

Title:	Document Version:
D5.1 WiseGRID cloud-based big data infrastructure	1.0

Project Number:	Project Acronym:	Project Title:
H2020-731205	WiseGRID	Wide scale demonstration of Integrated Solutions for European SmartGrid

Contractual Delivery Date:	Actual Delivery Date:	Deliverable Type*-Security*:
M18 (April 2018)	M18 (April 2018)	R-PU

*Type: P: Prototype; R: Report; D: Demonstrator; O: Other.

**Security Class: PU: Public; PP: Restricted to other programme participants (including the Commission); RE: Restricted to a group defined by the consortium (including the Commission); CO: Confidential, only for members of the consortium (including the Commission).

Responsible:	Organisation:	Contributing WP:
Paul Lacatus	CRE	WP5

Authors (organisation):
Paul Lacatus (CRE), Catalin Chimirel (CRE), Mihai Sanduleac(CRE), Álvaro Nofuentes (ETRA), Alberto Zambrano(ETRA), Giuseppa Caruso (ENG), Leandro Lombardo(ENG), Stefan Meir(VS)

Abstract:

In the context of WiseGRID WP5 this document will define the architecture of a cloud based Big Data platform. Specifications of the WiseGRID Big Data Cloud-based infrastructure towards enabling: (i) Data Integration from a variety of heterogeneous data sources, (ii) Storage and Processing of huge data volumes in a highly efficient and effective manner, (iii) adaptation of Service-Oriented-Architectures (SOA) towards enhancing flexibility, scalability, re-usability, loose coupling, integration of a variety of functionalities, and (iv) Interoperability and Interconnection with a wide range of applications. Configuration and integration of the WiseGRID Big Data infrastructure with the WG IOP and the individual WiseGRID components.

Keywords:

Big Data, Big Data platforms, data mining, data analytics, JSON, NoSQL, data storing, data processing,

Revision History

Revision	Date	Description	Author (Organisation)
V0.1	28.01.2018	New document	Paul Lacatus (CRE)
V0.2	11.03.2018	Document development	Paul Lacatus (CRE)
V0.3	11.03.2018	Document development	Paul Lacatus (CRE)
V0.4	25.03.2018	Document development	Paul Lacatus (CRE)
V0.5	25.03.2018	Integration of Leandro Lombardo (ENG) contribution.	Paul Lacatus (CRE)
V0.6	29.03.2018	Integration of Alberto Zambrano (ETRA) contribution.	Paul Lacatus (CRE)
V0.7	07.04.2018	Corrections, Changing Citation type to IEEE	Paul Lacatus (CRE)
V0.8	14.04.2018	Corrections, adding Executive summary	Paul Lacatus(CRE)
V1.0	24.04.2018	Including peer review , including Stefan Meir (VS) contribution	Paul Lacatus(CRE)

INDEX

EXECUTIVE SUMMARY	7
1 INTRODUCTION	12
1.1 Purpose of the document	12
1.2 Scope of the document.....	12
1.3 Structure of the document	12
2 BIG DATA CONCEPTS.....	13
2.1 Big data definitions	13
2.2 Big Data platforms	16
3 NoSQL CLOUD BASED DATA MANAGERMENTS.....	17
3.1 NoSQL versus RDBMS in Big data applications.....	17
3.2 MongoDB NoSQL database for WiseGRID architecture	17
3.2.1 MongoDB advantages.....	18
3.2.2 MongoDB drivers.....	19
3.3 Understanding MongoDB	21
3.3.1 MongoDB Structure.....	21
3.3.2 MongoDB basics overview.....	21
3.3.3 Datatypes used by MongoDB	23
3.3.4 MongoDB Methods	24
4 ARCHITECTURE OF BIG DATA CLOUD BASED INFRASTRUCTURE	28
4.1 Specific requirements in WiseGRID Project	29
4.1.1 MongoDB replication.....	29
4.1.2 MongoDB sharding	30
4.1.3 MongoDB cluster definition	30
4.2 Architecture adapted for WiseGRID project	30
4.2.1 General concepts.....	30
4.2.2 Computer cluster	32
4.2.3 Managing computer clusters.....	33

4.2.4	MongoDB computer cluster	38
4.2.5	Setting a MongoDB computer cluster	39
4.2.6	Administration of a MongoDB computer cluster	43
4.2.7	Horizontally scaling a MongoDB computer cluster	45
5	PRIVACY AND DATA PROTECTION IN WISEGRID BIG DATA INFRASTRUCTURE	47
5.1	Big data cloud based architecture location of sensitive data	47
6	Interface with IOP and applications.....	48
6.1	Application requirements from WiseGRID applications to the Big Data Platform.....	48
6.1.1	Big data requirements from WG IOP application	48
6.1.2	Big data requirements from WG Cockpit application	49
6.1.3	Big data requirements from WG CORP application.....	53
6.1.4	Big data requirements from WG COOP application	55
6.1.5	Big data requirements from WiseHOME application	57
6.1.6	Big data requirements from WG EVP application	59
6.1.7	Big data requirements from WG FastV2G application	61
6.1.8	Big data requirements from WG Staas/VPP application	61
6.1.9	Big data requirements from WG RESCO application (ENG)	64
7	BIG DATA PROCESSING MODULES	67
7.1	Data mining concept	67
7.2	Data analytics concept	67
7.3	Data mining and analytics platforms	67
7.3.1	Apache™ Hadoop® Project	68
7.3.2	Hadoop compatibility with MongoDB.....	69
7.3.3	Hadoop Cluster setup	70
7.3.4	Apache Spark™ engine for large scale data processing.....	71
7.4	Architecture for WiseGRID offline Big Data processing	71
8	CONCLUSIONS AND NEXT STEPS.....	72
8.1	Conclusions.....	72
8.2	Next steps to be implemented.....	72
9	REFERENCES AND ACRONYMS.....	73
9.1	References	73
9.2	Acronyms.....	74

LIST OF FIGURES

Figure 1 – Big Data [5]	13
Figure 2 – Big Data Volume [5]	14
Figure 3 – Big Data Velocity and Variety [5]	15
Figure 4 – MongoDB the leading NoSQL Database	18
Figure 5 – Replication and Sharding	19
Figure 6 – MongoDB drivers	20
Figure 7 – MongoDB partners	20
Figure 8 – MongoDB database structure vs RDBMS	21
Figure 9 – MongoDB terminology vs RDBMS	21
Figure 10 – MongoDB Data Model	23
Figure 11 – Replication Diagram.....	29
Figure 12 – Sharding Diagram	30
Figure 13 – WiseGRID Big Data Architecture overview.....	31
Figure 14 – RedHat Cluster structure example	33
Figure 15 – Atlas Peloton cluster [11]	34
Figure 16 – Webmin dashboard	35
Figure 17 – Webmin, scanning for instances.....	36
Figure 18 – Webmin Registering found servers	37
Figure 19 – Webmin cluster operation.....	38
Figure 20 – Data distribution over multiple shards.....	39
Figure 21– MongoDB Compass screenshot.....	45
Figure 22 – WG IOP Architecture	49
Figure 23 - WiseGRID Cockpit.....	50
Figure 24 – WG Cockpit interface with Big Data platform	51
Figure 25 – WG Cockpit data mining	51
Figure 26 – WiseCORP	53
Figure 27 – WG CORP interface with Big Data platform	54
Figure 28 – WiseCORP data mining.....	55
Figure 29 – WiseCOOP.....	56
Figure 30 – WG WiseCOOP Big Data Platform interface	56
Figure 31 – WG WiseCOOP Big Data mining	57
Figure 32 – WiseHOME Interaction Diagram	58
Figure 33 – WG Wise EVP	59
Figure 34 – WiseEVP interface with Big Data platform	60
Figure 35 – WiseEVP Big Data mining.....	61

