) interact with each other to share the results of the cluster's HPC jobs. The UAS's cluster calculates the location to which the rover's cluster must move. The UAS master shares this information with the rover master, which shares it with the slave rover.
3. The rovers, once knowing the location to which they should move, are directed there, take the ground samples and process them with its computing cards using parallel algorithms (executed on both rovers at the same time).

All the previous information relates to the first purpose of this conceptual description: Facilitate the design of scalable missions carried out by HPC cluster of robots. Regarding the second purpose: Mapping of a mission into HPC jobs, **Figure 2** depicts such mapping.

According to the ontology here presented, a mission is described as a workflow composed of HPC jobs, where each HPC job is a workflow of software applications, executed into the mission infrastructure (cluster of robots).

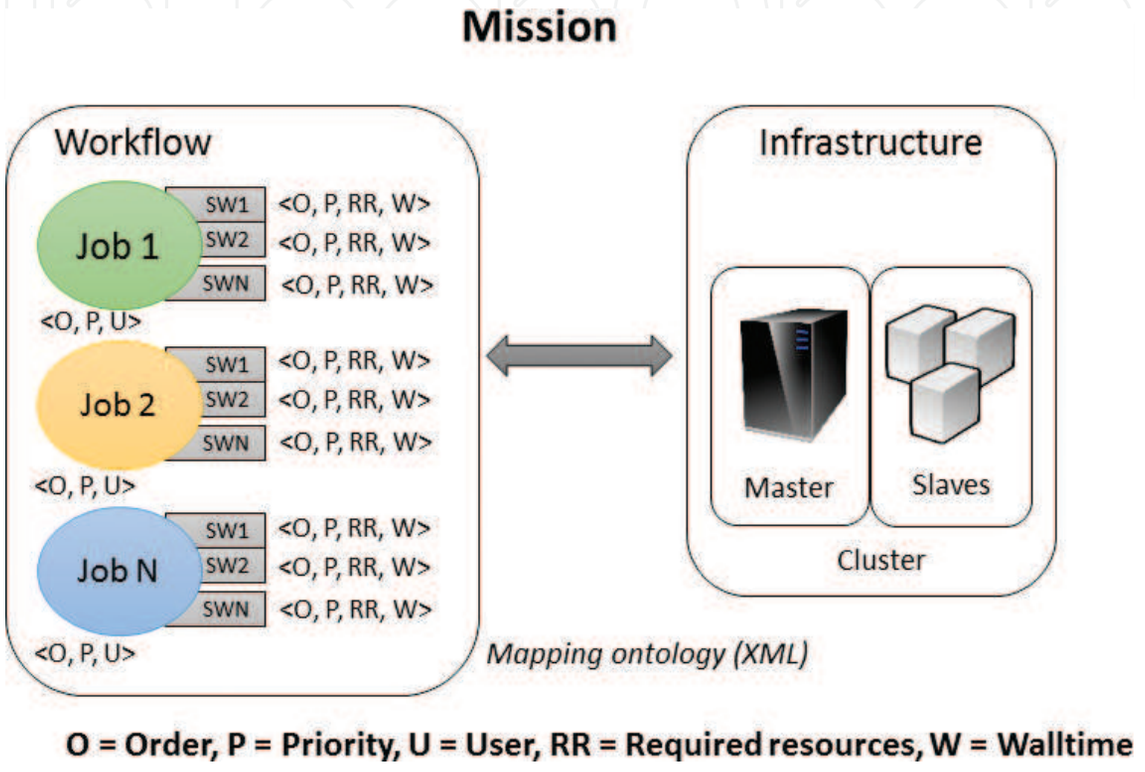


**Figure 2.** Mission workflow to HPC job mapping.

Each job is executed by a user (U), it has a priority (P), and an order (O). The priority is used by the batch system to decide which job to execute first in the infrastructure. The order is used to describe parallel jobs, jobs that must be executed at the same time. In the case of software applications (SW), order and priority also applies, plus each one has a set of required resources (RR) and a walltime (W). This information is introduced in an XML file.

