Current and Planned IoT Cloud Research at Digital Science Center

Geoffrey Fox, Supun Kamburugamuve, Hengjing He Indiana University

1. Introduction

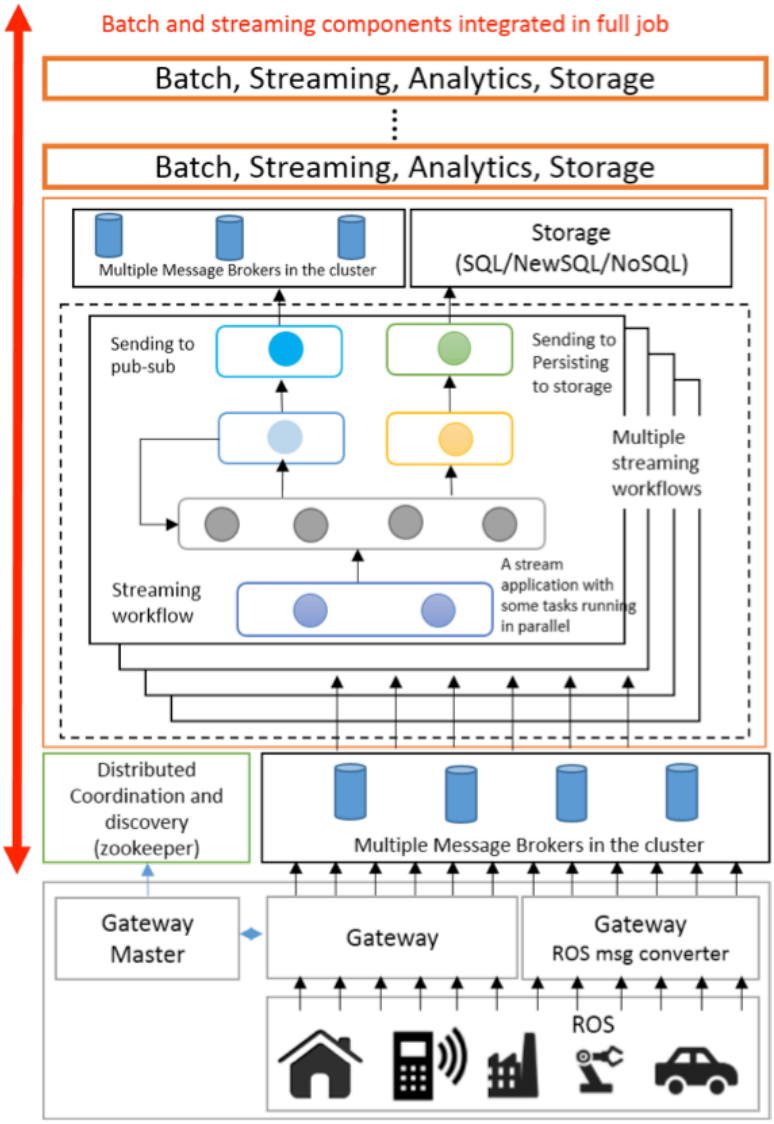

Figure 1 IoTCloud Architecture

Cloud Computing has long being identified as a key enabling technology for Internet of Things applications. We have developed an open source framework called IoTCloud[1] to connect IoT devices to cloud services. IoTCloud was developed as part of a research funded by AFOSR for Cloud-Based Perception and Control of Sensor Nets and Robot Swarms. IoTCloud consists of; a set of distributed nodes running close to the devices to gather 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 [2] uses Apache Storm[3] as the DSPF, RabbitMQ[4] or Kafka[5] as the message broker and an OpenStack academic cloud[6] (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 achieved with a ZooKeeper[7] 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 produces 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 5. Provide 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 the normal processing to continue with minimal effect to the applications. A detailed study of recovery methods possible for streaming applications is available in [8]. In our work we term a real time applications that produces correct answers but violates timing requirements as having performance faults. This proposal addresses (with same mechanisms) both explicit hardware/software and performance faults.

