BigClue: Towards a generic IoT cross-domain data processing platform

Dan Huru and Cătălin Leordeanu and Elena Apostol and Valentin Cristea
Faculty of Automatic Control and Computers, University Politehnica of Bucharest, Romania
Email:alexandru.huru2208@cti.pub.ro, {elena.apostol, catalin.leordeanu, mariana.mocanu, valentin.cristea}@cs.pub.ro

***Abstract*—Initially IoT systems have been built as isolated solutions for each problem domain. This has implied a lack of standardization and interoperability. The global IoT vision aims to integrate distinct problem domains into a unified network in order to offer enriched context and meaningful correlations. Connecting global platforms with multiple IoT sensor networks will imply increased data processing requirements. In this paper we first present the main technical challenges and non-functional requirements demanded by a cross-domain IoT data processing platform. We then propose a cloud based data processing architecture that integrates a collection of suitable frameworks from existing state of the art work. In the end we validate the proposal with a reference implementation.**

***Index Terms*—IoT, data processing, cross-domain platform, big data, cloud computing**

I. Introduction

Today various embedded devices are capable of communicating and sharing data using the Internet. In this manner traditional web services are enriched with physical world services [1].

In addition to the IoT vision, which gives every device an IP address and interconnects them, there is also the notion of *Web of Things* (WoT) which enables the devices to be part of the world wide web. Both concepts relate to the common goal of achieving *ubiquitous* computing where computing is made to appear everywhere and anywhere [2].

Numerous research efforts such as [3], [1], [4] and [5] have concluded that reusing the existing *web architecture* to integrate new devices is the most viable solution in terms of scalability, costs and complexity. In this sense an important idea is to have smart devices act as tiny web servers that directly provide web services and thus expose their data.

The Representational State Transfer architecture (REST) [6] has been identified as the ideal candidate [7], [8] in WoT for its main two advantages: the low complexity and the loose-coupling stateless nature of its interactions. These two features allow REST services to be exposed by resource-constrained devices and enable an easy composition.

Finally a *mashup* of web services is built at an unprecedented scale, growing at high pace and producing massive amounts of data ready to be put to use to provide better services to our society. In this context, scalable, real-time management and reasoning over sensor data are becoming vital aspects. The way in which this data is integrated and processed has led to significant research efforts and has yet to reach maturity.

With regards to data processing, the tendency is to keep time series processing close to the device itself (aka edge computing) so that traffic flowing through the mesh is minimized. On the other hand, more complex processing is needed in a centralized place such as the cloud which has the big picture of the network. This helps to do further correlations, to make predictions based on historical data and to take decisions.

The work presented in this paper focuses on the cloud computing environment dedicated to collect, process and interpret IoT data. This data is represented by sensor data but also relevant metadata such as structured human knowledge, equipment details or spatial information.

We use the examples in the next section to identify common patterns of data processing and the main research implications. This ensures the motivation to propose a generic platform called BigClue that can handle data processing for different IoT domains. In the end we demonstrate a reference implementation in a smart farming use case.

II. Applications of sensor data processing

1) Smart farming: Sensors can contribute with data to enable smart farming environments. For example information about humidity, temperature or light can be used in conjunction with weather data to understand and better monitor farm crops. Sensor information can also be used to understand the development of plants and verify their behavior with existing domain knowledge. In addition location data and statistical information can provide a better context (e.g. soil type, wind, air composition, animals). By gathering information from several farms, different strategies can be shared among farms while the farming process becomes more solid. Legal and compliance information services can be used by both farmers and agencies to manage agriculture better. Examples of such an application is project ClueFarm [9] where authors propose to join the information obtained locally in a cluster of farms.

2) Water management: Water management is undergoing a rich transformation from the sectoral approach to integrated water management systems with interrelated processes within the water cycle. Due to the use of instrumentation and telemetry of water systems a palette of smart data applications is now possible. Obvious examples are the early detection of floods, excessive pollutants and the ability to track and understand the overall water lifecycle and its implications. In order for this to be achieved data must be collected and accessed in real-time and correlated with historical facts. Initiatives such

as Data4Water [insert ref] project are investing in raising the awareness and developing systems to tackle these challenges.

3) Smart homes: Houses and offices can make use of distributed sensors to automatically handle room heating, lighting and save energy with appropriate monitoring and alert systems. For example in [10] an energy-aware/cost ware platform for smart homes is implemented to monitor energy consumption. It also benefits real-time information about tariffs by integrating smart grid web services. Another example is EnergyVisualizer [11], which offers a web interface to control different home appliances and their energy consumption.