Following Section 3 introduces the *HPC-ROS package*, an enabler technology for ubiquitous supercomputing.

## 3. Design and implementation details

This section introduces the *HPC-ROS package*, a technology based on the conceptual description described in Section 2. The HPC-ROS package is a Ubiquitous Supercomputing enabler that can be used to set up HPC cluster of computers in all types of devices (e.g., robots,

http://dx.doi.org/10.5772/intechopen.69443

computers, ground stations, augmented reality devices, etc.) that can be installed with a Linux operating system, Ubuntu for the moment.

The HPC-ROS packages allow (features), as proposed in Section 2:

1. The automatic deployment (installation and configuration) of the HPC software layers: Setting up a cluster of computers specifically the HPC software layers is a complex process that requires deep knowledge of Linux operating systems and HPC in general. Easing its deployment facilitates the robotics community to focus only on the robotics application to be executed on the applications layer of the cluster of computers.
2. The use of standard HPC software applications for each layer: By using standard HPC software applications, normally open-source, an HPC cluster of robots could interact with standard cluster of computers, for example, for missions requiring computing power not available in the robots.
3. Serving as a middleware where users and researchers can develop software applications to be deployed on the HPC applications layer (uppermost) relying on scalable subjacent software layers: Once a HPC cluster of robots is set, researchers could write mission software exploiting parallelism provided by the subjacent layers, and
4. The adaptation to the specific conditions of the devices and the missions are carried out by such devices. For example, the nodes in an HPC cluster of computers are normally connected through high-speed high-bandwidth low-latency communication technologies. Such conditions are not available for robotic applications nowadays: In a traditional cluster of computers, it is imperative that the connectivity between master and slaves is maintained during the execution of a specific software, since all the server side of all HPC software layers is confined to the master node. A common strategy is the deployment of a shadow master. In the case of mobile robots, operating in outdoor environments, a solution to maintain the master-slaves connectivity must be devised. Strategies for such purpose are being currently studied as part of this research work. However, they are not discussed in this chapter.

Moreover, ROS has been widely adopted by the robotics community. It is our belief that developing a ROS package, for the automatic deployment of HPC cluster of robots, is an important contribution that could be used for the development of very complex robotics systems by a community that keeps embracing ROS, more and more as a standard. Nevertheless, the HPC features of the package can be decoupled from ROS easily if the specific robotic system does not use ROS.

The HPC-ROS package, first version (1.0.0) consists of two software components: *HPC deployer component* and *System management component*.

### 3.1. HPC deployer component

This component relates to features 1, 2, and 3 of the HPC-ROS package. Using this component, the HPC-ROS package installs and configures all software layers in a device installed

with Ubuntu. It is currently tested in Ubuntu 14.04, 16.04, MATE, and with ROS Jade version, upon virtual machines and Raspberry PI 3 model B [23].

**Figure 3** shows the software architecture of the HPC deployer component. The python scripts (ROS nodes [24]) *Commander* and *Deployer* are used to set different roles in a particular device.

Four options are available for deployment in the current version: *setDefault*, *setMaster*, *setSlave*, and *testCluster*.

- The option setDefault installs and configures base software for high performance computing in Ubuntu, required by all HPC software layers. With this option deployed over a device, the node is capable of performing parallel computing.
- The option setMaster installs and configures the server-side of each HPC software layer configuring therefore a master node.
- The option setSlave installs and configures the client-side of each HPC software layer configuring therefore a slave node.
- Finally, the option testCluster executes a parallel software (MPI) test over the configured cluster.
- All options, except test Cluster, install and configure standard HPC software in each layer.

