2012 IEEE International Conference on
Multisensor Fusion and Integration for Intelligent Systems (MFI)
September 13-15, 2012. Hamburg, Germany

Towards Real-time Multi-Sensor Information Retrieval in Cloud Robotic System

Lujia Wang*, Ming Liu†, Max Q.-H, Meng*, Roland Siegwart†
* Department of Electronic Engineering, The Chinese University of Hong Kong
† Autonomous Systems Lab, ETH Zurich, Switzerland
{ljwang,max@ee.edu.cuhk.hk}, ming.liu@mavt.ethz.ch, rsiegwart@ethz.ch

***Abstract*—Cloud Robotics is currently driving interest in both academia and industry. It allows different types of robots to share information and develop new skills even without specific sensors. They can also perform intensive tasks by combining multiple robots with a cooperative manner. Multi-sensor data retrieval is one of the fundamental tasks for resource sharing demanded by Cloud Robotic system. However, many technical challenges persist, for example Multi-Sensor Data Retrieval (MSDR) is particularly difficult when Cloud Cluster Hosts accommodate unpredictable data requested by multi robots in parallel. Moreover, the synchronization of multi-sensor data mostly requires near real-time response of different message types. In this paper, we describe a MSDR framework which is comprised of priority scheduling method and buffer management scheme. It is validated by assessing the quality of service (QoS) model in the sense of facilitating data retrieval management. Experiments show that the proposed framework achieves better performance in typical Cloud Robotics scenarios.**

I. Introduction

Nowadays, there is a growing need of service robots in society. However, the development of a universal robot that covers all possible services is infeasible due to limitations of power consumption, payload, sensory and kinematic constraints. Instead, it is reasonable to combine multiple robots, though with limited capabilities for each robot, to produce and access vast amounts of information in a distributed fashion. The cooperation of client robots with various abilities could make new service jointly available. A representation of the aforementioned multi-robot system is the so-called Cloud Robotics [1]. It combines robot technology with ubiquitous network and Cloud-computing infrastructure that connects amount of robots, sensors, portable devices and most important a data-center. It is an attractive model, since it allows the provision of resource retrieval and allocation.

The related research can be categorized into three major aspects: architecture design, database building and data retrieval management. Service robot systems in a Cloud usually utilize the Service Oriented Architecture (SOA) [2], where host and clients are synthesized under a star-shaped structure, which requires the management of resource allocation to run the system efficiently. Several works have been proposed in this field, many of them use web-based platform to configure the infrastructure (processing power, memory capacity, and communication bandwidth), and perform tasks. Microsoft Robotics Developer Studio (MRDS) [3] was a vital product in applying SOA to embedded systems [4] [5]. Most existing works with this framework is web-service based and database dependent. For example, Waibel et al build a web community called RoboEarth [6] for robots to autonomously share descriptions of environments and object models, as well as the tasks they have been assigned. However, as discussed in [7], Cloud interoperability faces both vertical and horizontal heterogeneity, which is usually addressed by a common middle-ware to get interoperability throughout the entire infrastructure. Nowadays, research on resource management has been limited to e-commerce and enterprise computing systems, such as resource management architectures of Eucalypus of Amazon EC2 [8] OpenNebula [9] and Nimbus [10].

In this paper, we address the following characteristics, which are supposed to be deployed in physical devices and embedded systems for Cloud Robotics.

Firstly, Time-of-Respond (ToR) in near real-time need to be considered, because sophisticated collaborate robotic tasks are usually timely sensitive. For instance, cooperative semantic mapping or 3D mapping using several robots are to be completed in real-time, though with bottle-necks in data transmission for off-line computation.

Secondly, Reliability of Response (RoR) is a key criterion of all services as well. The on-board computational capability of robot clients are usually weak. As a result, it implies two inherent requirements as follows: 1) The computation and analysis are preferred off-board from clients. Therefore, data retrieval from existing database is inevitable; 2) Consequently, the precision of the retrieved data and the reliability of the transmission is crucial. The reliability determines the potentials to extend an existing system to large scale, such as Internet-based robotic system. Typically, in large scale system, the perception results need to be shared and retrieved in time, whenever the corresponding resources are queried.

Thirdly, compatibility of data retrieval management framework is another important factor, since there are various types of robots that access to the Cloud for request. The design of the system need to take the weakness of various clients such as processor, memory and sensors into account.

