An AHP-Based Evaluation of Real-Time Stream Processing Technologies in IoT

Patrick Killeen[1] and Alireza Parvizimosaed[2]

Abstract— **These days, Internet of Things (IoT) technologies gather high volumes of data from millions of Internet connected devices and sensors, trying to process data streams with the lowest latency possible. The primary applications of IoT are smart health, smart city, smart farming, smart homes, smart grid, and smart transportation. Some of these applications have been implemented using various IoT technologies and frameworks without necessarily considering the major requirements for developing real-time stream processing systems. Even though each application satisfy some of the requirements of IoT, their requirements are aligned to specific categories. In this paper, we propose critical requirements for stream processing in IoT, and evaluate Database Management System, Rule Engine, and Stream Processing Engine technologies with Analytical Hierarchy Process(AHP) algorithm based on the requirements.**

I. INTRODUCTION

A. What is IoT

When you think of a smart house and the Internet of Things (IoT), what do you imagine? Many people would imagine IoT as simply being able to start their microwave using their smartphones. IoT is much more than that. Some predict that by year 2020, there will be more than 10 billion devices (or things) connected to the Internet [1][2]. The primary applications of IoT are smart health, smart city, smart farming, smart homes, smart grid, and smart transportation[1][3][4][5][6][7][8]. A thing is an Internet connected device with embedded sensors and/or actuators [3][4][9], that are used for sensing the environment around the thing to possibly act and/or be controlled [2][5][7][10]. IoT involves real-time data analytics, processing and mining [5].

*This work was not supported by any organization

[1]Patrick Killeen is with Faculty of Computer Science, University of Ottawa, Ottawa, Canada pkill013@uottawa.ca

[2]Alireza Parvizimosaed is with Faculty of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada aparv007@uottawa.ca

B. Big Data and Big Data Streams

The current cloud architectures will have trouble dealing with all these devices and their data [1][5]. That leads us to the notion of Big Data, which is high volume, veracity, velocity, value, and variety data [10]. In the context of IoT 'Big' refers to the massive number of data sources, sensors, and other smart devices [1]. This large number of data streams (and devices) are heterogeneous [2][5][10]. Some streams may be sent over LTE and others over Wi-Fi. The different formats (or structures) of data sources render the problem not trivial from a scalability point of view [2][5]. [10] refers to such streams as Big Streams, that is, streams with many data sources, and real-time/predictable latency requirements as well as stream output managers (what consumer receives what stream).

C. Requirements

With IoT quickly evolving, if the future scalability issue predictions are correct, requirements for IoT systems need to be defined to make IoT system implementation and design feasible. Since IoT is a type of real-time stream processing system, to begin analyzing the requirements for this type of system, it is necessary to understand the requirements of the more general real-time stream processing systems. It is also important to understand what tools and technologies can be used to best meet these stream processing requirements. This paper analyzes real-time stream processing system requirements and attempts to adjust(add, remove, or modify) them so that they apply to IoT systems.

II. BACKGROUND

This section will focus on related work to IoT systems and their requirements. The core literature of this paper is founded on [11], which discusses in detail what a real-time stream processing system requires, as well as comparing which tool and how effectively the tool meets the requirements.

A. Eight Stream Processing Requirements

This section explains the primary requirements that should be considered while developing real-time stream processing systems. The below requirements were defined by [11].