4) Transportation and logistics: In supply chains, *real-time monitoring* of products can be achieved by using RFID and NFC (e.g. purchases of raw material, transportation, storage, distribution). Product related information is obtained fast and accurately so that companies are able to adapt quickly to market changes. Some of the advanced companies in the industry like Walmart and Metro use these kind of technologies and are able to achieve zero stock [12], [13].

5) Health tracking: Today a variety of medical sensor devices can be used to track personal health information or make predictions about individuals' lifestyle and possible diseases. Background information can help identify and authenticate patients and reduce harmful incidents. Data collection can reduce form processing time and enhance automated care and medical inventory management. Sensing devices may automatically diagnose patient conditions and retrieve health indicators.

III. Main Research challenges

The current state of the art borrows many models from those that deal with existing Internet data. Large web companies such as Google, Yahoo or Facebook have led the advance in processing and reasoning techniques and thus have produced many pioneering technologies that deal with Internet data. Although these technologies have proven effective in relation to their business scenarios they have to be adapted in order to be able to scale within the IoT environment. This process of adaption creates new research challenges. We outline several of them in the following paragraphs.

1) Data processing: One challenge of processing sensor network is to understand how to efficiently balance in-network with centralized processing without losing important information or straggling the computational environment. Since data streams from sensor networks are noisy significant research is needed to understand how to perform real-time data cleaning in order to build reliable models. In addition, due to large volumes of collected data processing may benefit from improved compression and filtering techniques.

2) Processing platforms: Current real-time processing is mainly done on existing web data but the extension to considerably larger amounts of data produced by multiple sensor networks requires research and design of robust and scalable processing platforms. In addition these frameworks need to consider numerous other functionalities such as correctness debugging, performance debugging, scheduling policies or memory management.

3) New reasoning opportunities: Although reasoning has been addressed on many stand-alone systems (e.g. wireless networks), new opportunities and challenges for deriving useful information arise from the availability of various and heterogeneous new data sources. Their integration with the existing Internet domain knowledge creates room for considerable innovation.

4) Context sharing: So far neglected, contextual information sharing needs to be addressed when designing new middleware solutions. Many existing solutions concentrate on isolated applications. In the IoT paradigm, the inter-middleware communication (e.g. of semantic relationships) will be an essential requirement if the extraction of knowledge and its reusability is to be leveraged.

IV. Motivation

Initially IoT systems have been built as isolated solutions for each problem domain. This has implied a lack of standardization and interoperability. The global IoT vision aims to integrate distinct problem domains into a unified network in order to offer enriched context and meaningful correlations. There is a significant body of research in developing interoperable frameworks and systems to be able to aggregate multiple sensor communication technologies as seen in [4]. Since the global platforms will connect multiple IoT sensor networks, this will imply increasing data processing requirements.

The examples in the previous section reveal common data processing patterns between different IoT use cases. Specifically, there is a considerable number of use cases where real-time information must be processed and presented in a meaningful way.

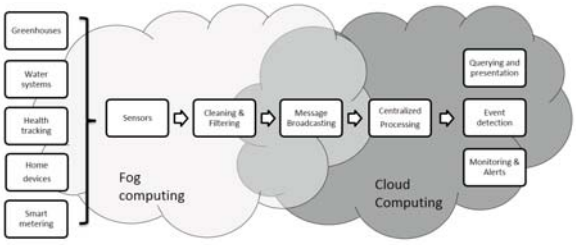

Fig. 1: IoT implementations follow a similar data flow pattern

The key observation in 1 is that, regardless of domain, time series data obtained from sensors must follow several common steps until end-user functionality is provided. In the left side of the picture 1 sensor data must be collected, filtered, cleaned and aggregated.

The requirements at this level are simple event processing mechanisms, low battery consumption, user friendly programmable interfaces.

The data is then broadcasted to centralized processing units to the right side of the picture within the cloud computing space. Here richer functionality can be provided such as advanced monitoring, event detection, alarm generation or fault root cause analysis, cross-domain and metadata correlations, context reasoning, reporting functionality and many others. These functionalities generally require increased processing power, persistence, machine learning models, metadata management, visualization and others.

In addition, the successful implementation of these use cases depends on the ability of the underlying platform to overcome several technical challenges. We present them below.

