AFRL-AFOSR-VA-TR-2016-0149

Cloud-Based Perception and Control of Sensor Nets and Robot Swarms

Geoffrey Fox
TRUSTEES OF INDIANA UNIVERSITY

04/01/2016
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
22-03-2016

2. REPORT TYPE
FINAL

3. DATES COVERED *(From - To)*
09/30/2013 - 12/31/2015

4. TITLE AND SUBTITLE
Cloud-Based Perception and Control of Sensor Nets and Robot Swarms

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-13-1-0225

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Fox, Geoffrey, C.
Crandall, David

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TRUSTEES OF INDIANA UNIVERSITY
INDIANA UNIVERSITY
205C BRYAN HALL, 105 N INDIANA AVE
BLOOMINGTON IN 47405-1106

8. PERFORMING ORGANIZATION REPORT NUMBER
00118707; 063210-00002B

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
USAF, AFRL DUNS 143574726
AF OFFICE OF SCIENTIFIC RESEARCH
875 NORTH RANDOLPH STREET, RM 3112
ARLINGTON VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFOSR

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFOSR/PKR2

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project "Cloud-Based Perception and Control of Sensor Nets and Robot Swarms" was performed by an interdisciplinary team at Indiana University with expertise in cloud and parallel computing, computer vision and robotics. It investigated the use of Cloud Computing as a key technology for Internet of Things (IoT) and DDDAS applications. The project developed an open source framework called IoTCloud to connect IoT devices to cloud services and used it to investigated three algorithms controlling robots from the cloud. Our three major applications were a parallel particle filtering based SLAM algorithm; a deep learning based drone control algorithm; and a robot swarm algorithm for n-body collision avoidance. These applications had significant time complexity needing the additional computer power offered by the cloud and we parallelized applications so that they could respond quickly to the edge devices. This project produced six major papers and a report. It also contributed to the STREAM2015 workshop and its final report. The final report summarizes highlights here of published work and give some comments for follow on activities.

15. SUBJECT TERMS
sensors, performance, cloud computing for DDDAS applications, robot swarm algorithm, parallel particle filtering, SLAM algorithm, deep learning, drone control

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON / 19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	8	Dr. Frederica Darema, AFOSR/RTC / (703) 588-1926

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18
Adobe Professional 7.0

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

Final Report AFOSR FA9550-13-1-0225:

Cloud-Based Perception and Control of Sensor Nets and Robot Swarms

Geoffrey Fox, David Crandall

Indiana University, March 2016

1. Introduction

This project investigated the use of Cloud Computing as a key technology for Internet of Things and DDDAS applications. We developed an open source framework called IoTCloud[1] to connect IoT devices to cloud services and used it to investigated algorithms controlling robots from the cloud. These were major applications needing the additional computer power offered by the cloud and we parallelized applications so that they could respond quickly to the edge devices. This project produced six major papers [2-7] and a report [8]. It also contributed to the STREAM2015 workshop and its final report [9]. We summarize highlights here of published work and give some comments for follow on activities.

