2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation

Data Flow Management and Visual Analytic for Big Data Smart City/IOT

P. Bellini, F. Bugli, P. Nesi, G. Pantaleo, M. Paolucci, I. Zaza
DISIT Lab, Dept. of Information Engineering, University of Florence, Italy
https://www.disit.org, https://www.snap4city.org

***Abstract* — In recent years, the number of Internet of Things and Internet of Everything (IOT/IOE) paradigms has increased significantly. The large number of devices contributed to generate a huge amount of data (Big Data) inserted in Smart City solutions, which are experiencing an explosion of complexity, also due to the increment of protocols, formats and providers. In this perspective it becomes essential to create a data indexing infrastructure that can optimize the performance of the system itself, for creating the so called data shadowing on IOT and other data on cloud. Therefore, it is fundamental to study paradigms to manage the indexing and visual analytics a great variety of data including IOT/IOE. One of the important aspects to be addressed for managing data in the smart city context are: the uniform model, the performance and scalability, response times in research, and the possibilities of performing visual analytic such as data flow analysis and drill down. All these needs imply the creation of a Smart Solution capable of managing and analysing heterogeneous kinds of data, providing a multitude of final applications based on the type of user who requires a certain service. To this end, in this paper, a unified model for IOT/IOE and data ingestion is presented. In addition, two possible architectural solutions have been implemented and compared in terms of performance, resource consumption, reliability and visual analytic tools for *data flow*. The solutions proposed for data indexing and shadowing have been tested in the context of Snap4City pilot Helsinki and Antwerp for smart city of EC project Select4Cities.**

Keywords* — *Smart City, IoT/IoE, Big Data Infrastructure, Big Data indexing, IOT data shadow.

I. INTRODUCTION

Smart City solutions are strongly influenced by Internet of Things (IoT) and Internet of Everything (IoE) potentialities and technologies. The **first wave** of Smart City was strongly focused on open data with solutions such as CKAN [1], OpenDataSoft [2], mainly oriented on data sets production, collection, publication and exchange. In some cases, they provided access to effective datasets, by using data integration and visualization tools which allowed the creation of graphic charts, such as distributions, pies and histograms, based on the values contained in the dataset. A **second wave** of Smart City technologies proposed data aggregators integrating various data sets in a unique model and storage, to make them accessible via some Smart City APIs [3], [4]. The large variety of different and heterogeneous formats and protocols used for data warehouse required the adoption of data mining reconciliation algorithms in the data aggregation process to enhance interoperability among tools and services, as well as for connecting different representations of the same city entities coming from different data sets and/or providers. Besides, such data aggregation solutions also provide access to aggregated data as Linked Data (LD), Linked Open Data (LOD), coding data information in terms of RDF triples [5]. There are several advantages in creating a sort of knowledge base for the city, since it can be regarded as the first step to create an expert system on the city entities and their relationships. A **third wave** arrived with the IoT/IoE solutions. On the other hand, the present evolution of smart city also demanded for Big Data techniques for the presence of a great variety of heterogeneous data coming from different and rapidly changing sources and on which the data analytics is needed for predictions, decision support, etc. Administrators need to know the progress of the state of their city, for instance through the use of visual tools such as a set of dashboards to be supported in the decision-making processes [6]. The expectations of **end users** in receiving information about city services are growing. Moreover, citizens which are taking part of the evolutionary process of a Smart City may contribute by providing their data, only if the solution is respecting their privacy (on their data on mobile App, IoT/IoE, Social Media, also more traditional systems such as web pages forms, blogs, mail, etc.). In addition, they are interested in receiving notifications, suggestions to be involved in their city. Among the **city stakeholders**, developers are interested in real-time logs of activities and features of the data management system, which may allow them to create new applications, for the administrators and for the final users.

In order to manage the rapidly growing number of IoT connected devices (sensors, actuators and various agents) and the large amount of heterogeneous data produced in a Smart City environment (representing a significant stress case for the emerging Industry 4.0 paradigm), a Big Data approach is necessary, in order to efficiently integrate data and produce additional knowledge.

Thus, the aggregated data can be used to produce smart services by generating predictions and suggestions for final users and decision makers based on machine learning. These kinds of solutions are adopted not only at research level, but also in commercial activities (medium-sized and large enterprises) [6]. Advantages of such approaches are becoming increasingly evident both in economic terms (reduction of costs, development of more competitive products on the market, involvement of the greatest number of possible buyers, etc.) and in technical aspects (e.g., energy saving, use of safe and reliable infrastructures).

As a conclusion, the modern Smart City solutions must be capable of: i) ingesting a variety of data coming from different providers and adopting different paradigms/ protocols, and formats, etc.; ii) aggregating, managing and indexing a Big amount of Data; iii) supporting Big Data analytics; iv) providing a set of tools to the final users to manage IoT devices, visualize the status of the city via a set of dashboards, receive suggestions and recommendations, log the system/device activities, etc. depending on their role, needs and permissions.

The presented work has been realized in the context of Snap4City Pilot, https://www.snap4city.org of Select4Cities PCP European Commission project. Snap4City presents a Smart City Big Data architecture enabling and supporting

978-1-7281-4034-6/19/$31.00 ©2019 IEEE
DOI 10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00276

IoT/IoE, providing tools to ingest, process, enrich, visualize and monitor Smart City data. It is evident, that from the above discussion that the data management, indexing and analysis is enabling a large amount of applications. In fact, the data analytics and the decision making are viable only if the data can be retrieve efficiently and taking into account all data entities relationships. Therefore, the experiment performed and reported in this paper, aimed at comparing two different kinds of indexing solutions in terms of performance, resource consumption, reliability and visual analytic tools for data flow. The solutions proposed for data indexing and shadowing have been tested in the context of Snap4City pilot Helsinki and Antwerp for smart city of EC project Select4Cities.

