Resource Optimization in Fog Enabled IoT Deployments

Visali Mushunuri, Ajay Kattepur, Hemant Kumar Rath & Anantha Simha

Embedded Systems and Robotics, Tata Consultancy Services (TCS) Research, Bangalore, India.
Email: visali.m | ajay.kattepur | hemant.rath | anantha.simha @tcs.com

***Abstract*—Internet of Things (IoT) devices are typically deployed in resource (energy, computational capacity) constrained environments. Connecting such devices to the cloud is not practical due to variable network behavior as well as high latency overheads. *Fog computing* refers to a scalable, distributed computing architecture which moves computational tasks closer to Edge devices or smart gateways. As an example of mobile IoT scenarios, in robotic deployments, computationally intensive tasks such as run time mapping may be performed on peer robots or smart gateways. Most of these computational tasks involve running optimization algorithms inside compute nodes at run time and taking rapid decisions based on results. In this paper, we incorporate optimization libraries within the *Robot Operating System (ROS)* deployed on robotic sensor–actuators. Using the ROS based simulation environment *Gazebo*, we demonstrate case-study scenarios for runtime optimization. The use of optimized distributed computations are shown to provide significant improvement in latency and battery saving for large computational loads. The possibility to perform run time optimization opens up a wide range of use-cases in mobile IoT deployments.**

***Index Terms*—Fog Computing, Offloading, Optimization, IoT, Robotics, ROS.**

I. Introduction

Mobile Internet of Things (IoT) [1] devices that typically operate on the sense-compute-actuate framework [2], suffer from common restrictions:

1) Low computational capacities on sensing devices.
2) Low energy and battery capacities.
3) Limited on-board memory and storage capacities.
4) Low-latency turnaround times needed between sensing and actuation.
5) Variability in network connectivity due to mobility patterns of sensor/actuating devices.

In order to overcome these challenges in IoT devices, cloud based architectures [3] have been proposed. Cloud computing [4] offers scalable hardware resources on demand for storing and computing data. This minimizes the computational load on network devices and increases their battery lifetime. However the communication latency and power consumed for communication with the cloud turn down its advantages for delay constrained applications.

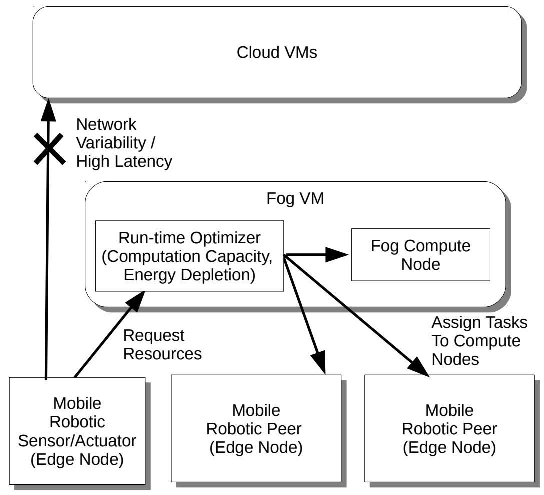
Cloud VMs
Network Variability / High Latency
Fog VM
Run-time Optimizer (Computation Capacity, Energy Depletion)
Fog Compute Node
Request Resources
Assign Tasks To Compute Nodes
Mobile Robotic Sensor/Actuator (Edge Node)
Mobile Robotic Peer (Edge Node)
Mobile Robotic Peer (Edge Node)

Fig. 1. Mobile IoT Deployments with Runtime Optimization.

For low latency requirements, *Fog Computing* has been proposed, which brings the merits of cloud computing to the Edge of network [5], [6]. In Fog Computing, Fog nodes[1] (such as smart gateways) that have high computational capacities, storage power and network access are used to perform computational tasks. As a result, computational capacities of both Edge nodes (low end peripheral nodes) and Fog nodes may be exploited for distributed computations. Typically, latency overheads and limited reliability of centralized cloud access is also mitigated, as a result of this distributed computation.