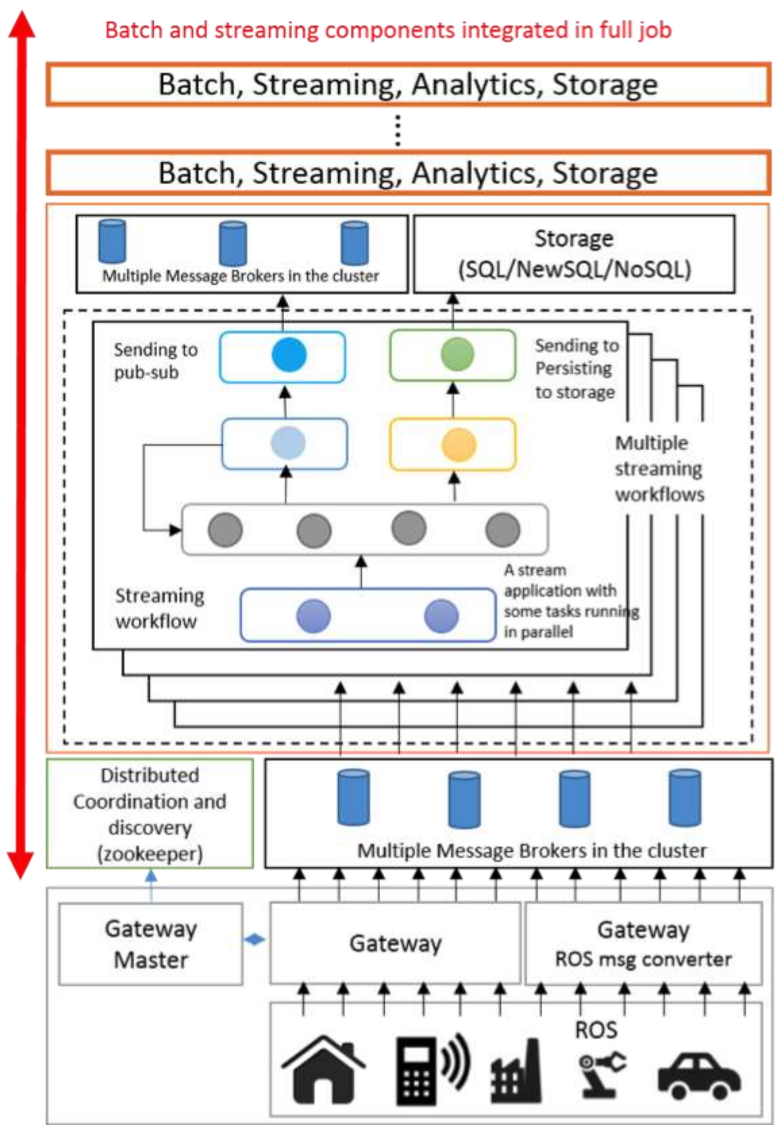


Figure 1 IoTCloud Architecture

IoTCloud consists of: a set of distributed nodes running close to the devices to gather and do initial processing (sometimes called fog layer) of the data, a set of publish-subscribe brokers to relay the information to the cloud services, and a distributed stream processing framework (DSPF) coupled with batch processing engines in the cloud to process the data and return (control) information to the IoT devices. Real time applications execute data analytics at the DSPF layer achieving streaming real-time processing. Our open-source IoTCloud platform [5] uses Apache Storm [10] as the DSPF, RabbitMQ [11] or Kafka [12] as the message broker and an OpenStack academic cloud [13] (or bare-metal cluster) as the platform. To scale the applications with number of devices we need distributed coordination among parallel tasks and discovery of devices; both were achieved with a ZooKeeper [14] based coordination and discovery service.

In general a real time application running in a DSPF can be modeled as a directed graph consisting of streams and stream processing tasks. Stream tasks are at the nodes of the graph and streams are the edges connecting the nodes. A stream is an unbounded sequence of events flowing through the edges of the graph and each such event consists of data represented in some format. The processing tasks at the nodes consume input streams and produce output streams. A distributed stream processing framework provides the necessary API and infrastructure to develop and execute such applications in a cluster of computation nodes. The main tasks of a DSPF include 1) Providing an API to develop streaming applications, 2) Distributing the stream tasks in the cluster and managing the life cycle of tasks, 3) Creating the communication fabric, 4) Monitoring and gathering statistics about the applications, and 5) Providing mechanisms to recover from faults. These frameworks generally allow the same task to be executed in parallel and provide rich communication channels among the tasks. Some DSPF's allow the applications to define the graph explicitly and some create the graph dynamically at run time from implicit information.

For most streaming applications, latency is of utmost importance and the system should be able to recover fast enough from faults for normal processing to continue with minimal effect to the applications. A detailed study of recovery methods possible for streaming applications is available in [15]. In our work, we term real time applications that produce correct answers but violate timing requirements as having performance faults. Our research addresses (with the same mechanisms) both explicit hardware/software and performance faults.