Thus, the paper is structured as follows. In Section II, the main features of Snap4City requirements and architecture are presented. Section III discusses the Unified Data Indexing model and layer, and two possible solutions for its implementation based on SolrCloud and Elasticsearch. In Section IV, the comparison among these two solutions in terms of performance, reliability and scalability are reported. In Section V, the Dashboards realized to perform visual analytics of the two different solutions are described and compared (by using Banana and Kibana, respectively). Conclusions are drawn in Section VI.

II. SNAP4CITY REQUIREMENTS AND ARCHITECTURE

The Snap4City solution allows to ingest and manage Big Data coming from IoT devices, applications and services, and compute actions for users, for instance providing notifications and a set of visual tools enabling the production of interactive dashboards for data analytics and supporting decision-making processes (useful for many different kinds of users: Public Administrations, final users, developers etc.). Ingested data are collected and aggregated in the Snap4City Knowledge Base, connected to the Km4City multi domain ontology, and indexed in order to speed up and facilitate data retrieval actions. Snap4City allows also the creation of data-driven applications, based on Microservices, exploiting mobile and web apps, flows of processing data and IoT data running on the platform [7].

A. General data flow

In a Smart City infrastructure **many different kinds of data flows** have to be managed and analysed in real time for value. The sources from which the data can be collected could be typically classified into: IoT Devices, External Services, Storages, as described in the following. After the data ingestion (both IOT and External Services), the data have to be hopefully managed uniformly to facilitate the exploitation of data ingested and also of those produced by the intermediate results or for predictions, heatmaps, suggestions, etc. To this end, a semantic regularization process is also needed. To this purpose, a number of ontologies have been proposed [10], [11], while in the context of Snap4City, Km4City ontology is used [14]. Finally, it has to be possible to perform on data flow with respect to Storage: indexing and searches, data analytics, visualize and render them with customized dashboards or other visual tools etc., for business intelligence and/or visual analytics.

- ***IoT Devices*** (sensors and actuators) exploit many different protocols, (e.g., MQTT, COAP, NGSI, AMQP), most of which are **data driven**. The devices are usually connected through to pone or more IoT Brokers, which manage subscription to their services presenting APIs for subscribing to data updates (e.g., telemetry). IoT Brokers represent the main interface to reach IoT Devices: subscribe, and in most cases also updates, or for just monitoring the data flow without addressing historical data. Most of the IoT Brokers are typically capable to address only one communication standard, such as: MQTT, COAP, AMQP, NGSI, OneM2M, SigFOX; only few of them cope with multiple protocols, and few of them support secure connection in HTTPS/TLS with mutual authentication. In a Smart City, the amount of data flows due to IoT Devices may create a relevant traffic in the cloud network and may need a certain computational workload to justify the usage of IoT Edge devices, in which some decentralized distributed resources can be allocated (for instance, data analytics for estimating the average values of environmental variables, to take local decision on controlling the public light, to control the road directions and signages) [8], [9].
- ***External Services*** are typically exploited by periodic processes running on the infrastructure, typically called ETL, Extract Transform Load. For example, for collecting partial resulting data (typically static and real time data) from third parties, providing data flows entering into the infrastructure. Examples of External Services are ITS (Intelligent Transport Systems) which collect data regarding traffic flow sensors, Traffic flow reconstruction, parking management, semaphore status, connected drive, public transport data and status (e.g. bus schedule, paths etc.); weather forecast and environmental agencies; waste collectors; hospitals and healthcare centres. External Services may provide data and service access via APIs, Rest Calls, WS, FTP, etc., as well as trough upper level protocols such as: DATEX II, OneM2M, etc. [9].
- ***Storage:*** are the data transfer activities performed by the internal processes with respect to the different storages in the infrastructure. For example, (i) to recover data to perform data analytics, (ii) to save data into the storage once collected from the external services, or computed by Data Analytics processes.

In the first two cases, the **data quality** in input has to be monitored in real time. For example, by detecting anomalies with respect to the typical values, delay or missing transmissions, strange ranges in the data, etc. The control of the data arriving according can be performed on the basis of the historical/typical data values and data description, but also taking into account the data type. For example, by estimating predictions and/or anomaly detections on the basis of historical values, and/or by defining healthiness criteria for each data flow, i.e., rules based on data retrieval frequency, non-stationarities, conformity into bounds, etc.

In **Figure 1**, the main components of Snap4City solution regarding data ingestion for data shadow and indexing are reported. The architecture presents areas for: data ingestion tools (IOT Brokers, External Services), data management tools (storage for collecting raw data and results of processing), data processing (including data analytics and IOT applications), and finally, as in the focus of this paper, the **Back-End Architecture** for data management, indexing. Once the data are collected and indexed, they can be

exploited by Smart City API, providing access to web and mobile Apps and also to Dashboards.

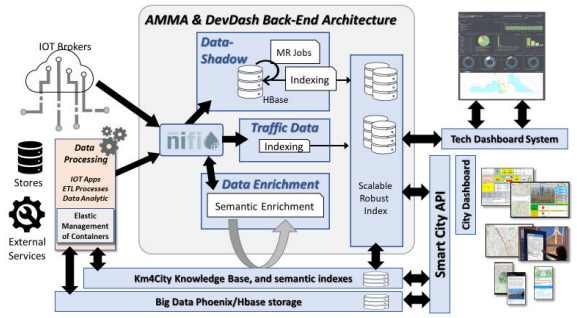

Fig. 1. Snap4City Indexing Back-End Architecture.