- **Keep the Data Moving**
  Systems must be able to process streams of data without storing data, since storing data creates latency. If persistence is necessary, the system must only store data inside its process space (using an embedded DBMS for example)[11].
- **Execute Queries on Streams of Data**
  Since data flows into real-time systems continuously, SQL queries cannot be performed on these data streams, because SQL queries can only be run on tables of finite size. Therefore, an alternative mechanism such as StreamSQL is required, which is a type of SQL that is designed to perform queries on streams of data[11]. The mechanism must provide extendable operators. On that note, function and parameter replacement are two methods which modify a query operation at runtime[12].
- **Handling Delayed, Missed and Out-of-Order Data**
  The system generates valid results of data processing whenever it processes sequences of data in some correct order. If packets arrive late, the system must not wait infinitely, which can be achieved by ignoring delayed data and continuing to process the next available data, and by supporting timeouts. An alternative consists of keeping an open timestamp window of delayed messages for an extra period of time[11]. Another option is dynamic revision of query results, which replaces predefined messages with delayed messages and revises query results to correct inconsistent data[12].
- **Create Predictable Data**
  The system must forecast repeatable results of processing time-series messages. In fact, the system must predict missed parts of data and continue data processing using the predictions[11].
- **Merging Stored and Streaming Data**
  Some real-time stream processing systems require the integration of stored data with live data[11][5]. For instance, fraud detection applications must analyze stored and live credit card transaction data. To avoid latency, the system should fetch data from embedded databases (SQLite for example) and process it with a uniform language[11].
- **Ensure Data Safety and Availability**
  To reach high availability, the system can either make a hardware backup of data or recover data from log files. Both methods create latency. A solution to this is Tandem-style hot backup, which keeps a secondary system synchronized with the primary system. Upon primary system failure the secondary system continues data processing until the primary system successfully restarts[11]. Another solution is rebuilding lost data by tracking a history of operations, which are then reapplied to recompute the lost data[13]. Precise, Rollback, and Gap recovery are also utilized to ensure availability and data safety[14].
- **Distribute and Scale Applications Automatically**
  Multi-threading and process distribution are two general methods for achieving high performance and low latency[11]. Processing data in different levels of granularity is an approach for ensuring scalability. That is, this method organizes multiple levels of granularity ranged from sensors to servers and processes data in each level separately[12]. Storing data and their structure in memory also has a positive effect on performance. Another method is clustering data processing nodes such that the data be processed in multiple clusters simultaneously[15].
- **Real-time Processing and Responding**
  The system must choose optimum execution paths for processing high volume data with low latency[11].

B. Technologies of IoT

In general, an IoT architecture can be separated into 5 layers (see below) and IoT technologies are involved in at least one of these layers.

- **Sensing Layer**: Things with sensors and actuators sense the environment around them for desired phenomenon[3][6][4].
- **Network Layer**: The heterogeneous data streams of various devices are connected to the Internet via various means (e.g., LTE and Wi-fi)[3][4].
- **Storage Layer**: A persistence mechanism processes and stores the huge number of data streams[3][6].
- **Learning Layer**: Live and stored data streams are analyzed to learn something about the data and the environment[3][6].

- **Application Layer**: The tool or service uses the information gathered and learned from the various data streams[3][6][4].

MQTT and CoAP, Gateway and Fog Computing are three IoT technologies which involve some of the aforementioned layers.

Gateway

- **Sensing Layer**: Gateways are used to handle the sensor data streams coming from various things (or devices). Their role is to manage the data stream heterogeneity of the various streams [4].
- **Network Layer**: Gateways connect smaller sensor/device networks (a Zigbee network for example [1]) to the Internet. It provides an abstraction layer, an API, over the sensor streams, to provide higher layers access to the sensors without needing knowledge of the underlying sensor communication protocols [4].

Fog Computing

- **Network Layer**: Fog computing is a platform with hierarchical distributed architecture which processes some parts of data in the edge of the network, in order to process high volume data with low latency[1]. The Fog brings IoT devices closer to the cloud to help meet IoT requirements the cloud doesn't meet[5].
- **Storage Layer**: The Fog's hierarchical structure also involves resource allocation to support both of the following processes: machine-to-machine(M2M), which may be real-time, and human-to-machine (H2M), which isn't real-time. [1].
- **Learning Layer**: In addition, the Fog supports many different data sources (a dimension of Big Data in IoT), ideal for real-time big data analytics. Fog computing middle-ware classifies data into separate groups and processes them in different levels from network edge to cloud based on their priorities[1].
- **Application Layer**: Provides high level access APIs on data sources whereby things can connect to Fog platform and feed their streams of data to the platform[1]. It is able to utilize the Policy-based service orchestration approach, which distributes service requests among Fog services according to the predefined policy constraints. Even if the platform does not find an active instance of the desired service, it creates a new instance. Consequently, whenever constraints are defined perfectly, data will be processed by means of free services [1].

