Real-time Processing of IoT Events using a Software as a Service (SaaS) Architecture with Graph Database

Mr. Godson Michael D'silva 1, Mr. Sanket Thakare 2, Dr. Vinayak Ashok Bharadi 3

1Information Technology Department, St. John College of Engineering & Technology, Palghar, India, dsilvagodson@gmail.com
2 Computer Department, Thakur College of Engineering & Technology, Mumbai, India, sanketmthakare@gmail.com
3 Information Technology Department, Thakur College of Engineering & Technology, Mumbai, India, vinayak.bharadi@thakureducation.org

Abstract— IoT is a computational notion where each of the tangible objects around are termed as things, which will be connected to other things through the web and will communicate or transfer data between each other. But if the focus is just on the things or devices, than there may be a key ingredient missing, that is all these internet of things are connected to each other. What this entitled is that all of the devices, from appliances to smart phones will be able to share information with one another. When gone into details about Internet of things (IoT), it has lots of different types of things it has devices, it has locations, it has people and they all are connected. That mean a big and a complex connection data is generated. When to model this connected data a relational database or other type of No SQL database is used, they under perform. As those relational and No SQL type abstractions are not optimized for connection data, they are optimized for different kinds of connections. To overcome this, a highly robust, scalable, pluggable and faster architecture is proposed which tries to solve the issues of connected data in IoT domain with the unification of three open source technologies such as Ejabberd, Apache spark and Neo4j database. And to a get more secure and efficient performance this three open source technologies are implemented on a Microsoft azure public cloud. This proposed architecture stack is equally suitable for projects with high ambitions.

Keywords— IOT, Ejabberd; Apache Spark; GraphX; Internet of Things; Neo4J database; Azure; Graph Database.

I. INTRODUCTION

The Internet of Things domain is growing and maturing at a very fast pace, in the upcoming years. The evolution of the iot is an inescapable, shift in the way users communicate with their devices and their surroundings. Along with these new blue sky of opportunities, the iot also faces a new hurdles for industry to be able to quickly, effectively and efficiently adapt to this changing environment. In internet of things all the things are connected to each other. But if the user focus is just on the things than the user may be missing a key ingredient that is all these things are connected to each other. In internet of things the system has to be certain that the fruit market mall is linked up to the users fridge itself and not anyone else fridge. So if any of the fruits get over in the fridge, then automatically an order will be placed and that fruit will be delivered to the user's house. Or it can be when user is going out with his android device and his application of it is giving him recommendation in real-time of different things based on where he is located, who he is meeting, etc. Those recommendations are made on the fly in context with a relationship with connection around the user, the connection between people, connection between locations, connection between devices etc.

When gone into details about Internet of things (IOT), it has lots of different types of things it has devices, it has locations, it has people and they all are connected. And in order to through a app with fully leverages the IOT and in whatever subset the IOT lives in whether it's a mobile app, or whether is a hardware device, whether it's a infrastructure to manage a network than your application needs to be connection aware. What is traditionally done is modelled the connected data into ER model and then making tables and connecting them with joins, foreign and primary key, etc. and after that write queries to get the data. It make take a time to execute. And if the results for a real-time application take time than it cannot employed in real time.

This is the challenges that are faced, whether a relational database or other type of No SQL database is used, those abstractions do have a way to deal with connections, but they are typically not optimized for connection data, they are optimized for different kinds of connections.

The Graph Compute Engines provide batch or streaming graph computation based on optimized graph algorithm implementations. Usually message passing algorithms like pregel is used. It depends on the complexity of the graph. No matter how many strongly connected components are there, but since some graph algorithms like PageRank are iterative, it has to iterate from one stage and use the results of the previous stage. For iterative graph algorithms the complexity of the graph will make the system or break the system. Graphs with high complexity need a lot of memory to be processed iteratively. To resolve this issue, apache spark is proposed in this architecture. The apache Spark is scalable, fault-tolerant in- memory big data and machine learning platform. Apache Spark integrates well with Hadoop, but is more modern and way faster in processing. Also, let's do in-memory analytics on a graph dataset using the optimized graphics library.