In the next paragraphs, a description of the main components of the Back-End Architecture is provided:

NiFi is an enterprise data flow bus that allows to automate, monitor and track dataflows among different systems, in a safe and reliable way [12], [16]. NiFi tool has been adopted in two instances to manage Big Data flows coming from both IOT Brokers and EventLogger.

- The first instance is capable to perform subscription to all the IOT Devices via a set of IOT Brokers: MQTT, NGSI, etc., and thus it is exploited by the Data Shadow.
- The second instance is dedicated to perform estimation of Traffic Flow data. Not discussed in this paper.

In both cases, the NIFI processes need to semantically enrich the received data with contextual information which are recovered from the knowledge base. For example, static IOT Devices can be enriched to quality control criteria on the basis of their ID, which cannot be recovered from the IOT Broker. In order to reduce the workload on the Knowledge base, a cache has been installed.

Data-Shadow. The data shadow is a feature provided as a plus by many IOT solutions for example AWS and IOT Azure. Data shadow is a mandatory feature to allow access at the IOT Device data when they are not sending a new data. In fact, IOT devices are typically addressed as data driven devices, and the historical data are not collected brokers. In some cases, the Data Shadow is a feature provided by the IOT services for example SigFOX, The Things Network services. On the other end, in full stack infrastructure, the data shadow has to be implemented to allow collecting data for data analytics, machine learning and in general for data analysis, drill down on time, anomaly detection, etc. In Snap4City, Data Shadow module collects in a persistent data storage the data generated by all IOT Devices as time series with needed contextual enriched data to facilitate the indexing. In fact, the indexing is another relevant plus of advanced data shadow, that in most of the IOT Services, as SigFOX, The Things Network, is not provided. The Data Shadow can be also used by ETL processes. This approach allows treating both periodic and even driven data uniformly and thus may simplify the access to the historical data for all kinds of data sources.

In Data Shadow activities of **data indexing** for creating a **Scalable Index** are performed. This component can be implemented with different solutions as described in the sequel of the paper. To this end, two competing solutions may be used for example SOLR Cloud and Elastic Search Cluster, have been considered. The produced index has to be scalable, has to capable of providing services via API REST Call and has to be efficient in terms of memory consumption. Section III, two different technologies (Solr Cloud and Elastic Search) are compared in in terms of performance in Section IV. With the aim of realizing a **Visual Analytic** tool (as Tech Dashboard System, see Section V) that has to provide support for understanding eventual dysfunction and anomalies, thus the data shadow index need to be browsed with faceted solution in space, time, relationships. Details on this regard are reported in the rest of the paper.

Smart City API is a layer of REST Call API which abstract from the complexity of the Smart City Back office. A number of services are accessible via Smart City API [14], [3]. Among the most relevant services, there are the API for accessing to semantic queries for spatiotemporal searches, full text search on KB elements, search on data shadow, search on historical data, search on data analytics results, etc. [3]. All the high level applications and services communicate through Smart City APIs, for many different purposes, dashboards, web and mobile Applications [13], [14]. Applications for the final Users (public administrations, citizens but also system managers) are based on Smart City API, in form of visual and interactive tools allowing the creation of Mobile and web Apps, as well as graphic dashboards, thanks to the presence of the **City Dashboard** System [15].

III. Data Indexing Layer

For the Data Shadow, the NIFI processes collect and enrich data (i.e., IoT and ETL) regularizing and enriching (as described above) them towards a unified structure that is adopted in the index: (<d1, v1>, …, <d16, v16>), where the keys d1, …, d16 as described in TABLE I. The unified data structure has been obtained generalizing data models communing from multiple IOT devices/Brokers and ETL processes, thus producing an unified interoperable IOT/ETL Data index.

TABLE I. Unified Data Structure for Data Shadow

Key	Id	Description
d_1	*Id*	Numerical Id of the IoT device or ETL data source.
d_2	*serviceUri*	URI of the IoT device or city entity.
d_3	*src*	Data source kind (*IoT* or *ETL*).
d_4	*kind*	Kind of IoT (*sensor* or *actuator*)
d_5	*deviceName*	Name of the IoT device or ETL.
d_6	*healthiness_ criteria*	It shows if the device is "healthy", according to specific rules (based, for instance, on value refresh rate) checked by dedicated scripts in the back-office.
d_7	*sensorID*	Unique identifier of the IoT device or ETL data source.
d_8	*date_time*	Date and time related to the instant in which the measure/data has been provided by the IoT device or ETL.
d_9	*latlon*	Pair of latitude and longitude coordinates.
d_{10}	*geolocation*	Special geolocation format for geographical faceting functionalities.

Key	Id	Description
d_{11}	*data_type*	The type of data representing the measure provided by the IoT device or ETL (*integer, float, string. etc.*).
d_{12}	*value_refresh_rate*	Frequency to which data or measure is provided (used also for checking the above-described *healthiness_criteria*)
d_{13}	*value_type*	Type of the measure provided by the IoT device or ETL (*temperature, humidity, speed* etc.)
d_{14}	*value*	Actual value of the provided measure.
d_{15}	*value_name*	Name of the provided measure (*MyRoomTemperature, AirHumidity* etc.).
d_{16}	*Organization*	The identification of the organization for which the data is collected

According to the above description and requirements, the indexes for Data Shadow could be implemented by using several different technologies. On the other hand, restricting the search to Open Source solutions for BigData and high performance, we focussed the comparison to: SolrCloud and Elasticsearch. In this paper, we are reporting the comparison between these two approaches and thus their peculiarities are described in the following subsections.