MQTT and CoAp

- **Network Layer**: These OSI Application Layer protocols are based on subscribe and publish model and are resistant to failure[1]. Both MQTT and CoAp are used extensively in IoT systems, meeting the availability (by nature of publish-subscribe model), expressiveness, and resource requirements of IoT systems[2].

III. ANALYZE THE EIGHT REQUIREMENTS

The following section will present our results. We investigate how meaningful it would be to include each of the requirements mentioned above into IoT system requirements. We found results by analyzing the traditional real-time stream processing system requirements (see section II-A) when applied to IoT systems. There are some of the eight requirements we argue that don't make sense, are missing, or need modification when focusing on IoT.

A. Modified Rules

- **Keep Data Moving:** Most IoT systems suppose that streams of data are flowing instead of storing the steams in a repository[16].
- **Execute Queries on Streams of Data:** There are many possibilities for querying IoT streams. A Database Stream Management System (DSMS), which is an extension of DBMS, processes streams of data instead of executing a query on stored data. Recently, some SQL-based languages and platforms such as Continuous Query Language, Event processing language, IBM Infosphere Stream, SAP Sybase Event Stream Processor, and SQL-stream Blaze have been developed to support data stream processing. Complex Event Processing (CEP), which is another option, monitors incoming events that satisfy specific conditions[16].
- **Integrate Stored and Streaming Data:** Some IoT systems take advantage of machine learning algorithms such as Principal Component Analysis and feature selection to reduce the dimension of data. Other systems apply supervised machine learning algorithms to streams of data, and create training models. The purpose of these algorithms is minimizing the error between real and estimated data[16][17][18]. For example, smart city applications should process events in real-time and explore useful information in order to recognize

specific types of data. To this aim, stream reasoning methods are appropriate candidates thanks to reasoning based on background knowledge and streaming queries[19].

Fog computing can arguably meet this requirement for integrating live and stored data (see storage layer section II-B).

- **Partition and Scale Applications Automatically:** Since many devices may connect to IoT platforms, the platforms must lead data to the most suitable services [1]. To this aim, Fog computing and its Policy-based service orchestration approach can satisfy this requirement (see application layer section II-B). Furthermore, since the Fog network is dynamic and nodes may move around, simply partitioning data to a location that a user accesses frequently may not be sufficient. Nodes and users may be dynamic, which means static partitioning isn't sufficient. Dynamic partitioning based on the service requester's (user's) location will be more efficient. That is, being able to predict the next location of a service requester will allow the data partitioning to be optimized and accessible by the user by sending the desired data to the user's future location where the request will take place [20]. The Fog is ideal for this, since it can perform machine learning (see learning layer section II-B).

  Information Flow Of Things (IFOT) discovers tags of flows by machine learning algorithms, and assign tags to data flows. Since metadata of data flows are available, IFOT nodes can efficiently search and process data of flows. In addition, it reduces granularity of data due to using dynamic granularity adjustment function[16].
- **Process and Respond Instantaneously:** To achieve high performance, Fog computing (see section II-B) processes data in the edge of network. In addition, some Fog computing platforms run a software agent on the connected devices to process data before inserting the data into the system[1]. IFOT also analyzes and aggregates data flows near data sources instead of sending raw data to the cloud and storing them on the cloud[16].
- **Ensure Data Safety and Availability:** Availability should be handled differently compared to the cloud paradigm. To increase availability, data locality should be used. That is, instead of sending all raw data streams over the network, which may create bottlenecks[9], the streams should only be sent over their local network (Personal Area Network (e.g., PAN and LAN), and aggregated before being sent to the cloud[1]. In other words, availability in an IoT system requires edge analytics for sending stream aggregations[20]. Therefore, the following IoT technologies meet this availability requirement (see section II-B): the Fog, gateways, and MQTT and CoAp.