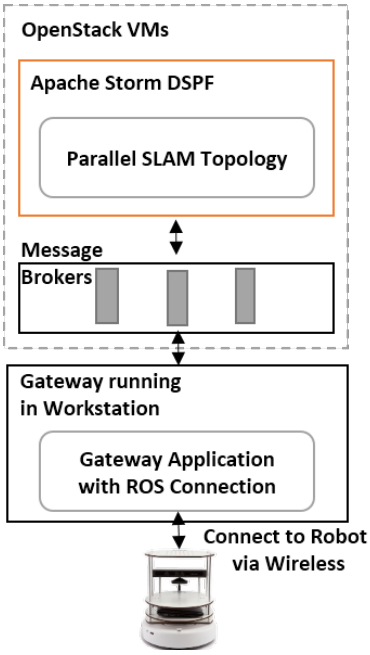

Figure 2 Parallel SLAM Application

We are exploring cloud controlled real time IoT applications in two 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 number of devices and are inherently suitable for central data processing. We are developing a parallel particle filtering based SLAM [9, 10] algorithm and Deep learning based drone[11] control algorithm both fit in to the first category. As an application to second category, we are developing a robot swarm algorithm for n-body collision avoidance [12-14] that can scale for large number of robots. We have made good progress in the parallel particle filtering algorithm and n-body collision avoidance algorithm, having working versions with very good performance characteristics. The deep learning based drone control algorithm is at the beginning stages and we are making progress on that front. The parallel SLAM and n-body collision avoidance algorithms use Turtlebot[15] as the robot and ROS[16] as the SDK for connecting to the robot. The overall parallel SLAM application is shown in Figure 2.

Through the work we have done in developing these applications we have identified shortcomings in the current technologies, current and future requirements. We will continue to drive and evaluate IoTCloud extensions, termed IoTCloud++, for scaling with performance guarantees with these applications and hope to add other applications in the future.

2. Streaming Application Challenges for IoT Cloud Controller

We present five categories of streaming 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 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.

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 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 don't address except by over-provisioning.

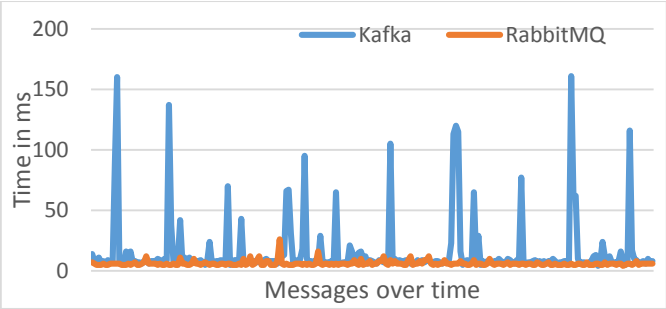

Figure 2a

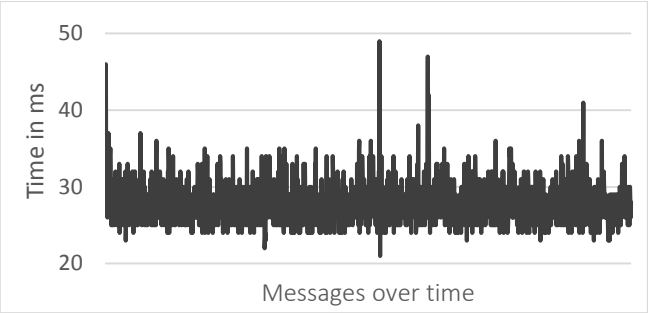

Figure 3b

Figure 2a Fluctuations in Time of IoTCloud using RabbitMQ and Kafka with Minimal Processing in Storm
Figure 2b Fluctuation in Time of IoTCloud with processing Kinect data from TurtleBot with RabbitMQ

This proposal will build IoTCloud++ where we 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.