A. Solution 1: Solr Cloud (description and architecture)

Apache Solr, [17], is an open source indexing and search engine that permits rapid searching based on the java library called Apache Lucene [18]. The base unit in the system is the Solr document, consisting of fields and field types, which are defined in the Schema file. Solr provides REST APIs to allow client applications interact with the system for indexing and searching. The main features for searching are Full-text Search, Highlighting, Facet Search, Query Suggestion and Geospatial Search. In particular, Faceted Search is a technique that allows exploring data applying dynamic filters in multiple steps and dividing the results in Facets, which have one or more values, called the facet values, used as filter for refining search query, interactively. A facet counter is associated to each *facet* value, representing the number of records matching with this value [19]. In addition, Pivot Facets can also be considered as decision trees because they are used for multifaceted hierarchical or tree searches. These searches allow performing multifaceted searches on a list of comma-delimited fields, where each field is recursively faceted from the field listed before it in the list.

Solr is configured in SolrCloud mode, a distributed architecture consisting of a cluster of 4 servers/Virtual Machine on cloud (see Fig. 2). It provides a centralized configuration scheme, automated failover and recovery, as well as highly scalable, reliable and fault tolerant capabilities. The automated failover is implemented through the integration of a cluster of Apache Zookepeer as a distributed coordination service, responsible for monitoring and maintaining the status of the cluster nodes. In the SolrCloud architecture, the Zookepeer is composed by 3 of the SolrCloud cluster nodes. SolrCloud has a flexible distributed indexing and search, without a main node to allocate nodes, shards and replicas, instead using the Zookeeper service. In order to make the system fault-tolerant, the index (called "collection") in Solr is divided into shards, which are logical sections of a collection (made up of one or more replicas) organized in the SolrCloud physical cluster. In this case, (see Fig. 2), a distributed collection is created and partitioned on 4 *shards* (number of servers/VM) with 4 replicas for each shard.

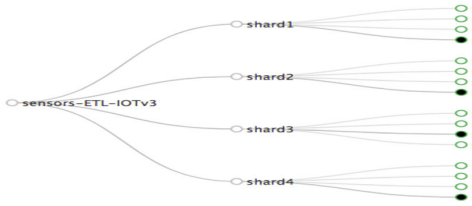

Fig. 2. SolrCloud Architecture.

B. Solution 2: Elasticsearch Cluster

Elasticsearch [20] is an open source, distributed, highly scalable search engine which is based on Lucene java library for indexing and index lookup [21]. It is a document oriented index in which the document basic unit of information is represented by an object in JSON format, with a collection of fields. However, there is not a configuration file corresponding to the Solr Schema file; actually, the *Mapping* process [22] is used to define how the documents fields are indexed before adding the first document to the index. Once defined for a certain index, the mapping cannot be changed. Elasticsearch has a RESTful API interface which provides endpoints to perform create, retrieve, update and delete (CRUD) operations on stored data and endpoints to configure the cluster using HTTP requests [23]. The searching functionalities of Elasticsearch are similar to the ones present in Solr: Highlighting, Query Suggestion and Spell-checking. It inherits these from Lucene. However, the Faceted Search functionality is replaced and improved with the Aggregations Framework, which provide aggregated data based on a search query [24]. In addition, it is possible to exploit Nested Aggregations, which allow aggregating a nested document and creating aggregation hierarchies, grouping documents based on one or more search criteria. Elasticsearch uses its own query language called Query DSL (Domain Specific Language) [25], which uses JSON objects.

Elasticsearch can be installed and configured on a cluster made up of a number of machines equal/similar to SolrCloud, in order to consider a completely equivalent architecture, as in the experiment we performed, as described in the following. Differently from SOLR, Elasticsearch cluster is based on the Master-Slave paradigm and for example may consist of 4 servers (nodes): 1 Master Node and 3 Slave nodes called Data Nodes (see Fig. 3). Data Nodes are used for handling queries and indexing, while the Master node contains all the data structures and plugins necessary to perform analysis and maintenance through a user interface.

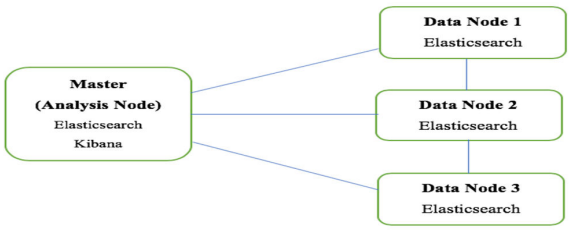

Fig. 3. Elasticsearch Cluster architecture.

Elasticsearch does not use a Zookeeper cluster, like SolrCloud, since a default built-in module called Zen

Discovery allows the automatic detection of nodes, Master Node detection and fault detection (bidirectional). As in Solr, a single index is created and split into *shards* which reside on different nodes. The *shards* are classified in two types: primary and replica. The primary shard is the place where the document is stored when it is indexed, while replica shards are just copies of the primary shard. For example, the index can be distributed over 5 primary shards and 2 replica shards for each primary shard, for a total of 15 shards distributed over 3 Data Nodes. A wide *sharding* of the index decreases search times by performing parallel searches across multiple Lucene instances (shard), as well as making the system more fault tolerant and scalable with replication shards.