1) Layered architecture: A complete solution should be based on a layered architecture with multiple components. Each component should serve specific purposes and should be able to run independently. At the very most the components should be loosely coupled and should not provide too much functionality.

2) Scalable, flexible, distributed framework: Scalability can be addressed by using standards in place of customized solutions. Also the solutions need to support plug-in architectures which ease the adding or removing of components dynamically. New functionalities should be added without modifying existing modules. The architecture should be distributed and allow context sharing (real-time, historic) between components or frameworks at different levels.

3) Automated context life cycle management and model independency: Solutions should be able to dynamically discover available sources such as physical, logical or virtual sensors as manually managing the activation and deactivation of large number of devices becomes inefficient. Appropriate data models should be used to store and clean the raw information while requiring minimum human input. Data models should support complex relationships and be able to represent a variety of knowledge domains while being managed separately from framework specific models.

4) Exposed Application Programmable Interfaces (APIs): Exposing simple and efficient application programming interfaces will enable the functionalities of the framework to be accessed by other services and thus will simplify the composition of more complex solutions. Applications will use the APIs to implement specific business logic.

5) Multi-model reasoning: No unique reasoning method can satisfy all the requirements therefore multiple reasoning techniques must be applied depending on the use case. An ideal framework should provide a complete palette of data mining and analysis techniques in order to deliver rich functionalities.

6) Monitoring and event detection: Detecting events in real time will be an important asset of applications in the IoT environment. Actions, recommendations and other context information will depend on the successful implementation of monitoring and event detection strategies.

7) Real-time processing: Processing and reasoning must be done in a real-time fashion. Efficient methods for streaming need to be considered as well as robust algorithms that must analyze the data in one pass.. For example applications that trigger alerts are time-sensitive and the time of response may be significantly influenced by the large number of monitored devices.

V. Similar data processing platforms

There is a considerable number of platforms providing end-to-end data management solutions however the majority do not support real-time processing and advanced reasoning. With the exception of MoCA [14] which provides an extra semantic context, most of the solutions which do provide realtime capabilities present limited reasoning techniques such as the use of rules I.

MoCA [14] is a service based distributed middleware that employs ontologies to model and manage context. It consists of three key components: context providers, context consumers and context service. The providers are responsible for generating or retrieving context from other sources available to be used by the context management system while consumers consume the context gathered and processed by the system. The context service receives, stores and disseminates context information.

SCONSTREAM [15] is a system with a centralized architecture that processes streams of data coming from physical sensors and applies rules to detect events. It highlights the challenges that appear in an IoT environment from a real-time perspective.

UbiQuSE [16] is a middleware that connects with any type of sensor. It provides real-time query processing based on live streaming data and historic context. Reasoning is achieved using rules.

COPAL [17] is a middleware with a loosely coupled plugin architecture. It provides automatic code generation using abstracts for re-usability and extendibility. Distribution includes public-subscriber and query architectures.

Project name	Real-time processing	Reasoning	Distributed architecture	History and storage	Knowledge management	Event detection	Dynamic composition	Registry maintenance
MoCA	Y	Ontology-based	N	N	N	Y	N	Y
SCONSTREAM	Y	Rule-based	N	Y	N	Y	N	N
UbiQuSE	Y	Rule-based	N	Y	N	Y	N	Y
COPAL	Y	Rule-based	N	N	N	Y	N	Y

TABLE I: Comparison of systems with real-time processing capabilities

In addition to these research frameworks, the major cloud providers, Google, Amazon, Microsoft [insert ref] now offer IoT services and tools that can be composed to build such platforms. These services can be used in a serverless computing scenario where several of the non-functional requirements described previously are treated behind the scenes while

the platform developer can concentrate on implementing the orchestration of these services.

VI. Proposed architecture

In this section we propose a data processing platform called BigClue that integrates a collection of suitable frameworks from existing state of the art work. We describe the platform and how we tackle each of the technical constraints discussed previously.

While we argue that our proposal has enhanced capabilities compared to other platforms, our arguments rely on the existing research work that can be consulted in the bibliography. It is not the scope of this thesis to rework or re-validate what has been previously demonstrated.

We consider that the originality of the work consists in putting the right pieces together in the right way. Therefore we limit our experiments to a reference implementation to show that such a platform is viable and we do not engage in additional benchmarking or performance comparisons with other similar platforms.

We propose a reusable, scalable platform that employs a multi-tenant approach. It relies on a service oriented architecture and offers real-time processing capabilities, advanced reasoning and auto-scaling.