**Figure 4** shows the use of the HPC deployer component for automatic cluster computing setup. Setting up and administrating a cluster of computers is a complex task that requires deep knowledge of Linux OS, networking, information security, parallelization approaches, software architecture, etc. This component facilitates this process so the users or researchers

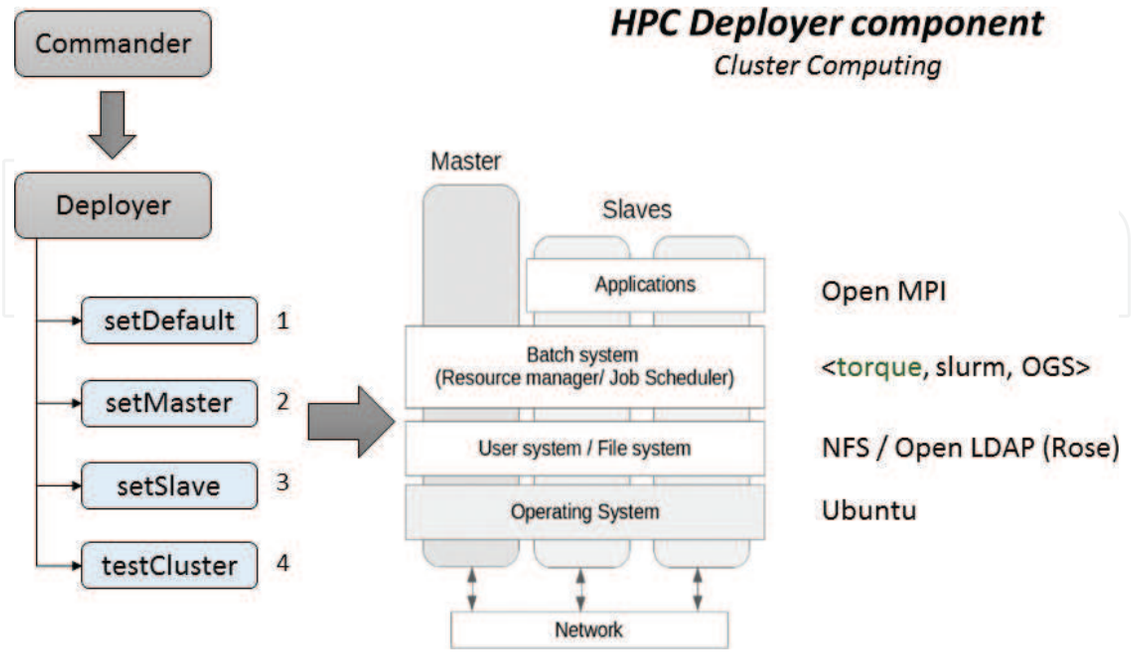


**Figure 3.** HPC-ROS package–HPC deployer component.

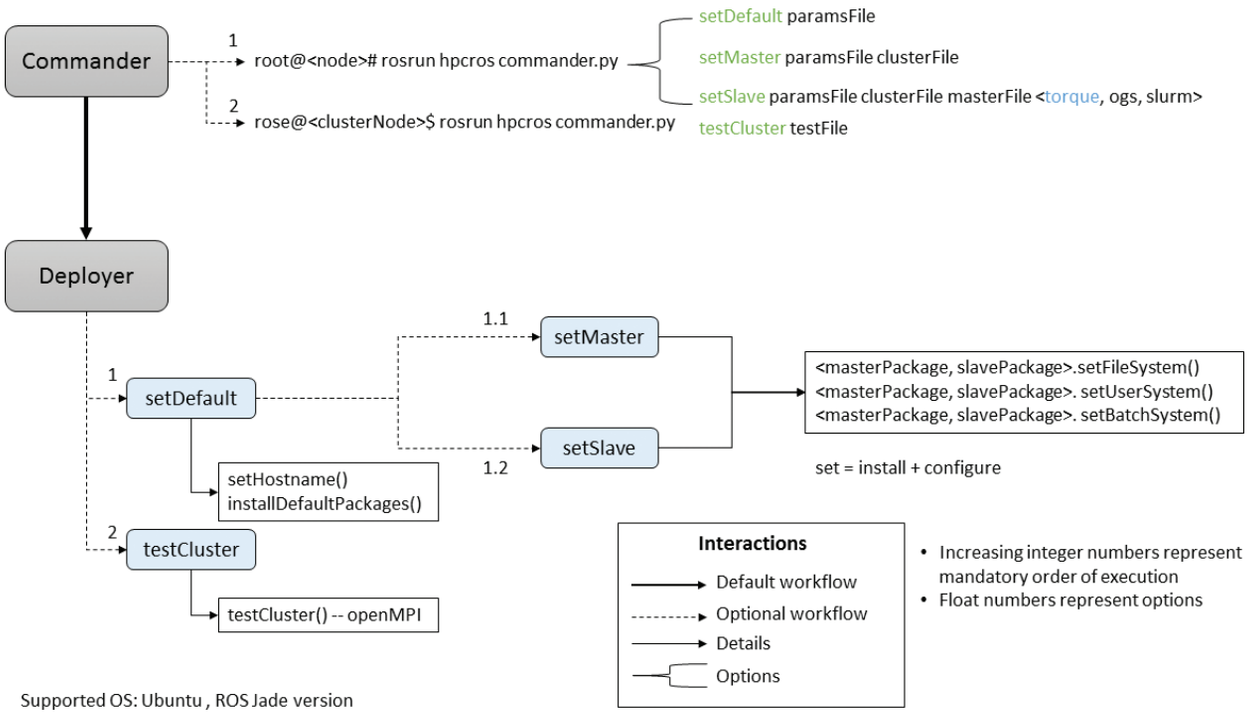


**Figure 4.** Automatic cluster computing setup.

can focus on the specific software applications (e.g., for robotics) to be implemented upon the applications layer (feature 3 of the HPC-ROS package).

### 3.2. System management component

This component relates to features 3 and 4 of the HPC-ROS package. This component is currently in design phase. **Figure 5** presents the system management component.

In the current design stage of this component, three subcomponents are foreseen: *dynamic HPC*, *mission controller*, and *system controller*. The three subcomponents serve two purposes:

1. Act as ROS nodes
2. Provide a set of libraries that can be used by software applications to be deployed upon the application layers

**Dynamic HPC**—This subcomponent is associated to the HPC features: scalability and failure tolerance. Given the conditions of missions to be carried out by an HPC cluster of robots, where constant connectivity is not granted, four options are available for this subcomponent: *Node addition*, *Node removal*, *Cluster connection*, and *Leader selector*.

- The option node addition allows a system to include a node into a specific mission infrastructure. To do so, the dynamic HPC subcomponent interacts with the HPC deployer component.

