CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Towards a Distributed Computation Offloading Architecture for Cloud Robotics

Rihab Chaari
Omar Cheikhrouhou
Anis Koubâa
Habib Youssef
Habib Hmam

CISTER-TR-190409

Towards a Distributed Computation Offloading Architecture for Cloud Robotics

Rihab Chaari, Omar Cheikhrouhou, Anis Koubâa, Habib Youssef, Habib Hmam

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

Cloud robotics is incessantly gaining ground, especiallywith the rapid expansion of wireless networks and Internetresources. In particular, computation offloading is emerging asa new trend, enabling robots with more powerful computationresources. It helps them to overcome the hardware and softwarelimitations by leveraging parallel computing capabilities andthe availability of large amounts of resources in the cloud.However, the performance gain of computation offloading incloud robotics is still an ongoing research problem becauseof the conflicting factors that affect the performance. In thispaper, we investigate this issue and we design a distributed cloudrobotic architecture for computation offloading based on Kafkamiddleware as messaging broker. We experimentally validatedour solution and tested its performance using image processingalgorithms. Experimental results show a significant reductionin robot CPU load, as expected, with an increase in robotcommunication delays.

Towards a Distributed Computation Offloading Architecture for Cloud Robotics

Rihab Chaari
Prince Research Laboratory
University of Mannouba, Tunisia
rihab.chaari@coins-lab.org

Omar Cheikhrouhou
College of CIT, Taif University
Taif, Saudi Arabia
Computer and Embedded Systems Laboratory
University of Sfax, Sfax, Tunisia
o.cheikhrouhou@tu.edu.sa

Anis Koubâa
Prince Sultan University, Saudi Arabia
CISTER/INESC-TEC, ISEP, Portugal
Gaitech Robotics, China
akoubaa@psu.edu.sa

Habib Youssef
Prince Research Laboratory
University of Sousse, Tunisia
Habib.Youssef@fsm.rnu.tn

Habib Hmam
Faculty of Engineering
University of Moncton, NB, E1A 3E9, Canada
habib.hamam@umoncton.ca

***Abstract*—Cloud robotics is incessantly gaining ground, especially with the rapid expansion of wireless networks and Internet resources. In particular, computation offloading is emerging as a new trend, enabling robots with more powerful computation resources. It helps them to overcome the hardware and software limitations by leveraging parallel computing capabilities and the availability of large amounts of resources in the cloud. However, the performance gain of computation offloading in cloud robotics is still an ongoing research problem because of the conflicting factors that affect the performance. In this paper, we investigate this issue and we design a distributed cloud robotic architecture for computation offloading based on Kafka middleware as messaging broker. We experimentally validated our solution and tested its performance using image processing algorithms. Experimental results show a significant reduction in robot CPU load, as expected, with an increase in robot communication delays.**

***Index Terms*—cloud robotics, processing offloading, distributed computing**

I. Introduction

Mobile robots have enabled a large number of applications in different domains, including manufacturing, inspection, surveillance [1], disaster response [2], agriculture, healthcare and medical, domestic services [3], to name a few. For several years, robots were designed as standalone systems to perform specific tasks. However, with the emergence of the Internet-of-Things and cloud computing there is an increasing tendency to leverage the Internet connectivity to facilitate the accessibility to services everywhere and anytime, and on the other hand, take benefit from the abundant computing and storage resources on the cloud to boost the performance of applications [4]. In this direction, the cloud robotics concept has emerged since 2010, when it was coined by James Kuffner in [5] and the idea is to connect robots to the cloud to take benefits from their resources. In [6] have presented a survey on cyber-physical cloud including cloud robotics. In [7], Koubaa has classified cloud robotics into two categories: (1) *virtualization*: which means providing seamless access to robots through service interfaces (ii.) *computation offloading* also known as remote brains, where computation is offloaded from the robots to the cloud through service interfaces. Several works in the literature have addressed the virtualization aspects of cloud robotics. For example, in [8], the authors proposed Dronemap Planner, a cloud-based management system of robots and drones over the Internet. It allows to access robots and drones, operating with the MAVLink [9] and ROSLink [10] protocols anytime and anywhere through the cloud. In [2], the authors proposed a search and rescue algorithm for disaster response using a swarm of drones by leveraging computation offloading on the cloud. In [11], the authors have proposed ROSBridge which allows to access robots through web interfaces. However, the Web server was located in the robot itself, thus making its access outside a local area network challenging. This problem was addressed by ROSLink [10], that adopted a three-tier architecture where the server is located on the cloud and the robot and user communicate through it.

On the other hand, several cloud robotics research works focused on computation offloading [12]. The idea behind computation offloading is that robots usually are battery powered and have limited computing and storage resources. Thus, running computation extensive applications on-board will deplete their energy faster. In addition, robots might be even incapable of handling complex computation intensive tasks due to the limitation of their resources (e.g. computer vision applications that require GPU). Furthermore, the use of the cloud enables robots to communicate with each other, share knowledge and collaborate to perform complex tasks. In this respect, the RoboEarth European project [13] has developed a cloud robotics system that promotes knowledge sharing and collaboration among robots. Later, in later recent works [14], researchers have investigated the effectiveness of computation offloading in cloud robotics, because the gain of performance is not always guaranteed. In fact, computation offloading also requires communication with the cloud server, which also consumes energy and can be expensive due to retransmission and bad communication quality. In [15], the authors addressed

the problem of how effectively assign/offload computational tasks. They proposed a genetic algorithm to optimize the task offloading process. In [16], the authors applied cloud robotics to improve the performance of the FastSLAM 2.0 algorithm and compared the performance and speed between the cloud-based and the traditional approach. In [17], the authors have proposed an architecture for cooperative robots using cloud robotics.