Figure 36 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools. 62

Figure 37 – WG RESCO.....64

Figure 38 – Object Mapping between Node and MongoDB managed via Mongoose.....66

Figure 39 – WG RESCO Tool interface with Big Data Platform.....66

Figure 40 – Apache Hadoop Logo68

Figure 41 – Hadoop Cluster structure70

Figure 42 – Apache Spark Logo.....71

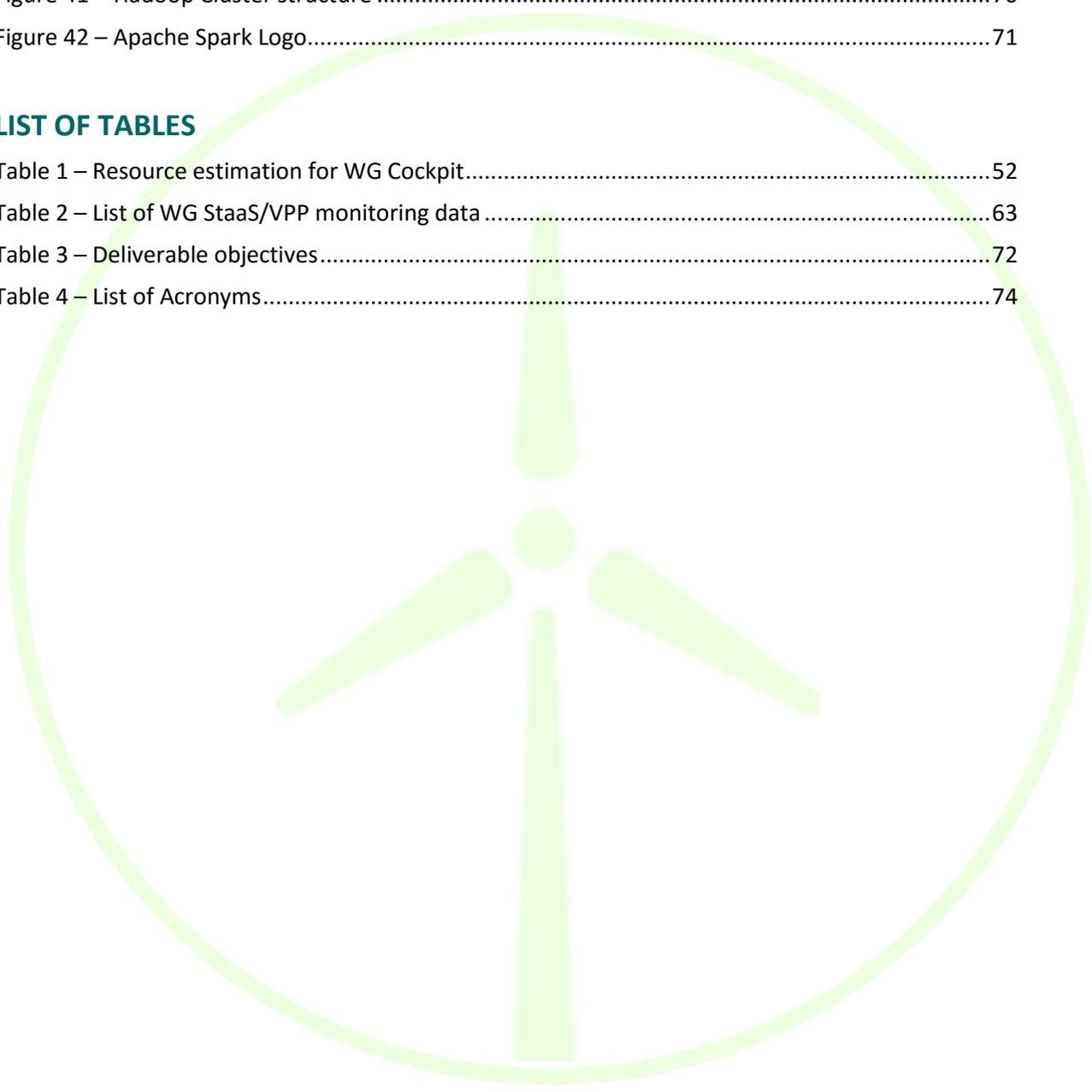
LIST OF TABLES

Table 1 – Resource estimation for WG Cockpit.....52

Table 2 – List of WG StaaS/VPP monitoring data63

Table 3 – Deliverable objectives.....72

Table 4 – List of Acronyms.....74



EXECUTIVE SUMMARY

The executive summary shortly summarizes the contents of the main document chapter by chapter.

Chapter 1. Introduction describes the purpose of the document, the scope of the document and the structure. The main purpose of the document is to elaborate the definition of the architecture for a Big Data cloud-based platform to suit the requirements of the WiseGRID applications.

Chapter 2. Big Data concepts defines the general concepts that are used for designing Big Data platforms. The big data platforms are providing to the users two types of services: Online services for storing, retrieving and indexing the data, and Offline services for getting insights of the data stored like data mining and data analytics. All the services of Big Data platforms should comply with the characteristics defined by Doug Laney in early 2001 in the document "3V Data management: Controlling Data Volume, Velocity and Variety". [1]. Later the number of 'V's increases to 5 considering also "Veracity" and "Value" between Big Data characteristics.

Chapter 3. NoSQL cloud-based data managements details the use of NoSQL (that should be read as Not Only SQL) in Big Data platforms. NoSQL databases management are complementary to the relational database management systems and are preferred in Big Data platforms due to following criteria:

- For operational Big Data workloads, NoSQL Big Data systems such as document databases [2] have emerged to address a broad set of applications. Other architectures, such as key-value stores, column family stores and graph databases are optimized for more specific applications. NoSQL technologies, which were developed to address the shortcomings of relational databases in the modern computing environment, are faster and scale much more quickly and inexpensively [2] than relational databases.
- Critically, NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational Big Data workloads much easier to manage, and cheaper and faster to implement.
- Traditional database systems are also designed to operate on a single server, making increased capacity expensive and finite. As applications have evolved to serve large volumes of users, and as application development practices have become agile, the traditional use of the relational database has become a liability [2]

In the subchapter 3.2 **MongoDB NoSQL Database for WiseGRID Architecture**, the reasons for choosing opensource MongoDB database management system in the WiseGRID Big Data platform are detailed.

Subchapter 3.3 **UNDERSTANDING MONGODB** is analysing the main characteristics of MongoDB like structure-less document database. The same subchapter contains a basic overview, a description of datatypes used by MongoDB and a brief review of MongoDB interactive methods for storing and retrieving data with easy to understand examples.

Chapter 4. Architecture of Big Data Cloud based infrastructure is defining the requirements of the WiseGRID set of applications from a Big Data platform that will run like a cloud service accessible to all WiseGRID applications. The chapter is detailing how the two characteristics of MongoDB: replications and sharding are needed by WiseGrid Big Data platform.

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers. Replication protects a database from the loss of a single server.

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling.

With sharding, more machines can be added in order to support data growth and the demands of read and write operations.

In order to be able to sustain the requirements of the WiseGRID Big Data platform the implementation will use a computer cluster.

The Big Data platform will be implemented by a Linux computers cluster. Linux clustering is more and more popular today in many industries. The acceptance of open source software is very important today and more and more ICT application are migrating towards. Linux as an open source operating system which becomes more and more utilized due his high reliability and efficiency. The clustering technology for Linux computer is mature and allows now to create supercomputers at a fraction of the cost of traditional high performance machines. Now, by using a number of normal high accessible computers, it is possible to get the computation power that was otherwise only accessible to high performance machines

Subchapter 4.2 **ARCHITECTURE ADAPTED FOR WISEGRID PROJECT** describes the main characteristics of a computer cluster providing the characteristic requirements for the WiseGRID Big Data platform.

The techniques used for configuring a computer cluster and managing it are detailed in the same subchapter. In this document is proposed for the management of the Linux Computer cluster the use of an opensource application named Webmin that enables remote management and specific techniques used for managing cluster with multiple nodes.

The MongoDB database cluster configuration is detailed in subchapter **5.2.5 Setting a MongoDB computer cluster**. The process is mainly based on allocation of the roles of each node, computer, in the cluster. All nodes are running the MongoDB database manager and depending of the necessities they play a specific role:

- **Configuration Server** that is storing and defining the complete structure of the cluster. On the config servers node will run config server processes that are mongod instances that store the cluster's metadata. Mongod instances are mongoDB database server applications.
- **Application router** that is processing the user interaction with the database and is running the mongos process. The mongos instances are lightweight and do not require data directories since they will not participate directly to data storage and retrieval. Mongos instances can be ran on a system that runs other cluster components, such as on an application server or a server running a mongod process. By default, a mongos instance runs on port 27017. The WiseGRID applications will have each one a dedicated application router on a separate node or sharing a nodes based by the load generated by the applications.
- **Shards** That are cluster nodes containing and processing the directly the databases. A shard can be a standalone mongod instance on a cluster node or a replica set running on multiple cluster nodes. In a production environment, each shard should be a replica set in order to increase the availability of the database based on replication process.

After the role allocation into the computer cluster, the cluster is ready to provide the database processes and has to be actively managed through the production time. The administration documentation addresses the ongoing operation and maintenance of MongoDB instances and deployments. This documentation includes both high level overviews of these concerns as well as tutorials that cover specific procedures and processes for operating MongoDB.

A popular tool for MongoDB administration is MongoDB Compass developed by the same developers as MongoDB. This Document is briefly describing the use of MongoDB Compass.

A very important technique for the MongoDB computer cluster is **the horizontally scaling of a MongoDB computer cluster** that is described in **subchapter 4.2.7**.

The WiseGRID applications are now in the development phase. Right now, only an estimation of needed resources can be done. MongoDB provides the flexibility to start the production cluster with a minimal set of resources and, after deploying in production when the real resources requirements will be calculated, scale up the computer cluster in order to fulfil the application requirements. The horizontally scaling will be done by adding new shards nodes to the cluster.

New shards can be added to a sharded cluster after the creation of the cluster or any time when needed to add capacity to the cluster.

Chapter 5 Privacy and data Protection in WiseGRID Big Data Infrastructure is defining the DPIA requirements in direct connection with the special deliverable D3.2 “WiseGRID architecture, data models, standards and data protection (v2)”.

Chapter 6 Interface with IOP and applications is describing the interactions between the different applications of WiseGRID package with the Big Data platform and a first estimation of the resources needed to store the data and process them in a satisfying way.

The applications that will interact with the Big Data platform are:

- **WG IOP application:** The WiseGRID Interoperable Platform (WG IOP) is a scalable, secure and open ICT platform, with interoperable interfaces, for real-time monitoring and decentralized control to support effective operation of the energy network. The objective of the platform is to manage and process the heterogeneous and massive data streams coming from the distributed energy infrastructure deployed. As better described in the D4.2 deliverable architecture, the WG IOP does not directly access the big data platform but uses external services (DB interaction services).
- **WG Cockpit application:** WiseGRID Cockpit is the WiseGRID technological solution targeting DSOs and microgrid operators, allowing them to control, manage and monitor their own grid, improving flexibility, stability and security of their network. Taking into account the goals of the project, the features to be implemented within WiseGRID cockpit consider a scenario of increasing share of distributed renewable resources and services provided by communities of prosumers (aggregated in the form of VPPs or cooperatives in order to achieve higher participation and environmental, social and economic benefits). The big data platform will be used by the WiseGRID Cockpit in order to hold the history of data provided by the sensors of interest of the application. As depicted in the architecture of the WiseGRID Cockpit, the main data sources for this information are:
 - Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds)
 - Advanced Metering Infrastructure systems, retrieving data from already deployed Smart Meters – usually hourly curves retrieved once a day
 - SCADA systems, retrieving real-time electric measurements from monitored infrastructure under control of the DSO (buses of the distribution grid) and status of safety elements

This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseGRID Cockpit application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

- **WiseCORP application:** WiseCORP is the WiseGRID technological solution targeting businesses, industries, ESCOs and public facility consumers and prosumers, with the objective of providing them the necessary mechanisms to become smarter energy players. By means of energy usage monitoring and analysis, proper information can be given to facility managers helping them to reduce energy costs and environmental impact.

The big data platform will be used by the WiseCORP in order to hold the history of data provided by the different elements of interest of the application. As depicted in the architecture of the WiseCORP, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds)

- Advanced Metering Infrastructure systems from the DSO, retrieving data from already deployed Smart Meters – usually hourly curves retrieved once a day
 - Storage systems
 - CHP devices, publishing their operation setpoint
 - HVAC devices, publishing their operation setpoint
 - Sensors for measuring indoor conditions, mainly temperature
 - This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseCORP application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.
- **WiseCOOP application:** WiseCOOP is the WiseGRID technological solution targeting aggregators of consumers and prosumers – particularly focused on domestic and small businesses -, supporting them in their roles of energy retailers, local communities and cooperatives – which may have different objectives. The main goal of the solution is helping consumers and prosumers to work together in order to achieve better energy deals while relieving them from administrative procedures and cumbersome research.
The big data platform will be used by the WiseCOOP mainly with the objective of storing the history of energy demand and production of the members of the portfolio aggregated by the organization using the application. As depicted in the architecture of the WiseCOOP, the main data sources for this information are:
 - Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds)
 - Advanced Metering Infrastructure systems from the DSO, retrieving data from already deployed Smart Meters – usually hourly trends retrieved once a day.
 - **WiseHOME application:** WG Home is an application used by the home users that are in the WiseCOOP user structure in order to get information of the status of their energy performance. WG HOME interacts with other application as Wise COOP and WG Staas/VPP trough IOP sending data requests and generation simple and comprehensive graphic interface for the user accessible by a large type of devices as computers and mobile devices. WiseHome does not interact directly to the Big Data platform and is not storing any data in the long term database from the Big Data Platform. WG Home is using a small local SQL database in order to process the user authentication data and data got from WG COOP and WG Staas/VPP.
 - **WiseEVP application :** WiseEVP is the WiseGRID technological solution for:
 - Vehicle-sharing companies or electric vehicle fleet managers
 - Electric vehicle infrastructure (EVSE) operators.