Fig. 1 shows the general model of our approach, with a Fog node and multiple mobile IoT sensor/actuator nodes. Directly offloading computations to cloud based VMs is not viable due to network variations and latency overheads [7]. Furthermore, the Edge nodes are low end nodes that have limited battery and computational resources for carrying out high computational tasks. Hence, distributing a heavy task among the Edge/Fog nodes reduces computational burden on a single node and improves overall battery lifetime. In this paper, we leverage the advantages of Fog computing for performing high computational tasks on mobile robotic sensor/actuators. We optimally divide

[1]Fog nodes, smart gateways and Edge nodes are referred interchangeably in this paper

the computational load among varied Edge/Fog nodes to reduce battery usage and computational latency. For performing this compute division, the Fog node (server node) is used to divide the computational data over peer robots in an optimal fashion, dependent on run time constraints such as mobility, location, current battery capacities and available computation power. Such optimization constraints are common in most mobile IoT deployments and require run time solutions.

For implementing our model, we utilize the open source ROS (Robot Operating System) [8] to spawn robotic sensors, actuators and computation nodes. ROS is a widely used software development framework with packaging tools and communication infrastructure for robotic scenarios. ROS can also be easily deployed on virtual machines running on Android devices, smart mobile gateway devices as well as robotic hardware [8]. We link the open-source optimization library NLopt [9], which is a widely used optimization library, with ROS, for running optimization algorithms. The NLopt library has implementations for different types of optimization algorithms (global optimization, local derivative-free and local gradient based algorithms). Using this library, we present several use case scenarios and optimize the compute division with different constraints. This may be extended to multiple resource constrained mobile IoT environments.

The main contributions of this paper are:

1) Motivate the need for Fog/Edge compute nodes for distributing computations when cloud connectivity is infeasible.
2) Use Fog Computing for co-operative communication between mobile IoT for a task accomplishment.
3) Integration of an external library NLopt, into ROS, for running optimization algorithms at runtime for robotic deployments.
4) Implement a ROS Service based model for our communication framework.
5) Demonstrate, over multiple case studies, the improvements in energy/latency provided by runtime optimization.

The rest of the paper is organized as follows. In section II, we explain our system model along with associated communication and battery depletion models. Then we briefly discuss different case studies deployed in our robotic scenario in section III. ROS based implementation details are covered in section IV. The results for optimal deployment and energy/latency improvements are shown in section V. In section VI, we discuss the literature related to our work. We conclude this paper with section VII.

II. Mobile System Model

In this section, we provide a brief overview of the Fog based runtime optimization model, battery depletion and communication path loss models.

A. Fog based Optimal Computations

IoT devices typically operate on the sense–compute–actuate paradigm [2]. In order to demonstrate the utility of optimization over such devices, we make use of robotic deployments. These involve i) constrained devices (energy, actuation latency, computation) ii) mobile devices that affect communication transmission and path loss iii) limited connectivity to the cloud due to constraints (i,ii).

In most cases, the connectivity of robots to the cloud may not be available or not reliable or with high communication latency [7] as shown in Fig. 1. In such cases, a Fog node is used for performing large computations. A node intending to perform a large computation cannot directly deploy this to cloud; instead it requests for resources from a nearby Fog node. A Fog node, in addition to the capability to provide cloud connectivity, has its own resources for performing high end computations. A runtime optimizer (running on the Fog node) is implemented in order to divide the computational load to peer nodes depending on their communication path loss, computational power and battery power constraints. This is then assigned to both Fog and Edge peer nodes in an optimal fashion (total energy usage, latency, computational capacities). In general, we work with large computational tasks that need to be divided or parallelized among nodes for faster completion. We assume these tasks (sorting, searching, indexing) are embarrassingly parallel.

B. Power Depletion Model

In order to optimally utilize battery resources, the battery depletion rate is updated on a regular basis to the optimizing server by the computational nodes. This power is computed by finding the power required for data transmission, reception and to perform computations on data. The relation between total battery capacity B_p, actual discharge current in amperes I^p and time taken to discharge current t is given by well known Peukert's law [10]:

$$B_p = I^{\alpha} \cdot t \tag{1}$$

where the parameter α is dependent on the battery type (lead-acid, lithium-ion).

C. Communication Power Path-loss Model

The communication path loss is modeled using Friis transmission equation with P_t as transmission power at sender and P_r as received power at the receiver: [11]

$$\frac{P_t}{P_r} = \frac{1}{G_t G_r} \cdot \left(\frac{4\pi d_i}{\lambda}\right)^{\gamma} \tag{2}$$

where, d_i is the distance between sender and receivers, γ is the path loss exponent which takes values in the range of 2 to 5, G_t and G_r are gains of transmit and receipt antenna and λ is the wavelength. We have assumed that the transfer of data occurs wireless using IEEE 802.11n [12] with maximum throughput of 54 Mbps.