In order to achieve a **layered architecture** and deliver this kind of functionality the system is split into the following components:

- Processing - processing can take various forms (cleaning/rules/analytics)
- Messaging - data is collected from multiple sources and replicated to multiple destinations
- Storage - data needs to be stored in a persistent layer for later availability
- Service registry - services need to be auto-discovered
- Visualization - data must be presented in an useful way

In this sense we choose a micro-services architecture where each of the above components acts as an independent service with its own life-cycle.

Exposed APIs are achieved by choosing a communication based on HTTP protocol. We choose RESTful [6] in comparison to Simple Object Access Protocol (SOAP) because of the following benefits [7], [8]: low complexity, stateless nature, easy to implement, standardized (HTTP Protocol)

Although there are solutions which perform multiple roles we strive to keep a separation of concerns and discuss each component separately.

A. Data processing

Since processing is a broad subject and our task is to choose the best fit, in the following paragraphs we share more details about existing work and discuss pros and cons. We motivate our choices for the adopted techniques and technologies.

The map reduce paradigm and its implementation Hadoop has been widely adopted for situations where processing large amounts of data is needed but, due to the nature of its programming model, it has proven suitable only for offline batch processing. Efforts have been made to make it suitable for real-time analysis [18] and some of them may fit some near real-time use cases. However, in an IoT environment, the processing and analysis will be done in real-time and in one pass so tools to process streaming data need to be considered instead.

Nevertheless there will be still the need to mine historic data to allow a deeper analysis in parallel with the real-time counterpart. In fact today there are many implementations data that rely on a two tiered architecture (e.g. lambda architecture) that leverages the advantages of both components. In general the results of the offline analysis are fed to the real-time algorithms to infer better rules and classification models.

Model	Characteristics	Drawbacks	Implementations
Map-reduce	distributed, Batch-oriented, file-system storage	Not suitable for real-time processing	Hadoop
Streaming databases	Concepts similar to relational database systems	most not scalable; those distributed require complex fault tolerance protocols	Aurora, Borealis
NewSql databases	Distributed + ACID + Sql interfaces	Still young, not tested	H-Store, S-Store, Google Spanner
Large-scale streaming frameworks	Distributed, expose high-level APIs	Costly replication (usually done in an external storage); Do not support mixing streaming and batch tasks	TimeStream, Map Reduce Online
Messaging systems	Based on the publish/subscribe model; fast messaging systems	Replication done on external systems, kafka is less suited for stream processing	Storm & Trident, Kafka
Incremental processing	Same benefits of map-reduce	they store all their state in a replicated disk file system, incurring high overheads	Incoop [19], CBP [20] and Comet [21]
Stateless, deterministic computations	Distributed, batch-oriented, in-memory computing, easy computation of lineage through RDDs, both batch and streaming in one system	Supports only the point-to-point pattern	Spark + D-Streams (Spark Streaming) (said to be 20x faster than hadoop)
Lambda architecture	Batch + real-time (eg. Hadoop + Storm)	Integration may be difficult, usually based on customized solutions	Applied at Yahoo, Netflix

TABLE II: Comparison of real-time processing models

A full comparison of the possible approaches is listed in table II. Taking into account the IoT **real-time processing** and the **multi-model reasoning** requirements we choose to implement a lambda architecture presented in figure 2:

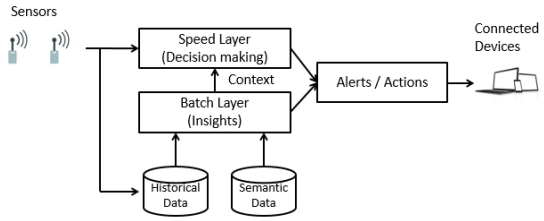

Fig. 2: Processing architecture

The first step is to collect sensor data and route it to a central endpoint. Then this data is processed two times: once in the speed layer where fast decision making is required, second in the batch layer where a time series algorithm models the historical pattern and generates predictions. The results are used in the speed layer to raise alarms and recommendations to the user. In this way **monitoring and event detection** functionality can be provided

A cross-domain IoT platform implementation requires to be **scalable, flexible and distributed** due to the increased data volumes, data models and business logic from various domains. Large-scale streaming is an essential functionality. There are numerous technical approaches for achieving such functionality including streaming databases, large-scale streaming engines, message-driven systems, or bulk incremental processing.