In this paper, we investigate the performance of computation offloading in cloud robotics. To this end, we design a fully distributed cloud robotics architecture for seamless computation offloading. The architecture is based on Kafka, which is a scalable distributed publish-subscribe messaging system with a robust queue that can handle a large amount of data, as used in the cloud robotics. Then, we conduct an extensive performance evaluation study to assess the effectiveness of the architecture. The proposed architecture significantly outperforms robots as demonstrated in our performance evaluation.

The paper is organized as follows: In section 2, we discuss the related work and the motivation behind this work. Then, in Section 3, we describe the design of our architecture and provide a detailed description of its actors and the relation among them. In Section 4, we implement a first prototype of the architecture and evaluate its performance. Finally, we conclude in Section 5.

II. Related Work

The idea of offering computation offloading through architecture was firstly presented through Davinci project [18]. The researchers provide the DavinCi server that acts like a proxy server coupling the two ecosystems, Hadoop Distributed File System (HDFS) [19] and Robot Operating Systems (ROS) [20]. The service provides a publish-subscribe model for robots to send/update their information as well as requests, which are logged and maintained on the server. The server pushes this data and/or requests to the backend HDFS. The server triggers MapReduce tasks to execute data and collects results produced by the Hadoop ecosystem. Finally, these results are transmitted to the ROS subscribers. They validated this work by implementing the FastSLAM algorithm and reported significant performance gains in execution times provided by the framework. Although the focus of the paper is the evaluation of FastSLAM algorithms under large distributed environments, the communication issues between internal processes as well as the cloud are not detailed.

Authors in [21] designed a cloud robotic architecture to handle the processing demands of a set of heterogeneous large-scale robot agents. It consists of reliably link robots abstracted by ROS with cloud resources, namely cloud storage and cloud computation. The architecture includes also networking platforms used to manage communication between robots and cloud infrastructure. The architecture was implemented using the Hadoop Distributed File System (HDFS) as a cloud storage engine, and MapReduce as a processing engine. The cloud resources are managed by OpenStack [22] platforms. But, neither a quantitative performance analysis was presented, nor a prototype was implemented.

While the aforementioned works proposed general purpose architectures, authors in [23] proposed a collaborative Visual simultaneous localization and mapping (SLAM) framework. The framework is a cloud-based solution for sharing and processing data of a large team of robots. The proposed framework is composed of four main components: 1.) storage layer, contains the data collected and shared among robots, 2.) shepherd Layer, responsible for the cloud coordination, 3.) computation layer, consists of processing the SLAM algorithm, and 4.) communication layer, maintains connections between robots and computation hosts. Besides, they have designed an algorithm to divide the sets of robots demands on the computation resources. Through simulation, they demonstrated the scalability of the proposed framework, and its efficiency compared to other SLAM solutions.

Compared to the aforementioned works, we propose a distributed cloud robotic architecture. The communication engine in the architecture utilizes Apache Kafka [24] to seamlessly handle data transmission. In fact, Kafka is designed with a good message delivery concepts that 1.) persists message data of different formats on the cluster to support a variety of message consumption and failure handling, and 2.) loose coupling between messages consumers and producers. Besides, we can linearly scale servers without affecting the existing cluster setup or the data in.

III. System Architecture

In this section, we specify the functional requirements of the proposed architecture in addition to the non-functional requirements. Then, we focus on architecture design for achieving our objectives.

A. Requirements

Our main objective is to assess the resource efficiency and utilization of robots by offloading computationally intensive tasks to remote cloud servers. For that, we propose to design a cloud-based architecture where streams of intensive data are transmitted to the cloud infrastructure. The basic idea of the offloading system consists of 1) the design of a distributed architecture to dynamically handle the robots computation demands, and 2) the implementation of an offloading middleware enabling bidirectional communication between robots and cloud resources. As multiple cloud resources are available, the middleware aims at ensuring a seamless and a transparent way to exchange data, therefore reducing the difficulties raised by the heterogeneity of robots and cloud.

Designing such a system is challenging since we need to consider many factors including:

- *Transparency*: Transparency is crucial for robots to transmit their data without having the internal cloud information. In a transparent cloud environment, robots access remote resources while ignoring their location. The address of the remote cloud servers and/or the access mechanisms are totally hidden. Such characteristics will

facilitate the efficient delivery of the robots data to remote cloud servers.

- *Scalability*: The system must be scalable in many aspects. The architecture should handle all requests of all robots even if their number increases. The system is recommended to provide a sufficient number of computing machines to store and analyze huge amounts of data. The system should also provide scalable distributed storage systems for big size files.
- *Reliability*: Since robots are transmitting their data to the cloud, the architecture should guarantee that messages reached their final destination in the cloud. The set of computing resources should be available and functional to receive and service all the robot requests. The system should build a continuous and stable connection between the robot team and the cloud resources.