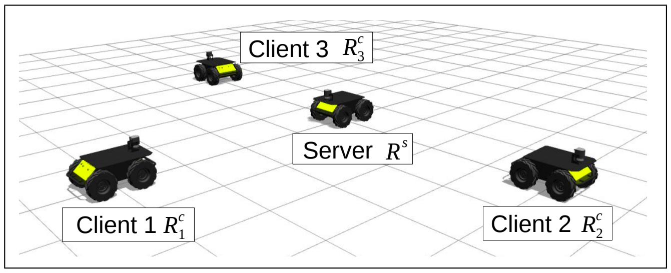

Fig. 2. Robots spawned into ROS Gazebo.

III. Robotic Case Study Deployment

To demonstrate our approach on mobile sensor-actuators, we spawn four Husky robots [13] into the ROS-Gazebo simulator as shown in Fig. 2. ROS (Robot Operating System) is a widely used software development framework with packaging tools, simulators and communication infrastructure for robotic scenarios. The robots are equipped with odometry sensors that may be used to actuate velocity and angular movement for localization and path planning. Robots are defined as nodes in ROS and all these nodes communicate with each other using ROS Communication framework. We make use of ROS Services [14] for communication between server and client robots and ROS topics for position updates.

One robot acts as a master (server) node with the other three robots as client nodes. The server node (Fog node) is denoted as R^s and the three Edge client nodes as R_1^c, R_2^c and R_3^c. In our model, the client node R_1^c moves around and collect location information to perform some computations (like constructing a global map or finding the nearest location to server). Whenever there is some data available for computation, the robot R_1^c share its computation load among peers in order to extend its battery lifetime. The Edge client robot having some data for computation offloads its entire data to Fog server robot R^s for computational data division. R^s then divides the whole computation among peer robots depending on their communication path loss and power availability. It then sends the divided computational data to corresponding client node for carrying out computations. We formulate this computational data division by R^s as a constrained optimization problem with battery capacity, communication path loss and computational capacity of the client robot acting as main constraints. Note that robotic nodes may be replaced with higher capacity devices, representing Fog nodes, such as smart gateway devices.

Since the communication between server and client robots is one-to-one, we use the *ROS Service Call* method. ROS services allow nodes to communicate via Request/Response messages. The server node offers different *services* to client nodes for communication. The client nodes call the corresponding Service whenever they need to communicate with the server node. The implementation details of the Client-Service model are given in Section IV.

The power calculations are performed using the parameters available from Husky Data sheet and given in Table I. The battery power calculations uses this data to find the power consumed when robot is in movement or performing computation.

TABLE I
Husky Data Sheet Specifications

Parameter	Values
Dimensions	39 x 26.4 x 14.6 in
Weight	50 kg
Maximum Speed	1.0 m/s
Battery	Sealed Lead Acid, 24 V, 20 Ah
Runtime	8 hours (standby), 3 Hours (usage)
Sensors	Odometry, Battery levels
Actuators	Wheel Velocity, Torque

A. Sensing Implementation

One robot R_1^c senses the location coordinates and sends it to Fog server robot R^s. The goal is to find the minimum location distance from server among the collected coordinate information. The Fog server robot then runs an optimization algorithm in ROS and finds out the optimal allocation factors. The allocated ratio of computational data is sent to the respective client nodes by R^s. All the three Edge robots perform computations, in our case sorting the location coordinates using *Bubble Sort* and finding the minimum location distance from server. The computed minimum distances are returned back to the server, which then finds global minimum among the three distances and returns the result to Robot R_1^c. All the three robots then update their remaining battery power and location to server.

B. Optimization Formulation

In order to divide the computation load among the three Edge client robots in an efficient manner, the optimization algorithm is formulated with runtime communication path loss, remaining battery capacity and computational capacity as constraints. In order to prevent overloading one Edge robot with most of the computational load, another constraint is added which restricts the allocation to a fraction $q(t)$ of the total available computation load to each robot. The fraction $q(t)$ is the maximum fraction of computation that a robot shares with other peer's computational load so that its own computational requirements are not affected. It depends on the amount of battery available and hence is a temporal value.

$$\begin{aligned} \text{Minimize:} \quad & f(X) = \sum_i a_i x_i \\ \text{Such that:} \quad & \sum_i x_i = C \qquad \forall i \in N \\ & x_i \le q(t) \cdot C \qquad \forall i \in N \end{aligned} \tag{3}$$

where, a_i is the cost assigned to robot R_i^c depending on its communication path loss d_i^γ, battery power available B_p and computational capacity (CPU cores) H_i:

$$a_i = F[B_p, d_i^\gamma, H_i] \tag{4}$$

The cost values are normalized to unity. $f(X)$ is the objective function and x_i are the allocation factors assigned to each client robot R^c_i. The first constraint specifies that the sum of three allocation factors should be equal to the total available computation load C. The other three constraints restricts the computation allocated to each robot to a fraction $q(t)$ of the total available computation C.