There have been a number of recent systems that enable large-scale streaming processing with the use of high-level APIs (see table III). Unfortunately these systems do not provide efficient fault and straggler recovery techniques. Also, most of them do not support mixing streaming with batch and ad-hoc queries.

Apache Spark [27] has gained a lot of attention due to its different approach and increased performance. Its authors claim that Spark is 20 times faster than Hadoop for iterative applications and can process 1TB in about 5-7 seconds.

The key concept in Spark is represented by RDDs [28] (resilient distributed datasets). They consist of a restricted form of shared memory which is based on coarse-grained operations and transformations (e.g. map, filter, join) to the shared state, as opposed to other systems which process fine-grained updates.

By applying the same operations to many data item sets it is possible to log the transformations and compute the lineage for each RDD, instead of the actual data. An RDD has enough lineage information to compute its partitions from stable storage. RDDs can express cluster programming models such as map-reduce, DryadLINQ, Haloop, Pregel or Sql and allow a more efficient fault tolerance than previous systems but are restricted to applications that perform bulk reads and writes.

D-Stream [29], [30] (or Spark Streaming) is an add-on to the Spark engine and is based on the idea of treating streaming computations as series of short interval batch computations. Because it is based on RDDs the process is also deterministic,

System	Characteristics	Drawbacks
TimeStream [22]	runs continuous, stateful operators in Microsoft StreamInsight	Recovery takes places on a single node for each operator; is proportional to the operator's processing window (e.g., 15 seconds for a 15-second sliding window)
	uses a recovery mechanism similar to upstream backup	
MillWheel [23]	runs stateful computations	reliability consists of writing all state in replicated storage systems (e.g. BigTable)
MapReduce Online [18]	a streaming Hadoop engine that pushes records between maps and reduces	recovery of reduce tasks with long-lived state is not possible
	uses upstream backup for reliability	does not handle stragglers
Meteor Shower [24]	uses upstream backup	can take up to minutes to recover state
iMR [25]	exposes a MapReduce API for log processing	can lose data on failure
Percolator [26]	runs incremental computations using specific triggers	does not offer high-level operators (e.g. map or join)

TABLE III: Comparison of large scale stream processing platforms

so lost data can be recomputed without replication and in parallel with the active computations. Consistency is ensured by atomically processing each record within the time interval in which it arrives. Spark streaming inter-operates efficiently with Spark's batch features. Users can express ad-hoc queries and use the same high level API for processing both historical and streaming data.

We choose to use Apache Spark for both speed and batch layers. In summary Apache Spark has the following differentiators:

- Low latency processing
- In-memory distributed computing
- Data recovery is fast and straightforward
- Can be easily integrated with other technologies
- Opensource, community is large

B. Messaging, storage and service registration

Internet companies invested significant time in distributed messaging systems which are able to process and send large amounts of data. Using these platforms (see table IV), developers can write stateful code to process individual records.

Apache Kafka [32] is a lightweight message broker that handles the delivery of sensor data to the processing and storage layers. We choose Apache Kafka because of the following differentiators:

- Lightweight, very fast
- Distributed system, fault tolerant
- Guarantees at-least-once message delivery
- Easy integration with other technologies
- Open-source, community is large

Platform	Characteristics	Drawbacks
Storm [31]	"at least-once" delivery	keeps all state in a replicated databases, expensive
Trident	exposes a high-level API on top of storm	
Kafka [32]	faster than other messaging systems, "at least once" delivery, allows automatic failover to replicas, supports point-to-point and publish subscribe models	is less suited for legacy systems, less capability for stream processing, no guarantee on the ordering of messages coming from different partitions, requires deduplication logic at consumer

TABLE IV: Comparison of big data messaging systems

One of the key functions of streaming databases is the possibility of running continuous queries that produce different results as new data arrives.

In this sense recent research has resulted in S-Store [33], a data management system that combines OLTP transactions with stream processing. It is based on H-Store [34], a distributed, row-store based relational database which runs on clusters of shared-nothing, main memory nodes. H-Store is a type of MMDBS (Main Memory Database Systems) which means that it performs computations in-memory.

H-Store is part of the emerging class of new database systems, called NewSql which seek to retain the advantages of traditional database systems (ACID and Sql interface) while leveraging the scalability and speed of NoSql engines. In H-Store fault tolerance is realized using a combination of command logging and periodic snapshotting. Each node hosts one or more sites that execute an autonomous instance of a storage engine. Because multi-side nodes don't share data structures with collocated sites, there is no need to solve concurrency issues.