In this paper a highly robust, scalable, pluggable and faster architecture is proposed which tries to solve the issues of connected data in IOT domain with the unification of three open source technologies such as ejabberd, apache spark and neo4j database. To a get more efficient performance this three open source technologies are implemented on a Microsoft azure

public cloud [15][16]. This proposed architecture stack is equally suitable for projects with high ambitions.

II. Literature Survey

The Internet of Things (IOT) is the network of devices that contain embedded technology to interact and sense or interact with other devices or the external environment [1]. The future of communication will be in Internet of Things, which is certainly the most well-known after technology today. The use of IOT is diverse, and it can be from ordinary voice recognition to critical space programs [2]. According to new research from international data corporation, by 2020 number of networked devices will be 30 billion more than 4 times the entire global population and IOT market will grow $ 1.7 trillion [3]. When it comes to IOT, the main things is huge and continuous stream of data generated. The main provocation is to examine huge datasets from heterogeneous Io devices and provide near real-time solutions. Network, disk, and compute power all will be affected and should be planned to take care of this new type of data. The efficient technology should be used to handle this additional load of heterogeneous data [4].

For the past few years, spark streaming has emerged as the leading platform to enforce IOT and similar different real-time use cases. The apache spark is tools for streaming data and processing continuously generated machine or sensor data. It processes tuples in group-like method during specific intervals of time in which events are continuously collected. It also have non streaming data capabilities. The apache spark also provides graph processing framework called graphX. The apache spark ensure that it is able to process real-time analytics on big data for transactions in an internet of things world.

IOT devices, data, and applications are inherently connected to each other. Managing this number of huge connection is became a challenging task. IOT performs many operation at real time. So, there is a need of tools that can show these connections quickly and easily. The graph database provide an efficient solution to store such large number of connection in the form of graph. It became an essential tool in discovering, making sense of intricate relationship and interdependencies [5]. The neo4j is a leading graph database which stores information in the form of nodes and relationship between the nodes instead of rigid data structure [6]. The neo4j traverse database on the basis of relationship between their nodes and properties where everything is connected to each other. In many cases, It complete queries that are impossible in relational database and gives 10000 times faster than similar queries in relational database.

III. Related Theory

In this paper the public cloud based architecture is discussed in detail to solve the connected dataset problems in internet of thing domain. In this proposed architecture the internet of things system is implemented majorly over the three open source technologies, such as ejabberd server, apache spark and neo4j database. Moreover to enhance the performance and efficiency of the architecture this three technologies are put on Microsoft azure public cloud with improves the scalability, robustness of the architecture. This three big open source technologies are elaborated in detail in this section below.

A. Neo4j Graph Database

The neo4j is a one of the popular graph databases. The graph database is a database which stores data in the form of graph structures. It is a scalable, robust and high-performance database. It supports features like ACID transactions, high availability,high-speed querying through traversals, declarative graph query language, and scales to billions of nodes and relationships between them [1]. It does not require complex joins to retrieve connected linked data as it is very comfortable to retrieve its adjacent node or relationship details without joins or indexes. The neo4j stores application's data in terms of nodes, relationships, and properties [3]. The nodes are typically used to represent entities. The properties are key/value pairs. It can have zero or more relationships connecting them to other nodes. The relationships represent the relation between two nodes as illustrated in Figure 1.