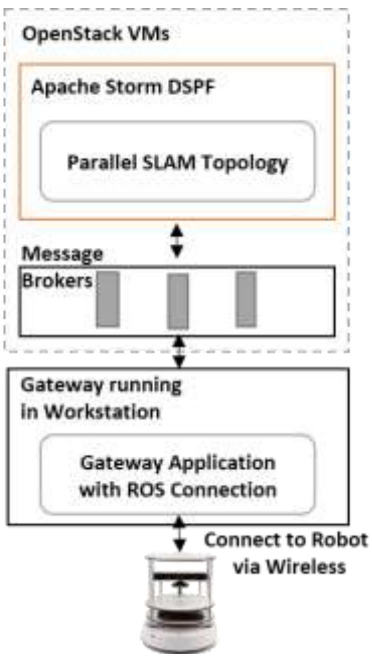


Figure 2 Parallel SLAM Application

We are exploring cloud controlled real time IoT applications in two distinct dimensions. In one dimension there are computationally intensive algorithms for processing device data that can benefit from cloud based processing for real time response. These methods are powerful but impossible to run near the devices due to high computational and specialized hardware requirements. In the other dimension there are applications that have to be scaled to support vast numbers of devices and are inherently suitable for central data processing. We have developed a parallel particle filtering based SLAM [16, 17] algorithm [6] and deep learning based drone [18] control algorithm [4], which both fit into the first category. As an application of the second category, we have developed a robot swarm algorithm [3] for n-body collision avoidance [19-21] that can scale for a large number of robots. In all three cases, we have working versions with good performance characteristics and papers published or under consideration. The parallel SLAM and n-body collision avoidance algorithms use Turtlebot [22] as the robot and ROS [23] as the SDK for connecting to the robot. The overall parallel SLAM application is shown in Figure 2.

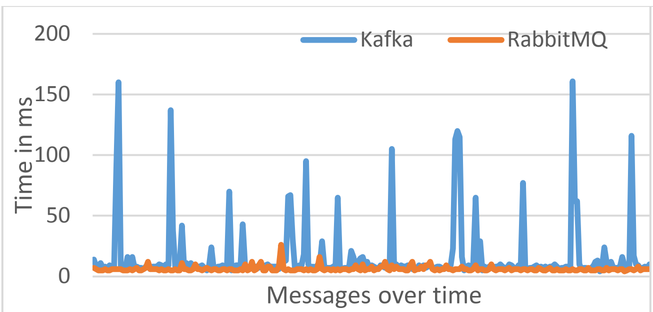
Through our work in developing these applications, we have identified shortcomings in the current technologies, based on current and future requirements. These imply IoTCloud extensions, termed IoTCloud++, that can give scaling with performance guarantees and represent possible future research illustrated by our recent extensions [7] to Apache Storm to improve its communication performance. Our future work includes more extensive performance testing and additional applications.

2. Streaming Application DDDAS Challenges for IoT Cloud Controller

We present five categories of streaming DDDAS applications based on challenges they present to the backend Cloud control system.

1) Set of independent events where precise time sequencing unimportant. **Example:** independent search requests or tweets from users
2) Time series of connected small events where time ordering is important. **Example:** streaming audio or video; robot monitoring
3) Set of independent large events where each event needs parallel processing with time sequencing not critical **Example:** processing images from telescopes or light sources with material or biological sciences.
4) Set of connected large events where each event needs parallel processing with time sequencing critical. **Example:** processing high resolution monitoring (including video) information from robots (self-driving cars) with real time response needed
5) Stream of connected small or large events that need to be integrated in a complex way. **Example:** tweets or other online data where we are using them to update old and find new clusters rather just classifying tweets based on previous clusters as in 1), i.e. where we update model as well as using it to classify event.

6)



7) **Figure 2a**

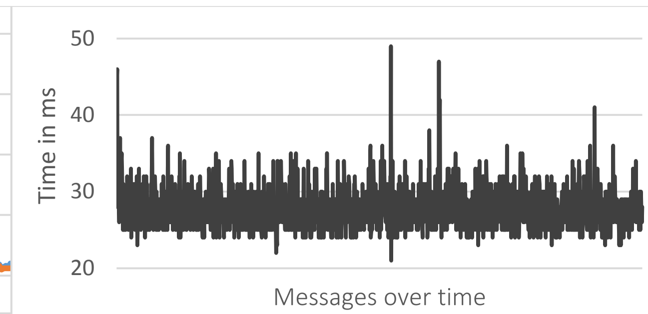
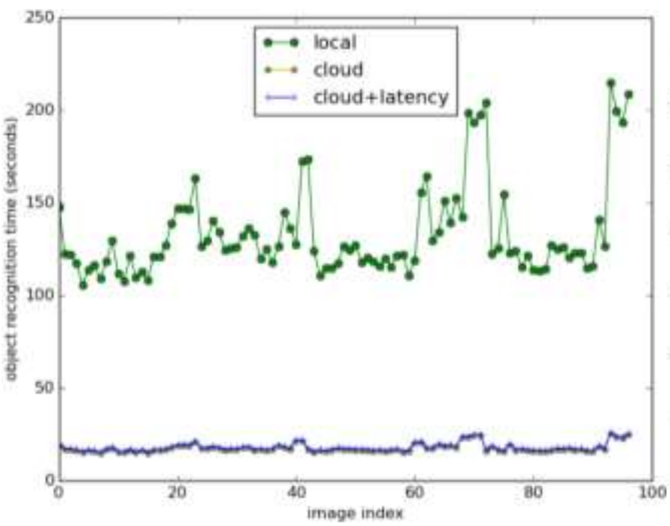