S-Store is a research effort that adds streaming primitives to H-Store to support streams, windows, triggers, and workflows. S-Store inherits all benefits from H-Store and adds the lacking streaming functionalities.

Although they are good fit for a significant number of uses cases, streaming databases generally lack scalability and fault tolerance. Distributed databases such as Borealis [35] use replication or upstream backup for recovery but require complex protocols. A more efficient recovery mechanism, namely parallel recovery is done in [36], but does not handle stragglers and can tolerate at most one node failure.

S-store is still under development and has not benefited from extended benchmarks and real use cases. It is still not clear how it performs under various workloads. Also, because it does not keep any data in non-volatile storage, it must use other strategic data distribution schemes to maintain high-availability.

Our requirements imply the need for scalability and fault tolerance so we focus our attention on NoSql category of databases. Since in our use case we value consistency and partition tolerance over availability we choose Apache Hbase. Hbase has the following features:

- Distributed key store
- Compatible with Hadoop (HDFS)
- Offers random access
- Big community, open-source

The underlying file system is HDFS (Hadoop Distributed File System) with the following benefits:

- Low cost per byte
- Solid data reliability
- High bandwidth to support MapReduce workloads

Our choice for service registry and auto-discovery is Consul, from HashiCorp, an open-source software. Consul has the following features and advantages:

- Service discovery
- Health checking
- Key value store
- Distributed architecture
- Multi data-center

At startup time a service sends information to Consul which registers it and exposes it to other services.

C. Technology stack

The overall technology stack is presented in figure 3:

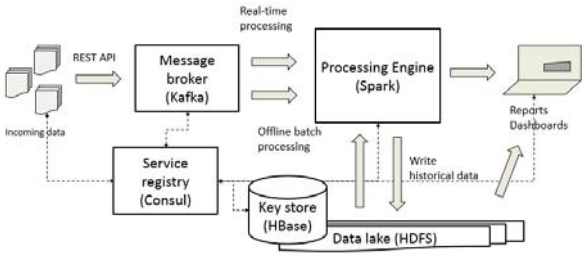

Fig. 3: Technology stack

VII. Reference implementation

One of the implemented use cases is ClueFarm project (http://cluefarm.fitx.ro/) which aims to aggregate, process, analyze and present farm related information. It offers cloud-based farm management services for indoor agriculture in order to increase crop productivity and lower knowledge barriers.

ClueFarm project requires the following services to be implemented:

- Monitor the development of farm resources and the environmental conditions
- Crops or livestock management
- Estimation of the productivity and management solutions for the future
- Imposing farm regulations by authorities and tracing how the farms meet them
- Other third party functionalities

The monitoring process is very important for farming. First of all, it provides records regarding the farm's development. This is regarded as the farm resource monitoring process and it may vary from supervising the crops growing to tracking the welfare of the farm's livestock. By analyzing this monitoring

data, we can obtain an overview of the production rate increases and decreases. Secondly, the monitoring process can be used for supervising the environmental conditions, such as water quality or land condition. This helps keep the environment in a good condition, according to the environmental regulations, and indirectly it also leads to long-term production growth.

We choose a typical use case of monitoring to test our platform: to detect anomalies on streaming sensor data. This time we rely on basic statistical functions. These functions are applied on a window of time-series data. Using mean and standard deviation we calculate vectors of min and max values. Values outside the limiting vectors are considered outliers (an arbitrary stdFactor is manually chosen).

Algorithm 1 Outlier detection using statistical functions

```
Compute mean vector
Compute variance vector
STD ← sqrt(variance)
minValues ← (std − mean)
maxValues ← (std + mean)
if newValue between [MinValues, MaxValues] then
   NOK
end if
return OK
```

We are considering two sources of time series: internal temperature and humidity, collected from greenhouse sensors. The data is collected for a 2 months' period (beginning of September till end of October) from a local greenhouse. The frequency of collected values is 15 minutes.The data set records contain:

$$sensorID|timestamp|temp|hum|externalmeasurements$$

The sensors are collocated and they present high data correlation with external weather conditions. In 4 a day/night cycle can be observed.

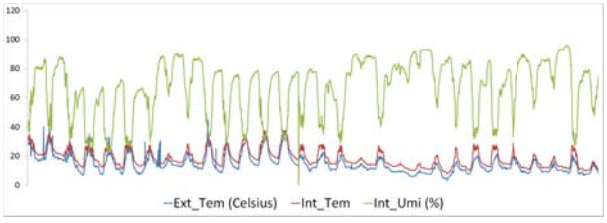

Fig. 4: 2 months of sensor data

In figure 5 a screen-shot is taken displaying outliers in streaming real-time sensor data.