IV. COMPARISON BETWEEN THE TWO SOLUTIONS

In order to compare the two solutions. A cluster of Virtual Machine has been reserved and used in both scenarios. Each node in the cluster had: *CPU:* Intel® Xeon® Gold 5118, 16 CPUs, 4 cores per CPU, 2.3 GHz, 1 thread per core; *RAM:* 24 GB; *HDD:* 500 GB 15krps; *Operating System:* Ubuntu 16.04.04 LTS. The reference data set has been based of 30 million documents for a total size of 7 Gbyte of data.

The performance analysis has been performed recreating a realistic workload by executing a set of queries in at the same time for searching data into the two indexing architectures: Elasticsearch Cluster and SolrCloud. The test has been focussed on evaluating performance, reliability and efficiency. Thus estimating for both the solutions

- response time at request in search, since indexing time is less relevant in Smart City where the indexing is performed once and search many times;
- consumption of resources in terms of CPU, memory, network used by each VM of the cluster.

The testbed has been implemented with a script capable of performing 15 queries (Requests) to the solutions. The queries for the workload have been performed through REST APIs to both the Elasticsearch and SolrCloud Clusters, in separate sections of the whole test. The 15 queries have been designed to stress the relevant aspects of the two indexing architectures according to the requirements described above, and in particular:

- executing Faceted Search and Facets Pivot on SolrCloud, and Aggregations and Nested Aggregations on Elasticsearch;
- requesting sorted results for one or more fields, of unified model of TABLE I. ;
- perform textual searches on textual fields of the unified model of TABLE I. (e.g., the equivalent of the LIKE operator used in the SQL language);
- perform temporal filtering on data.

In order to replicate a realistic workload, the processes with 15 queries have been executed several times and in parallel by a number of processes, obtaining a workload varying from 150, 750, 1500, 2250, 4500 and 7500 simultaneous queries. The scripts have been executed on external VMs, with respect to the two indexing clusters. The Master Node is the endpoint to query the Elasticsearch Cluster, while the node constituting the Shard number 4 has been chosen as the queries endpoint for SolrCloud, and in the case of time out we shifted to the next.

TABLE II. PERFORMANCE TEST RESULTS OF RESPONSE TIMES FOR SOLRCLOUD AND ELASTICSEARCH CLUSTER

tests	Number of Parallel Requests	Elasticsearch		SolrCloud	
		Average Response Time (s)	*Variance*	*Average Response Time (s)*	*Variance*
Test 1	**150**	23,216	0,848	48,669	0,532
Test 2	**750**	75,097	18,031	407,444	3768,326
Test 3	**1500**	147,463	153,106	318,046	2633,315
Test 4	**2250**	484,289	452,823	601,223	10434,492
Test 5	**4500**	1060,429	2363,393	981,419	10407,077
Test 6	**7500**	1775,962	281418,250	Cluster Breakdown	

The results of the tests, in terms of response times of Elasticsearch Cluster and SolrCloud, are shown in TABLE II. From Table II, it is possible to observe that the average response time of the two architectures has a linear trend, up to about 4000 Requests in parallel, even if the Elasticsearch is better ranked. From 4000 to 4500 requests this trend is reversed due to the failure of a Data Node of Elasticsearch Cluster that causes a deterioration of the performance, even if Elasticsearch Cluster does not stop working and continues to respond to the incoming ***Requests***, since the Master Node remains active (in addition to the two Data Nodes). For 4500 requests or more, the comparative analysis was not possible since SolrCloud crashes (Break Point) due to the failure of the query endpoint shard (Shard 4), on which the test queries are executed, hence interrupting the whole service, neither the others were capable to provide answer to queries. And thus, a long recovery time of several minutes after the stopping of the workload was necessary for the failing Shard to be restored.

The **assessment of resource consumption** have been carried out during all the above mentioned test for estimating the performance in searching. The assessment have been performed monitoring: “CPU Usage in MHz”, “CPU Usage (percent)”; “Memory Active”, “Memory Consumed” and “Memory Granted”; “Network Data Receive Rate”, “Network Data Transmit Rate” and “Network Usage”. These metrics have been collected on ESX/ESXi servers and vCenter Servers of the Cloud VMWare vSphere [26]. In this paper, only the most significant metrics are reported.

In Figures 4 and 5, the typical trends of CPU percentage are reported during the full testing execution for the above reported Test 2, Test 3, Test 4 and Test 5, of Table II, respectively for SolrCloud and Elasticsearch. From the analysis of **CPU resource consumption**, it is obvious that CPU usage is higher in the moments of testing. Regarding SolrCloud, the virtual machine that most uses the CPU, compared to other shards, is the Shard 4 (that represents the query endpoint for testing) with peaks up to 60% (see Fig. 4). On the contrary, for Elasticsearch Cluster, the CPU usage for all the Data Nodes is comparable (at most 50%), while the CPU usage of Master Node is quite low with respect to those of the nodes (see Fig. 5). In fact, the Master Node has no indexing data inside, and it is only responsible for lightweight cluster-wide actions such as creating or deleting an index, tracking which nodes are part of the cluster, and deciding which shards have to be allocated to which nodes [27].