4. Related Work

Industrial companies are realizing the need of data analytics driven approaches to support efficient operations at all levels to reduce the costs and be innovative. The machines are getting intelligent with software controls and communication to outside services. The industries can benefit immensely from real time central management to deploy, manage, upgrade, and decommission these intelligent machines. Concepts like Brilliant machines[17] by GE Software is pushing the industry towards such connected and intelligent infrastructure. A Brilliant machine is connected to the Industrial internet, 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. Years of research have produced software frameworks capable of executing distributed computations on top of event streams. Examples of such early event stream processing frameworks include, Aurora[18], Borealis[19], StreamIt[20] and SPADE[21]. With the emergence of Internet Scale applications in the recent years, new distributed stream processing systems like, Apache S4[22], Apache Storm[3], Apache Samza[23], Spark Streaming[24] and commercial solutions including Google Millwheel[25] and Amazon Kinesis[26] 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 distribute 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 are technical differences in how they distribute the tasks and how they manage the communications. Because 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 streaming case Spark reads the 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 does not seem 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 PE architecture is that, key attributes with very large domains can create large number of PEs in the system at a given time.

A comprehensive list of optimizations possible to reduce the latency of the stream processing applications are mentioned in [27]. 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[28], RabbitMQ[4], Kafka[5], Kestrel, HornetMQ etc. 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 will revisit this question when the performance enhancements of IoTCloud++ are implemented.

Real time applications with critical time requirements in the vanilla Java virtual machine is a challenge itself due to garbage collection, virtual machine etc. There have been efforts to improve the Java runtime and JDK to fit these requirements [29-31]. 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, simulators can create a powerful environment for the researchers. In some of our cloud applications 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[32] and MTConnect[33] is 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 the data in MQTT protocol, they can be send without transformation at the gateways directly to the cloud.

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

A major goal is to achieve real-time QoS in spite of fluctuations in the computation time. The architecture of the new IoTCloud++ platform is shown in Figure 3. 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 consist 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 done in MillWheel[25]. 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.