  We propose a second modification to this requirement. Standard real-time stream processing systems meet availability by using machine cloning, offloading, and duplication[11][20] (see data safety and availability section II-A). However, in IoT things are more dynamic. For example, Fog networks are more dynamic, which complicates duplication and renders Fog node duplication a more difficult process to achieve [20]. Thus, machine cloning, offloading, and duplication may not be a suitable solution for this requirement for IoT systems.

B. Added Rules

- **Support Diverse Set of Distributed Applications:** Since IoT systems connect a lot of devices and applications together[1][3][2][5], they must provide a mechanism for supporting various devices and applications[1]. There are three categories of applications mentioned by [5], namely: real-time, data analytics, and device interaction. From the perspective of the device interaction category, Fog computing meets this category's requirement. It does so by providing the APIs which allow various heterogeneous devices to interact with each other[6](see application layer section II-B). To some extent, the gateways also meet this category's requirement by interacting with sensors and the Internet (see section II-B).

  Onto the data analytics category, smart city applications for example, must apply some simplification functions like aggregation and summarization of data to efficiently analyze data[19]. There is so much data that simply processing all the raw streams would not be feasible. The Fog, gateways, and MQTT and CoAp meet these requirements (see modified data safety and availability section III-A). Furthermore, streaming applications using IoT systems also may visualize analytical data as part of data flow instead of generating visual charts in clients' browsers[17].

- **Resource Management:** Fog computing (see section II-B) meets this requirement, since the Fog computing platform supports virtualization of resources, networks, and computing. After virtualization, multiple operating systems and service containers can be executed on a common physical machine to improve resource management. It can perform its policy-based orchestration and provisioning over the virtualization layer[1]. For example, a smart city may have a software-defined network and the nodes will compete for resources, requiring a network manager to dynamically self optimize the network resources[2].
- **Geographical Distribution:** IoT systems must be able to handle many distributed sets/networks of devices around the globe, instead of using a centralized solution such as the cloud[1][8][20]. Distributing flows of data among remote IFOT nodes is an issue that is solved by searching multiple paths of data to a distinct target, measuring transformation parameters, and then setting up multiple paths to the target[16]. We argue MQTT and CoAp meets this requirement by connecting devices together over the Internet (see section II-B).
- **Connecting Networks of Sensors:** In IoT systems, there are networks of sensor networks[6][7][8]. Therefore, the sensor networks must be connected together to act in coordination and achieve a common task[7][8]. The gateways arguably can achieve this task, since they have their own local network of sensors, and the gateways can interact with each other using their own virtual networks over the Internet (see section II-B).
- **Data Multi-Tenancy:** The data gathered (sensed and deduced) on a thing should be only accessible by the proper tenant. That is, the different types of data should be protected and only accessed by users with the right privileges. This is required since there are legal and privacy requirement on data ownership[3]. For example, a hospital patient may own his or her vital sign data, but doesn't own the vital sign device's metadata, such as the heart rate monitoring frequency. The gateways can achieve this since they are near the data source and thus can manage these privileges before the data are sent over the Internet (see section II-B).

C. *Removal of Requirements*

- **Handling Delayed, Missed and Out-of-Order Data:** The things that receive streams of data from sensors don't need to handle missing or out-of-order data. The data elements may not necessarily have a timestamp, since the sensors can simply output the raw sensor value they have read from the environment. The thing itself (gateway for example) is responsible for creating the timestamp[21]. Therefore, this requirement does not matter for gateways because sensors (data sources) and gateways are physically located near each other instead of being geographically dispersed, and consequently gateways receive data from sensors with correct sequences. In addition, since sensors are constantly generating data, gateways cannot wait for missed data. In general, the requirement for handling out-of-order data or missing data is important to higher-layer nodes in the IoT architecture, but not necessarily important to sensing layer nodes.