Figure 3b

Figure 3a Fluctuations in Time of IoTCloud using RabbitMQ and Kafka with Minimal Processing in Storm
Figure 3b Fluctuation in Time of IoTCloud with processing Kinect data from TurtleBot with RabbitMQ



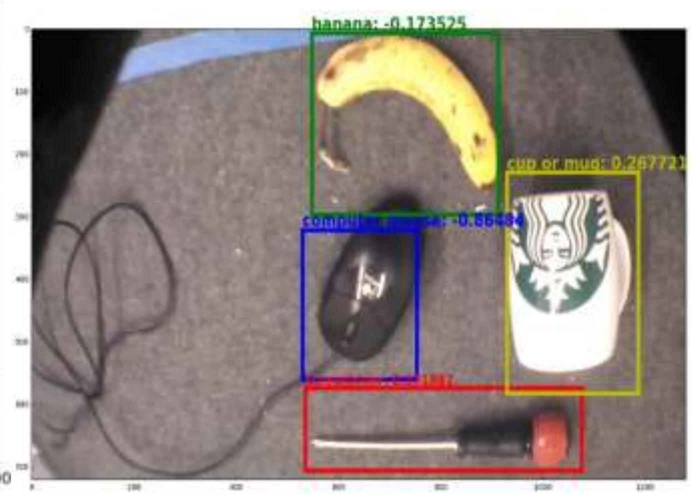


Figure 4 a) Performance of Cloud-based Deep Learning and 4 b) Typical region split and recognition of multiple objects in a single image.

These 5 categories can be considered for single or multiple heterogeneous streams. Our initial work has identified difficulties in meeting real time constraints in cloud controlled IoT due to either the intrinsic time needed to process events or due to fluctuations in processing time caused by virtualization, multi-stream interference and messaging fluctuations. Figure 3a shows the fluctuations we observed with RabbitMQ and Kafka with minimal processing in

Apache Storm and Figure 3b show fluctuations in processing 3d point cloud Kinect data in Storm from a Turtlebot with RabbitMQ. Large computational complexity in event processing is naturally addressed by using parallelism in the Storm bolts, but that also can lead to further sensitivity to fluctuations. Currently IoTCloud can handle 1) automatically and 3) with user designed parallelism. The other cases require careful tuning on a case by case basis and still can see unexpected large fluctuations in processing time that currently we do not address except by over-provisioning.

Category 4) is illustrated by our work on deep learning for drones. The idea is that state of the art deep learning-based object detectors can recognize among hundreds of object classes and this capability would be very useful for mobile devices, including robots. However as a model for a single object can have billions of parameters, the compute requirements are enormous with classification requiring ~20 sec/image on a high end CPU, and ~2 sec/image on a high-end GPU. Our results using Regions with Convolutional Neural Networks CNNs (R-CNNs) trained on ImageNet are shown in figure 4. Note for this problem latency is unimportant as the cloud processing time is so long.

A future IoTCloud++ will enhance IoTCloud to allow real-time guarantees and fault tolerance in both execution and performance. We will achieve this autonomic behavior by allowing dynamic replication and elastic parallelism in a self-monitored environment. This work will be delivered as an enhancement to Storm extending the work in [7].

4. Related Work

Industry is realizing the need of data analytics driven approaches to support efficient operations at all levels to reduce the costs and increase innovation. The machines are getting intelligent with software controls and communication to outside services. Industry can benefit immensely from real time central management to deploy, manage, upgrade, and decommission these intelligent machines. Concepts like Brilliant machines [24] by GE Software are pushing the industry towards such connected and intelligent infrastructure. A Brilliant machine connected to the Industrial Internet of Things can run software that will make the machines react to changes in data and its environment both in operation and configuration and can communicate with other machines. Software Defined Machines (SDM) is a software environment to program such machines with a generic API hiding the underlying details such as hardware details. A SDM for a brilliant machine can run close to the machine or can be hosted in the cloud. Having generic distributed open platforms such as IoTCloud to execute both data analytics and SDMs in cloud will be beneficial for such applications.