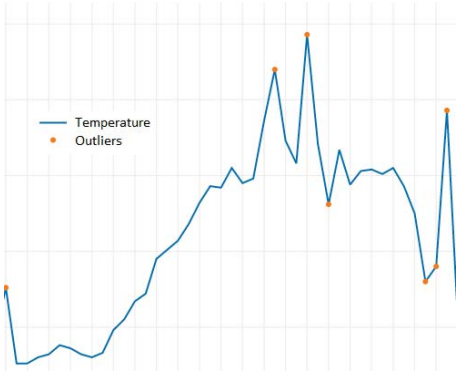

Fig. 5: Outliers in temperature data

When talking about the vizualization it is important to choose a technology that enhances reports and views on streaming data. We choose to use JSF, Amcharts and Primefaces for this implementation. In figure 6 we can see the main class diagram used for the implementation:

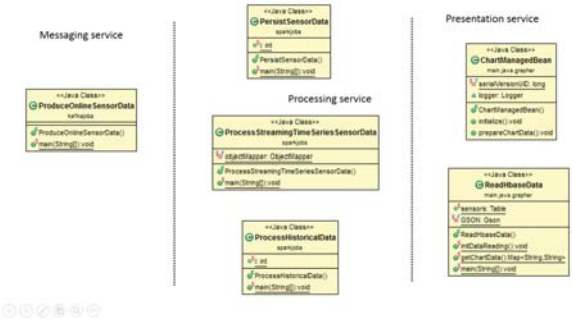

Fig. 6: Reference implementation main class diagram

The functionality of the Java classes of each component is described below:

- *Message broker*: ProduceOnlineSensorData reads and broadcasts JMS messages with sensor readings.
- *Data processing*: PersistSensorData reads and stores sensor information in the database. ProcessingStreamingTimeSeriesSensorData applies statistical functions to incomings time series windows. ProcessHistoricalData is used to apply prediction algorithms on historical data.
- *Data visualization*: ReadHbaseData reads sensor data persisted in the database to be shown in the graphical user interface. ChartManagedBean is used for visualizing the incoming data and detected anomalies

VIII. Summary

In this paper we first motivated the needs and discussed the technical challenges for a cross-domain IoT platform. We proposed BigClue, a data processing solution suitable for multiple IoT domains. It is a platform which integrates multiple existing big data technologies in order to accommodate a cross-domain IoT implementation and to provide room for value added services. While presenting the solution

we motivated our architectural and technology choices and explained how these fulfill the overall requirements. We then presented a reference implementation developed for a smart farming use case.

IX. Acknowledgement

The work has been supported by the project Data4Water: Excellence in Smart Data and Services for Supporting Water Management, number 690900/H2020-TWINN-2015, and the UPB project GEX AU 11-17-11 Activ. 4000.130.

References

[1] J. He, Y. Zhang, G. Huang, and J. Cao, "A smart web service based on the context of things," *ACM Transactions on Internet Technology*, vol. 11, no. 3, pp. 1–23, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2078316.2078321

[2] "Ubiquitous computing - wikipedia," https://en.wikipedia.org/wiki/Ubiquitous_computing.

[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Thinags: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1389128610001568

[4] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[5] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey," *JOURNAL OF COMMUNICATIONS*, vol. 6, no. 6, pp. 424–454, 2011.

[6] "Representational state transfer - wikipedia," https://en.wikipedia.org/wiki/Representational_state_transfer.

[7] "TinyREST: A protocol for integrating sensor networks into the internet," *Proceedings of REALWSN*, pp. 101–105, 2005. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.112.5129{&}rep=rep1{&}type=pdf

[8] "Fixed-mobile hybrid mashups: Applying the REST principles to mobile-specific resources," vol. 5176 LNCS, 2008, pp. 172–182.

[9] E. Apostol, C. Leordeanu, M. Mocanu, and V. Cristea, "Towards a hybrid local-cloud framework for smart farms," in *Proceedings - 2015 20th International Conference on Control Systems and Computer Science, CSCS 2015*, 2015, pp. 820–824.