IV. TECHNOLOGIES SUPPORTING IoT STREAM PROCESSING

- **Rule Engine(RE):** As Fig.1 illustrates, a RE is an active system that processes data whenever data enters the engine. A RE has a collection of rules, each rule being a condition-action pair. When data comes into the engine, it is compared to many conditions. If a condition is triggered, its corresponding action is fired. A RE often stores temporary data in memory instead of a repository. Hence, its capacity of storing data is limited[11].

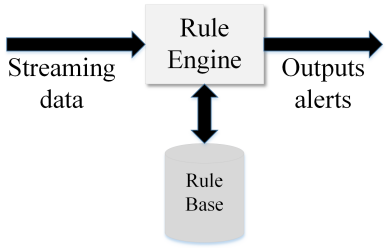


Fig. 1: Basic Architecture of RE[11]

- **Stream Processing Engine(SPE):** A SPE is also an active system that performs SQL-style queries on streams of data. Fig.4 shows the basic architecture of a SPE. Whenever data enters into the engine, it processes data without waiting for external requests. Similar to a RE, a SPE is designed for processing flows of data instead of

processing stored data. However, a SPE can store data in an embedded database which isn't straight forward for a RE. In addition, a SPE takes advantage of timestamps and stream-oriented operators like merging streams, timeouts, and floating windows[11].

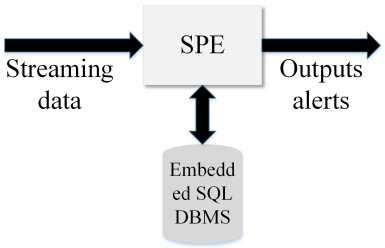


Fig. 2: Basic Architecture of SPE[11]

- **Database Management System(DBMS):** A DBMS is a passive technology that stores data in a permanent repository and executes SQL queries on the repository's fixed-size tables. In other words, if data is entered into a DBMS, it must be written to the disk before a DBMS processes the data. As Fig.3 shows, its processing will start whenever an external request arrives, for example an application's query, which triggers the DBMS to process data. Furthermore, a DBMS does not support timeouts, timestamps, or stream-oriented operators[11].

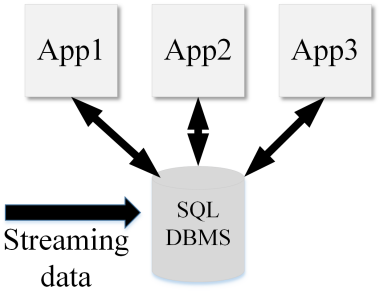


Fig. 3: Basic Architecture of DBMS[11]

V. EVALUATION

To evaluate the aforementioned technologies (see section IV) against the IoT requirements we compiled (see section III), we use Analytic Hierarchy Process (AHP) method. AHP is a multi-criteria decision making method which organizes evaluation factors and alternative decisions in a hierarchical structure, and then compares pair decisions regarding factors[18]. Fig.4 depicts AHP hierarchy of IoT requirements and the three technologies: SPE, RE, and DBMS. We rename the requirements (see below) to simplify the rest of this section when referencing them.

R1-Keep Data Moving; R2-Execute Queries on Streams of Data; R3-Integrate Stored and Streaming Data; R4-Partition and Scale Applications Automatically; R5-Process and Respond Instantaneously; R6-Ensure Data Safety and Availability; R7-Support Diverse Set of Distributed Applications; R8-Resource Management; R9-Geographical Distribution; R10-Connecting Networks of Sensors; R11-Data Multi-Tenancy.

We also classify R1, R2 and R3 in Stream Processing, and classify R4 and R9 in Distributed Management. We suppose that the importance of a source factor in comparison with a target factor is a value of -Strong(-S), -Moderate(-M), Equal(E), +Moderate(+M), +Strong(+S) which shows the importance of a source factor against a target factor. Their importance order is -S, -M, E, +M, +S from least important to highest one. We provided a voting meeting with experts, and assigned average values of each factor, which were proposed by voters, to the factor. Then, we created comparison matrices of requirements and technologies to sort their importance.