Different scenarios are deployed and tested in ROS and the cost functions vary in the optimization problem. Note that latency is not added as an explicit constraint in this formulation, though a trade off between latency and battery consumption may also be included in this model. Further details on the optimization formulation can be found in our previous work [15].

1) Static Robots with Random Deployments: In this scenario, all the three Edge client robots are placed randomly at unequal distances from the Fog server R^s. Hence, the optimization problem takes into account constraints on the battery power and the communication path loss of client robots.

2) Mobile Robots: In this scenario, we add mobility to the three Edge client robots R^c_1, R^c_2 and R^c_3. The optimization problem is similar to the previous case, except that the distances of three client robots varies from one iteration to the other. These distances are updated after each iteration to the Fog server robot. Hence, the optimization problem in this scenario takes battery power and varying communication path loss of client robots into consideration.

3) Robots having Heterogeneous Computational Capacity: In this case, the three robots are given heterogeneous computational capacity by assigning different number of CPU cores to each robot. In our case, we make R^c_3 as a node with higher computational power (such as a smart gateway Fog node) by assigning two CPU cores (2 and 3) out of four available cores using the linux *taskset* command (`taskset -c 2,3`). The computational power of each node is communicated with the Fog server after each iteration. The optimization function is formulated as a function of ratio of cores, battery powers and communication path loss values. In this case, a_i is the function of robot's core capacity ratio, battery power and its path loss values from server robot. All these values are normalized to 1 for simulations. The maximum computation allocation constraint $q(t)$ is increased from forty percent to fifty percent in this case for better perception in results.

IV. Implementation in ROS

In this section, we provide details about integrating an optimization library NLopt in ROS and implementation of the Client-Service model in our simulations.

A. Optimization Implementation using NLopt

We use the NLopt Library [9] for implementing the optimization algorithm in ROS. The external libraries can be added to ROS by linking the target library to ROS build system in CMakeLists file as shown below:

```
FIND_LIBRARY (EXT_LIBRARY extName
  library_full_path)
target_link_libraries(executable_name
  ${catkin_LIBRARIES} ${EXT_LIBRARY})
```

We link the NLopt library to the server node in the same way by modifying CMakeLists file. Now we can use the external library directly from the server node by including the NLopt header file (`#include<nlopt.hpp>`). The NLopt library has implementations for different gradient-based and derivative-free optimization algorithms. For our scenario, we use a simple local derivative-free optimization algorithm named COBYLA (Constrained Optimization BY Linear Approximations). This algorithm supports both nonlinear equality and inequality constraints and optimizes by constructing successive linear approximations of objective function and constraint functions. This algorithm can be called using the following commands given there are three allocation vectors in the optimization problem:

```
nlopt_opt opt;
double lb[3] = {1,1,1};
opt = nlopt_create(NLOPT_LN_COBYLA, 3);
nlopt_set_lower_bounds(opt, lb);
```

where `lb` is used to set lower bounds for the three allocation vectors.
Similarly, the objective function and equality and inequality constraints can be defined and added using the following commands:

```
nlopt_set_min_objective(opt, myfunc, &costData[0]);
nlopt_add_equality_constraint(opt, eqconstr,
  &d1[0], 1e-8);
nlopt_add_inequality_constraint(opt, ineqconstr,
  &d2[0], 1e-8);
```

The detailed version of using this optimization library is explained in the NLopt tutorial [9].