Distributed stream processing provides frameworks to deploy, execute and manage event based applications at large scale. Many years of research [8] have produced software frameworks capable of executing distributed computations on top of event streams. Examples of such early event stream processing frameworks include Aurora [25], Borealis [26], StreamIt [27] and SPADE[28]. With the emergence of Internet scale applications in recent years, new distributed stream processing systems like Apache S4 [29], Apache Storm [10], Apache Samza [30], Spark Streaming [31] and commercial solutions including Google Millwheel [32] and Amazon Kinesis [33] have been developed.

Apache Storm applications are developed in the model of the graphical dataflow we introduced earlier. A Storm application consists of Spouts, Bolts, and Streams. Spouts and Bolts are the nodes in the graph connected by streams and a single such application is called a Topology. Storm uses its own servers to manage and distributes the tasks among the cluster nodes. The communication fabric is built on top of TCP using the Netty library. Storm provides at least once processing guarantees at its core. Apache Samza is another open source stream-processing framework developed on top of Kafka message broker and Apache Yarn. Samza applications are similar to Storm applications in the graph structure, and differences between Storm and Samza include technical details in how they distribute the tasks and how they manage the communications. Because the Samza messaging layer is backed by a file based message broker Kafka, its latency is expected to be higher compared to other processing engines.

Apache Spark streaming extends the Spark batch processing system. Spark is a batch processing system targeting iterative algorithms and interactive analytics problems on top of large data sets. In the streaming case, Spark reads input from a stream source like a message queue. It uses small batches of incoming data as input to the running jobs, creating the illusion of continuous processing. Such batching of the inputs is not very attractive for real time applications. S4 is another fully distributed real time stream processing framework. The processing model is inspired by map-reduce and uses a key-value based programming model. S4 creates a dynamic network of processing elements (PEs) and these are arranged in a DAG at runtime. One of the biggest challenges in the PE architecture is that key attributes with very large domains can create large numbers of PEs in the system at any given time.

A comprehensive list of optimizations possible to reduce the latency of the stream processing applications are mentioned in [34]. These optimizations include features like operator reordering, load balancing, fusion, fission, etc. All the operations mentioned are targeted towards optimizing the average performance metrics of the system. For real time applications individual tuple latency is also very important.

There are many open source message brokers available that can act as gateways to the stream processing platforms. Such brokers includes ActiveMQ [35], RabbitMQ [11], Kafka [12], Kestrel, and HornetMQ. ActiveMQ, RabbitMQ, Kestrel and HornetMQ, are all in memory message brokers with optional persistent storages. On the other hand, Kafka is a store first broker backed by a message log. Compared to other message brokers Kafka has better parallel consumption semantics, scalability and fault tolerance due to its topic partition and replication across the cluster. Our measurements [2] showed that RabbitMQ illustrated in fig. 2 has comparable or superior performance compared to other brokers and Kafka has large fluctuations in latency. We should revisit this question when the performance enhancements of IoTCloud++ are implemented.

Implementing real time applications with critical time requirements in the vanilla Java virtual machine is a challenge itself due to garbage collection and other unpredictable factors. There has been efforts to improve the Java runtime and JDK to fit these requirements [36-38]. Most of these studies are related to real time requirements in embedded systems that control the devices. In our platform the actual software controlling the IoT devices will be running near the device and the cloud processing will enhance this processing for stages where some latency (~few 100ms) can be tolerated.

Robot Operating System (ROS) is an open source platform that offers a set of software libraries to build robotics applications. Popular off the shelf robots have ROS applications already written and these applications combined with the available wide range of tools such as visualization tools and simulators create a powerful environment for researchers. In some of our cloud applicationsm we use ROS as the first layer to connect to the robot, collect data and control it. We transform the ROS data structures to data structures required by cloud applications at the gateways.

Open standards like MQTT [39] and MTConnect [40] are being developed to bridge the gap between the application data requirements and the device data. IoTCloud support the MQTT transport and can transfer data between gateways and cloud using MQTT. If the devices send data with the MQTT protocol, they can be sent without transformation at the gateways directly to the cloud.