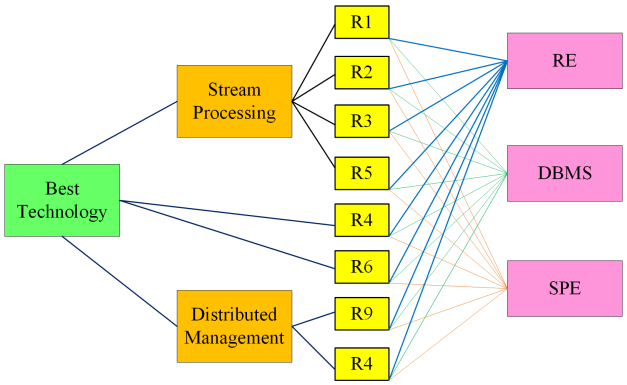


Fig. 4: AHP hierarchy of IoT requirements and three technologies

A. Comparison Matrices of Technologies Regarding Requirements

In this part, we evaluate pair technologies regarding per requirements. Tables I, II and III show the result of the evaluation. Since three technologies do not have an effect on R7, R8 and R10, we ignore these requirements in the evaluation.

B. Comparison of Requirements

In the next step, the importance of requirements are compared against each other. In other words, R1,

TABLE I: Evaluation of scenarios 1 through 4

Pair Tech.	Value	Reason
Scenario R1		
SPE VS RE	E	Both support stream of data.
SPE VS DBMS	+S	DBMS is a passive system and SQL queries perform on stored data whereas SPE is active and processes stream of data without storing.
RE VS DBMS	+S	DBMS is a passive system and SQLs perform on stored data whereas RE is active and processes stream of data without storing.
Scenario R2		
SPE VS RE	E	RE compares incoming data with rules, and SPE executes stream-oriented queries on flow of data
SPE VS DBMS	+S	DBMS executes SQL queries while SQL is performed on stored data.
RE VS DBMS	+S	
Scenario R3		
SPE VS RE	+S	SPE has access to both stored and streaming data whereas RE does not store and fetch data from permanent database.
SPE VS DBMS	+M	DSMS, which is an extension of DBMS, can support data streaming but its capability is lower than SPE.
RE VS DBMS	-M	DSMS can integrate both streaming and stored data whereas RE does not have database.
Scenario R4		
SPE VS RE	+M	Although RE can allocate flow of data to services well, it does not have any database for machine learning, and then it needs a third party application to update rules dynamically based on historical data. In contrast, SPE has a storage which can update rules more easily.
SPE VS DBMS	+S	Although DBMS can partition data statically, SQL commands cannot classify data dynamically for making tags.
RE VS DBMS	+M	Even though RE does not have storage, it can support tagging but DBMS does not handle classification of data as easily.

TABLE II: Evaluation of scenarios 5 through 10

Pair Tech.	Value	Reason
Scenario R5		
SPE VS RE	E	Although SPE processes data with more latency than RE due to checking windows, timestamps and referring to database, it can reduce latency by aggregating and analyzing data in the edge of network.
SPE VS DBMS	+S	DBMS has to refer to disks per query while SPE and RE do not necessarily need accesses to disks.
RE VS DBMS	+S	
Scenario R6		
SPE VS RE	+S	SPE can aggregate and analyze data on the edge but RE just has a temporary memory and does not propose any mechanism for crash recovery.
SPE VS DBMS	+M	SPE avoids of bottleneck thanks to using aggregation whereas DBMS does not support it. However, DBMS can replicate data in the backup database.
RE VS DBMS	-M	RE does not support any recovery methods while DBMS at least proposes replication.
Scenario R7		
SPE VS RE	E	Supporting diversity of devices does not related to these technologies.
SPE VS DBMS	E	
RE VS DBMS	E	
Scenario R8		
SPE VS RE	E	Virtualization and resource management do not related to these technologies.
SPE VS DBMS	E	
RE VS DBMS	E	
Scenario R9		
SPE VS RE	E	Distributed databases fragment data among remote sites and look for optimum integration queries whereas RE and SPE do not offer distribution mechanisms.
SPE VS DBMS	-S	
RE VS DBMS	-S	
Scenario R10		
SPE VS RE	E	Connection of sensor networks does not relate to these technologies.
SPE VS DBMS	E	
RE VS DBMS	E	