B. Client-Service Model Implementation

The client server model is implemented using *ROS Services* which uses a pair of Request/Response messages as shown in Fig. 3. This is a one-to one mechanism where the client robot sends a request message and gets a response message back from the server. To create a service, we create a service file (eg: $srv - file$) and define the Request and Response fields. A service can be advertised in ROS using the following commands:

```
ros::ServiceServer service = n.advertiseService
  ("service_name", function);
```

Then, the client to the advertised service can be created in the client nodes using:

```
ros::ServiceClient client = n.serviceClient
```

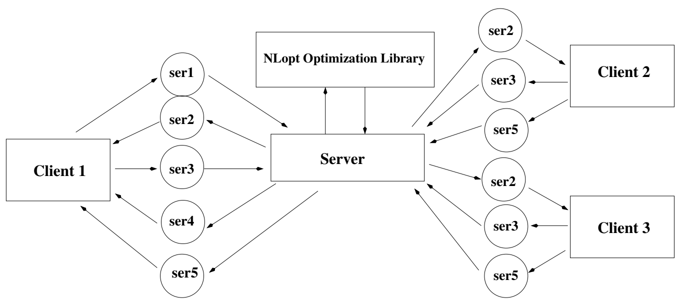

Fig. 3. Services Implemented in ROS.

```
<package_name::srv_file>
   ("service_name");
```

To call the service from client, we create an object to call the service (eg. *srv*) and the request data should be copied using this object. More detailed explanation can be found in ROS Services tutorial in ROS Wiki [14].

```
package_name::srv_file srv;
bool isSuccess = client.call(srv);
```

In our scenario, the server R^s advertises five services to the client robots (Fig. 3):

```
ros::ServiceServer ser1 = n.advertiseService
  ("sendData", add);
ros::ServiceServer ser2 = n.advertiseService
  ("optimizeAndDivide", optimize);
ros::ServiceServer ser3 = n.advertiseService
  ("sendSortedResult", sortResult);
ros::ServiceServer ser4 = n.advertiseService
  ("receiveFinalResult", finalResult);
ros::ServiceServer ser5 = n.advertiseService
  ("batteryLocationUpdate", updateDB);
```

The services `ser2`, `ser3` and `ser5` are run on the three client robots whereas the services `ser1` and `ser4` are run on the client robot R_1^c.

1) `ser1` - R_1^c uploads computational data to the server R^s requesting it to share the computation among peers.
2) `ser2` - Server offloads the ratio of allocated computation to each client for performing computations.
3) `ser3` - Each client robot sends back individually computed sorted minimum distances to the server.
4) `ser4` - The server sends the final result to client R_1^c.
5) `ser5` - The three client robot update their location and battery levels to server R^s.

Note that the services are invoked sequentially with the client waiting for a synchronous response from the server.

The sensing client R_1^c uses first service `ser1` to upload computational data to Fog server R^s as a request message. The server runs the optimization algorithm to find optimal allocation factors to all the three robots based on their cost function. Based on the allocation factor, the corresponding ratio of computational data is offloaded to each client robot in the second service `ser2`. The robots then call the third service `ser3` to send the individually computed sorted minimum distances to the server. From the three sorted coordinates, the server robot finds the final global minimum value and is offloaded to client R_1^c in the fourth service `ser4`. Then all the client robots use the fifth service `ser5` to update their location and battery levels to server R^s.

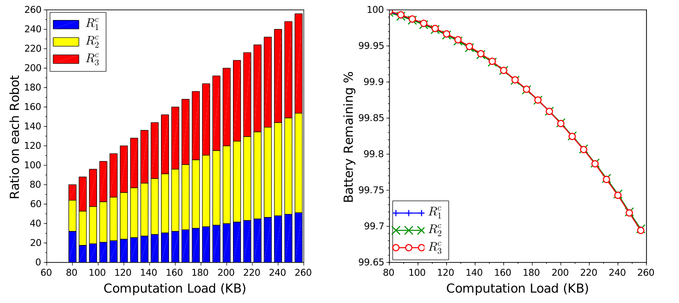

Fig. 4. Computation Division and Battery Usage for Static Nodes.

The primary advantage of implementing our solution in ROS is that it can be easily ported to a variety of robots, drones, android smart phones and smart gateways [8]. As robotic nodes come equipped with a variety of sensors and actuators, this platform may be extended to a variety of IoT deployments with runtime computation capabilities.

V. Results

In this section, the simulation results are presented for four cases. All three client robots are: (i) placed randomly and static, (ii) moving around and updating their location information to server after each iteration, (iii) assigned with different computational capacity and are static at equal locations from server robot. We also include a (iv) Baseline comparison of the distributed computation along with single robot computation and cloud offloading. The discussion of above results are also presented.