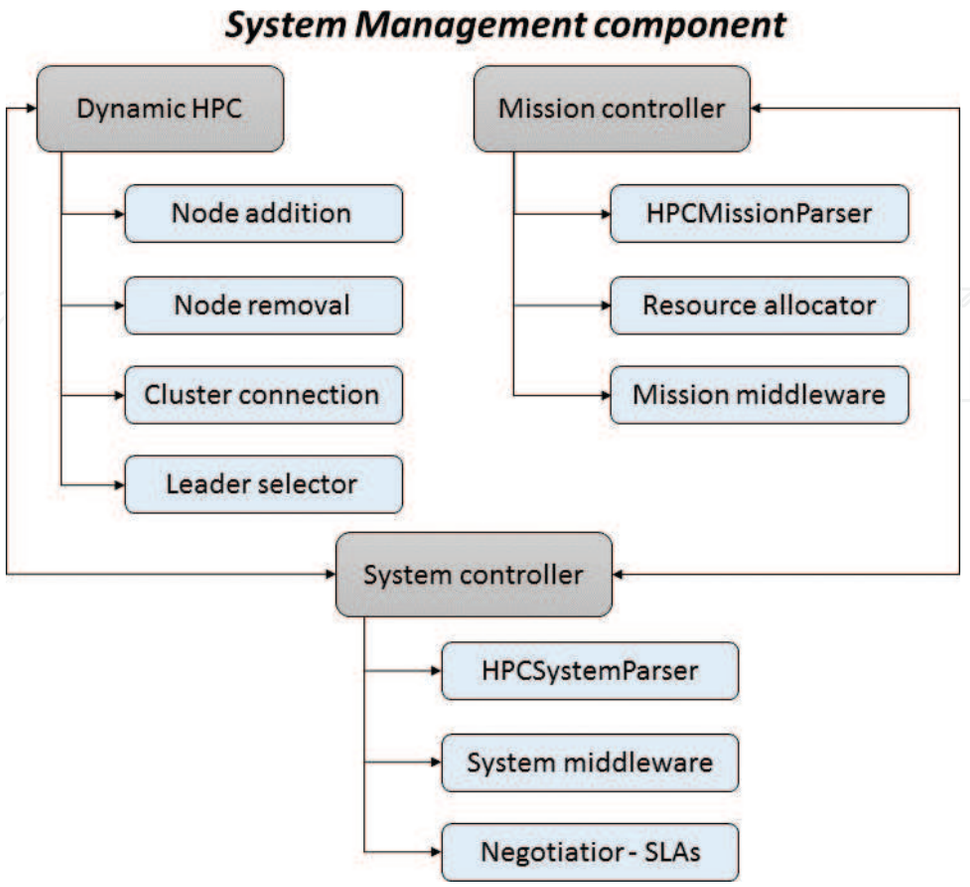


**Figure 5.** HPC-ROS package–System management component.

- The option node removal allows the removal of a node in case of detected failure, lack of connectivity, etc.
- The option cluster connection facilitates the interaction between HPC software layers in different clusters belonging to the same system or different systems (multi-system level).
- The option leader selector allows selecting a new master node (leader) in case of master and/or shadow master failure.

**Mission controller**—This subcomponent is to be deployed upon the mission controller node (RRD). It provides three options: *HPCMissionParser*, *Resource Allocator*, and *Mission middleware*.

- The option HPCMissionParser allows mapping of missions into HPC jobs, in order to request transparently (without user-interaction) resources to the batch system. This applies at mission level (intra-cluster)
- The option Resource Allocator facilitates to allocate or reallocate (node adding, node removal) resources to mission jobs.
- The option Mission middleware offers a set of features for message exchange, cooperation, load distribution, etc., between nodes belonging to the same cluster. It relies on the dynamic HPC subcomponent–option cluster connection.

**System controller**—This subcomponent is to be deployed upon the system controller node (RRD). It provides three options: *HPCSystemParser*, *System middleware*, and *Negotiator–SLAs*.

- The option HPCSystemParser allows mapping of a system into HPC jobs, in order to request transparently (without user-interaction) resources to the batch system of each infrastructure within the system. This applies at system or multi-system level.
- The option System middleware offers a set of features for message exchange, cooperation, load distribution, etc., between clusters belonging to the same system or different systems. It relies on the dynamic HPC subcomponent–option cluster connection.
- The option Negotiator–SLAs allows to negotiate adding of nodes to specific infrastructures, using an infrastructure for the execution of particular jobs, use of resources belonging to a node, a cluster, etc.

This negotiation is based on a cost function, which considers aspects such as: computing capacity (e.g., floating operations per second–FLOPS, quantity of cores, etc.), max endurance (e.g., for the UASs), max range, weight, energy consumption, price, etc.

Each system, mission, job, software, infrastructure, cluster, or node has a cost that can be used for inter (multi-system) or intra (mission-wise, cluster-wise, job-wise, etc.) negotiation.