Last but not least, it is preferable to have the filtered information registered to a data center once obtained, in order to avoid further re-computation. The activities of robots are usually regular, which means that the same resource from the same work space may be repetitively queried. Therefore, a local data buffer on the host will be a great help in order to realize fast response.

Though the concept of data retrieval is not new [11], [12], it is a fundamental element for the implementation of Cloud Robotic systems. Besides, data retrieval and allocation problem is compromised of the following subtle difficulties.

- Synchronization of data is hard for distributed system when different clients perform asynchronized tasks [13].
- If multi sensor data or information are distributed, resource retrieval by robot addressing is low-efficient.
- Applications on real-time systems such as Cloud Robotics are not much reported, though data fusion for multi sensor data is extensively applied for distributed systems, such as target tracking [14], [15], automated identification of targets [16], and automated behavior reasoning [17].

In this paper, centralized relation database is utilized to enable efficient retrieval of the raw data at first, Second, a software data retrieval management framework is proposed, and we set up a data buffer on host side for frequently retrieved data and predicted data to be asked in near future. At the same time, a response scheduler is used to rank priority of the requests from clients. Quality of Service (QoS) based on response time and response reliability is used as primary criterion. We show a significant improvement when the proposed framework is applied.

These technical contributions are validated by experiments on several physical robot clients that performed services based on the information retrieval. The retrieved information includes multi-type data, e.g. local map, local omnidirectional images etc. The results indicate the feasibility of real-time response for data retrieval.

The rest of the paper is organized as follows. Section II introduces the framework of MSDR in Cloud Robotics. Section III proposes formulation and solutions of MSDR in Cloud Robotics, followed by the experiments implementation of several scenarios, and the experiment results evaluation of the proposed framework and MSDR protocol is demonstrated in section IV. Section V is conclusion and future work.

II. Framework of a Cloud Robotic System

In Cloud Robotics, it is common that computational resources are allocated to distributed clients. Our expectation is that resources are shared fairly and well organized in the Cloud data center. In this section, we introduce the framework of resource allocation for Cloud Robotics as shown in Fig.1. It enables heterogeneous robots to share knowledge and services, such as retrieval of interdependent multi sensor data and management of power consumption.

A. Structure Design

Resource allocation in Cloud Robotics system is regarded as a mechanism that aims to fulfill any requirements of target applications. The requirements are usually proposed by clients who connected to the network. There are three main entities involved in the architecture for supporting MSDR for Cloud Robotic System, Robot Clients, Cloud Cluster Host (CCH) and Database.

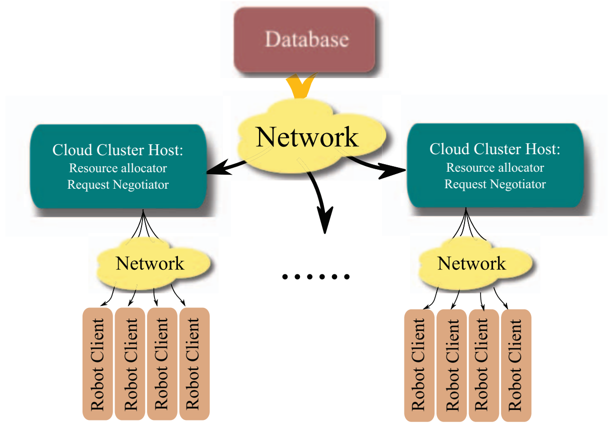


Fig. 1. Resource allocation framework for Cloud Robotics

The CCH is a server running data retrieval management. At the same time, it is the controller to accommodate unpredictable demands of data, which are parallel requests from multi robot clients. It consists of two major functionalities: Data Request Negotiator and Resource Allocator as shown in figure 1. A Data Request Negotiator deals with interface between Robot and Database. It classifies requests and provides them with different prices by pricing scheme. A Resource Allocator, which can access a data-center in Cloud, is performed as an admission controller and buffer queue manager. It distributes resources to clients regarding the priority which derived from the Data Request Negotiator. At the highest level, Database stores information such as point-cloud, images and services.

B. Management and Communication Software Platform

In order to implement the SOA of Cloud Robotic system, we compare two parallel management and communication software platforms, Hadoop MapReduce and Twisted-based network management. It is preferable to choose Twisted [18] as the software platform considering its multi thread, easy implementation and compilation to current Database.