B. Design

In this section, we present the design of the software architecture, including the actors and the relation among them. Figure 1 gives a first insight into the different components of the architecture.

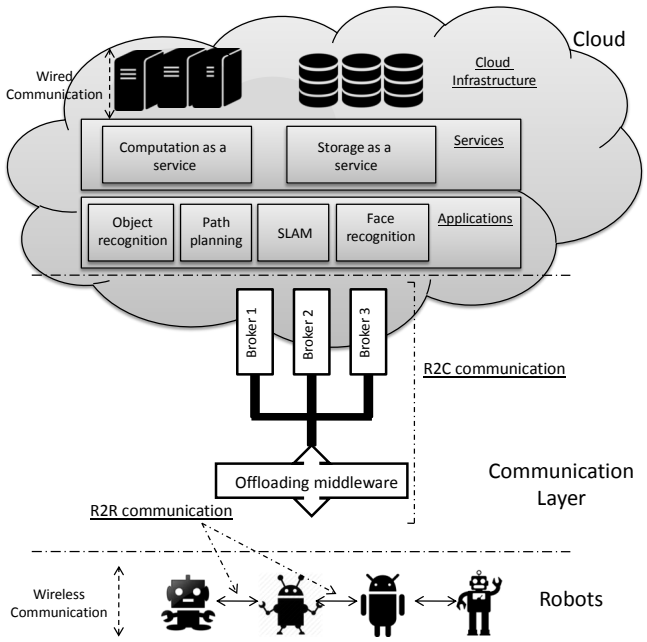

Fig. 1: A High Level View of the Architecture

The offloading computation architecture relies on three main layers: robots, communication, and cloud. As depicted in Figure 1, robots represent the tangible part of the architecture, which is responsible for monitoring its surrounding environment and collecting information using its sensors and/or multimedia features. Then, the collected information is sent to the cloud for i.) stored as data archives, and ii.) computation of a set of software programs. We assume that each robot is equipped with a wireless communication module to transmit data through WiFi, 3G or 4G connection. We assume also that the robots are abstracted by ROS (Robot Operating System) packages. In fact, ROS is the most ubiquitous platform designed for robots as it is an open source project widely known by a large community of users [20]. It is a software robotic platform designed to facilitate resources handling and inter platform interoperability. It provides all the software required to deal with different categories of sensors and actuators.

The middle layer is called communication layer which includes: Robot to Robot (R2R) communication and Robot to Cloud (R2C) communication. Messaging and communication across the robots (R2R communication) are managed by named buses called topics. Topics are an asynchronous publish and subscribe semantics where nodes do not have any idea about data origin. Within topics, data is transported using TCP or UDP protocols. R2C communication is responsible for exchanging robots collected information with cloud computing and storage infrastructure over the Internet. It is used to distribute the robots message streams over several brokers to be consumed by cloud applications and vice versa. It offers a seamless transmission of i.) information flows generated by robots to cloud instances, and ii.) results produced by the execution of cloud applications back to the robots. R2C communication is responsible for exchanging robots collected information with cloud computing and storage infrastructure over the Internet. It is used to distribute the robots' message streams over several brokers to be consumed by cloud applications and vice versa. It offers a seamless transmission of i.) information flows generated by robots to cloud instances, and ii.) results produced by the execution of cloud applications back to the robots.

The topmost layer presents the cloud where the processing of complex applications is performed and data are aggregated in archival databases in the cloud. The cloud is internally made of three sublayers: applications, services, and cloud infrastructure. The application layer exposes various applications remotely accessed by robots such as object recognition, path planning, face recognition, etc. In the services layer, we have two main services: computation as a service and storage as a service. Computation as a service is the provisioning of shared computing resources on demand. While storage as a service is offering a storage infrastructure to archive data. These services allow the allocation of the computation and storage resources required for the application being executed. The cloud infrastructure is a set of nodes constructed from commodity hardware connected to each other through a wired Internet connection. They may be virtual machine instances provided by an IaaS (Infrastructure as a Service) API.

The main contribution of this architecture is distributing streams of data collected from a set of robots to be remotely processed on processing units in the cloud. The achievement of this idea expounds on the design of a middleware enabling bidirectional and stable communication between robots and cloud resources. The offloading solution is modeled by the component diagram presented in Figure 2.