TABLE III: Evaluation of scenario 11

Pair Tech.	Value	Reason
Scenario R11		
SPE VS RE	+S	DBMS intrinsically guarantees privacy. SPE also guarantees it due to using embedded DBMS whereas RE just distributes data without checking privacy.
SPE VS DBMS	E	
RE VS DBMS	-S	

TABLE IV: Comparison of requirements based on (a) SP, (b) IoT Systems, (c) DM

(a) SP

	R1	R2	R3	R5
R1		E	+S	+M
R2			+S	+M
R3				-S
R5				

(b) IoT Systems

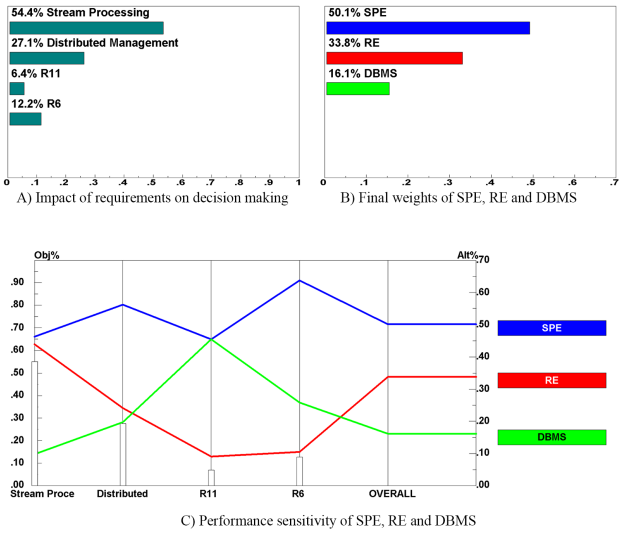
	R6	R11	SP	DM
R6		+M	-S	-M
R11			-S	-S
SP				+M
DM				

(c) DM

	R4	R9
R4		+S
R9		

R2, R3 and R5 are compared together against capability of processing data streams. R4 and R9 are compared against capability of supporting distributed management. R6 and R11, Stream Processing(SP) and Distributed Management(DM) are compared together against their importance in IoT systems. Table IV summarizes the aforementioned comparisons.

After analyzing the mentioned technologies with AHP, the results indicate that the SPE is the most appropriate choice for an IoT system, since it can support the most significant requirements as a whole. The second alternative is the RE. As Fig.5 shows, SPE, RE, and DBMS satisfy primary requirements by 50.1%, 33.8% and 16.1% respectively. In addition, the picture illustrates the effect of SP, DM, R11 and R6 on the final results. Since overall inconsistency is 0.05, there are few inconsistencies among values of comparison matrices. Comparison between these results and results mentioned in [11] suggests that using SPE satisfies stream processing requirements in both gateway level and application level.



A) Impact of requirements on decision making
B) Final weights of SPE, RE and DBMS
C) Performance sensitivity of SPE, RE and DBMS

Fig. 5: Analysis of Impact and performance sensitivity of SPE, RE and DBMS

VI. CONCLUSION

As technologies such as sensors and computing power become cheaper and smaller, it makes IoT possible and hence the accelerated rate at which IoT is evolving. IoT is still a new paradigm and it isn't fully understood yet. The infrastructure we currently have isn't ready for the full transition to IoT. The cloud isn't going anywhere, but may be tightly coupled with IoT systems. The requirements for such systems must therefore be defined to help ease the transition and understanding from the cloud paradigm to the IoT. There are quite a few technologies that exist which help meet the requirements for IoT systems, but their use depends on the problem at hand. For example, smart farming is arguably much more static by nature than smart transportation. Thus, some IoT problems may have different requirements, but they will all have some in common. Upon understanding what technology performs best in any given IoT situation, it will help when making informed decisions when designing and implementing IoT solutions.