5. Proposed Research Plan for Robust Open Source Cloud IoT Controller with Real Time QoS

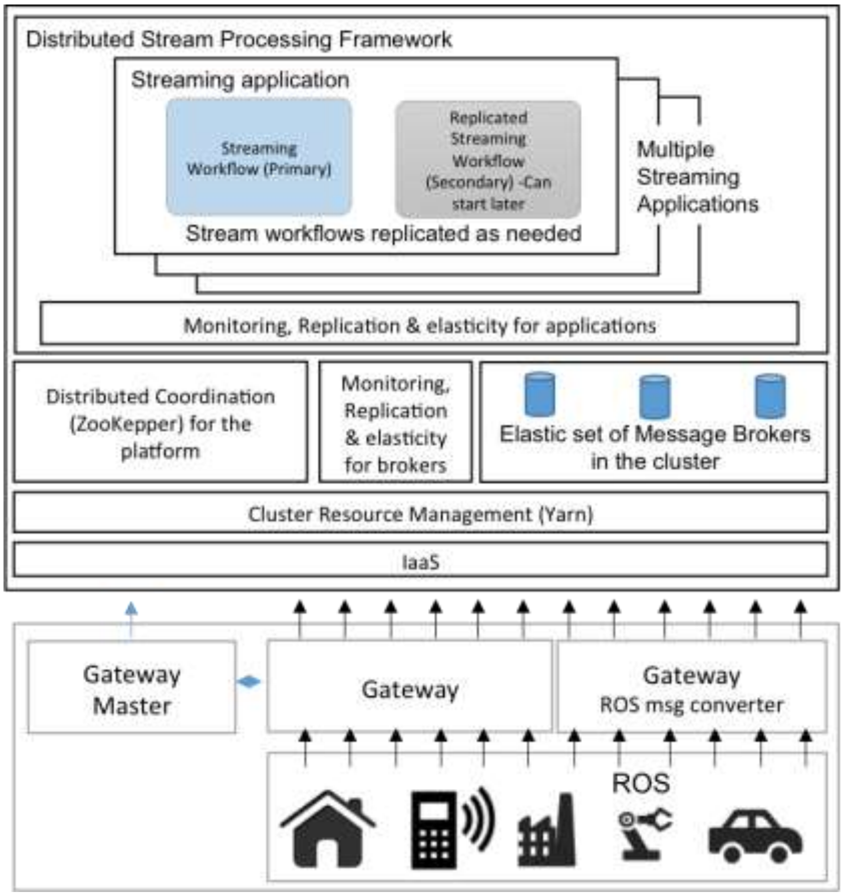


Figure 5 IOTCloud++ Architecture

Our research has identified the need to achieve real-time QoS in spite of fluctuations in computation time and we have designed IoTCloud++ to address this, although we did not have resources to address this outside the recently published extensions to Storm [7]. The architecture of the new IoTCloud++ platform is shown in Figure 5. In this architecture, we propose to dynamically replicate the streaming computation tasks within cloud clusters to achieve good performance in at least one replica. This replication will not be universal but rather done only when achieving QoS demands it as for example when monitoring shows that initial task is delayed. This as-needed replication will drastically reduce overhead from replication in many cases. We will dynamically identify the streaming tasks that require replication and replicate at the task level rather than at the streaming application level. This dynamic replication of streaming tasks will be implemented for Apache Storm described above. Apache Storm consists of two types of servers called Nimbus and Supervisor. Nimbus manages the streaming applications running in the cluster. Each Supervisor consists of a fixed number of workers capable of executing the stream tasks belonging to a job. To dynamically increase the Storm servers, we will use a resource manager such as Apache Yarn. Apache Storm is already ported to run on top of Apache Yarn and we will extend this framework to support elastic cluster resizing.

A resource management framework such as Yarn only works with the allocated computation resources. We will use the IaaS layer to dynamically scale computation nodes in the cluster. We have extensive expertise at the infrastructure level where we can instantiate systems on demand that can then support the dynamic scaling of the system. We will explore the Google Compute engine for the infrastructure level. Streaming computation nodes will be managed by the resource management layer and this will be controlled by a separate component. We can either use the messaging system or a distributed key value store to replicate the state as is done in MillWheel [32]. The fluctuations in time at the broker are from fig. 3 much less (than those in processing stage) in RabbitMQ but important in Kafka. We will scale the brokers at runtime to minimize such effects to the system by monitoring performance of brokers. Then a controller will directly use the IaaS infrastructure to scale the brokers as needed by increasing the number of assigned VMs.