1) Drawback of Hadoop and MapReduce: Apache Hadoop is a framework for running applications on large cluster built of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named Map/Reduce, where the application is divided into many small fragments of work, each of which may be executed or re-executed on any node in the cluster. However, DeWitt and Stonebraker considered MapReduce as a major step backwards in [19]. We summarize their primary arguments as follows, regarding Cloud Robotics applications.

- Not flexible enough for Cloud Robotics as a distributed computing and database, since robots need near real-time response to solve problems in parallel. However, in MapReduce, each of the N map instances produces M outputs files – each destined for a different reduce instance. It would be more complex when these files are written to a local disk on computer.

- It is not efficient enough for algorithms such as parallel SLAM, which requires computation on several robots. Because MapReduce includes large number of disk seeks, by which the bottle-neck of disk access significantly slows down the process. DaVinci [20] Cloud Robotics system, which claimed the efficiency of their approach, did not considere this problem in their implementation.

2) Benefits of Twisted on Parallel Communication: Twisted [18] is a framework for deploying asynchronous, event-driven and multi-thread supported network system in Python. It is compromised of the following three primary elements.

a) Reactor: The reactor is the core of the event loop within Twisted – the loop which drives applications using Twisted. The event loop is a programming construct that waits for and dispatches events or messages in a program. It works by calling some internal or external "event providers", which generally blocks until an event has arrived. It then calls the relevant event handler respectively. The reactor provides basic interfaces to a number of services, including network communications, threading, and event dispatching.

In figure 2, we use looped circles to represent the running reactors. Please note that the reactor pattern separates application code from the reactor implementation, which has been mostly predefined by Twisted. It means that application functions can be simply divided into modular, compact parts. We could also easily write specific data query and data retrieval modules for the clients and hosts respectively.

b) Protocol: The protocol defines the specifications for transmitting and receiving behaviors. Functions regarding received data and sent data can be constructed following the predefined virtual function names. A protocol begins and ends its life with two predefined virtual functions: *connectionMade* and *connectionLost*, which are called whenever a connection is established or dropped.

c) Factory: The factory is responsible for two tasks: creating new protocols, and keeping global configuration and state. In addition to abstractions of low-level system calls, it also includes a large number of utility functions and classes, which facilitates the establishment of new types of servers. Twisted includes support for popular network protocols, e.g. SOCKETS, HTTP and SMTP etc. We define several global visible variables in the Factory, which is however omitted in the figure 2 to save space. One of them is the Local Data Buffer which stores the recent requests and queries. The host factory will manage the connections to all the client reactor loops. At the same time, it is also in charge of updating the existing relation database, registering to new multi sensor readings. The detail of the framework structure and its assessment are outlined in the next section.

III. Data Flow Management and Protocol Design

In this section, we introduce the proposed framework, followed by defining criteria for evaluation of MSDR problem.

A. Data Flow Management

The data flow of multi data retrieval and communication in our Cloud Robotic system is shown in Fig. 2. The host and clients are built on Twisted framework. In Twisted program, it includes a main loop called reactor and a callback system. This system automatically launch new thread for each client which attempts to connect to the network with approved address and port. We only need to define the specific functionalities in the host and clients separately. Amongst, the major functions are introduced as follows.

1) Database Query: This function is launched and managed only by host. It utilizes standard SQL [21] syntax to retrieve target information from a dynamic updated relation database. Therefore, the database access may be one of the bottle-neck in the system. Through the management of host, the bottle-neck is supposed to be alleviated. To this end, we use the following two sub-functions to assist the retrieval.

2) Filter and Pre-process: In the proposed data flow structure, the filter and pre-process blocks stand for general data pre-process, for example, data fusion, feature fusion and decision fusion [22], [11]. They are the major means to decrease the frequency of database access and reduce the data noise. At the same time, it solves the reasoning and predication problem for specific applications.

3) Buffer Management: As depicted in Fig. 2, a Local Data Buffer is deployed for storage of the frequently requested data. Since activities of robots are usually regular, the same resource may be queried repetitively. Therefore, we build such a buffered mechanism will help to alleviate the database access bottle-neck to some extent.