In order to optimize the activities related with smart charging and discharging of the EVs including V2G (vehicle to grid, energy injection in the distribution network) and V2B (vehicle to building, energy injection in the household electric installation). The Big Data platform will be used by the WiseEVP for storing the history of energy supplied or injected by the different EVSEs, and hold the history of the parameters read out from the vehicles of the fleet, which will be used for analysing the usage of those and optimize the charging schemes. As depicted in the architecture of the WiseEVP, the main data sources for this information are:

- Electric Vehicle Supply Equipment (EVSEs), providing readings of energy supplied (or injected under V2G schemes) and the identification of the vehicle associated to these energy flows.
- Electric Vehicles (EVs), which will be monitored to regularly extract information to model their usage (State of Charge, travelled distances, location, energy charged)

- **WG FastV2G application:** The WG FastV2G is the product from WiseGrid Project used for the control of the power flow from an EVSE capable to feed energy into the grid. The WG FastV2G does not interact directly to the Big Data Platform.
- **WG Staas/VPP application:** WG Staas/VPP (“Storage as a Service / Virtual Power Plant”) is WiseGrid’s solution to clustering distributed storage systems and making them usable for the grid in general and for DSOs in particular.
- **WG RESCO application:** The WG RESCO will be a tool conceived for RESCOs – Renewable Energy Service Companies that want to provide RES services to end-users (households or businesses) that do not own nor wish to maintain the necessary equipment. In this perspective, three potential scenarios are envisaged:

Chapter 7 Big data processing modules is described the other type of services provided to the user application by the Big Data platform, the offline services. This services are of two main categories:

- **Data mining:** Data mining is a process to structure the raw data and formulate or recognise the various patterns in the data through the mathematical and computational algorithms, data mining helps to generate new information and unlock the various insights. The data is first placed into a data warehouse to do the required the required extraction of data to produce meaningful relationships and patterns. There is two type of data mining one is descriptive, which gives information about existing data of the organisation, while the other is predictive: which makes forecasts based on the data.
- **Data Analytics:** Data analytics is the art of exploring the facts from the data with specific to answer specific questions, i.e. there is a test hypothesis framework for data analytics. The techniques used in analytics also are same as used in business analytics & business intelligence.

For Data mining and data analytics an important need of computation power should be considered. Large amount of data has to be processed so the answer for this request is also a computer cluster. The database cluster is mainly a load balancing cluster but now, for data mining and analytics the type of cluster to be used is a high performance cluster. This type of cluster is sharing the resources of the computers that are composing the cluster so the applications will run on a high performance machine with higher resources than each of the components of the cluster.

For reaching this goal special software will run on each computer in the cluster enabling the distributed processing and resources sharing. In the WiseGRID project will be used the Apache Hadoop Framework and the Apache Spark tools collection.

Chapter 8 Conclusions and next steps is concluding the main facts of the Big Data platform that will be implemented in a Cloud like platform providing database as a service. The implementation will be done in two separate computer cluster one running MongoDB providing NoSQL data base service and another one running Apache Hadoop framework in order to provide a running environment for WiseGrid applications written using Apache Spark tools for data mining or data analytics. The Apache Hadoop computer cluster will connect the MongoDB databases through a special MongoDB Hadoop connector.

1 INTRODUCTION

1.1 PURPOSE OF THE DOCUMENT

In the context of WiseGRID's WP5, this document will define the architecture of a cloud based Big Data platform.

Specifications of the WiseGRID Big Data Cloud-based infrastructure towards enabling: (i) Data Integration from a variety of heterogeneous data sources, (ii) Storage and Processing of huge data volumes in a highly efficient and effective manner, (iii) adaptation of Service-Oriented-Architectures (SOA) towards enhancing flexibility, scalability, reusability, loose coupling, integration of a variety of functionalities, and (iv) Interoperability and Interconnection with a wide range of applications. The purpose is also to define the configuration and integration of the WiseGRID Big Data infrastructure with the WG IOP and the individual WiseGRID components.

1.2 SCOPE OF THE DOCUMENT

This document will focus on the definition of the architecture for a Big Data cloud based platform to suit the requirements of the WiseGRID applications.

1.3 STRUCTURE OF THE DOCUMENT

The document will define general concepts and notions about Big Data platforms. The NoSQL database will be defined integrated in the Big Data concepts and it will be explained why the NoSQL databases are suitable to be used in Big Data Platforms in Chapter 3.

The MongoDB database was introduced, a leading open source NoSQL database that has been proposed to be used for implementing the Big Data platform for WiseGRID project in chapter 3.2

The document will introduce further the architecture of a clustered MongoDB database suitable to handle the requirements of WiseGRID Project.

The Privacy and Data protection will be analysed, evaluated and treated in connection with D3.2, deliverable in WiseGRID project in chapter 5.

The interface between the applications of WiseGRID project and the Big Data platform will be direct from the applications routers dedicated to each application in chapter 6.

The Data mining and Data analytics will be described in order to be used in WiseGRID project in appropriate applications in chapter 7

2 BIG DATA CONCEPTS

2.1 BIG DATA DEFINITIONS

Big Data is a concept that emerges early in 1990 due to the continuously increases in the amount of data produced in all human activities that has to be stored, managed and processed. Big Data is in fact your data, every data. It is information owned by you, by your company, obtained and processed through new techniques in order to produce value in the best way possible.

Big Data is a concept that echoes across all corners of the ICT business today. Now the whole world is producing, storing and processing huge amount of data:

As only an example, Facebook processes more than 500 TB of data daily [3]

In 2015 Every Day Big Data Statistics – 2.5 Quintillion Bytes of Data Created Daily [4]

In 2001 Gartner analyst Doug Laney started to define concepts and issued the: "The three V's" - "volume, velocity and variety" and anticipated in his article [1]:



Figure 1 – Big Data [5]

" META Trend: During 2001/02, leading enterprises will increasingly use a centralized data warehouse to define a common business vocabulary that improves internal and external collaboration. Through 2003/04, data quality and integration woes will be tempered by data profiling technologies (for generating

metadata, consolidated schemas, and integration logic) and information logistics agents. By 2005/06, data, document, and knowledge management will coalesce, driven by schema-agnostic indexing strategies and portal maturity.”

Volume

A typical PC might have had 10 gigabytes of storage in 2000. Today, Facebook ingests 500 terabytes of new data every day; a Boeing 747 will generate 240 terabytes of flight data during a single flight across the US; the proliferation of smart phones, the data they create and consume; sensors embedded into everyday objects will soon result in billions of new, constantly-updated data feeds containing environmental, location, and other information, including video

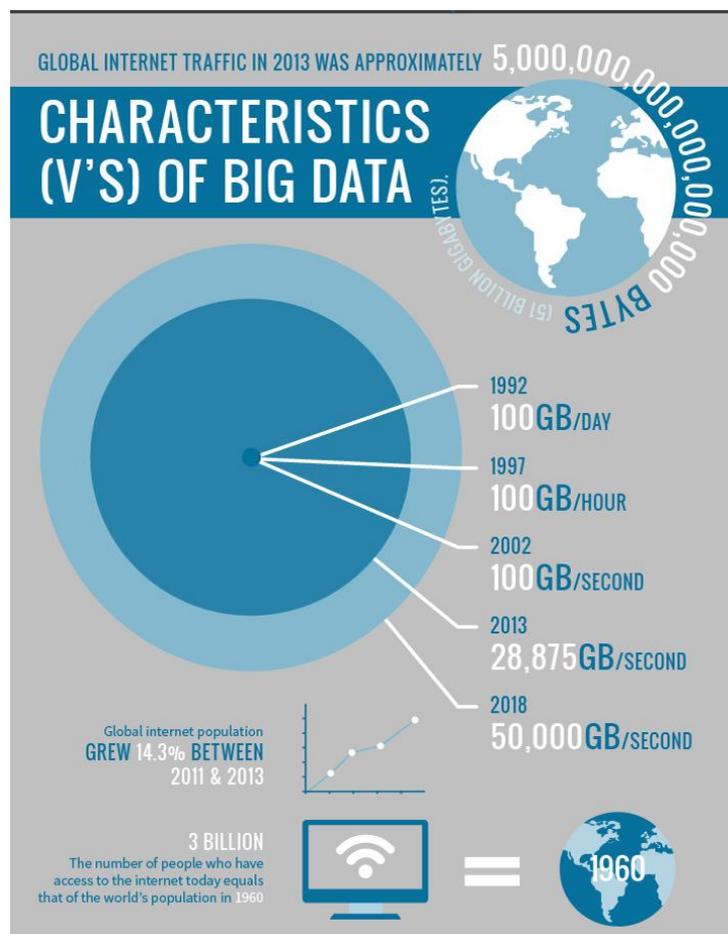


Figure 2 – Big Data Volume [5]

Velocity

Clickstreams and ad impressions capture user behavior at millions of events per second; high-frequency stock trading algorithms reflect market changes within microseconds; machine to machine processes exchange data between billions of devices; infrastructure and sensors generate massive log data in real-time; on-line gaming systems support millions of concurrent users, each producing multiple inputs per second, traffic monitoring by the online traffic applications gather the traffic over each section of road worldwide.

Variety

Big Data information isn't just numbers, dates, and strings. Big Data is also geospatial data, 3D data, audio and video, and unstructured text, including log files and social media. Traditional database systems were designed to address smaller volumes of structured data, fewer updates or a predictable, consistent data structure.