To scale an application that receives input from multiple sources as a single stream and needs to differentiate each source, the larger stream must be partitioned into sub streams according to the source. This can be done with current processing frameworks but when parallel processing and state tracking is needed, the user code becomes complex. Also for parallel processing the scheduling is not adequate because each task will get a sub task for every sub stream. We will solve this by introducing new data abstractions and scheduling at the sub stream level and task level. The IoTCloud project is largely built on top of Apache Open Source projects. We have extensive experience in working with Apache projects (as users, committers and ASF members) and will contribute the results of this research back to the open source community.

References

1. Community Grids Lab and Indiana University. *IoTCloud*. 2015 [cited 2015 January 16]; Available from: http://iotcloud.github.io/.
2. Supun Kamburugamuve, Leif Christiansen, and Geoffrey Fox, *A Framework for Real-Time Processing of Sensor Data in the Cloud*. 2014.
3. Hengjing He, Supun Kamburugamuve, and G.C. Fox, *Cloud based real-time multi-robot collision avoidance for swarm robotics.* International Journal of Grid and Distributed Computing, 2015.
4. Jangwon Lee, et al., *Real-Time Object Detection for Unmanned Aerial Vehicles based on Cloud-based Convolutional Neural Networks*. 2015.
5. Supun Kamburugamuve, Leif Christiansen, and Geoffrey Fox, *A Framework for Real Time Processing of Sensor Data in the Cloud.* Journal of Sensors, 2015. **2015**: p. 11.
6. Supun Kamburugamuve, et al., *Cloud-based Parallel Implementation of SLAM for Mobile Robots*, in *International Conference on Internet of things and Cloud Computing (ICC 2016)*. 2016: Cambridge, UK.
7. Supun Kamburugamuve, et al., *Towards High Performance Processing of Streaming Data in Large Data Centers*, in *HPBDC 2016 IEEE International Workshop on High-Performance Big Data Computing in conjunction with The 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016)*. 2016: Chicago.
8. Supun Kamburugamuve and Geoffrey Fox, *Survey of Distributed Stream Processing*. 2016.
9. Geoffrey Fox, Shantenu Jha, and Lavanya Ramakrishnan. *Streaming and Steering Applications: Requirements and Infrastructure STREAM2015* 2015. Indianapolis.
10. Anderson, Q., *Storm Real-time Processing Cookbook*. 2013: Packt Publishing Ltd.
11. Videla, A. and J.J. Williams, *RabbitMQ in action*. 2012: Manning.
12. Kreps, J., N. Narkhede, and J. Rao. *Kafka: A distributed messaging system for log processing*. in *Proceedings of the NetDB*. 2011.
13. Fox, G., et al., *FutureGrid—A reconfigurable testbed for Cloud, HPC and Grid Computing.* Contemporary High Performance Computing: From Petascale toward Exascale, Computational Science. Chapman and Hall/CRC, 2013.
14. Hunt, P., et al. *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. in *USENIX Annual Technical Conference*. 2010.
15. Hwang, J.-H., et al. *High-availability algorithms for distributed stream processing*. in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. 2005. IEEE.
16. Grisetti, G., C. Stachniss, and W. Burgard. *Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling*. in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. 2005. IEEE.
17. Grisetti, G., C. Stachniss, and W. Burgard, *Improved techniques for grid mapping with rao-blackwellized particle filters.* Robotics, IEEE Transactions on, 2007. **23**(1): p. 34-46.
18. Bristeau, P.-J., et al. *The navigation and control technology inside the ar. drone micro uav*. in *18th IFAC world congress*. 2011.
19. Claes, D., et al. *Collision avoidance under bounded localization uncertainty*. in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. 2012. IEEE.
20. Alonso-Mora, J., et al., *Optimal reciprocal collision avoidance for multiple non-holonomic robots*. 2013: Springer.
21. Van Den Berg, J., et al., *Reciprocal n-body collision avoidance*, in *Robotics research*. 2011, Springer. p. 3-19.