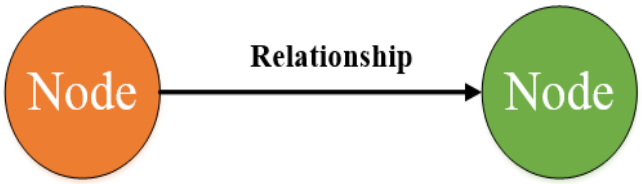

Figure 1: Representation of Nodes and Relationship

Each relationship must have a relationship type. In relationships the properties can be a key/value pairs. Both nodes and relationships can have properties, this properties can quantify relationships. For a node, zero or more labels can be assigned. The labels used to represent roles, categories or types. It is used to define indexes and constraints of nodes. The neo4j database uses cypher as a query language, that allows faster and efficient querying, updating and deleting of the graph database [3]. It focuses on the clarity of conveying what needs to be retrieved from a graph, not on how it should be retrieved. It is a relatively simple but still very powerful language. The complicated graph database queries can easily be written in cypher.

B. Apache Spark

The apache spark is an open-source framework built for solving big data problems which is faster and easy to use [4]. It has several advantages compared to other big data technologies like hadoop and storm. In apache hadoop, the programed output data between each step need to be stored in the distributed file system before the next step can begin. Thus, this approach tends to be sluggish due to replication & disk storage. It also requires the combination of different tools for different big data use cases like mahout which is used for machine learning and storm which is used for streaming data processing. With spark, programmers can develop complex and multi-step data pipelines using directed acyclic graph (DAG) pattern.

It holds intermediate results in memory rather than composing them to disk which is very useful, especially when need to work on the same dataset multiple times. It is designed to be an execution engine that works both in-memory and on-disk. It can store part of a data set in memory and the remaining data on the disk. The apache spark operators perform external operations only when data does not fit in memory. With the help of in-memory feature, in-memory spark enables applications in hadoop clusters to run up to 100 times faster in memory and 10 times faster as compared to running on a disk. In addition to map and reduce operations, it supports SQL queries, machine learning, and streaming data and graph data processing. The apache spark takes map reduce to the next level with more cost effective shuffles in the data processing. Due to its features like in-memory data storage and real-time processing makes apache spark faster as compared to other big data technologies.

C. *Ejabberd Server*

The ejabberd server is an instant messaging solution of the open source community, which is cross-platform, distributed, fault-tolerant and reliable [5]. It is built in erlang language and uses XMPP protocol at the backend. It is a protocol which is used as paradigm for message sharing between two devices over the internet. All the message transfer through ejabberd can be encrypted to enhance the security and privacy of the user.

In ejabberd server, the mobile reliability is provided with the help of stream management and message delivery receipts modules. The ejabberd server also supports interaction with the browser and the web app, facilitating real-time messaging. The role of web sockets provides the power to seamlessly send and get messages while a browser tab is opened. The BOSH capability is offered as a fallback for XMPP connection. Now all ejabberd core modules like mobile reliability features, are available from any mobile app or web page thanks to API interfaces of ejabberd server. The ejabberd server provides libraries, for web pages and mobile apps, to be easily integrated within customized source code of user.

There is no message loss in ejabberd server even during bad network connectivity, as each time when message is send, ejabberd check message delivery with an acknowledgement provided by device. If end user device is disconnected from server, a message is stored in server database called mnesia and it is sent back to the device whenever device again gets connected to ejabberd server. The ejabberd server can reliably support thousands of simultaneous users on a single node and has been designed to provide exceptional standards of fault tolerance.

IV. CONCEPTUAL FRAMEWORK

There are five key dataset in the internet of things they are listed

- Device graph
- Customer graph
- Location graph
- Permission graph
- Network graph

The first is the device graph, in this graph all the devices are connected to each other directly through a bluetooth connection or a wi-fi or a VPN or indirectly over the internet. The second type of graph is a customer graph, in this graph the people are collaborated and engaged with each other so it can serve up recommendation based on how people are connected with each other, what they like etc. The third graph is a location graph, in this graph people move to different location and based on this location the recommendation of different services are made to the users. The fourth graph is a permission graph, in this graph the permission of a user are checked, what the users has the access rights to talk to a certain device. Mostly used for service authorization, user privacy, user and device access control. The last graph is a network graph, in this graph the network is laid down by which the devices are connected to each other. It can be used for impact and dependency analysis, root cause detection or predictive analysis.