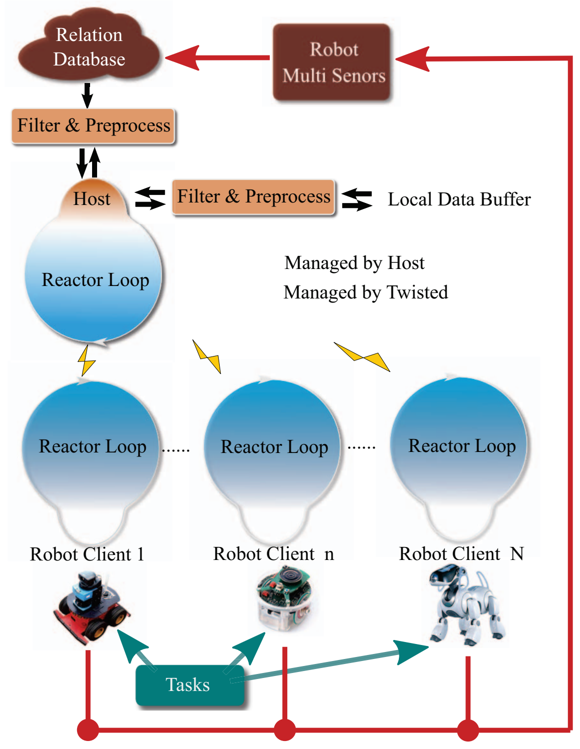


Fig. 2. Data Flow of Multi Data Retrieval and Communication in Cloud Robotic System

B. Quality of Service (QoS)

Generally, QoS is used to assess the performance of a SOA. It advertises performance quality levels of service which is provided by service providers; at the same time, clients use it to select an optimal candidate data/service, which could in part fulfill the request. Therefore, a well-defined set of QoS's will greatly help the assessment of the quality of a service framework. In common cases, bandwidth usage is one of the most important factor to define QoS, since the response of most network-based applications are sensitive to it. In Cloud Robotic systems, it is not the case. Moreover, instead of taking bandwidth usage as the only criterion, QoS definition can be extended to other aspects regarding the processing or storage capabilities of nodes. We selectively define the following QoS's as primary criteria to assess the proposed framework.

- **Definition 1 - Time of Response (ToR)**
  ToR is the needed time for a client to received a response after a request has been sent. It is formulated as.
  $$ToR = T_{Data\_received} - T_{Request\_sent}$$
- **Definition 2 - Reliability of Response (RoR)**
  RoR is a measure of confidence that the retrieval data is free from errors. Its value is given in percent and calculated as follows.
  $$RoR = \frac{\#_{Succeeded\_Requests}}{\#_{Total\_Requests}}$$

C. Protocol for Multi Sensor Data Retrieval and Scheduling Problem Model

The paradigm of Cloud Robotics enables harnessing large numbers of low-end robots working in parallel to make use of relation database, in order to share knowledge, or solve a problem. Therefore, for multi data retrieval management, a protocol which includes admission control and scheduling policy is necessary to optimize the QoS in term of fast response, reliable response, fairness of allocation etc.

1) Admission Control: When a service request is first submitted, Request Negotiator utilizes the proposed admission control mechanism to interpret the submitted request before determining whether to accept or reject it regarding QoS requirements. Thus, it ensures that there is no overloading of information, where overwhelmed client requests can not fulfilled successfully due to limited resources.

In Twisted Factory, we set a threshold for admitted number of robot client according to CPU, bandwidth usage and ToR.

2) Scheduling Problem Formulation: As the number of services and data increases, efficiency of multi data retrieval becomes more challenging. Therefore, we formulate the scheduling problem with QoS constraints and provide with concise solutions in this paper.

The goal is to minimize total cost of information retrieval for each task. The cost is evaluated by completing time of all request tasks. Suppose that n clients share m resources. Each task C_i consists of k_i parallel and dependent requests. Each resource R_j has a fixed price p_j according to its capacity and expense. The a_{ij} is the amount of requests of client C_i allocated to resource R_j. Therefore, the total cost of task C_i is $u_i(a_i) = \omega_t \max_{t_{ij} \in t_i} t_{ij} + \omega_e \sum_j e_{ij}$. The utility of task C_i defined as:

$$u_i(a_i) = \frac{1}{\omega_t \max_{t_{ij} \in t_i} t_{ij} + \omega_e \sum_j e_{ij}} \tag{1}$$

where t_{ij} denotes the response time, ω_t and ω_e denote the weights of completion time and expense respectively.