22. Garage, W., *TurtleBot.* Website: http://turtlebot. com/last visited, 2011: p. 11-25.
23. Quigley, M., et al. *ROS: an open-source Robot Operating System*. in *ICRA workshop on open source software*. 2009.
24. Chauhan, N. *Modernizing Machine-to-Machine Interactions*. 2014; Available from: https://www.gesoftware.com/sites/default/files/GE-Software-Modernizing-Machine-to-Machine-Interactions.pdf.
25. Cherniack, M., et al. *Scalable Distributed Stream Processing*. in *CIDR*. 2003.
26. Abadi, D.J., et al. *The Design of the Borealis Stream Processing Engine*. in *CIDR*. 2005.
27. Thies, W., M. Karczmarek, and S. Amarasinghe. *StreamIt: A language for streaming applications*. in *Compiler Construction*. 2002. Springer.
28. Gedik, B., et al. *SPADE: the system s declarative stream processing engine*. in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008. ACM.
29. Neumeyer, L., et al. *S4: Distributed stream computing platform*. in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. 2010. IEEE.
30. Kamburugamuve, S., *Survey of distributed stream processing for large stream sources*. 2013.
31. Zaharia, M., et al. *Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters*. in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*. 2012. USENIX Association.
32. Akidau, T., et al., *MillWheel: fault-tolerant stream processing at internet scale.* Proceedings of the VLDB Endowment, 2013. **6**(11): p. 1033-1044.
33. Varia, J. and S. Mathew. *Overview of amazon web services*. 2013; Available from: http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/intro.html.
34. Hirzel, M., et al., *A catalog of stream processing optimizations.* ACM Computing Surveys (CSUR), 2014. **46**(4): p. 46.
35. Snyder, B., D. Bosnanac, and R. Davies, *ActiveMQ in action*. 2011: Manning.
36. Anderson, J.S. and E.D. Jensen. *Distributed real-time specification for Java: a status report (digest)*. in *Proceedings of the 4th international workshop on Java technologies for real-time and embedded systems*. 2006. ACM.
37. Borg, A. and A. Wellings. *A real-time RMI framework for the RTSJ*. in *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*. 2003. IEEE.
38. Bollella, G. and J. Gosling, *The real-time specification for Java.* Computer, 2000. **33**(6): p. 47-54.
39. Locke, D., *Mq telemetry transport (mqtt) v3. 1 protocol specification.* IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html, 2010.
40. Vijayaraghavan, A. *MTConnect for realtime monitoring and analysis of manufacturing enterprises*. in *Proceedings of the international conference on digital enterprise technology, Hong Kong*. 2009.

1.

1. Report Type

Final Report

Primary Contact E-mail
Contact email if there is a problem with the report.

gmiksik@indiana.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

812-856-0484

Organization / Institution name

Indiana University

Grant/Contract Title
The full title of the funded effort.

CLOUD-BASED PERCEPTION AND CONTROL OF SENSOR NETS AND ROBOT SWARMS

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-13-1-0225

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Geoffrey C Fox

Program Manager
The AFOSR Program Manager currently assigned to the award

Dr. Frederica Darema, AFOSR/RTC

Reporting Period Start Date

09/30/2013

Reporting Period End Date

12/31/2015

Abstract

This project "Cloud-Based Perception and Control of Sensor Nets and Robot Swarms" was performed by an interdisciplinary team at Indiana University with expertise in cloud and parallel computing, computer vision and robotics. It investigated the use of Cloud Computing as a key technology for Internet of Things (IoT) and DDDAS applications. The project developed an open source framework called IoTCloud to connect IoT devices to cloud services and used it to investigated three algorithms controlling robots from the cloud. Our three major applications were a parallel particle filtering based SLAM algorithm; a deep learning based drone control algorithm; and a robot swarm algorithm for n-body collision avoidance. These applications had significant time complexity needing the additional computer power offered by the cloud and we parallelized applications so that they could respond quickly to the edge devices. This project produced six major papers and a report. It also contributed to the STREAM2015 workshop and its final report.

Distribution Statement
This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

AFD-070820-035.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

AFOSR FA9550-13-1-0225 FINAL REPORT.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

Changes in research objectives (if any):

Change in AFOSR Program Manager, if any:

Extensions granted or milestones slipped, if any:

No Cost Extension granted for 90 days, extending the project deadline from 09/30/2016 to 12/31/2016

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Mar 25, 2016 12:34:51 Success: Email Sent to: gmiksik@indiana.edu