Using this option, a whole new range of missions could emerge from the use of robotic systems belonging to different companies, organizations, etc. by the implantation of service level agreements (SLAs).

## 4. Conclusions

The conceptual description presented in Section 2 and the HPC-ROS package in Section 3 could be used to build very complex systems, carrying out missions with multiple heterogeneous components such as UASs, UGSs (rovers), Internet of things devices, sensors, etc., acting as a single distributed collective entity, as a cluster of computers is (collection of multiple heterogeneous computing units).

Two aspects are being addressed with the development of the HPC-ROS package. Quickly and easily deploying a HPC cluster of robots and maintaining its virtual existence (master-slaves connectivity) during the execution of a mission.

Deploying a cluster of computers is not an impossible task but it requires deep knowledge of Linux operating systems, understanding of the ideas behind HPC and ultimately time effort. In this sense, the HPC-ROS package *HPC deployer component* facilitates this task by automating the deployment (installation and default configuration) of a cluster of computers. It can do so in any computing cards installed with Ubuntu.

Current communications technologies prevent a fully operating standard cluster of computers (high bandwidth, low latency, constant connectivity, etc.) to be deployed with multiple robots operating outdoors (e.g., outside a Wi-Fi area). Locally (e.g., in one UAS), cluster computing could be easily implemented (e.g., multiple computing cards embedded on an UAS).

However, the HPC-ROS package aims at multi-robot systems, and it is our belief that in the following years, communication technologies will be ready for mobile embedded HPC. Nevertheless, the HPC-ROS package *System management component* looks for adapting to nowadays communication technologies. We are exploring strategies to maintain the master-slaves connection, at all time, during the execution of a mission like bioinspired models facilitating, therefore, HPC cluster of robots operating outdoors.

## Acknowledgements

The authors would like to thank the Colombian government, in particular to the Colombian Administrative Department of Science, Technology and Innovation–Colciencias for the funding of the Ph. D studies of the first author, which include the work presented hereby.

## Author details

Leonardo Camargo Forero*, Pablo Royo and Xavier Prats

*Address all correspondence to: leonardo.camargo@upc.edu

Universidad Politécnica de Cataluña, Castelldefels, Spain

## References

[1] Message P Forum. Mpi: A Message-Passing Interface Standard. Technical Report. Knoxville, TN, USA: University of Tennessee; 1994

[2] Holland O, Woods J, De Nardi R, Clark A. Beyond swarm intelligence: The ultraswarm. IEEE. In: Proceedings of the IEEE Swarm Intelligence Symposium (SIS '05); 8-10 June 2005. Pasadena, California; 2005, June. pp. 217-224

[3] Marjovi A, Choobdar S, Marques L. Robotic clusters: Multi-robot systems as computer clusters: A topological map merging demonstration. Robotics and Autonomous Systems. 2012;60(9):1191-1204

[4] Dagum L, Menon R. Open MP: An industry standard API for shared-memory programming. IEEE Computational Science and Engineering. 1998;5(1):46-55. DOI: 10.1109/99.660313

[5] Gabriel E, Fagg G, Bosilca G, Angskun T, Dongarra J, Squyres J, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain R, Daniel D, Graham R, Woodall T. Open MPI: Goals, concept, and design of a next generation MPI implementation. In: European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting; 19-22 September 2004; Budapest, Hungary: pp. 97-104

[6] Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with CUDA. Queue. 2008;6(2):40-53

[7] Salamí E, Barrado C, Pastor E. UAV flight experiments applied to the remote sensing of vegetated areas. Remote Sensing. 2014;6(11):11051-11081

[8] Intel® Edison Compute Module [Internet]. 2017 Available from: http://www.intel.com/content/www/us/en/do-it-yourself/edison.html [Accessed: 11-01-2017]