Now to graph this internet of connective things, the idea is very simple its starts out with a simplistic of graph which is node a thing connected to another node by a relationship. This concept is illustrated in Figure 2, where a user A is connected to a house B by a relationship owns. The ways it's modelled is shown in Figure 2, so cypher is graph query language used to query the graph database.

This is the way graph database is being used to store the connected dataset of the internet of things. But as discussed earlier even though neo4j is being a highly scalable and clustered database, it becomes evil when dealing with iterative operations and give the results in a slower fashion. To solve this issues in this proposed architecture apache spark is used in collaboration with neo4j database through a extension called as neo4j-Mazerunnner. This mazerunner distributes the graph processing jobs of neo4j to apache spark graph module, which allows neo4j to do big data graph processing jobs while persisting the results back to neo4j database

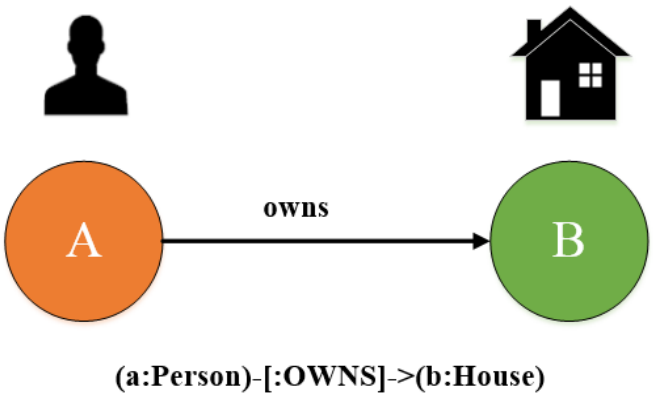

Figure 2: Person Node with relationship owns to House Node

The mazerunner distribute a graph processing jobs to graphX module of spark using a message broker. So whenever an agent job is shipped, a subgraph is send from Neo4j Graph database and written to HDFS. Once the neo4j graph database has written a subgraph to HDFS, a mazerunner services for

apache spark gets notified to start processing this data. This service will then resume and call a distributed graph processing algorithm using scala and spark's graphX module. The graphX algorithm is serialized and dispatched to apache spark for processing. Once the apache spark job completes, the results are written back to HDFS as a key-value list of property updates to be applied back to neo4j. The neo4j is then notified that a property update list is available from apache spark on HDFS. The neo4j batch imports the results and applies the updates back to the original graph.

Now all the processing part is done by the integration of apache spark and neo4j database, but there should an event processor which will be able to handles all the events from the clients more efficiently and reliable. To cope with this high expectations, in this proposed system ejabberd server is used, which is seamlessly robust, faster, reliable and fault-tolerant server. The ejabberd server is normally use for instant messaging solution used by well know names such as WhatsApp, Facebook chat etc.

The ejabberd accepts the events and reliable send those events for further processing to Apache spark and when apache spark send back the result of the request, it reliably send those event results back to the client efficiently and securely. Moreover ejabberd server stores offline event, that is if suppose when the client send the event for processing to the ejabberd server, ejabberd server forwards those request to the apache spark for processing, and when apache spark send a result of the event to ejabberd to be send back to the client, the ejabberd server check if the client is online or offline, if its online it will send the event instantly but if its offline, than the ejabberd server will store the event result in its mnesia database. And whenever the client come online again it will send the stored offline event to the client.

V. IMPLEMENTATION

This proposed architecture is divided into three main section that is sending the events to the ejabberd server, processing those events in the apache spark, querying the neo4j database from apache spark. There are basically three operation for this proposed architecture. Out of which two operation are ambiguous in nature they are node registration or device registration and creating a relationship between nodes and the third operation is the process request operation, in which some recommendation, predictions, or actions are made based on the event data received. And this result can be send to all the devices connected to that device, or the devices which have a specific relation with that device, or it can be only to a specific device. This operations are elaborated as follows.