In this paper, we analyzed eight requirements of processing high volume data stream with low latency in the scope of IoT. We added, removed and modified some items to the aforementioned requirements, and then evaluated three primary IoT technologies including Stream Processing Engine, Rule Engine and Data Base Management System by an AHP method. The result represents that Stream Processing Engine satisfies stream processing requirements of IoT better than the other two technologies. Future work for this paper would be to adjust the requirements and technologies

as IoT evolves, since IoT is in its infant-stage. There may be new technologies that out-class the technologies mentioned in this paper, or new requirements that arise as IoT grows.

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Big data and internet of things: A roadmap for smart environments," *Studies in Computational Intelligence*, vol. 546, pp. 169–186, 2014.

[2] M. S. Obaidat and P. Nicopolitidis, *Smart cities and homes: Key enabling technologies*. Morgan Kaufmann, 2016.

[3] T. Chou, *Precision - Principles, Practices and Solutions for the Internet of Things:*. McGraw Hill Education, 2017. [Online]. Available: https://books.google.ca/books?id=OVpHDwAAQBAJ

[4] S. K. Datta, C. Bonnet, and N. Nikaein, "An iot gateway centric architecture to provide novel m2m services," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 514–519.

[5] M. Díaz, C. Martín, and B. Rubio, "State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing," *Journal of Network and Computer Applications*, vol. 67, pp. 99–117, 2016.

[6] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *Systems, Signals and Image Processing (IWSSIP), 2013 20th International Conference on*. IEEE, 2013, pp. 43–46.

[7] C. Fiandrino, F. Anjomshoa, B. Kantarci, D. Kliazovich, P. Bouvry, and J. N. Matthews, "Sociability-driven framework for data acquisition in mobile crowdsensing over fog computing platforms for smart cities," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 4, pp. 345–358, 2017.

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

[9] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[10] L. Belli, S. Cirani, G. Ferrari, L. Melegari, and M. Picone, "A graph-based cloud architecture for big stream real-time applications in the internet of things," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2014, pp. 91–105.

[11] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 requirements of real-time stream processing," *ACM Sigmod Record*, vol. 34, no. 4, pp. 42–47, 2005.

[12] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina *et al.*, "The design of the borealis stream processing engine." in *Cidr*, vol. 5, no. 2005, 2005, pp. 277–289.

[13] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters." *HotCloud*, vol. 12, pp. 10–10, 2012.

[14] J.-H. Hwang, M. Balazinska, A. Rasin, U. Cetintemel, M. Stonebraker, and S. Zdonik, "High-availability algorithms for distributed stream processing," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*. IEEE, 2005, pp. 779–790.

[15] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham *et al.*, "Storm@ twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 147–156.

[16] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of real-time processing technologies of iot data streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[17] A. Shukla, S. Chaturvedi, and Y. Simmhan, "Riotbench: a real-time iot benchmark for distributed stream processing platforms," *arXiv preprint arXiv:1701.08530*, 2017.

[18] T. L. Saaty, "Decision making with the analytic hierarchy process," *International journal of services sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[19] R. Tönjes, P. Barnaghi, M. Ali, A. Mileo, M. Hauswirth, F. Ganz, S. Ganea, B. Kjærgaard, D. Kuemper, S. Nechifor *et al.*, "Real time iot stream processing and large-scale data analytics for smart city applications," in *poster session, European Conference on Networks and Communications*, 2014.

[20] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[21] K. Aberer and M. Hauswirth, "Middleware support for the" internet of things"," 2006.