One of the most advantages of our architecture is distribut-

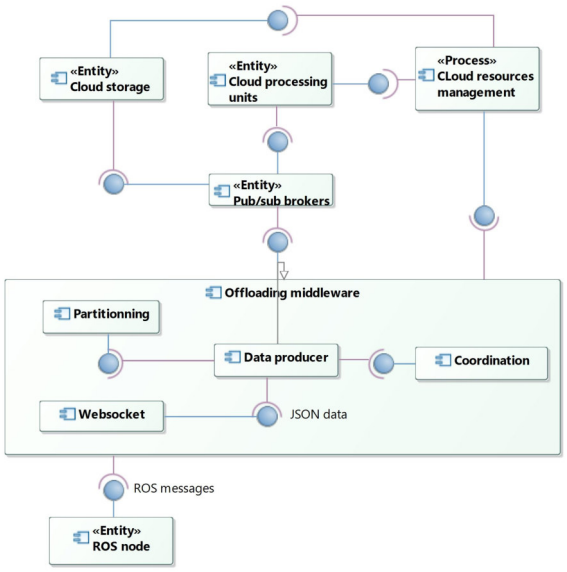

Fig. 2: Component diagram explaining the role of the offloading middleware in transmitting and distributing robots collected data

ing data by using the offloading middleware, through which the processing units in the cloud can process the massive and complex data in a very short time. The middleware acts as an intermediate layer that provides a seamless interaction between the robot ecosystem and the cloud infrastructure. Accordingly, it is responsible for: (i.) subscribing to ROS topics, and (ii.) safely delivering ROS published messages to the cloud. It collects data sent by robots and triggers the backend computation cluster to process it. One of the advantages of the middleware is enabling a transparent computation offloading process for robots. Robots do not have any idea about the cloud infrastructure behind. The middleware uses WebSocket interfaces to expose ROS messages to a set of computing machines in the cloud. In fact, WebSockets offer bidirectional communication using a single socket connection. They make real-time communication more efficient because they guarantee low latency in transmitting messages using long-held connection [25]. WebSocket component interchanges ROS messages to JSON (JavaScript Object Notation) serialized messages to appropriately prepare the data for cloud processing. We preferred to use JSON message format because it is more lightweight compared to XML. Besides, it is language independent and can represent any concrete data.

JSON messages are then partitioned over a pub/sub brokers. The partitioning process ensures that messages are equally distributed over all the cluster machines. If it receives two messages, it stores the first message on the first available machine and the second message on the next machine. It receives periodically the status of the cluster from the coordination process, and then shares the received data on an alive machine. It assigns to the message a key, store the message on the cluster and update the coordination process with information regarding the stored message.

Deploying a distributed system can be a precarious problem. While the main goals of distributed systems are reliability, transparency, scalability, and performance, they are prone to many problems like node failure and inconsistency. Designing and deploying a distributed architecture requires a coordination service, to maintain reliability. The coordination process simplifies data partitioning in the cluster and periodically maintains the status of the cluster. It keeps track of which broker is alive and which has just logged in. It stores the status of the cluster in a file and distributes this file to the entire cluster. The input data to the processing units comprises streams of data published on the brokers of the pub/sub messaging cluster. Upon receiving a publication, the cloud resources management process assigns the job of processing to an available processing unit, then notifies the coordination process. In our case, the maximum level of parallelism we can reach is defined by the number of brokers reserved for the application. In parallel, it stores a copy of this data in a storage entity to be accessed and analyzed at any time.

The interactions between the components of the architecture are best explained through the sequence diagram presented in Figure 3.

For now, our architecture supports the static offloading decision. All the tasks required in a scenario are deployed in the cloud as an application as a service during the design time.

IV. Implementation

A. General Background

1) Robot Operating System (ROS): Robot Operating System (ROS) [20] is a meta-operating system designed for the construction of distributed systems. It provides a set of extensible tools for managing distributed robotic applications. The main goals of ROS are package management, hardware abstraction, low-level device control, message exchange between processes, and implementation of several functionalities.

ROS consists of a set of executable called nodes, used to perform the system computation. Nodes in ROS are organized in Peer-to-Peer network forming the ROS computation graph.

Messages transmitted between ROS nodes are simple data structure. They are of two types: topics and services. A topic is a name of a stream of an object published by a node called the publisher. Other nodes interested in this topic may subscribe to it, and they are called listeners. The ROS master node tracks the list of publishers and listeners to a topic. A ROS master node is an XML-RPC server. Nodes connect to other nodes directly. The Master node provides only lookup information, much like a DNS server. ROS uses the transport layer TCPROS for reliably transporting message data. It uses standard TCP/IP sockets.

2) Apache Kafka: Apache Kafka [24] is a publish-subscribe message brokering system rethought as a commit log. It can be used for both online and offline message consumption. It was