A. Node Registration and Relationship Operation

In the Node creation operation, the request will be for device registration, in which the ejabberd server when received such kind of request will create a jabber id of that devices and send it forward to apache spark, which will fire a create a node cypher query on the Neo4j database and in the Relationship creation operation, in which the ejabberd server when received such kind of request will forward the request to apache spark, which will indeed fire a cypher query to create a relationship between two nodes. The Node creation and relationship operation is illustrated in Figure 3, which consist of the following steps:

1. The devices will send a event data with request type as node registration or relationship request to the ejabberd server.
2. The ejabberd server will check the request type of the event received and if it's a node registration request than it will create a jabber id to uniquely identify the device and forwards the request to Apache Spark and if the request is a relationship request than it will forward this received event as it is to Apache Spark.
3. Based on the request type apache spark will process the request accordingly. If it's a node registration request it will generate a create node cypher query or if it's a relationship request it will generate a create relationship cypher query.
4. Apache Spark fires relevant query on Neo4j graph database.
5. The resultant status about the query is send back to apache spark from Neo4j.
6. Apache spark send this status report about registration or relationship to ejabberd server.
7. Ejabberd check if the device is online or offline. If it's online then sends this status about the registration or relationship request to the device or if it's offline than store the status in tis offline storage and whenever the device comes online it will send this stored offline status to that device.

B. Process Operation

In the process operation, the request can be for recommendations to a user based on its location, likes, friends etc, it can also be some predictive suggestion based on the analysis of previous data or it can be just an event which all the other devices connected to this devices should be notified or only a specified device should be notified. The process event operation is illustrated in Figure 4, which consist of the following steps:

1. The registered device will send the event data with request type as process request to the ejabberd server.
2. The ejabberd server will check the request type of the event data. If it's a process request than it will directly forward this request to apache spark for processing it.
3. Apache spark based on the request type of the event and do the relevant processing of that event and generate a resultant cypher query.
4. Now Apache spark fires the queries on Neo4j graph database.
5. Neo4j database returns the relevant result of the query to Apache spark.
6. Apache spark forwards this result event to ejabberd server.
7. Ejabberd check if the device is online or offline. If it's online then sends this status about the registration or relationship request to the device or if it's offline than store the status in tis offline storage and whenever the device comes online it will send this stored offline status to that device or other connected devices.