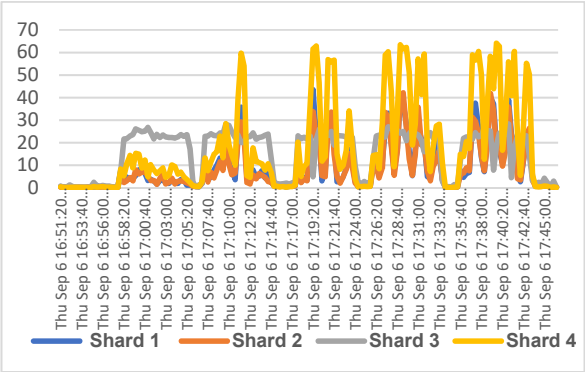

Fig. 4. Usage CPU (%) related to the SolrCloud architecture

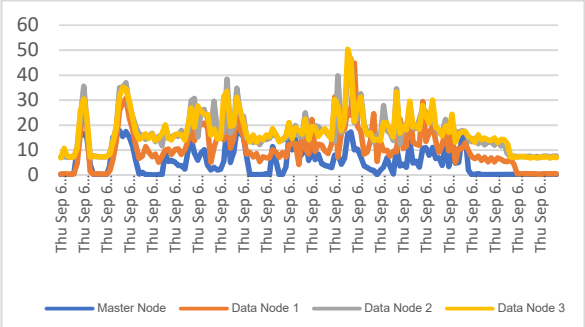

Fig. 5. Usage CPU (%) related to the Elasticsearch Cluster architecture.

In Figures 6 and 7, the typical trends of **memory consumption** in case of stress are reported during the execution Test 1 and Test 5 or 6 in sequence of Table II, respectively for SolrCloud and Elasticsearch (in fact Test 6 cannot be executed on SOLR since crashed systematically for the effort). The "Active Memory" values collected represent the amount of memory used during the tests phases.

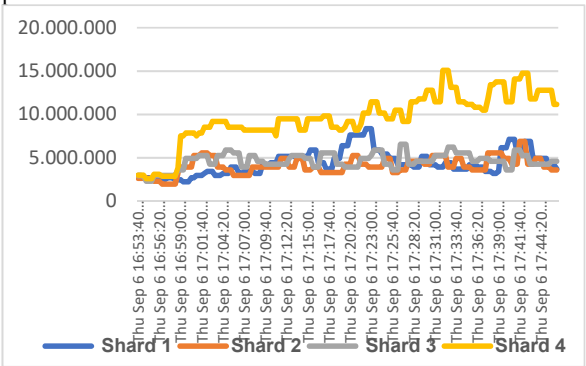

Fig. 6. Active Memory in KB related to SolrCloud architecture.

Please note that, from Figure 6, SolrCloud, the node Shard 4 (query endpoint) uses more memory than the other shards.
In Figure 7, the trend of Active Memory for the case of Elasticsearch, in which the trend of memory usage of each server of Elasticsearch Cluster (both Data Nodes and the Master Node) is comparable. In this case, the time trend shows how the use of CPU decreases to zero for Node 1 due to its failure. However, the architecture does not fail because the Master Node remains alive, allowing the Elasticsearch architecture to guarantee the service. The recovery of the failed node is automatic and guaranteeing the service despite the high workload, provided to wait for a certain recovery time. This is also visible with the increment of memory for Data Node 2. On the contrary, for SolrCloud the architecture stopped responding to queries when on Test 6 shard 4 crashed for a not affordable workload, and the service recovered only after several minutes of absence of workload.

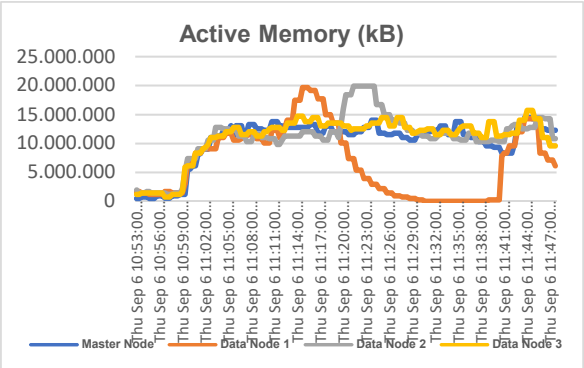

Fig. 7. Active Memory related to the Elasticsearch Cluster architecture.

V. Dashboards Banana and Kibana

Dashboards for visual analytic on the basis of indexes produced by Elasticsearch and SOLR, can be built using Kibana [28], [23], for Elasticsearch, and Banana [29] (which is a fork of Kibana) for Solr. Kibana and Banana are open source tools for data visualization and provide web-based interface for view, search and analyse data indexed in Elasticsearch Cluster and SolrCloud, respectively.

Both tools include a set of classical dashboard widgets to visualize data, such as: histograms, filters, time-pickers, facet selection on the different kind of data managed, heatmaps, pie charts, tables, and also customized panels created from scratch, such as a the Smart City linked map, which adds geo-faceting graphical filtering capabilities and enrich displayed data with Smart City related information (descriptions and additional metadata, time trends of indexed data etc.) by integrating the Km4City Smart City API interface [12]. Banana uses facet search functionalities of Solr, providing the real time analytic processing layer which is required for producing dynamic visualizations and different views. Instead Kibana uses aggregations functionalities of Elasticsearch.