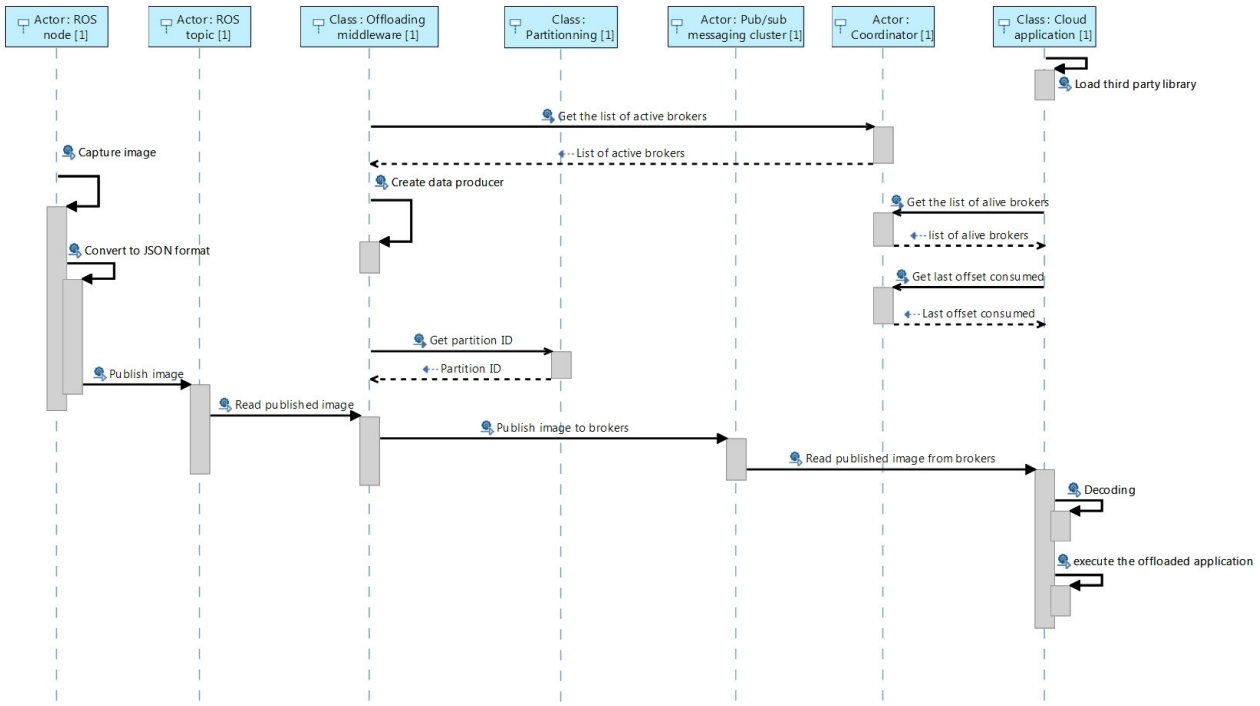

Fig. 3: Sequence diagram explaining the steps through which a message goes from being captured by the robot to the execution in the cloud

originally developed at LinkedIn for managing the mountains of data they are generating.

Kafka offers a high level of abstraction called topic. A topic is a name to which messages are stored in an unchangeable sequence and identified by an offset. For each topic is attributed at least one partition log. Partitions contain messages that are replicated over multiple servers for reliability. Producers are processes that publish data or messages to a specific topic by choosing the appropriate partition. Consumers are applications that subscribe to a topic and process the set of published messages.

3) Apache Storm: Apache Storm [26] is a distributed, open-source, real-time computation system. It is designed to manipulate large streams of data. Components in a Storm cluster are organized in a master-slave architecture. A Storm cluster contains a single master node called nimbus, and a set of worker nodes called supervisors. Nimbus node is responsible for (1) distributing application code across supervisors, (2) assigning tasks to different workers, (3) monitoring tasks for detecting any failures, and (4) restarting tasks when required. Supervisor nodes are responsible for managing worker processes that execute the tasks assigned to this node. Besides, Storm offers a coordination interface between the master and the worker nodes managed by a zookeeper cluster. Storm cluster is fault tolerant since all its configuration information is stored in Zookeeper. Any event of failure of the nimbus or any supervisor node will not disturb the cluster performance.

Data flow in Storm goes continuously through several transformation entities. The key abstraction of data flow in Storm is called stream, which represents an unbounded sequence of tuples. Tuples are a named list of key-value pairs.

Computation in Storm is abstracted as a graph called topology. A topology is simply a network of spouts and a chain of bolt components. Spouts are the source of the data stream while bolts are data processing.

B. First Prototype

Designing the software architecture was challenging. In fact, we faced the problem of choosing the appropriate technologies to bind robots with the cloud infrastructure. In what follows, we present in detail the frameworks used to build a first prototype of the architecture.

All robots are abstracted by ROS packages. Messaging and communication across the robots, and between the robots and the middleware are managed by named buses called topics. A topic is an asynchronous publish and subscribe semantics where nodes do not have any idea about data origin. Within topics, data is transported using TCP or UDP protocols.

ROS messages are collected using rosbridge server [11] which is a JSON interface to non-ROS programs. It uses WebSockets to enable any client to publish and subscribe to ROS topics. The rosbridge connects to the topic of interest, gathers published messages, and sends them immediately to the publish-subscribe brokers. Data transferred between the robots and the back-end computing cluster take place using streams of bytes enveloped in JSON format.

The design of the architecture focuses on publish-subscribe brokers. In fact, in a publish-subscribe messaging pattern publishers and subscribers are loosely coupled; they do not need to know the existence or the parameters (like IP address) of each other. Besides, they may offer better scalability through a parallel operation. We were actually interested in topic-based models.

There are many open source topic-based message brokers, available that fulfill the messaging needs in our architecture like RabbitMQ [27], Apache Kafka [24], Apache ActiveMQ [28], etc. In fact, Apache Kafka has good clustering capabilities which fit the requirements of our architecture. It offers a high throughput level for both publishing and subscribing; thus the architecture won't block under any circumstances. Kafka persists published messages on disk to be used for both batch and stream processing. Besides, the way Kafka stores the messages is simple and efficient. First, messages are stored in partitions of a topic physically implemented as a set of segment files of equal sizes. Each partition corresponds to a logical log used to expose messages to consumers. Second, message overhead in Kafka is small compared to the other solutions [29].