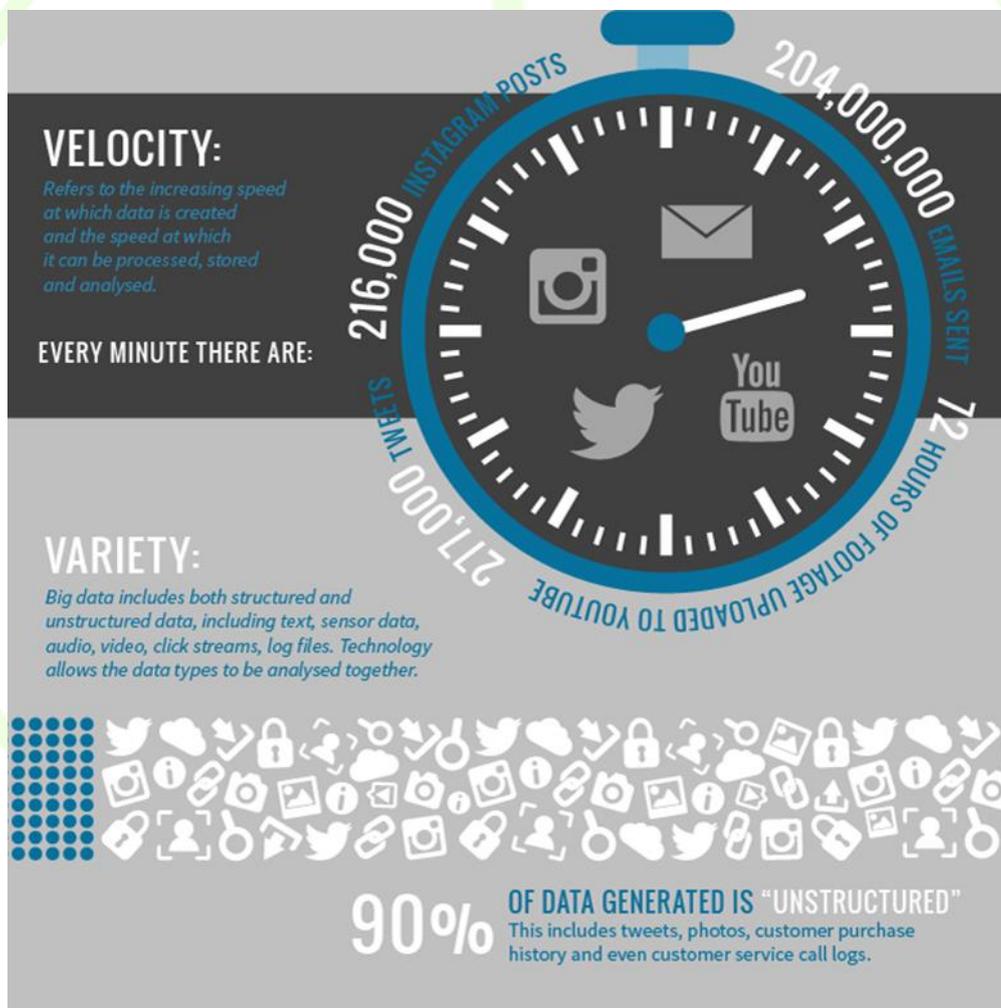


Figure 3 – Big Data Velocity and Variety [5]

Later in 2014 another two more 'V's were associated with Big Data:

Veracity

Refers to the messiness or trustworthiness of the data. With many forms of big data, quality and accuracy are less controllable (just think of Twitter posts with hash tags, abbreviations, typos and colloquial speech as well as the reliability and accuracy of content) but big data and analytics technology now allows us to work with these type of data. The volumes often make up for the lack of quality or accuracy [6].

Value

Then there is another V to take into account when looking at Big Data: Value! It is all well and well having access to big data but unless can be turned it into value it is useless. So it is possible to safely argue that 'value' is the most important V of Big Data. It is important that businesses make a business case for any attempt to collect and leverage big data. It is so easy to fall into the buzz trap and embark on big data initiatives without a clear understanding of costs and benefits [6].

2.2 BIG DATA PLATFORMS

What's unique about Big Data?

Companies are researching for decades how to make the best use of information to improve their business capabilities. However, it's the structure (or lack thereof) and size of Big Data that makes it so unique. Big Data is also special because it represents both significant information - which can open new doors, give new opportunities - and the way this information is analyzed to help open those doors. The analysis goes hand-in-hand with the information, so in this sense "Big Data" represents a noun - "the data" - and a verb - "combing the data to find value."

The days of keeping company data in Microsoft Office documents on carefully organized file shares are behind us, much like the bygone era of sailing across the ocean in small ships by looking on the start positions. One 50 gigabyte file share in 2002 was important but looks quite tiny compared to a modern-day 50 terabyte marketing database containing customer preferences and habits. How can be combed through all that material to spot trends suggesting which way consumer tastes are headed or what climate changes are occurring?

That's where the interpretive process comes in.

Big Data is not simply a huge pile of information. A good starting place is the following thought: "Big Data describes datasets so large they become very difficult to manage with traditional database tools."

Big Data Analytics Architecture

- Data is only as useful as the insights the user can get from it. Building a sophisticated analytics platform for user's data can spell the difference between outperforming and lagging behind competitors. However, this pursuit is not easy as today's applications have to handle data that is growing too fast, moving too fast, and too diverse in content to manage. Big Data has become a popular catchphrase to describe this challenge.
- To develop a solutions architecture for Big Data analytics, consider the difference between online and offline Big Data technologies.
- Online Big Data technologies, such as the database MongoDB, ingest and store data in real-time and in an operational capacity.
- Offline Big Data solutions such as Hadoop, process data in batch for retrospective analyses that may touch most or all of a company's data. These technologies are complementary and to develop a complete analytics solution, the user will likely need to employ both.

Big Data is quite complicated to be defined completely. Wikipedia is a huge encyclopaedia developed step

by step by his contributors. Gathering the data, correcting them is a continuous project that is giving most of the times very good and comprehensive information.

Wikipedia has his own definition of Big Data [7]:

Big data is data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. There are number of concepts associated with big data: originally there were 3 concepts volume, variety, velocity [1], other concepts were later attributed with Big data some of those are, veracity (i.e., how much noise is in the data), and value .

3 NoSQL CLOUD BASED DATA MANAGERMENTS

In time many type of databases were used for dealing with the data amount considered like Big Data and the most appropriate database management systems were the NoSQL database management systems.

3.1 NOSQL VERSUS RDBMS IN BIG DATA APPLICATIONS

NoSQL Means “Not Only SQL” and not “Not SQL” databases.

NoSQL is a complement to a traditional RDBMS, not necessarily as a replacement of them

Operational Big Data [8]

- For operational Big Data workloads, NoSQL Big Data systems such as document databases [9] have emerged to address a broad set of applications, and other architectures, such as key-value stores, column family stores, and graph databases are optimized for more specific applications. NoSQL technologies, which were developed to address the shortcomings of relational databases in the modern computing environment, are faster and scale much more quickly and inexpensively [2] than relational databases.
- Critically, NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational Big Data workloads much easier to manage, and cheaper and faster to implement.
- Traditional database systems are also designed to operate on a single server, making increased capacity expensive and finite. As applications have evolved to serve large volumes of users, and as application development practices have become agile, the traditional use of the relational database has become a liability.

3.2 MONGODB NOSQL DATABASE FOR WISEGRID ARCHITECTURE

For defining a database management system for WiseGRID project, the first step is to choose one from the existing ones. The first criteria was to ease as much as possible the interactions of all applications in WiseGRID ecosystem with the Big Data Platform. In the discussions had with application developers, the decision was that a Mongo DB database management system will be easily accessed by all the developing applications. The fact that MongoDB uses the JSON (JavaScript Object Notation) format for storing the documents into databases and the JSON format is very popular among the application developers in WiseGRID was an important condition that pushed MongoDB in the head of the list of Database management system considered.

3.2.1 MongoDB advantages

MongoDB is a NoSQL database system that provide users and application developers with the means to create tremendous business value from Big Data storing and processing.

Top 5 reasons to choose MongoDB:

1. **General purpose.** The MongoDB is not focused only on some type of data processing. Is a very flexible database management system able to be used in a wide variety of applications
2. **Agile.** MongoDB is a fast response database system , easy to be scaled for better performance
3. **JSON & JavaScript.** MongoDB is using the JSON format used by Javascript developers that is flexible, structure-less and easy to be used for data interchange
4. **Aggregation framework.** MongoDB provides a complete framework for data aggregation very useful in Big Data Platform
5. **Ecosystem.** MongoDB has a large ecosystem of users, developers that provides a dynamic response to the issues of Big Data platforms.
6. **OpenSource.** MongoDB is opensource that is providing a complete set of opportunities when developed and maintained.

MongoDB: The Leading NoSQL Database

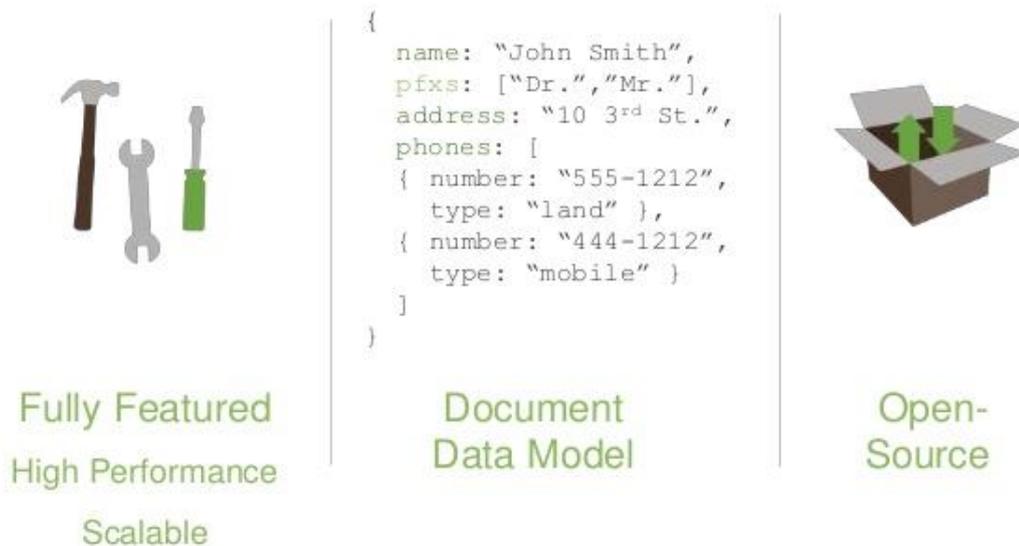


Figure 4 – MongoDB the leading NoSQL Database

The advantages for using MongoDB instead of Relational Databases in Big Data Application:

- Replication. The possibility of keeping data on more than one computer to increase the system availability.
- Sharding. The possibility of sharing the data on more than one computer in order to easy scale.

- Schema-less. The possibility of dealing of a wide range of data types that came from sources without a predefined data structure.
- Cloud based, Big Data.