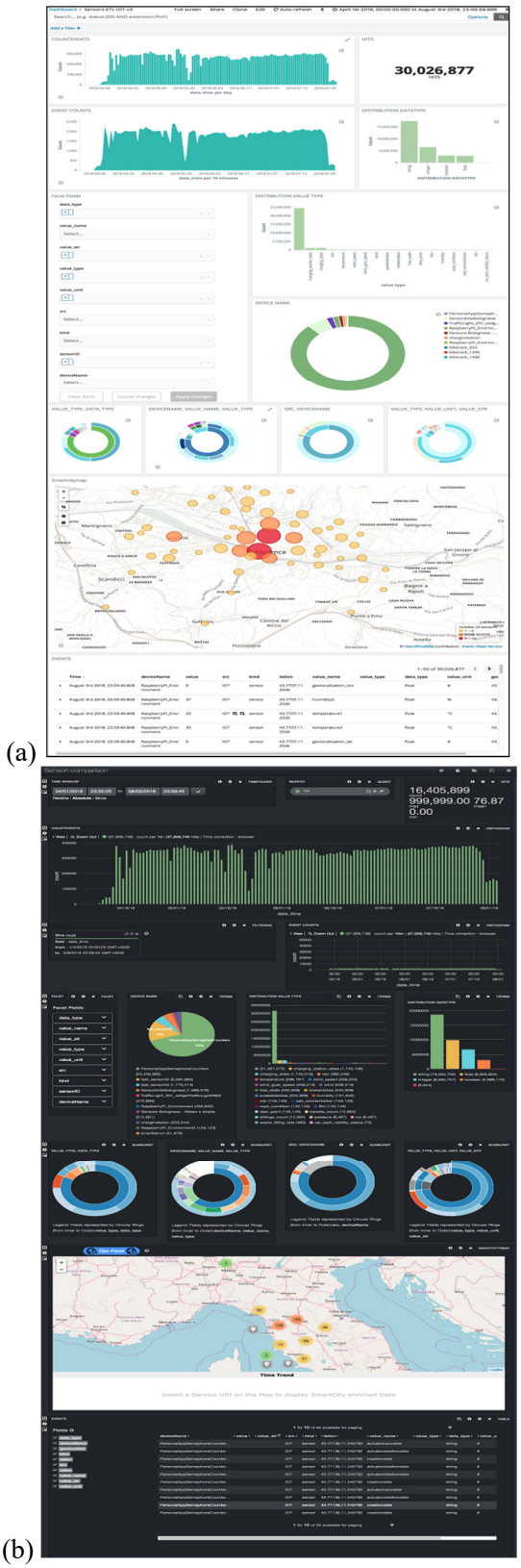

Fig. 8. (a) Dashboards Kibana and (b) Banana.

Banana and Kibana have been used for building two visual analytic dashboards with almost the same panels for data visualization (see Fig. 8a and b). A comparison has been made between the two dashboards measuring the response time, numbers of (http) requests and transferred bytes, using Developer Tools of Google Chrome [30]. Response time is the time which is needed to load the two dashboards (Banana and Kibana) after an action carried out on any of the dashboard's widgets. The execution time is particularly relevant when the facet/cluster actions are performed since the service has to apply specific filters on the millions of data and change all views into the panel according to the filter. For example, restricting the time frame of analysis, selecting only specific flow of data.

The assessment performed on visual analytics dashboards are reported in TABLE I. , and included:

- **Dashboard's loading**. First opening of the Visual Analytic tool on browser with the recent data view.
- **QueryA**: a drill down operation is performed, which selects a time frame of interest from the histogram panel (time filtering).
- **QueryB**: a temporal filtered search is performed on a histogram panel, and a Facet Fields filtering is applied on: device name (field "*deviceName*"), data type returned by the sensor (field "*value_type*") and unit of measure of the returned value (field "*value_unit*").
- **QueryC**: a specific sensor is searched by geo-spatial filtering from the map panel and a temporal range of interest is selected.
- **QueryD**: any value of a field from the search bar is searched, in this case the specific name of a device (field "*deviceName*").

The results are represented on TABLE III. From the resulting data, it is evident that Kibana-ElasticSearch is better ranked in complex visual analytic requests, while when specific value is recovered (such as QueryC) the performances are comparable.

TABLE III. AERAGE PERFORMANCE TEST OF DASHBOARD BUILT WITH BANANA-SOLRCLOUD AND KIBANA-ELASTICSEARCH

Banana-SOLRCloud	Number of Requests	Transferred Bytes (KB)	AVG Response Time (s)
Dash Loading	186	5900	28,55
QueryA	26	846	11,09
QueryB	76	1900	96
QueryC	175	3300	68,5
QueryD	28	776	14
Kibana-ElasticSercch	**Number of Requests**	**Transferred Bytes (KB)**	**AVG Response Time (s)**
Dash Loading	57	3500	6,93
QueryA	14	25,4	3,8
QueryB	10	52,2	48
QueryC	199	3300	35
QueryD	5	19,4	0,45

VI. Conclusions

In the present paper, two distributed and highly scalable architectures for data indexing and visual analytics for Smart City platforms have been compared: Solr-Banana and Elasticsearch-Kibana. Both search engines are based on the use of the Apache Lucene library. For Solr, the management of the index in terms of memory allocation is optimized thanks to the SolrCloud system, which uses the Zookeeper service to manage the cluster nodes. In addition, a useful feature that distinguishes Solr is Faceted Search that allows drilling down on the data. Elasticsearch is designed to be distributed and scalable, thanks to an easy cluster configuration. An important feature is represented by the Aggregations functionality, which aims at providing an evolution with respect to Faceted Search. Both search engines provide an intuitive graphical interface (exploiting the Banana and Kibana tools for building visual analytic dashboards) for viewing data, performing temporal drill down, multiple filtering, etc. This paper presented a performance evaluation, carried out between a SolrCloud and an Elasticsearch Cluster. By analysing the results of these performance tests, it is found that Elasticsearch Cluster present better performance to perform searches when it is massively queried. Elasticsearch is also more robust to fault than SolrCloud; actually, when one or more Data Nodes of the cluster fail, Elasticsearch can guarantee the service even if at the expense of performances. While for the distributed SolrCloud architecture, if one or more Shards fail, in most cases, it leads to the interruption of the search service and a long time for node recovery. In addition, a comparison has been made about the performances between visual analytics dashboards built with Banana for Solr and Kibana for Elasticsearch. It has been highlighted how the Kibana proved to be faster in loading and displaying the results of a search query, as well as when performing a drill down operation, if compared to the Banana.