In our architecture, we are actually interested in stream processing platforms. For that reason, we choose to work with the Apache storm as a fundamental cloud processing framework.

The input data to the storm cluster comprises streams of data published by Kafka brokers presented as "Kafka Spout". Kafka Spout is an adapter designed specifically for storm topologies to read Kafka's published messages. Actually, it is similar to a built-in Kafka consumer used to retrieve data from brokers. Actually, the maximum level of parallelism we can reach in a Kafka cluster is defined by the number of partitions we can create for a topic. Having so, we can share the message load into many partitions that many Kafka spout consumers read data from the same topic in parallel. We dynamically divide the load on multiple Kafka partitions and instantiate for each one a Kafka spout to handle it. In the middle, data is processed using bolts. The parallelism at this level is not bounded. It depends on the application.

As we have already mentioned it, the architecture can be used in different scenarios and application. We were interested in computer vision application using openCV [30] library. By default, Storm does not support emitting complex data like openCV object within bolds. To fix this, we have implemented a custom serializer for openCV objects using kryo serialization.

Apache Zookeeper [31] is a coordination service. It is designed for naming, maintaining configuration information, providing group service, and providing distributed synchronization. It runs in a cluster called an ensemble. One node serves as a leader that receives all clients requests. The other nodes are called followers. If the leader fails, one follower will be elected as a new leader to ensure reliability and availability of the service. With zookeeper, data is stored in shared hierarchical namespaces just like a file system. Each namespace is a set of data registers called znodes. Znodes can be files or directories that hold data or child nodes.

In our case, zookeeper is used by both Apache Kafka and Apache Storm. In Apache Kafka, zookeeper is used for topic configuration, cluster membership, the amount of data each client is allowed to write and read, and who is allowed to read and write from which topic. Zookeeper is also responsible for electing a new Kafka controller in case of failure. A Kafka controller is broker responsible for administrative tasks like managing partition states. Apache storm uses a zookeeper cluster to coordinate between nimbus and supervisor nodes. These nodes communicate with each other through the zookeeper ensemble. Besides, zookeeper stores the various tasks submitted and the state of the cluster. Both nimbus and supervisor nodes can be killed without affecting the cluster because all data is stored in zookeeper.

V. Performance evaluation

To evaluate the performance of the architecture, we propose a set of experiments to determine 1) the usefulness of the architecture in achieving high processing speed with low communication delay, and 2) the benefit of offloading the robots in terms of memory and processing.

A. Experimental Setup

We tested our architecture on a small OVH public cloud cluster [32] formed by eight virtual machines provisioned using OpenStack. Each virtual machine is equipped with 7 GB of memory, 50 GB storage, and two core processors running each with 2.3 GHz. The network bandwidth guarantees 250 Mbps for use and can reach until 300 Mbps. All machines run Ubuntu 16.04 operating system.

We deployed a Kafka cluster with three brokers, therefore allowing creating topics with three partitions. Accordingly, we deployed a storm cluster with three supervisor nodes running each on a single virtual machine, because the maximum parallelism we can reach on a Kafka spout is limited by the number of partitions a topic can have. The Nimbus and the zookeeper run on the same machine (see Figure 4). In the last virtual machine, we deployed the ROS node with the offloading middleware on the top. The worker and perception modules in ROS nodes were written in C, and the distributed algorithm running on a storm cluster is written in java.

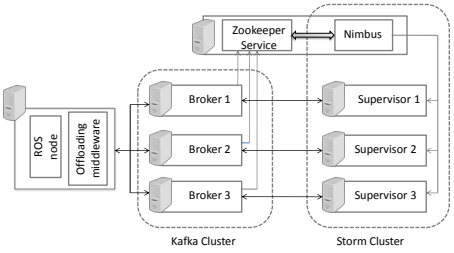

Fig. 4: Deployment Scenario of the Architecture

Nowadays, robots are extensively used in several fields including medicine and industry. As a result, the elements of the construction of a robotic application involve complex technologies. Object tracking is one of the challenging applications for robots since there are two types of information to be extracted: visual features and motion information. We have developed an object tracking algorithm for the evaluation of the architecture. Each image, being captured by the robot, is encoded in base 64 before transmission. Thus, the first bolt in the storm topology is concerned with image decoding. The next bolt is responsible for visual features extraction including color, texture, and shape. The following bolt extracts the motion information, especially the distance from the robot to the object. The last bolt distributes the result on Kafka partitions.

B. Experimental Results

Study 1: A useful computation offloading architecture should offer a high computation speed without introducing large delays. Robots require a quick answer since they may be deployed in critical real-time applications. However, many factors may degrade the performance of the architecture, including the network bandwidth and the amounts of data being exchanged.

In this experiment, we deployed a single ROS node that produces messages at different rates. This data is transmitted through the middleware and shared on the three Kafka brokers, which relay it to the storm cluster. We measured end-to-end delay for (1) robot to middleware, and (2) middleware to cloud servers transmissions with a 95% confidence interval.