Regarding game-theoretic studies on the task scheduling problem, players maximize their returns which depend on actions of other players. In our market mechanism, each task is a market participant, a_i is the strategy of request i. Given complete knowledge of the system, it would be natural for each participant to try to solve the following optimization problem:

$$\begin{gathered} \max u_i(a_i) \\ st. \sum_{a_{ij} \in a_i} a_{ij} = k(i) \\ a_{ij} \geq 0 \end{gathered} \tag{2}$$

Our assumptions are: the capacity of each resource is inelastic and undividable; clients are independent and don't communicate with each other. Please note that bandwidth cost in communication is not taken into consideration.

Nash equilibrium of the resource allocation game always exists when each request solves its optimal problem independently without considering the multiplexing resources [23],

3) Scheduling implementation in Negotiator: The Resource Allocator is the module run in the Twisted Factory. It is responsible for making resource management decisions. Having access to all robot clients allocation requests, the Resource Allocator can keep track of resource allocations for all the current clients, and make decisions about incoming robot clients requests simultaneous.

Basic operation policy of the Request Negotiator is demonstrated in Table I. The primal objective of the policy is to provide an index for ranking candidate requests. It can be made with considering quantity of free CPU, bandwidth and willingness payment of each request, etc. For example, when a request from robot client arrives, the Negotiator filters those requests from a rank list which records the minimum requirements. Then it creates a new list of candidate requests. Finally, the requests in the rank index is responded in parallel.

TABLE I
SCHEDULING ALGORITHM

Algorithm 1: Scheduling Algorithm

Inputs: willingness payment of request p_i, request ID i
Outputs: current_priority_list

```
1 BEGIN
2    function update_priority_list($p_i$)
3        current_priority_list.append($p_i$)
4    function is_lowest_priority($p_i$)
5        current_priority_list.sort($p_i$)
6    while $i \leq n_{threshold}$
7        update_priority_list($p_i$)
8        is_lowest_priority($p_i$)
9    return current_priority_list
10 END
```

IV. Experimental Validation

In this section, we set up the experiment and evaluate our framework with two scenarios for MSDR in Cloud Robotic system.

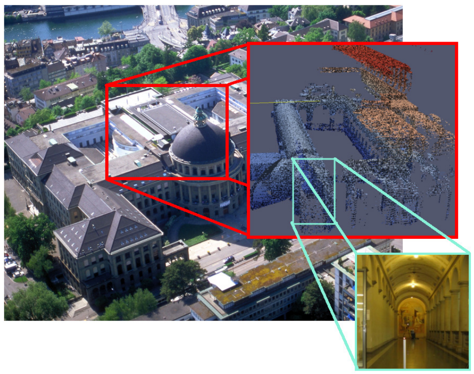


Fig. 3. 3D point-clouds and image instance of ETH HG building

A. Experiment Design

The proposed system enables several low-end robots without 3D laser working in parallel to get a 3D map, which is built by a high-end robot with elaborate 3D laser scanner. Details are as follows.

- Build a relation database of 3D map and image data of ETH HG floor as shown in Fig. 3 which derives from a high-end robot. All these data is stored in a collection of system catalogs and can be queried (in PostgreSQL) by any user to uncover such structure.
- Provided with its pose, each low-end robot send several requests once for maps or images to the host, which can access to the database and match the target data.
- Each request is managed by the host with predefined with scheduling algorithm and protocol mechanism.
- Requested messages include navigation odometry messages, navigation occupancy grids, rosgraph logs, battery states, sonar arrays and slam gmapping entropies. These message types are definitions adopted from ROS [1].
- We generate SQL requests according to ID and timestamp according to the data storage in the relation database.
- ToR is utilized to evaluate the experiment results.

TABLE II
Configuration of Host and Clients in Experiment

Node	CPU	Memory	Hard Disk
Host	Intel(R) Core(TM) i5 2540M processor 2.60GHz	8GB	350GB
Robot 1	Intel(R) Pentium(R) M processor 1000MHz	236.2MB	35GB
Robot 2	Intel(R) Core(TM) 2 Duo processor 3.00GHz	6GB	140GB

The test is carried out a Twisted multi thread loop which we introduce in section II. For comparison, different configurations of clients are selected as shown in Table II.