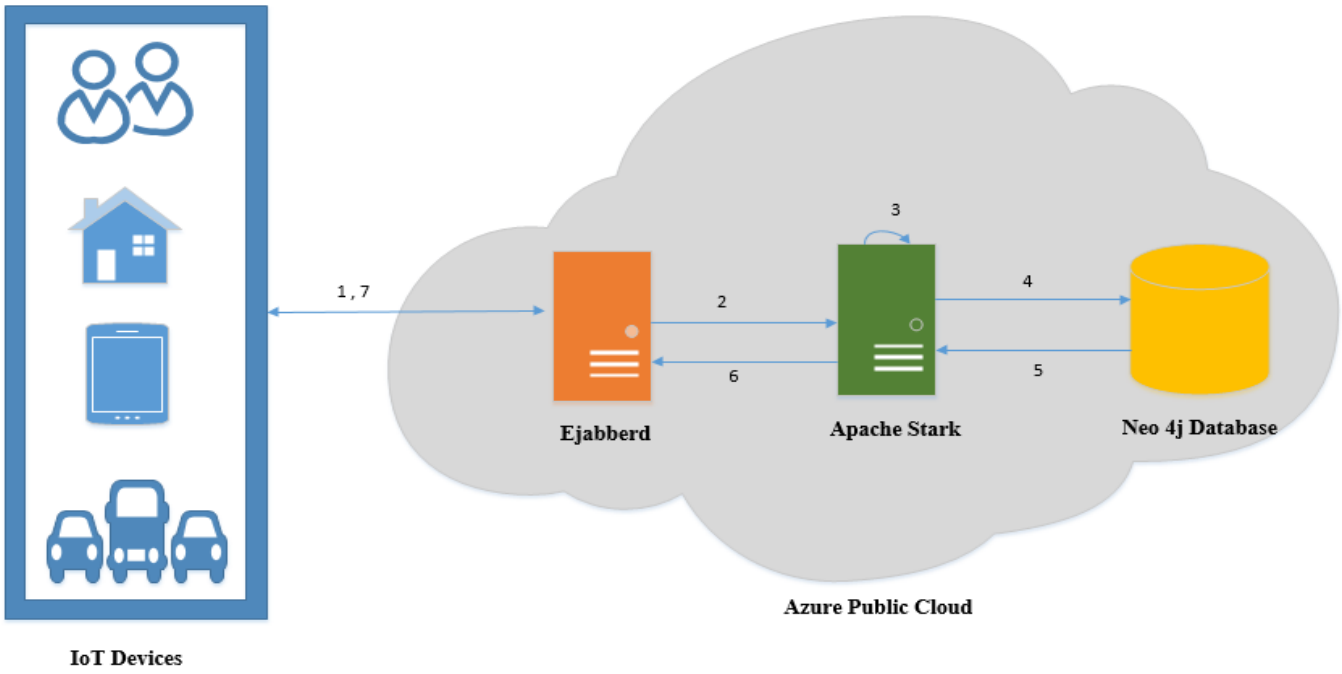

Figure 3: Node Registration and Relationship Request Operation

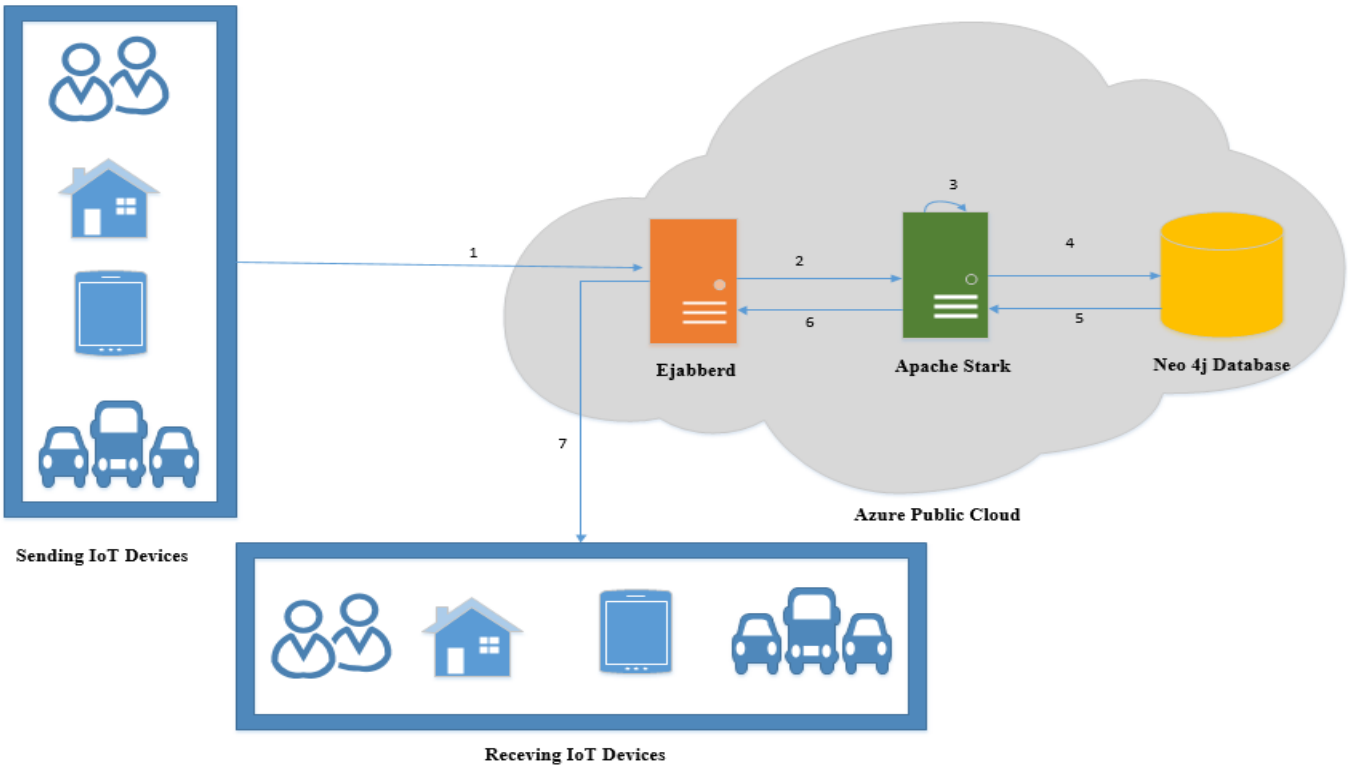

Figure 4: Process Request Operation

VI. RESULTS

A. Node Creation and Relationship operation

There are two nodes or things one is person and other is home node, where the person node is connected to home node with a relationship as owns. Ones the node registration request is received by the ejabberd it going to create two jabber id's one for the person node and other for the home node. After jabber id are created the ejabberd forward the node registration request along with the newly created jabber id's of both the node. The apache spark will then fire a create CQL query on the Neo4j database and create two nodes. Now for establishing connection between this two node with a relationship "OWNS", the client send a request to ejabberd, which will get the two jabber id's of the node with whom the connection is to be done with relationship "OWNS", than apache spark will fire a Match CQL on Noe4j database for this two nodes. The resultant graph is illustrated in Figure 5.

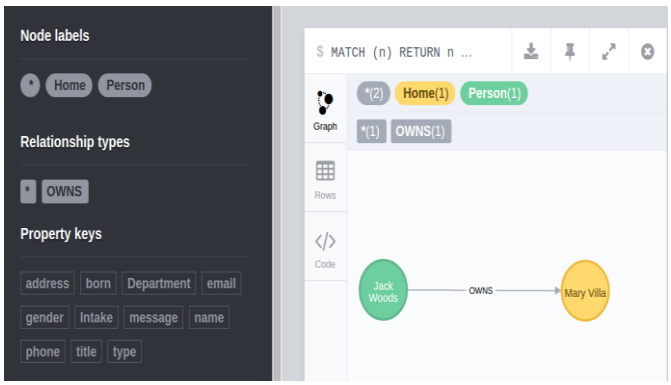