A. Static Robots

The client robots are assumed to be placed at unequal distances from server robot – robot R_1^c is placed near the server R^s. Now, the cost function takes the distance and battery capacity and optimizes the division of computational data. Since R_1^c is placed nearer to the server, the ratio allocated to R_1^c is higher for first iterations as seen in Fig. 4. Later, the battery constraint suppresses the distance constraint and hence R_1^c is allocated with less ratio (which is equal to the lower bound) to compensate battery reduction. Hence, battery depletions are almost same on all the three robots as shown in Fig. 4.

B. Mobile Robots

All three client robots are moving around the server robot, updating their locations and remaining battery capacity after each iteration. Hence, distance of three robots from

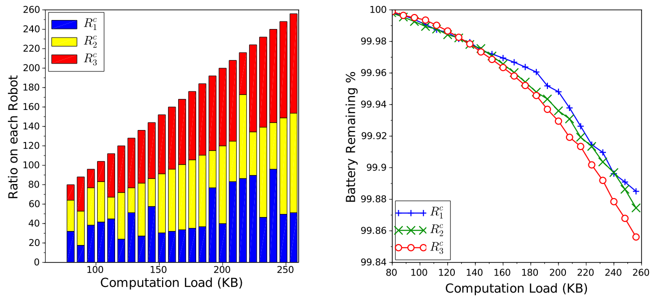

Fig. 5. Computation Division and Battery Usage for Mobile Nodes.

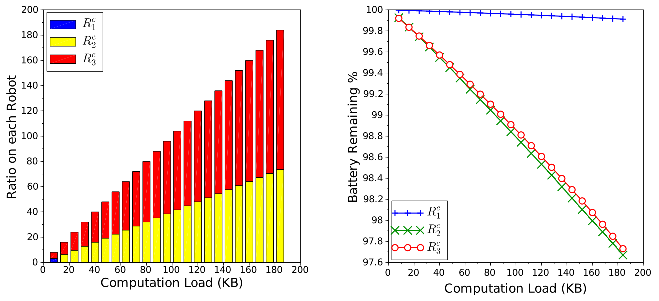

Fig. 6. Computation Division and Battery Usage for Heterogeneous Nodes.

the server robot varies dynamically after each iteration. Fig. 5 shows the battery capacity remaining after each iteration for the three robots and the ratio divided after each iteration for the three client robots. Even though the battery for client robot R_1^c is always less, dynamic distances of three robots is a factor and hence ratio allocated to each robot varies dynamically.

C. Robots with Heterogeneous Computational Capacities

In this case, the three robots are assigned with heterogeneous cores and since client robot R_3^c is assigned with more core capacity (similar to a Fog node), the ratio of computational data assigned to R_3^c is more. The maximum allocation is restricted to fifty percent because of the allocation factor $q(t)$. Fig. 6 demonstrates that assignment to robots R_2^c and R_3^c is far greater than that given to R_1^c. Note that despite the increased computational assignment, as a result of higher cores assigned to R_3^c, the battery depletion on this robot is lower than R_2^c.

D. Baseline Comparison

The computation done locally on a single robot R_1^c is compared with optimized distributed computation and the cloud computing case (where all the computational load is offloaded to cloud). We assume that the cloud is a virtual machine with 4 core CPU, 4 GB memory and a WiFi link with a maximum data rate of 54 Mbps.

1) In single robot scenario, the whole computation is performed by the robot itself using single core machine. The time taken to complete the given computational task is relatively high as can be observed from Fig. 7. The corresponding battery depletion is also high for carrying out entire computations by itself.
2) In the cloud computing scenario, the data has to be offloaded to a remote cloud based server. In spite of the improved computational cloud capacity, the communication latency is nearly twice that of the local processing. Hence, the response time recorded is higher than the optimized distributed computing scenario. The power depletion considered in this case is due to the communication power required for offloading the entire data to the cloud. Hence, battery power depletion is relatively low. Even though a reliable network has been considered for the cloud based offloading, this cannot be guaranteed in all cases. In addition, Edge and Fog nodes are severely underutilized in this scenario, as all the computations are performed on the cloud.
3) In case of distributed scenario, both the response time and battery depletion are relatively low. This is due to parallelization of data between three robots and exploiting parallel processing of computations. As can be observed from Fig. 7, the battery power in case of distributed scenario lasts longer compared to the single robot computation (190% improvement at 0.2 MB). Also, the total execution time is also very high for the singular case, demonstrating the efficacy of distributed computations (93% improvement at 0.2 MB). The improvement in latency against cloud offloading is 58% at 0.2 MB.