[1]http://www.ros.org

B. Experiment Results

In the first scenario, several robots attempt to query messages of large binary objects which are dense for network. We compare the CPU and bandwidth usage when there is buffer queue management in the CCH of host as shown in Fig.1. In this case, we use the local buffer to factory which saves the recent queries. Since robot 1 equips with lower configuration hardware while robot 2 has higher configuration. When they are requesting the same type of map message with large size from host, Fig. 4(a) depicts that robot 1 has a higher CPU usage than robot 2. Moreover, since robot 1 asks for message first and there is no message in buffer, the bandwidth usage is high while host processes its request and queries database. When the identical request of robot 2 arrives, such data persists in the buffer. Thus host is relieved from query again, which is bandwidth saving as shown in Fig. 4(b). We can also get the conclusion that QoS of ToR is greatly optimized.

In the second scenarios, we build several requests from clients, and set the threshold such that the maximum number of requests is 6 in the host. Then there can be only 6 requests to access, other requests with priority large than 6 can only wait until the previous ones received data and removed from the queue. All requests announce willingness to pay for the request. The higher willingness payment, the earlier response. We compare the ToR when several requests of multi type messages come and host implements a priority scheduler on it as shown in Table. I.

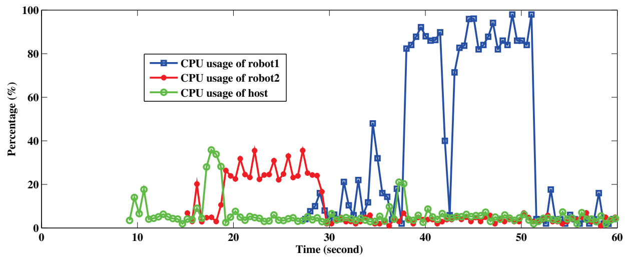
Table III demonstrates three different types of messages with quite different size. We set the priority order of request as $\{$map, pose, tf$\}$ in terms of data size. In Fig.5, it depicts ToR of different data types with and without buffer in the host. The results show that it has reduced more than half when there is a local buffer. Especially, with higher priority map message request get a faster response, although it is larger in data size than those with lower priority. Moreover, according to the column of Rank index of response, we can derive that whenever the host receives request from the high priority one, it delayed the response of following low priority requests. However, the Rank index has changed in two tests because data storage in buffer queue changed. Therefore, buffer queue management need further control if the request expect to be responded accurately in terms of their request priority. In all, if there is a buffer stored the frequent request data, we get a faster response in general.

V. Conclusion

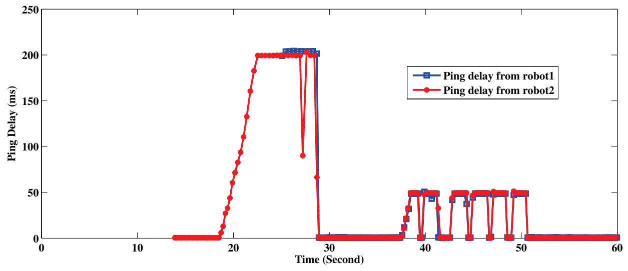
In this paper, we have presented the design, the implementation, and the evaluation of MSDR in Cloud Robotic systems. A priority scheduling method and a buffer management is proposed in the CCH module. The experiment results demonstrate significant improvement of the proposed framework in terms of ToR, CPU and bandwidth usage. For future study, we will explore a scheduler concerning dynamic priority, which includes a prediction model for the possibly required data in the next time interval. We aim at the optimization of the total

TABLE III
COMPARISON OF RANKING INDEXING OF RESPONSE

Request	**Data type**	**ROS tiopic**	**Request**	**Data size**	**Rank index of response**	
squence	(message)		**priority**	(Bytes)	Index 1	Index 2
1	odometry	pose	2	3825	2	4
2	odometry	pose	5	3829	1	1
3	navigation	map	1	19681557	4	3
4	navigation	map	4	19681561	3	5
5	transformation	tf	3	689	6	2
6	transformation	tf	6	763	5	6