Figure 5 – Replication and Sharding

3.2.2 MongoDB drivers

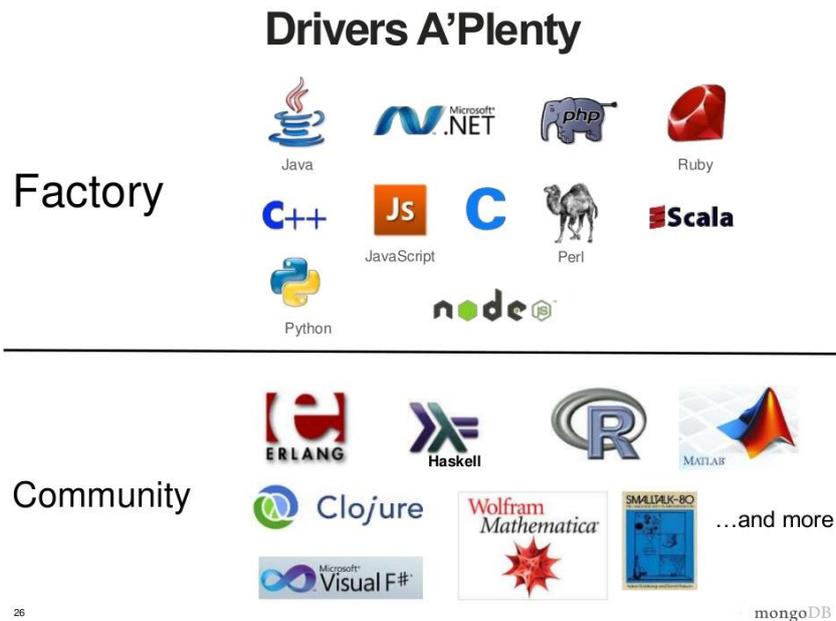
WiseGRID project is an heterogeneous project from the point of view of software platform used and Mongo Db provides drivers to the most important software developing platforms as:

1. Java
2. Microsoft .Net
3. PHP
4. Ruby
5. C++
6. JavaScript
7. C
8. Perl
9. Scala
10. Python
11. Node Js platform

All this software developing platforms can be used in WiseGRID project to interface the applications with the Big Data platform

There is also an important number of other special software development platforms that can be interfaced with Mongo DB trough the drivers developed by a community of developers connected to MongoDB as an open source software,

Mongo DB developers provide a picture with all the drivers supported by MongoDB

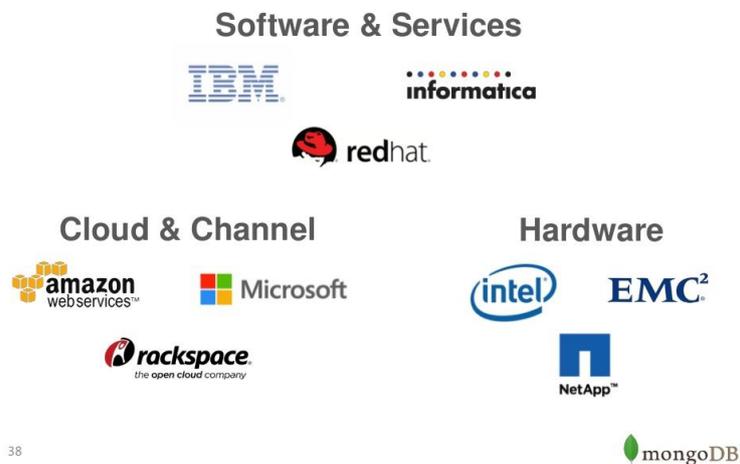


26

Figure 6 – MongoDB drivers

Also MongoDB is based on a strong ecosystem of partners,

MongoDB Partners (500)



38

Figure 7 – MongoDB partners

3.3 UNDERSTANDING MONGODB

3.3.1 MongoDB Structure

MongoDB is a document database. It stores in the database full documents with a flexible structure. In traditional Relational databases the structure of databases has to be defined in the beginning of building the database application. MongoDB comes with a lot of more flexibility since the structure of the documents to be stored is not defined in the beginning.

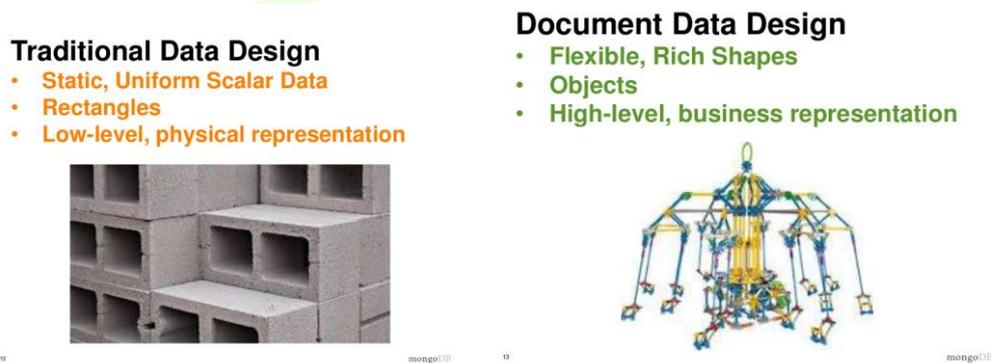


Figure 8 – MongoDB database structure vs RDBMS

MongoDB is structure-less. In this case there are some terminology changes versus the well-known RDBMS. The concept of data table from a RDBMS that indicates a data structure predefined in the application development is not anymore used and is replaced with the collection concept. The collection concept will be detailed in further chapters.

Terminology

RDBMS		MongoDB
Database	→	Database
Table	→	Collection
Index	→	Index
Row	→	Document
Column	→	Field
Join	→	Embedding & Linking & \$lookup

Figure 9 – MongoDB terminology vs RDBMS

3.3.2 MongoDB basics overview

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases [10].

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose [10].

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data [10].

Data in MongoDB has a flexible schema. Documents in the same collection. They do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Some considerations while designing Schema in MongoDB

- Design the schema according to user requirements.
- Combine objects into one document to be used together. Otherwise separate them (but make sure there should not be need of joins).
- Duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize the r schema for most frequent use cases.
- Do complex aggregation in the schema.

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'MongoDB getting started',
  url: 'http://www.getting_started.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

Data Model

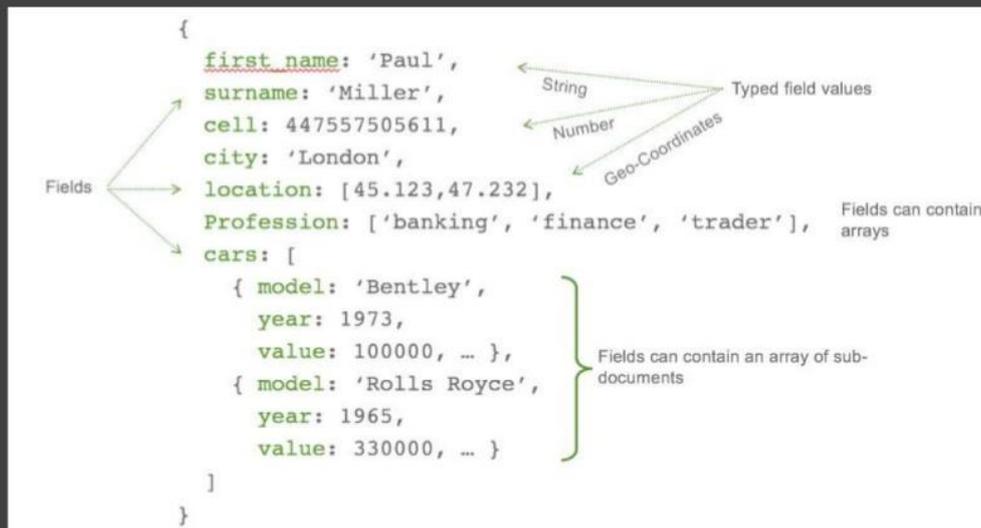


Figure 10 – MongoDB Data Model

3.3.3 Datatypes used by MongoDB

MongoDB support in his databases the using of several datatypes:

- **String:** This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer:** This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon server architecture
- **Boolean:** This type is used to store a boolean (true/ false) value.
- **Double:** This type is used to store floating point values.
- **Min/Max Keys:** This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays:** This type is used to store arrays or list or multiple values into one key.
- **Timestamp:** ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object:** This datatype is used for embedded documents.
- **Null:** This type is used to store a Null value.
- **Symbol:** This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date:** This datatype is used to store the current date or time in UNIX time format. A specific data format date time can be used by creating object of Date and passing day, month, year into it.
- **Object ID:** This datatype is used to store the document's ID.
- **Binary data:** This datatype is used to store binary data.

- **Code:** This datatype is used to store JavaScript code into the document.
- **Regular expression:** This datatype is used to store regular expression.

3.3.4 MongoDB Methods

MongoDB uses for processing data a set of methods. They are shortly indicated in the following parts of the document.

3.3.4.1 MongoDB methods for creating Database and collections

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows:

```
use DATABASE_NAME
```

Example

To create a database with name **<mydb>**, then **use DATABASE** statement would be as follows:

```
>use mydb
switched to db mydb
```

To check the currently selected database, use the command **db**

```
>db
Mydb
```

To check databases list, use the command **show dbs**.

```
>show dbs
local 0.78125GB
mydb 0.23012GB
test 0.23012GB
```

The createCollection() Method

MongoDB **db.createCollection(name, options)** is used to create collection.

Syntax

Basic syntax of **createCollection()** command is as follows: **db.createCollection(name, options)**

In the command, name is name of collection to be created. Options is a document and is used to specify configuration of collection.

Examples

Basic syntax of **createCollection()** method without options is as follows:

```
>use test
switched to db test
>db.createCollection("mycollection")
```

```
{"ok" : 1 }
```

To check the created collection can be done by using the command show collections.

```
>show collections
mycollection
system.indexes
```

3.3.4.2 MongoDB Insert document

The insert() Method

To insert data into MongoDB collection, methods to be used are MongoDB's **insert()** or **save()**.

Syntax

The basic syntax of **insert()** command is as follows :

```
>db.COLLECTION_NAME.insert(document)
```

Example

```
>db.mycol.insert({
  _id: ObjectId(7df78ad8902c),
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
})
```

3.3.4.3 MongoDB Query Document

The find() Method

To query data from MongoDB collection, it has to be used MongoDB's **find()** method.

Syntax

The basic syntax of **find()** method is as follows:

```
>db.COLLECTION_NAME.find()
```

find() method will display all the documents in a non-structured way.

The pretty() Method

To display the results in a formatted way, use **pretty()** method.