Acknowledgment

The authors would like to thank the Select4Cities EC project, the European Union's Horizon 2020 research and innovation program with grant agreement No 688196, and also all the companies and partners involved. Snap4City is an open technology of DISIT Lab, as Km4City.

References

[1] CKAN: http://ckan.org

[2] OpenDataSoft: https://www.opendatasoft.com

[3] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, Analysis and assessment of a knowledge based smart city architecture providing service APIs, Future Generation Computer Systems, Volume 75, 2017.

[4] CitySDK: http://www.citysdk.eu

[5] 5 Stars Open Data from Tim Barneers Lee. http://www.slideshare.net/TheODINC/tim-bernerslees-5star-open-data-scheme

[6] Y. Zhao, H.Zhang, L. An, Q. Liub. Improving the approaches of traffic demand forecasting in the big data era. Cities, Volume 82, December 2018, Pages 19-26.

[7] P. Bellini, P. Nesi, M. Paolucci, I. Zaza. Smart city architecture for data ingestion and analytics: Processes and solutions, Proceedings - IEEE 4th International Conference on Big Data Computing Service and Applications, BigDataService 2018.

[8] S.Yang. IoT Stream Processing and Analytics in The Fog. 2017 in IEEE Communications Magazine. DOI: 10.1109/MCOM.2017.1600840

[9] C. Badii, E. G. Belay, P. Bellini, D. Cenni, M. Marazzini, M. Mesiti, P. Nesi, G. Pantaleo, M. Paolucci, S. Valtolina, M. Soderi, I. Zaza, "Snap4City: Smart City IOT/IOE Platform", Int. Conf. IEEE Smart City Innovation, Cina 2018, IEEE Press.

[10] Gyrard, Amelie, Antoine Zimmermann, and Amit Sheth. "Building IoT based applications for Smart Cities: How can ontology catalogs help?." IEEE Internet of Things Journal (2018).

[11] Fox, Mark S. "The role of ontologies in publishing and analyzing city indicators." Computers, Environment and Urban Systems 54 (2015): 266-279.

[12] P. Nesi, G. Pantaleo, M. Paolucci, I. Zaza, "Auditing and Assessement of data traffic flows in an IoT Architecture", IEEE TeC4C'18, 1st International Workshop on Technology Convergence for Smart Cities, Philadelphia, PA, USA. IEEE 4 th International Conference on Collaboration and Internet Computing (CIC), pp. 388-391, 2018.

[13] C. Badii, P.Bellini, D.Cenni, A. Difino, P. Nesi, M. Paolucci, "User Engagement Engine for Smart City Strategies", IEEE International Conference on Smart Computing, IEEE SMARCOMP 2017, Hong Kong.

[14] Bellini, Pierfrancesco, et al. "Km4City ontology building vs data harvesting and cleaning for smart-city services." Journal of Visual Languages & Computing 25.6 (2014): 827-839. 2017, http://dx.doi.org/10.1016/j.future.2017.05.001

[15] P. Bellini, D. Cenni, M. Marazzini, N. Mitolo, P. Nesi, M. Paolucci, "Smart City Control Room Dashboards Exploiting Big Data Infrastructure", The 24th International DMS Conference on Visualization and Visual Languages, DMSVIVA 2018, Hotel Pullman, Redwood City, San Francisco Bay, California, USA, June 29 - 30, 2018.

[16] NIFI, https://nifi.apache.org

[17] Apache Solr, http://lucene.apache.org/solr/

[18] Apache Lucene, http://lucene.apache.org/

[19] G. Simonini, S. Zhu, "Big data exploration with faceted browsing", 2015 International Conference on Information Management (ICIM), 2015, pp. 541-544

[20] Elastic, https://www.elastic.co/

[21] X. Li and Y. Wang, "Design and Implementation of an Indexing Method Based on Fields for Elasticsearch", 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), 2015

[22] Elasticsearch 6.3.0 – Mapping [online], 2018 https://www.elastic.co/guide/en/elasticsearch/reference/6.3/mapping.html

[23] M. Bajer "Building an IoT Dataa Hub with Elasticsearch, Logstash and Kibana", 2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), 2017, pp. 63-68

[24] Elasticsearch 6.3.0 – Aggregations [online], 2018, https://www.elastic.co/guide/en/elasticsearch/reference/6.3/search-aggregations.html

[25] Elasticsearch 6.3.0 - Query DSL [online], 2018 https://www.elastic.co/guide/en/elasticsearch/reference/6.3/query-dsl.html

[26] Cloud VMware vSphere [online], 2018 https://pubs.vmware.com/vsphere-4-esx-vcenter/index.jsp?topic=/com.vmware.vsphere.bsa.doc_40/vc_admin_guide/performance_metrics/r_network_counters.html

[27] Elasticsearch 6.3.0 – Modules-Node [online], 2018 https://www.elastic.co/guide/en/elasticsearch/reference/6.3/modules-node.html#master-node

[28] Kibana, https://www.elastic.co/products/kibana

[29] Banana Lucid work, https://github.com/lucidworks/banana

[30] Chrome DevTools [online], 2018 https://developers.google.com/web/tools/chrome-devtools/