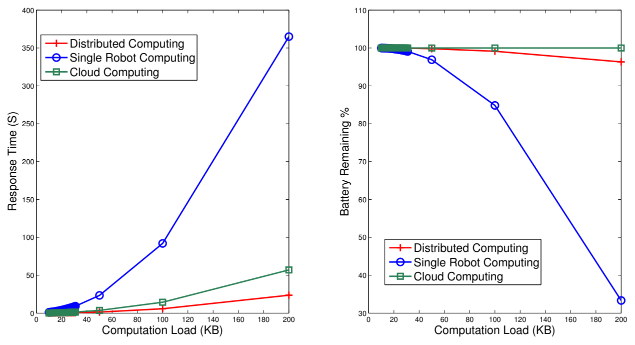

Fig. 7. Baseline Comparison with Single Node Computation.

E. Discussion of Results

While these results have been applied to mobile robotic sensors (gyroscope, accelerometer), compute nodes (coordinate sorting, mapping) and actuators (robot velocity, turn angles), these can be applied to other cases requiring runtime optimization. For the case of mobile IoT devices, location of sensors, energy levels, unused CPU cycles and so on are important metrics needing optimization. Reiterating, these results demonstrate:

1) Application of runtime optimization in varied scenarios by integrating optimization libraries within sensor-actuator IoT devices.
2) Demonstrating the utility of data and computational offloading, in conjunction with Fog computing.
3) Integrating our solution for ROS based deployment, that can be easily ported to mobile robotic nodes, android smart-phones and smart gateways.

VI. Related Work

The basic architecture of IoT is proposed in [1] and its technology aspects and applications are discussed in most of the literature [16], [17]. The generic architecture of IoT is broadly divided into 3-layers: perception layer, network layer and application layer [16]. Various versions of IoT architectures are proposed namely IoT-A [18], SPITFIRE [19], which are specific to the project domain and requirements. In [20], the heterogeneity of various architectures proposed in literature is addressed and the developments in designing a common IoT architectural framework being carried out in Europe are presented.

Towards expanding the integration of IoT with cloud computing, the authors in [21] present smart gateway based communication with Fog computing which helps in reducing burden on cloud and also reduces the latency in cloud based communication. The absolute latency provided by WiFi networks varies widely with time when compared to a cellular network [7]. The importance and benefits provided by Fog computing in IoT are stressed in [5] and [6] and have proposed a hierarchical architecture from network to Fog. In [15], the utility of Fog based offloading in resource constrained environments is analyzed.

In [3], the authors propose a cloud based architecture for networked robots with low computation power and memory and discussed the potential benefits of this architecture. The authors in [22] have considered the trade-off between the computation energy for mobile execution and communication energy for cloud execution. Using constraint optimization, they determine the optimal condition for offloading the data to cloud and prove that cloud execution is energy efficient as compared to mobile execution. Hartanto et. al. in [23] propose a cloud based publish/subscribe mechanism for a reliable communication in distributed robotic scenarios.

The authors in [24] propose a femtocloud architecture that uses mobile devices to provide cloud service at the Edge and optimizes the task assignment based on the time taken to transmit, execute and return the results back. This architecture leverages the available computation capacity of devices and also reduces the latency of transmission to the cloud. In [25], a survey of computational offloading in mobile systems is done, which reveals the improvements in performance and energy depletion rates due to computational offloading.

KalaKrishnan et. al. in [26] present an algorithm for motion planning using stochastic trajectory optimization framework. The authors in [27] develop a C++ framework for graph optimization which addresses robotics and computer vision problems like Simultaneous Localization and Mapping (SLAM) and Bundle Adjustment for integration into ROS [28].

In this paper, we integrate optimization libraries in tandem with resource constrained offloading. This is valuable in the case of latency/energy constrained deployments, such as robotic devices. With the advent of Fog and Edge computing paradigms, multiple offloading points may be optimally exploited by using our technique. As we have demonstrated this over ROS, it may be deployed on physical robots, Raspberry-pi and Android devices.