[10] A. Kamilaris and A. Pitsillides, "Exploiting Demand Response in Web-based Energy-aware Smart Homes," in *ENERGY 2011, The First International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, 2011, pp. 27–32. [Online]. Available: http://www.thinkmind.org/index.php?view=article{&}articleid=energy{_}2011{_}2{_}20{_}50047

[11] "Towards the Web of Things : Web Mashups for Embedded Devices," 2009, pp. 1–8.

[12] Yuan, Shumin, and Baogang, "Value Chain Oriented RFID System Framework and Enterprise Application," in *Science Press, Beijing, 2007*, 2007.

[13] K. Kalyanam, R. Lal, and G. Wolfram, *Future Store Technologies and Their Impact on Grocery Retailing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 141–158. [Online]. Available: https://doi.org/10.1007/978-3-540-72003-4_9

[14] J. Viterbo, V. Sacramento, R. Rocha, G. Baptista, M. Malcher, and M. Endler, "A middleware architecture for context-aware and location-based mobile applications," vol. 0, pp. 52–61, 10 2008.

[15] O. Kwon, Y.-S. Song, J.-H. Kim, and K.-J. Li, "Sconstream: A spatial context stream processing system," vol. 0, pp. 165–170, 03 2010.

[16] K. Conroy and M. Roantree, "Enrichment of raw sensor data to enable high-level queries," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6262 LNCS, no. PART 2, 2010, pp. 462–469.

[17] "COPAL: An adaptive approach to context provisioning," 2010, pp. 286–293.

[18] "MapReduce Online." *Proc. of the NSDI - Conf. on Networked Systems Design and Implementation*, p. 15, 2010. [Online]. Available: http://static.usenix.org/events/nsdi10/tech/full{_}papers/condie.pdf

[19] "Incoop: MapReduce for incremental computations," *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*, pp. 1–14, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2038916.2038923

[20] "Stateful bulk processing for incremental analytics," 2010, p. 51. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1807128.1807138

[21] B. He, M. Yang, Z. Guo, R. Chen, B. Su, W. Lin, and L. Zhou, "Comet: Batched stream processing for data intensive distributed computing," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 63–74. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807139

[22] "TimeStream: Reliable Stream Computation in the Cloud," *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*, p. 1, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2465351.2465353

[23] "MillWheel: Fault-Tolerant Stream Processing at Internet Scale," *Proceedings of the the VLDB Endowment*, vol. 6, no. 11, pp. 734–746, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2536229{%}5Cnhttp://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/41378.pdf

[24] H. Wang, L. S. Peh, E. Koukoumidis, S. Tao, and M. C. Chan, "Meteor shower: A reliable stream processing system for commodity data centers," in *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012*, 2012, pp. 1180–1191.

[25] "In-situ MapReduce for log processing," *USENIX Conference on Hot Topics in Cloud Computing*, p. 26, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2170444.2170470

[26] D. Peng and F. Dabek, "Large-scale incremental processing using distributed transactions and notifications." in *OSDI*, vol. 10, 2010, pp. 1–15.

[27] M. Zaharia, "An Architecture for Fast and General Data Processing on Large Clusters," Ph.D. dissertation, 2014. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.html

[28] M. Zaharia, M. Chowdhury, T. Das, and A. Dave, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," *Nsdi*, pp. 2–2, 2012. [Online]. Available: https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf

[29] "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," *Sosp*, no. 1, pp. 423–438, 2013. [Online]. Available: http://dx.doi.org/10.1145/2517349.2522737

[30] "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters," *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing*, pp. 10–10, 2012.

[31] A. Toshniwal, J. Donham, N. Bhagat, S. Mittal, D. Ryaboy, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, and M. Fu, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*, 2014, pp. 147–156. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2588555.2595641

[32] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," *ACM SIGMOD Workshop on Networking Meets Databases*, p. 6, 2011. [Online]. Available: http://research.microsoft.com/en-us/um/people/srikanth/netdb11/netdb11papers/netdb11-final12.pdf

[33] "S-Store: a streaming NewSQL system for big velocity applications," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1633–1636, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2733004.2733048

[34] "H-store: a high-performance, distributed main memory transaction processing system," *VLDB '08*, vol. 1, pp. 1496–1499, 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1454159.1454211$\delimiter"026E30F$nhttp://www.vldb.org/pvldb/1/1454211.pdf

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

[36] J. H. Hwang, Y. Xing, U. Çetintemel, and S. Zdonik, "A cooperative, self-configuring high-availability solution for stream processing," in *Proceedings - International Conference on Data Engineering*, 2007, pp. 176–185.