(a) CPU Usage Comparison between with and without Scheduler



(b) Bandwidth Usage Comparison between with and without Scheduler

Fig. 4. Experiment results

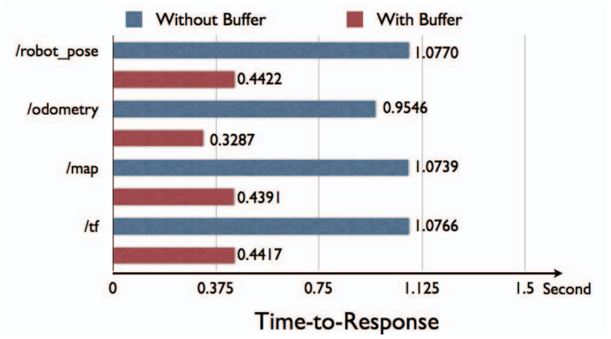


Fig. 5. Time-of-response comparison between with and without host buffer

utility for both Cloud and robot clients. Extended experiments on a larger number of robots will also be carried out.

REFERENCES

[1] E. Guizzo, "Robots with their heads in the clouds," *Spectrum, IEEE*, vol. 48, no. 3, pp. 16–18, march 2011.

[2] M. Bichier and K.-J. Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99–101, march 2006.

[3] M. Corporation, "Microsoft robotics developer studio." [Online]. Available: http://www.microsoft.com/Robotics

[4] W. Tsai, Q. Huang, and X. Sun, "A Collaborative Service-Oriented Simulation Framework with Microsoft Robotic Studio," in *Simulation Symposium, 2008. ANSS 2008. 41st Annual*, april 2008, pp. 263–270.

[5] "An ontology-based collaborative service-oriented simulation framework with microsoft robotics studio," *Simulation Modelling Practice and Theory*, vol. 16, no. 9, pp. 1392–1414, 2008.

[6] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. 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, 2011.

[7] A. Sheth and A. Ranabahu, "Semantic Modeling for Cloud Computing, Part 1," *Internet Computing, IEEE*, vol. 14, no. 3, pp. 81–83, may-june 2010.

[8] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus : A technical report on an elastic utility computing architecture linking your programs to useful systems," 2008.

[9] O. penNebula Project, "Opennebula.org - the open source toolkit for cloud computing." [Online]. Available: http://www.opennubula.org/

[10] P. Sempolinski and D. Thain, "A comparison and critique of eucalyptus, opennebula and nimbus," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 30 2010-dec. 3 2010, pp. 417–426.

[11] D. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[12] R. Luo and M. Kay, "Multisensor integration and fusion in intelligent systems," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 5, pp. 901–931, 1989.

[13] N. Kaempchen and K. Dietmayer, "Data synchronization strategies for multi-sensor fusion," in *Proceedings of the IEEE Conference on Intelligent Transportation Systems*. Citeseer, 2003.

[14] A. Brooks and S. Williams, "Tracking people with networks of heterogeneous sensors," in *Proceedings of the Australasian Conference on Robotics and Automation*. Citeseer, 2003, pp. 1–7.

[15] Y. Bar-Shalom, "Multitarget-multisensor tracking: advanced applications," *Norwood, MA, Artech House, 1990, 391 p.*, vol. 1, 1990.

[16] D. Klimentjew, N. Hendrich, and J. Zhang, "Multi sensor fusion of camera and 3d laser range finder for object recognition," in *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2010 IEEE Conference on*. IEEE, 2010, pp. 236–241.

[17] M. Kam, X. Zhu, and P. Kalata, "Sensor fusion for mobile robot navigation," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 108–119, 1997.

[18] G. L. Moshe Zadka, "The twisted network framework," 2010. [Online]. Available: http://twistedmatrix.com/user/glyph/ipc10/paper.html

[19] D. J. DeWitt and M. Stonebraker, "Mapreduce: A major step backwards," 2008. [Online]. Available: http://www.hpts.ws/papers/2009/session10/shekita.pdf

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

[21] C. Date and H. Darwen, *A Guide to the SQL Standard*. Addison-Wesley Reading (Ma) et al., 1987, vol. 3.

[22] B. Dasarathy, *Decision fusion*. IEEE Computer Society Press, 1994, vol. 1994.

[23] G. Wei, A. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation forcloud computing services," *The Journal of Supercomputing*, vol. 54, pp. 252–269.