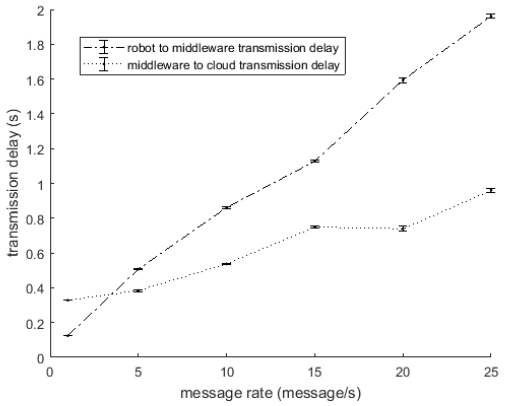

Fig. 5: Transmission Delay

Figure 5 shows the transmission delay of the messages from the robot to the middleware and from the middleware to the cloud. It is clear that the transmission delay in both cases increases linearly with the number of messages transmitted per second. In fact, images are encoded in base 64. The size of data encoded in base 64 is expanded by 1/3 larger size than their binary equivalent, which will raise the bandwidth utilization.

Besides, the robot to middleware transmission delay is larger compared to middleware to cloud transmission delay. This is actually normal since we have one data producer for Kafka cluster and three consumers. Among the top advantages of the design of Apache Kafka is handling high volume and high-velocity data without using large hardware. Apache Kafka splits the data load on multiple brokers (in our case 3). It receives stores and sends messages using different running servers.

Study 2: we demonstrate what benefit the architecture could offer for the robots in terms of memory usage, execution time, and the number of messages being processed. To do so, we deployed two ROS nodes: the first one runs the algorithm of object tracking, while the second one uses the offloading solution. They both run on machines with the characteristics described below.

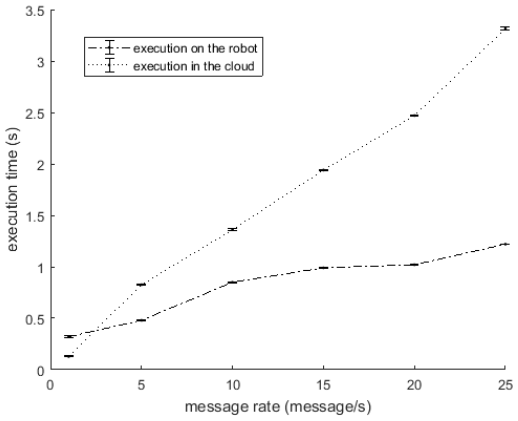

Fig. 6: Execution Time

The first observation, we noticed from this experiment, is that robot with the offloading solution can exchange a higher number of images. A robot running the object tracking algorithm can execute up to 19 messages per second, while a robot with the offloading solution can process up to 30 messages per second.

Figure 6 shows the execution time of the object tracking for both cases on the robot and in the cloud. Cloud execution was surprisingly higher than robot execution. The increase in cloud execution time can be attributed to many factors. First, servers are geographically located in different cloud regions. Data movement in clouds is related to the underlying network bandwidths. Besides, relay points binding to different cloud regions will introduce more delay. Second, since images are encoded in base64 for transmission, the object tracking algorithm running in the cloud runs supplementary tasks including decoding and restoring the initial image. Finally, Apache Kafka runs on the top of the Java Virtual Machine (JVM), which will even double the size of the stored data. As a result, this will increasingly slow down the Java garbage collection. As we can see in Figure 7, the CPU usage without offloading is extremely high compared to the CPU usage

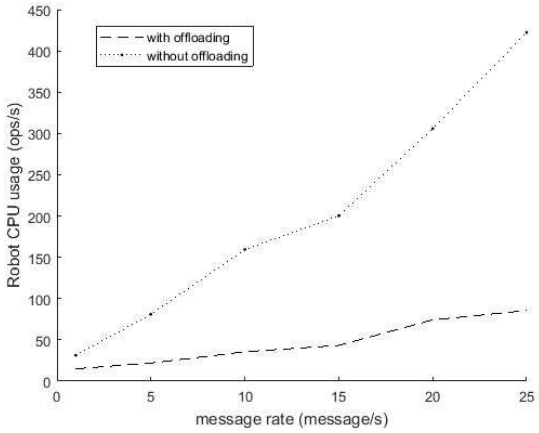

Fig. 7: Robot CPU Usage

with offloading. Offloading seeks to overcome performance bottlenecks by executing some network functions enabling data exchanges with cloud clusters. This has the advantages of augmenting the availability of the robot's CPU for other important functions, therefore improving its efficiency.

VI. Conclusion

This paper presented a cloud robotic architecture to offload the robots from their complex processing workload. The architecture distributes the robots workloads over cloud servers through the offloading middleware.

The architecture was implemented using Apache Kafka and Apache Storm. The primary benefit is the reduced cost of the offloading solution. When implemented using open source solutions, the economic cost of the architecture is less expensive.