Figure 5: Representation of Nodes and Relationship

B. Neo4j database operations

Consider the above case of person and the home which is connected to each other with relationship owns. The home node sends event notification to the person node regarding the entities inside the home. When both this node are online or connected to internet, that the ejabberd make the necessary updates regarding its availability on the ejabberd dashboard as illustrated in Figure 6.

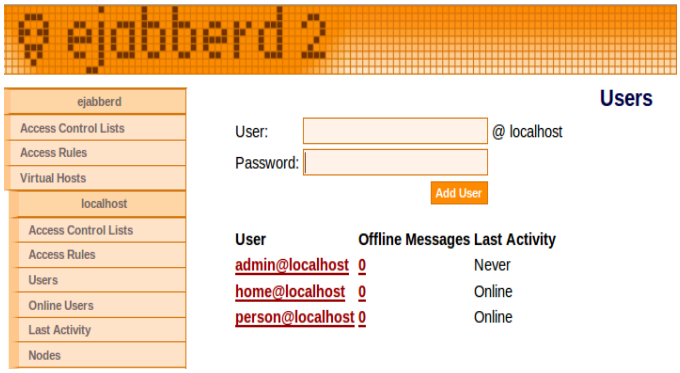

Figure 6: Ejabberd Dashboard

And when event messages are send from the home node to the person node, the ejabberd directly sends this event notifications to the person node. Now when the person node goes offline, ejabberd identifies it and make necessary updates regarding its availability on the ejabberd dashboard. And when home node is sending the event notifications to the person node, the ejabberd checks if the person node is online and offline, as its offline it will store the event notification in its mnesia database and once the person node comes online as illustrated in Figure 4, the ejabberd will send this event notification to the person node.