[9] NVIDIA Jetson TX1 [Internet]. 2017. Available from: http://www.nvidia.com/object/jetson-tx1-module.html [Accessed: 11-01-2017]

[10] Jeon D, Kim D, Ha Y, Tyan V. Image processing acceleration for intelligent unmanned aerial vehicle on mobile GPU. Soft Computing. 2016:20(5):1713-1720

[11] Cocchioni F, Frontoni E, Ippoliti G, Longhi S, Mancini A, Zingaretti P. Visual based landing for an unmanned quadrotor. Journal of Intelligent & Robotic Systems. 2016: 84(1-4):511-528

[12] Ribeiro C, Dutra M, Rabelo A, Oliveira F, Barbosa A, Frinhani L, Porto D, Milanez R. A robotic flying crane controlled by an embedded computer cluster. In: IEEE 12th Latin American Robotics Symposium (LARS) and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR); 29-31 October 2015; Uberlandia, Brazil: IEEE; 2015. pp. 91-96

[13] ODROID-X2 open development platform [Internet]. 2017. Available from: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G135235611947 [Accessed: 11-01-2017]

[14] Gentzsch W. (Sun Microsystems). Sun grid engine: Towards creating a compute power grid. In: Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID '01); 16-18 May 2001; Brisnane, Australia: IEEE/ACM; 2001. pp. 35-36

[15] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng A. ROS: An open-source Robot Operating System. In: Proceedings of the ICRA Workshop on Open Source Software; 12-17 May 2009; Kobe, Japan. Vol. 3, No. 3.2. p. 5

[16] Ma'sum M, Jati G, Arrofi M, Wibowo A, Mursanto P, Jatmiko W. Autonomous quadcopter swarm robots for object localization and tracking. In: International Symposium on Micro-NanoMechatronics and Human Science (MHS); 10-13 November 2013; Nagoya, Japan: IEEE; 2013. pp. 1-6

[17] Davis E, Nizette B, Yu C. Development of a low cost quadrotor platform for swarm experiments. In: 32nd Chinese Control Conference (CCC); 26-28 July 2013; Xi'an, China: IEEE; 2013. pp. 7072-7077

[18] Ramaithitima R, Whitzer M, Bhattacharya S, Kumar V. Automated creation of topological maps in unknown environments using a swarm of resource-constrained robots. IEEE Robotics and Automation Letters. 2016;1(2):746-753

[19] Pestana J, Sanchez-Lopez J, de la Puente P, Carrio A, Campoy P. A vision-based quadrotor swarm for the participation in the 2013 international micro air vehicle competition. In: International Conference on Unmanned Aircraft Systems (ICUAS); 27-30 May 2014; Orlando, USA: IEEE; 2014. pp. 617-622

[20] Arumugam R, Enti V, Bingbing L, Xiaojun W, Baskaran K, Kong F, Kumar A, Meng K, Kit G Davinci: A cloud computing framework for service robots. In: IEEE International Conference on Robotics and Automation (ICRA); 3-8 May 2010; Anchorage, USA: IEEE; 2010. pp. 3084-3089

[21] Remmersmann T, Tiderko A, Langerwisch M, Thamke S, Ax M. Commanding multi-robot systems with robot operating system using battle management language. In: IEEE Communications and Information Systems Military Conference (MCC); 8-9 October 2012; Gdansk, Poland: IEEE; 2012. pp. 1-6

[22] Hu C, Hu C, He D, Gu Q. A new ROS-based hybrid architecture for heterogeneous multi-robot systems. In: IEEE 27th Chinese Control and Decision Conference (CCDC); 23-25 May 2015; Qingdao, China: IEEE; 2015. pp. 4721-4726

[23] Raspberry PI 3 model [Internet]. 2017. Available from: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ [Accessed: 11-01-2017]

[24] ROS documentation concepts [Internet]. 2017. Available from: http://wiki.ros.org/ROS/Concepts [Accessed: 11-01-2017]