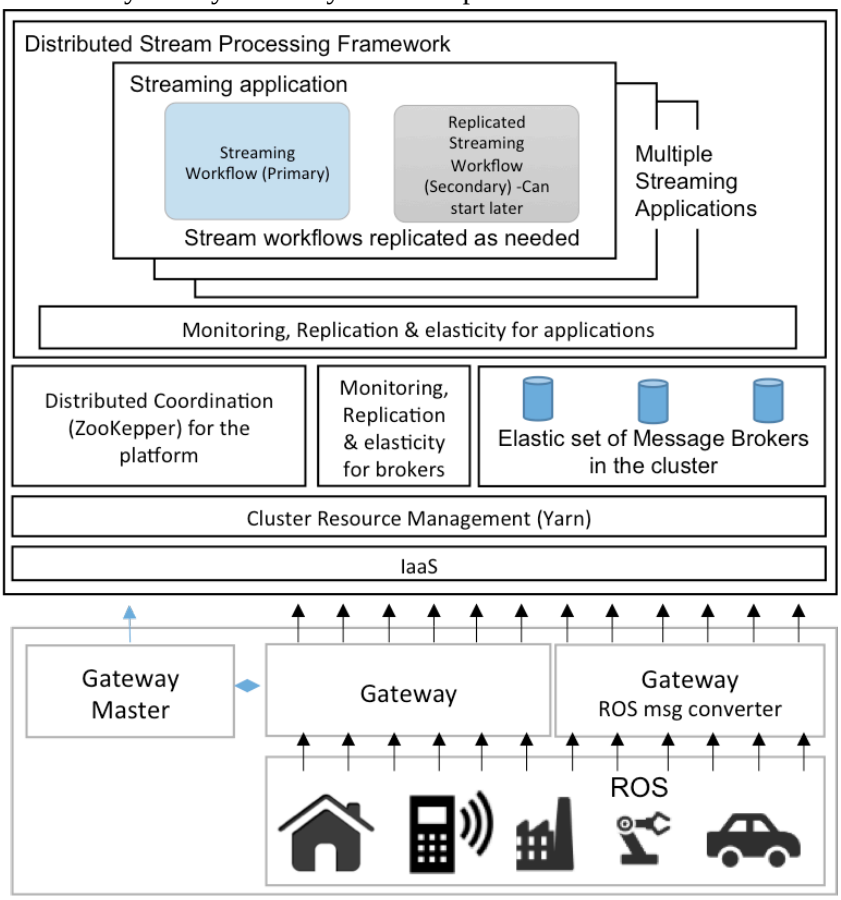

Figure 4 IOTCloud++ Architecture

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 in to 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 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. IoTCloud project is largely built on top of Apache Open Source projects. We have extensive experience in working with Apache projects (as user, committer and ASF member) 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. Anderson, Q., *Storm Real-time Processing Cookbook*. 2013: Packt Publishing Ltd.
4. Videla, A. and J.J. Williams, *RabbitMQ in action*. 2012: Manning.
5. Kreps, J., N. Narkhede, and J. Rao. *Kafka: A distributed messaging system for log processing*. in *Proceedings of the NetDB*. 2011.
6. 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.
7. Hunt, P., et al. *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. in *USENIX Annual Technical Conference*. 2010.
8. 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.
9. 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.
10. 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.
11. Bristeau, P.-J., et al. *The navigation and control technology inside the ar. drone micro uav*. in *18th IFAC world congress*. 2011.
12. Claes, D., et al. *Collision avoidance under bounded localization uncertainty*. in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. 2012. IEEE.
13. Alonso-Mora, J., et al., *Optimal reciprocal collision avoidance for multiple non-holonomic robots*. 2013: Springer.
14. Van Den Berg, J., et al., *Reciprocal n-body collision avoidance*, in *Robotics research*. 2011, Springer. p. 3-19.
15. Garage, W., *TurtleBot*. Website: http://turtlebot. com/last visited, 2011: p. 11-25.
16. Quigley, M., et al. *ROS: an open-source Robot Operating System*. in *ICRA workshop on open source software*. 2009.
17. 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.
18. Cherniack, M., et al. *Scalable Distributed Stream Processing*. in *CIDR*. 2003.
19. Abadi, D.J., et al. *The Design of the Borealis Stream Processing Engine*. in *CIDR*. 2005.
20. Thies, W., M. Karczmarek, and S. Amarasinghe. *StreamIt: A language for streaming applications*. in *Compiler Construction*. 2002. Springer.
21. 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.
22. Neumeyer, L., et al. *S4: Distributed stream computing platform*. in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. 2010. IEEE.
23. Kamburugamuve, S., *Survey of distributed stream processing for large stream sources*. 2013.
24. 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.
25. Akidau, T., et al., *MillWheel: fault-tolerant stream processing at internet scale*. Proceedings of the VLDB Endowment, 2013. **6**(11): p. 1033-1044.
26. Varia, J. and S. Mathew. *Overview of amazon web services*. 2013; Available from: http://docs.aws.amazon.com/gettingstarted/latest/awsgsg-intro/intro.html.
27. Hirzel, M., et al., *A catalog of stream processing optimizations*. ACM Computing Surveys (CSUR), 2014. **46**(4): p. 46.
28. Snyder, B., D. Bosnanac, and R. Davies, *ActiveMQ in action*. 2011: Manning.
29. 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.
30. 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.
31. Bollella, G. and J. Gosling, *The real-time specification for Java*. Computer, 2000. **33**(6): p. 47-54.
32. 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.
33. Vijayaraghavan, A. *MTConnect for realtime monitoring and analysis of manufacturing enterprises*. in *Proceedings of the international conference on digital enterprise technology, Hong Kong*. 2009.