VII. CONCLUSION

Any device or an application which fully leverages the internet of things, no matter in what subset that iot lives in whether it's a mobile app, or whether is a hardware device, whether it's a infrastructure to manage a network than the application needs to be connection aware, that is connected data. This proposed architecture work with connected data very well and give the results at faster pace than the traditional relational database approach. This paper aim is to resolve the issues of connected data in iot domain with the integration of three open source technologies such as Ejabberd server, Apache spark and Neo4j database. To a get more efficient performance this three open source technologies, they are implemented on a Microsoft azure public cloud.

VIII. REFERENCE

[1] Internet of things: http://www.gartner.com/it-glossary/internet-of-things/ Accessed on 08.04.2016, 9:07 AM.

[2] P. Gaur, M. P. Tahiliani, "Operating Systems for IoT Devices: A Critical Survey", in Region 10 Symposium (TENSYMP), Ahmedabad, IEEE 2015 pp. 33 – 36, May. 2015.

[3]International Data Corporation http://www.idc.com/getdoc.jsp?ContainerId =prUS25658015, Accessed on 09.04.2016, 7:09 AM.

[4] The Impact of Internet of things on big data: http://data-informed.com/the-impact-of-internet-of-things-on-big-data, Accessed on 09.04.2016, 10:09 AM.

[5] Why graph databases are perfect for Internet of Things: https://dzone.com/articles/why-graph-databases-are, Accessed on 10.04.2016, 08:09 AM.

[6] Neo4j:http://neo4j.com/blog/graph-databases-perfect-internet-things, Accessed on 10.04.2016, 10:15 AM.

[7] Introduction to Neo4j: http://neo4j.com/docs/stable/introduction-highlights.html, Accessed on 10.04.2016, 10:45 AM.

[8] H. Huang, Z. Dong, "Research on architecture and query performance based on distributed graph database Neo4j" in Consumer Electronics, Communications and Networks (CECNet), 2013 3rd International Conference, pp. 533 – 536, Nov. 2013 .

[9] Graph Database: http://neo4j.com/docs/stable/graphdb-neo4j.html, Accessed on 10.04.2016, 11:05 AM.

[10] Neo4j and cypher: https://www.airpair.com/neo4j/posts/getting-started-with-neo4j-and-cypher, Accessed on 10.04.2016, 11:30 AM

[11] Introduction to Apache Spark: http://www.infoq.com/articles/ apache-spark-introduction, Accessed on 10.04.2016, 11:40 AM.

[12] A. I. Maarala, M. Rautiainen , M. Salmi ;,S. Pirttikangas , J. Riekki, "Low latency analytics for streaming traffic data with Apache Spark" in Big Data (Big Data), 2015 IEEE International Conference pp. 2855 – 2858, Nov. 2015.

[13] Ejabberd: https://www.process-one.net/en/ejabberd, Accessed on 10.04.2016, 11:55 AM.

[14] V. A. Bharadi and G. M. DSilva, "Online Signature Recognition Using Software as a Service (SaaS) Model on Public Cloud," in 2015 IEEE International Conference on Computing Communication Control and Automation, pp. 65–72, Feb. 2015.

[15] G. M. DSilva and V. A. Bharadi, "Modified Online Signature Recognition Using Software as a Service (SaaS) Model on Public Cloud," IEEE International Conference on Information Processing, December 16-19, 2015.