Empowering robots with cloud computing comes with a fundamental tradeoff. Offloading the execution of a computationally intensive algorithm to the cloud can reduce resource utilization, including CPU, memory, and the battery. However, this comes with a cost: communicating with cloud resources over a congested network increases latency and can lead to delay for real-time applications. In future work, we will consider working on an offloading decision that reduces the overall execution time of the application.

Acknowledgment

The authors would like to thank the Center of Excellence and the Robotics and Internet-of-Things Unit at Prince Sultan University, Saudi Arabia, for their support of this work.

References

[1] B. Benjdira, T. Khursheed, A. Koubaa, A. Ammar, and K. Ouni, "Car detection using unmanned aerial vehicles: Comparison between faster r-cnn and yolov3," in *2019 1st International Conference on Unmanned Vehicle Systems-Oman (UVS)*, Feb 2019, pp. 1–6.

[2] E. T. Alotaibi, S. S. AlQefari, and A. Koubaa, "Lsar: Multi-uav collaboration for search and rescue missions," *IEEE Access*, pp. 1–1, 2019.

[3] A. Koubaa, M. Sriti, Y. Javed, M. Alajlan, B. Qureshi, F. Ellouze, and A. Mahmoud, "Turtlebot at office: A service-oriented software architecture for personal assistant robots using ros," in *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, May 2016, pp. 270–276.

[4] J. Dizdarevi, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 116:1–116:29, Jan. 2019.

[5] J. Kuffner, "Cloud-enabled robots," in *IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2010.

[6] R. Chari, F. Ellouze, A. Kouba, B. Qureshi, N. Pereira, H. Youssef, and E. Tovar, "Cyber-physical systems clouds: A survey," *Computer Networks*, vol. 108, pp. 260 – 278, 2016.

[7] A. Koubaa, "Service-oriented software architecture for cloud robotics," *CoRR*, vol. abs/1901.08173, 2019.

[8] A. Koubaa, B. Qureshi, M.-F. Sriti, A. Allouch, Y. Javed, M. Alajlan, O. Cheikhrouhou, M. Khalgui, and E. Tovar, "Dronemap planner: A service-oriented cloud-based management system for the internet-of-drones," *Ad Hoc Networks*, vol. 86, pp. 46 – 62, 2019.

[9] "Mavlink developer guide," https://mavlink.io/.

[10] A. Koubaa, M. Alajlan, and B. Qureshi, *ROSLink: Bridging ROS with the Internet-of-Things for Cloud Robotics*. Cham: Springer International Publishing, 2017, pp. 265–283.

[11] C. Crick, Jay, S. Osentoski, B. Pitzer, and O. C. Jenkins, *Rosbridge: ROS for Non-ROS Users*. Cham: Springer International Publishing, 2017, pp. 493–504.

[12] I. Chaari, A. Kouba, B. Qureshi, H. Youssef, R. Severino, and E. Tovar, "On the robot path planning using cloud computing for large grid maps," in *2018 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, April 2018, pp. 225–230.

[13] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. M. M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft, "Roboearth," *Robotics Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, June 2011.

[14] A. Koubaa and B. Qureshi, "Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet," *IEEE Access*, vol. 6, pp. 13 810–13 824, 2018.

[15] Y. Guo, Z. Mi, Y. Yang, and M. S. Obaidat, "An energy sensitive computation offloading strategy in cloud robotic network based on ga," *IEEE Systems Journal*, pp. 1–11, 2018.

[16] S. S. Ali, A. Hammad, and A. S. T. Eldien, "Fastslam 2.0 tracking and mapping as a cloud robotics service," *Computers & Electrical Engineering*, vol. 69, pp. 412 – 421, 2018.

[17] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, K. Nakamura, and J. Ogawa, "A study of robotic cooperation in cloud robotics: Architecture and challenges," *IEEE Access*, vol. 6, pp. 36 662–36 682, 2018.

[18] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit, "Davinci: A cloud computing framework for service robots," in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 3084–3089.

[19] "Apache hadoop," http://hadoop.apache.org/.

[20] "Robot operating system (ros)," www.ros.org/.

[21] S. A. Miratabzadeh, N. Gallardo, N. Gamez, K. Haradi, A. R. Puthussery, P. Rad, and M. Jamshidi, "Cloud robotics: A software architecture: For heterogeneous large-scale autonomous robots," in *2016 World Automation Congress (WAC)*, July 2016, pp. 1–6.

[22] "Openstack cloud software," https://www.openstack.org/.

[23] P. Zhang, H. Wang, B. Ding, and S. Shang, "Cloud-based framework for scalable and real-time multi-robot slam," in *2018 IEEE International Conference on Web Services (ICWS)*, July 2018, pp. 147–154.

[24] "Apache kafka," https://kafka.apache.org/.

[25] V. Wang, F. Salim, and P. Moskovits, *The Definitive Guide to HTML5 WebSocket*, 1st ed. Berkely, CA, USA: Apress, 2013.

[26] "Apache storm," www.storm.apache.org/.

[27] "Rabbitmq," www.rabbitmq.com/.

[28] "Apache activemq," www.activemq.apache.org/.

[29] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece*, 2011.

[30] "Opencv library," http://opencv.org/.

[31] "Apache zookeeper," https://zookeeper.apache.org/.

[32] "Ovh cloud service provider," www.ovh.com.