VII. Conclusion

IoT deployments with limited computation and energy resources require runtime optimization over multiple parameters. To overcome the limitations of IoT devices in terms of computation power, we use Fog computing, in which computationally intensive tasks are offloaded to Edge/Fog nodes with high computational capacities. In this paper, we have developed a Fog computing architecture for IoT scenarios for cooperative communication between peer nodes. We use ROS to demonstrate our model and integrate NLopt optimization libraries within ROS for deploying optimization algorithms during runtime. We believe that the developed framework, with integration of optimization within ROS, can be used to address wide range of application problems in the IoT world. Significant improvements in latency and battery usage are shown over non-optimal deployment of computations.

In this work, we have performed computations such as Bubble Sorting coordinate data to test our deployment. However, our future work involves extending this work to GMapping and creating a distributed global map in an optimal fashion.

References

[1] L. Tan and N. Wang, "Future internet: The internet of things," in *3rd Intl. Conf. on Advanced Computer Theory and Engineering(ICACTE)*, vol. 5, Aug 2010, pp. V5–376–V5–380.

[2] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen, "Sensor discovery and configuration framework for the internet of things paradigm," in *Internet of Things (WF-IoT)*, March 2014, pp. 94–99.

[3] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE Network*, vol. 26, no. 3, pp. 21–28, 2012.

[4] Q. Hassan, "Demystifying cloud computing," *The Journal of Defense Software Engineering CrossTalk*, pp. 16–21, 2011.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *MCC workshop on Mobile cloud computing.* ACM, 2012, pp. 13–16.

[6] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments.* Springer, 2014, pp. 169–186.

[7] J. Sommers and P. Barford, "Cell vs. wifi: On the performance of metro area mobile connections," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398808

[8] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[9] S. G. Johnson, "The NLopt nonlinear-optimization package," http://ab-initio.mit.edu/wiki/index.php/NLopt, 2014.

[10] D. Linden and T. Reddy, "Handbook of Batteries," *McGraw-Hill*, vol. 3, 2001.

[11] T. S. Rappaport, *Wireless communications : principles and practice.* Prentice Hall, 2002.

[12] D. Halperin, B. Greenstein, A. Sheth, and D. Wetherall, "Demystifying 802.11 n power consumption," in *Power aware computing and systems*, 2010, p. 1.

[13] "Husky UGV," https://www.clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/, accessed: 2016-08-22.

[14] "ROS Services," http://wiki.ros.org/Services, accessed: 2010-09-30.

[15] A. Kattepur, H. Dohare, V. Mushunuri, H. K. Rath, and A. Simha, "Resource Constrained Offloading in Fog Computing," in *ACM Middleware Workshops*, 2016.

[16] M. Yun and B. Yuxin, "Research on the architecture and key technology of internet of things (iot) applied on smart grid," in *Advances in Energy Engineering*, June 2010, pp. 69–72.

[17] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: The internet of things architecture, possible applications and key challenges," in *Frontiers of Information Technology (FIT)*, Dec 2012, pp. 257–260.

[18] "EU FP7 Internet of Things Architecture project," http://www.iot-a.eu/public, accessed: 2016-11-03.

[19] "Semantic-Service Provisioning for the Internet of Things using Future Internet Research by Experimentation, FP7," http://spitfire-project.eu/, accessed: 2016-11-03.

[20] S. Krčo, B. Pokrić, and F. Carrez, "Designing iot architecture(s): A european perspective," in *Internet of Things (WF-IoT)*, March 2014, pp. 79–84.

[21] M. Aazam and E. N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Future Internet of Things and Cloud (FiCloud)*, Aug 2014, pp. 464–470.

[22] Y. Wen, W. Zhang, K. Guan, D. Kilper, and H. Luo, "Energy-optimal execution policy for a cloud-assisted mobile application platform," *Nanyang Technol. Univ., Singapore, Tech. Rep*, 2011.

[23] R. Hartanto and M. Eich, "Reliable, cloud-based communication for multi-robot systems," in *Technologies for Practical Robot Applications (TePRA).* IEEE, 2014, pp. 1–8.

[24] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femto clouds: Leveraging mobile devices to provide cloud service at the edge," in *2015 IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 9–16.

[25] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A Survey of Computation Offloading for Mobile Systems," in *Mobile Network Applications.* Springer, 2012, pp. 1–12.

[26] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA).* IEEE, 2011, pp. 4569–4574.

[27] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *ICRA.* IEEE, 2011, pp. 3607–3613.

[28] "libg2o," http://wiki.ros.org/libg2o, accessed: 2010-09-30.