Syntax

```
>db.mycol.find().pretty()
```

Example

Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({"by": "tutorials point", "title": "MongoDB Overview"}).pretty()
{
```

```

    "_id": ObjectId(7df78ad8902c),
    "title": "MongoDB Overview",
    "description": "MongoDB is no sql database",
    "by": "tutorials point",
    "url": "http://www.tutorialspoint.com",
    "tags": ["mongodb", "database", "NoSQL"],
    "likes": "100"
  }
>

```

3.3.4.4 MongoDB Update Document Method

MongoDB Update() Method

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows:

```
>db.COLLECTION_NAME.update(SELECTIOIN_CRITERIA, UPDATED_DATA)
```

Example

Consider the mycol collection has the following data.

```

{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```

>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDBTutorial'}})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}

```

By default, MongoDB will update only a single document. To update multiple documents, set the parameter 'multi' to true.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB Tutorial'}},{multi:true})
```

3.3.4.5 MongoDB Sort records, Indexing, Agregation

The sort() Method

To sort documents in MongoDB, use **sort()** method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

Syntax

The basic syntax of **sort()** method is as follows:

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Example

Consider the collection mycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

Without specifying the sorting preference, the **sort()** method will display the documents in ascending order.

Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.

This scan is highly inefficient and require MongoDB to process a large volume of data.

Indexes are special data structures that store a small portion of the data set in an easy-to-traverse form. The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

The **ensureIndex()** Method

To create an index is needed to be used **ensureIndex()** method of MongoDB.

Syntax

The basic syntax of **ensureIndex()** method is as follows().

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Here key is the name of the file to create index and 1 is for ascending order. To create index in descending order is needed to be used –

1.

Example

```
>db.mycol.ensureIndex({"title":1})
```

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result. In SQL count (*) and with group by is an equivalent of mongodb aggregation.

The **aggregate()** Method

For the aggregation in MongoDB, should be used the **aggregate()** method.

Syntax

Basic syntax of **aggregate()** method is as follows:

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Example

In the collection are the following data:

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by_user: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100
}
```

```

    },
    {
      _id: ObjectId(7df78ad8902d)
      title: 'NoSQL Overview',
      description: 'No sql database is very fast',
      by_user: 'tutorials point',
      url: 'http://www.tutorialspoint.com',
      tags: ['mongodb', 'database', 'NoSQL'],
      likes: 10
    },
    {
      _id: ObjectId(7df78ad8902e)
      title: 'Neo4j Overview',
      description: 'Neo4j is no sql database',
      by_user: 'Neo4j',
      url: 'http://www.neo4j.com',
      tags: ['neo4j', 'database', 'NoSQL'],
      likes: 750
    }
  ]
}

```

Now from the above collection, in order to display a list stating how many tutorials are written by each user, then has to be used the following **aggregate()** method:

```

> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum :1}}}]
{
  "result" : [
    {
      "_id" : "tutorials point",
      "num_tutorial" : 2
    },
    {
      "_id" : "Neo4j",
      "num_tutorial" : 1
    }
  ],
  "ok" : 1
}
>

```

4 ARCHITECTURE OF BIG DATA CLOUD BASED INFRASTRUCTURE

The Big Data platform infrastructure has to be able to run a NoSQL database , accessible from all the WiseGRID application heterogeneous ecosystem , that will provide cloud based services to the applications that will need to store long term data , retrieve the stored data, process and filter the stored data and also to use special process for data analysis or mining.

4.1 SPECIFIC REQUIREMENTS IN WISEGRID PROJECT

The Big Data platform that will be specific designed for the WiseGRID project was defined like having some specific requirements:

1. High availability. The platform has to be able to run in a 24/7 manner, able to cope with hardware failures, able to be restored from a hardware failure and maintained without stopping the services to the WiseGRID Applications.
2. Easily scalable. The WiseGRID project is a research and development project that can have an important extension over the production phase. The Big Data system has to be able to be scaled in an inexpensive manner to the continuously growing requirements.
3. Secure. The data stored in the Big Data platform are vital for the good working of the applications and has to be well protected from intrusion from unauthorized parties.

4.1.1 MongoDB replication

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers. Replication protects a database from the loss of a single server.

Replication also allows to recover from hardware failure and service interruptions. With additional copies of the data, one can be dedicated to disaster recovery, reporting, or backup.

Why Replication?

1. To keep data safe
2. High (24/7) availability of data
3. Disaster recovery
4. No downtime for maintenance (like backups, index rebuilds, compaction)
5. Read scaling (extra copies to read from)
6. Replica set is transparent to the application

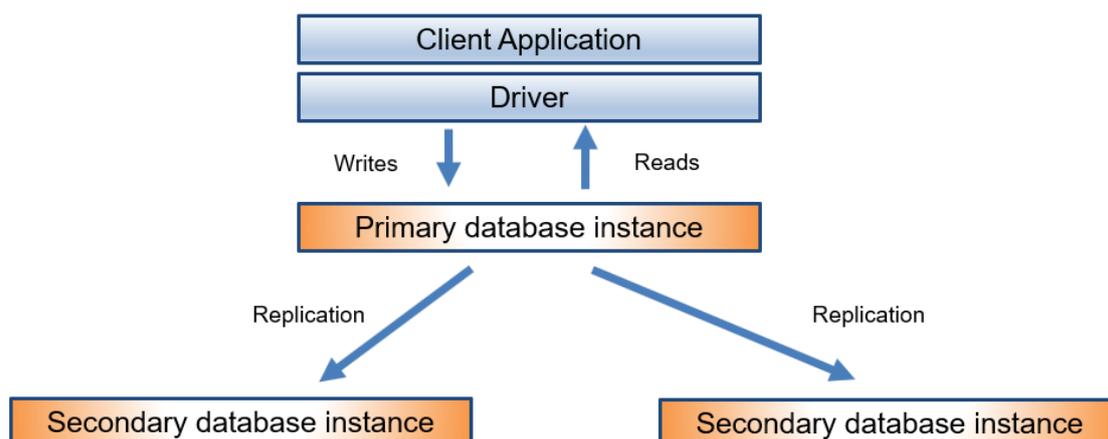


Figure 11 – Replication Diagram

4.1.2 MongoDB sharding

Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling.

With sharding, add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local disk is not big enough
- Vertical scaling is too expensive

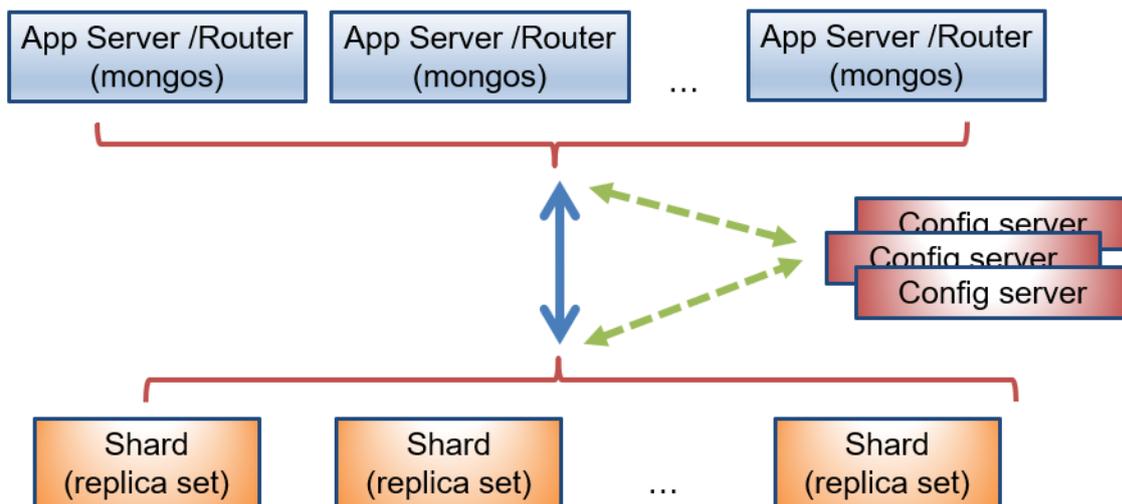


Figure 12 – Sharding Diagram

4.1.3 MongoDB cluster definition

The MongoDB cluster will be built over multiple nodes that are separated computers connected in a network on the clustering principles that will be detailed in the next chapter.

4.2 ARCHITECTURE ADAPTED FOR WISEGRID PROJECT

4.2.1 General concepts

The Big Data platform will provide database storage, retrieving and processing for all the application that will be developed in the WiseGRID project. The application will access the Big Data platform by direct Database engine access through dedicated cluster computers.

Integration in general WiseGRID Architecture specifications of the Big Data platform is described in the following picture.

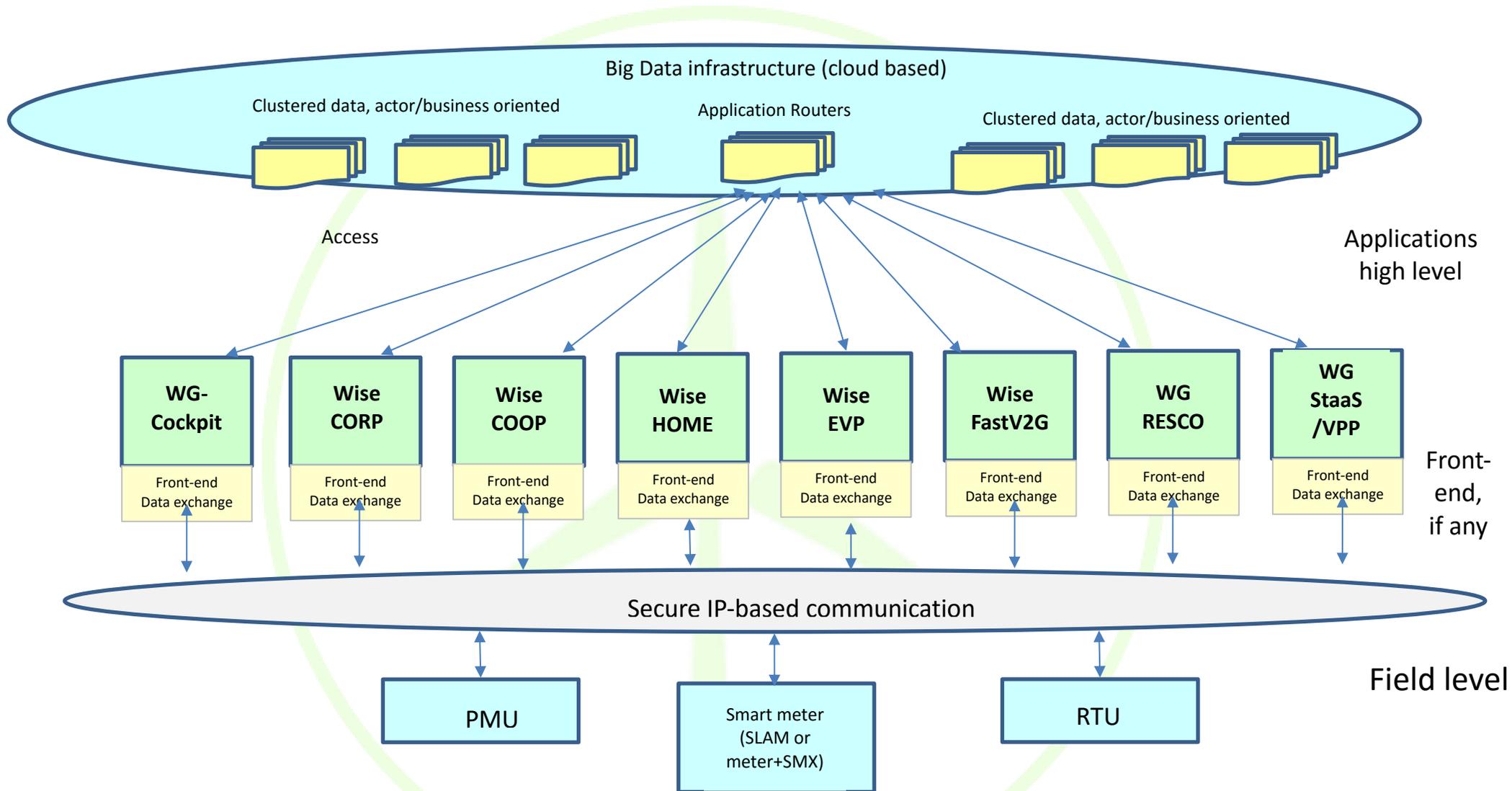


Figure 13 – WiseGRID Big Data Architecture overview

4.2.2 Computer cluster

The Big Data platform will be implemented by a Linux computer cluster. Linux clustering is more and more popular today in many industries. The acceptance of open source software is very important today and more and more ICT applications are migrating towards it. Linux as an open source operating system, becomes so utilized due his high reliability, efficiency. The clustering technology for Linux computer is mature and allows now to create supercomputers at a fraction of the cost of traditional high performance machines. Now by using a number of normal, high accessible computers can be obtained the computation power that was otherwise only accessible to high performance machines.

In these days there are some types of computer clusters:

- **Fail-over clusters** where the computers are working for reserving each other so the availability of the system goes higher. In case of one or more of the nodes computers are failing their job is taken over by other nodes without big performance lost and giving time to the maintenance team to cope with the problems of the failed computers.
- **Load balancing cluster** where the nodes of the cluster are providing the same type of service and balancing the computation power over the nodes giving the cluster the possibility to sustain a needed computation power near the sum of his nodes.
- **High performance** cluster where the nodes are running special applications that are sharing the resources of the nodes simulating a high performance machine with aggregated resources.

There is a possibility to run a combination of these types of clusters. For the online services of Big Data platform that are storing and retrieving data, it will be used a combination of the first two types: Load balancing cluster and Fail-over clusters.

The simplest fail-over cluster has two nodes: one stays active and the other stays on stand-by but constantly monitors the active one. In case the active node goes down, the stand-by node takes over, allowing a mission-critical system to continue functioning. The Big Data platform will have nodes that are running in the same time replicating the same data process as indicated in chapter 4.1.1

Load-balancing clusters are commonly used for busy Web sites where several nodes host the same site, and each new request for a Web page is dynamically routed to a node with a lower load. In our Big Data application the process of sharding described in chapter 4.1.2 will balance the load and the database capacity over multiples nodes of the cluster.

An example of clustering structure is indicated by Red-hat an enterprise Linux distribution developer in the next figure:

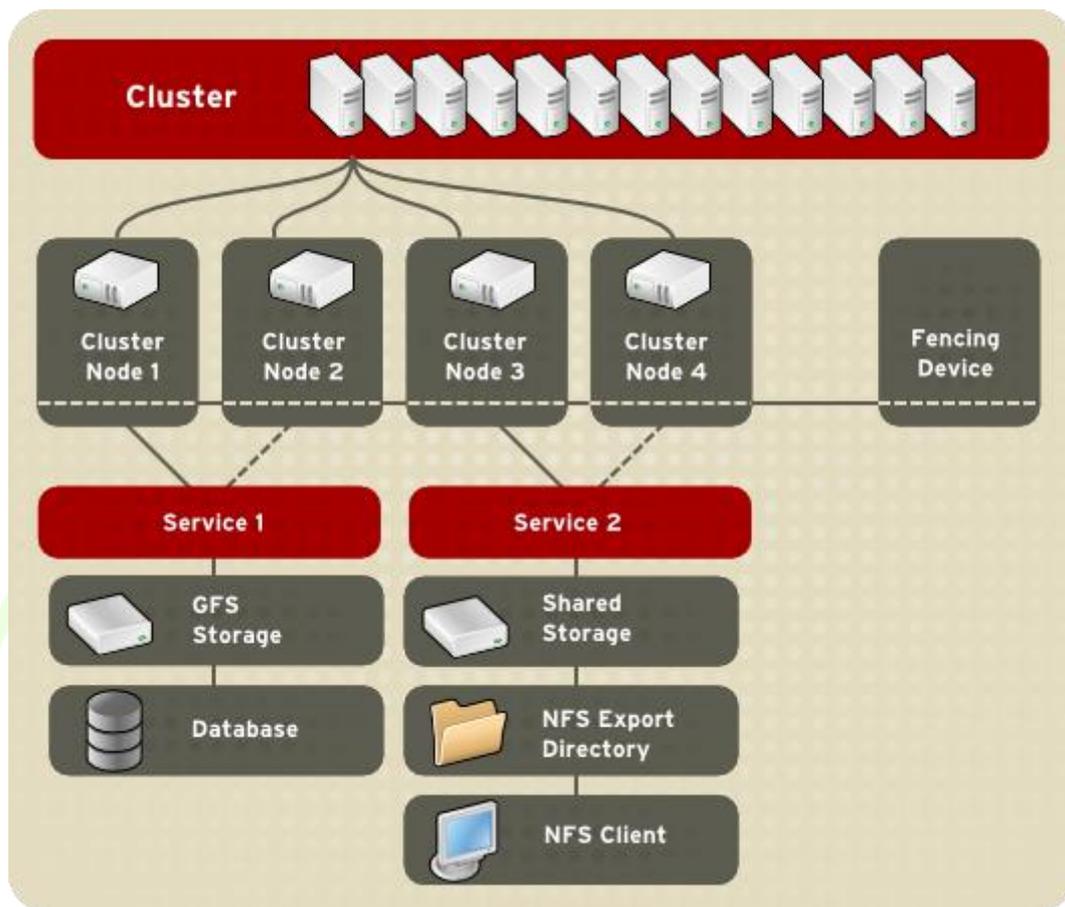


Figure 14 – RedHat Cluster structure example

4.2.3 Managing computer clusters

Establishing and managing a computer cluster can be a very complicated task depending of the number of nodes involved in the cluster. Due to high requirements in computing power the clusters can have hundreds or thousands of nodes.



Figure 15 – Atlas Peloton cluster [11]

The software side of setting up a cluster is a two-stage process: first, install the cluster management server and second, install the rest of the cluster. Following this process enables the user to use the management server to help to configure the rest of the cluster and to prepare for post-installation maintenance and operation. For managing the nodes of our Big Data application there will be used remote managing application able to install, apply settings and control processes from remote through the network so the direct access to the node computers will be not necessary.

There are many remote managing applications. It will be proposed further the open source application named Webmin.

The developers of Webmin are defining in their site that:

Webmin is a web-based interface for system administration for Unix. Using any modern web browser, you can setup user accounts, Apache, DNS, file sharing and much more. Webmin removes the need to manually edit Unix configuration files like /etc/passwd, and lets you manage a system from the console or remotely. See the standard modules page for a list of all the functions built into Webmin [12].

Webmin is continuously maintained and there are many repository that contain packages with updated webmin for Linux distributions.

Installing webmin on a cluster node based on Debian distribution is a simple and straight forward process described in the manual of webmin.

Using the Webmin APT repository

To install and update Webmin via APT, edit the `/etc/apt/sources.list` file on a system and add the line :

```
deb https://download.webmin.com/download/repository sarge contrib
```

It is needed also to fetch and install GPG key with which the repository is signed, with the commands:

```
cd /root
wget http://www.webmin.com/jcameron-key.asc
apt-key add jcameron-key.asc
```

Now is possible to install with the commands:

```
apt-get install apt-transport-https
apt-get update
apt-get install webmin
```

All dependencies should be resolved automatically [12].

An example of dashboard of webmin running on a node computer implemented on a single board computer type Raspberry Pi 3 can be seen on the figure bellow:

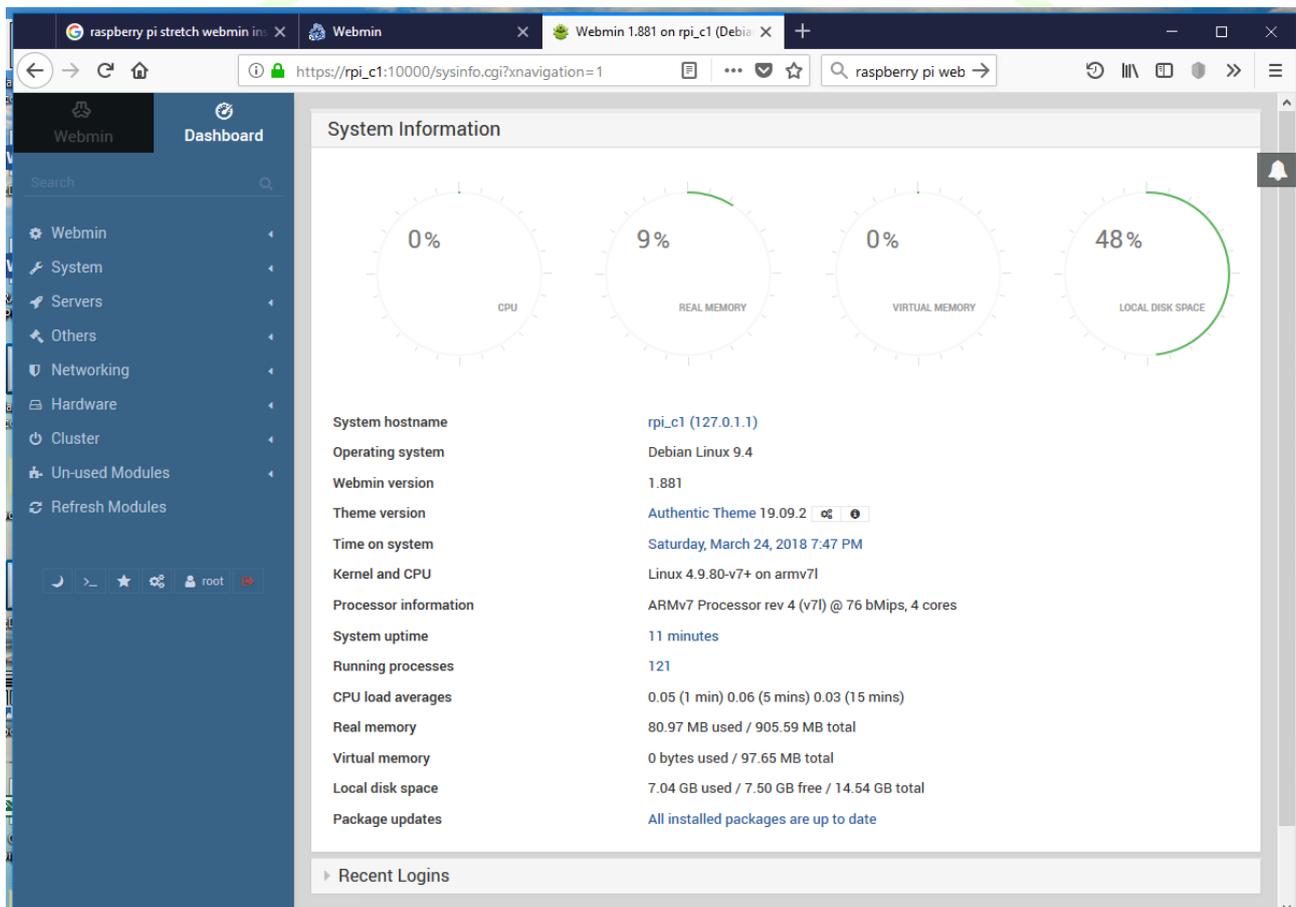


Figure 16 – Webmin dashboard

For managing the cluster over Webmin, this application should be prior installed on all nodes of the cluster and then registered on the webmin server manager.

After installing Webmin manually on cluster nodes, webmin provides methods of scanning the subnet for installed webmin instances like in the following image:

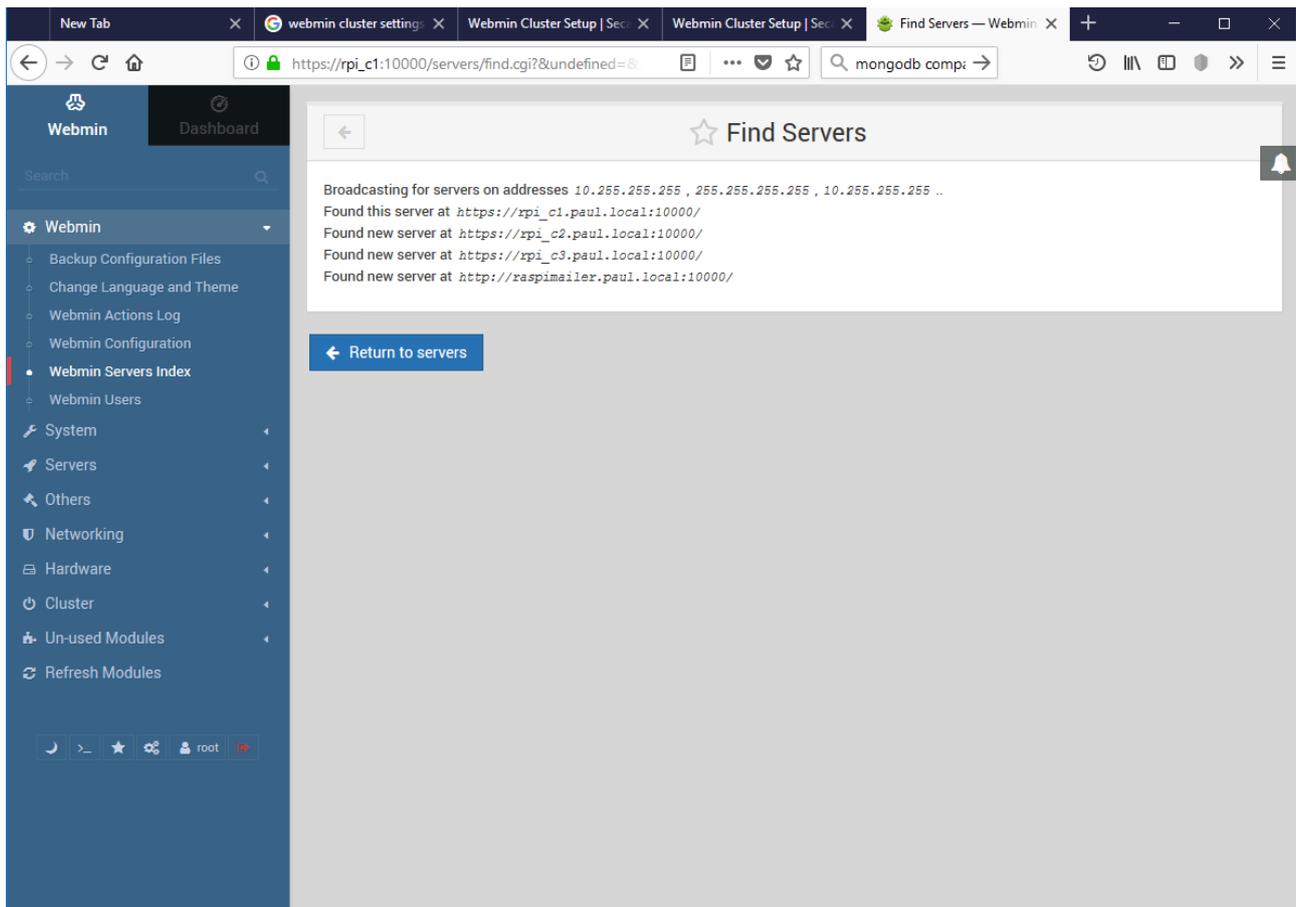


Figure 17 – Webmin, scanning for instances

The detected instances can be registered to webmin sever list:

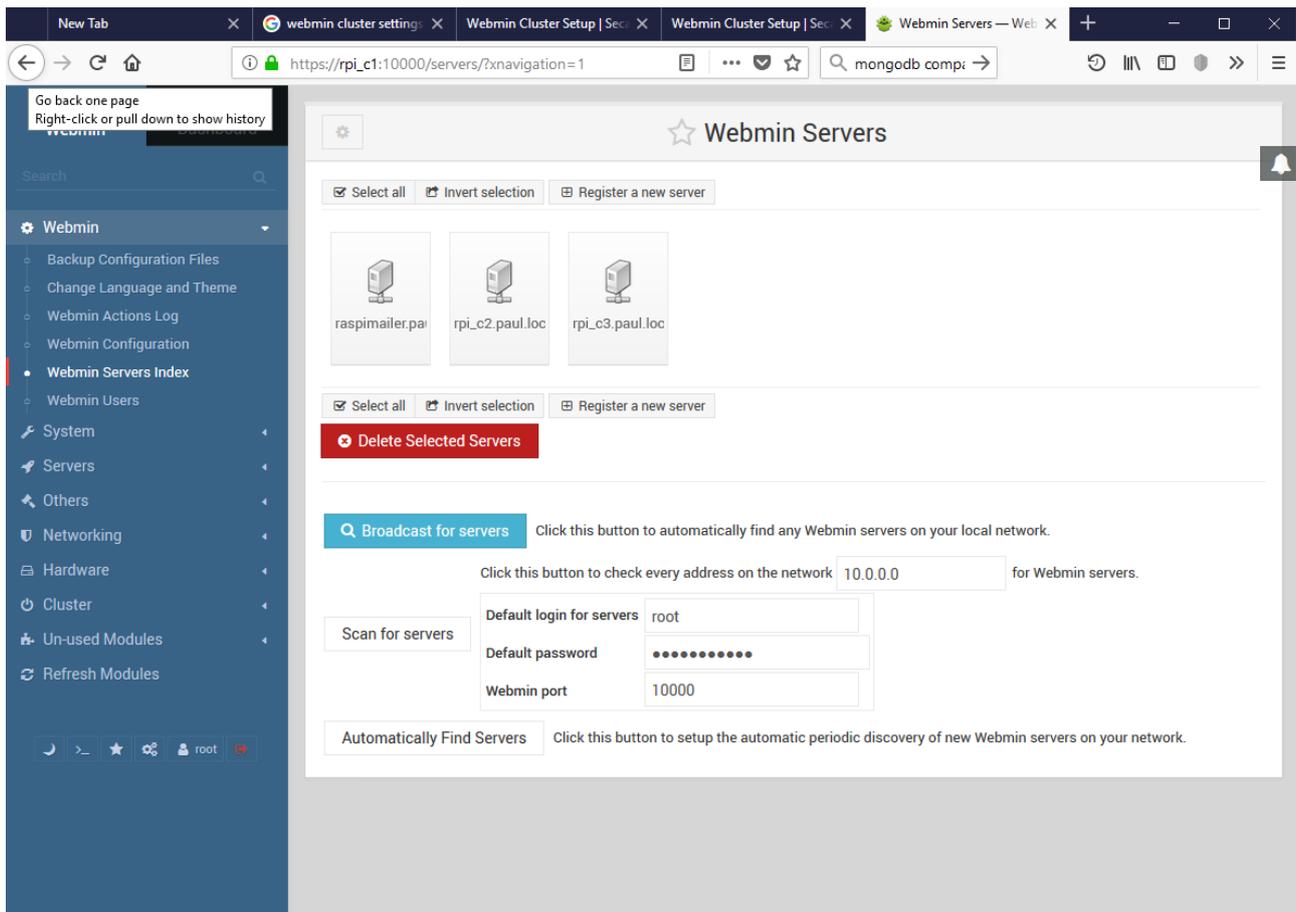


Figure 18 – Webmin Registering found servers

With registered servers Webmin allows now to process cluster commands as:

- Changing passwords on cluster nodes.
- Copy files on cluster nodes.
- Executing Cron jobs on cluster nodes. Cron is a Linux command that features scheduled execution of processes.
- Executing Shell commands over cluster nodes.
- Installing packages on cluster nodes.

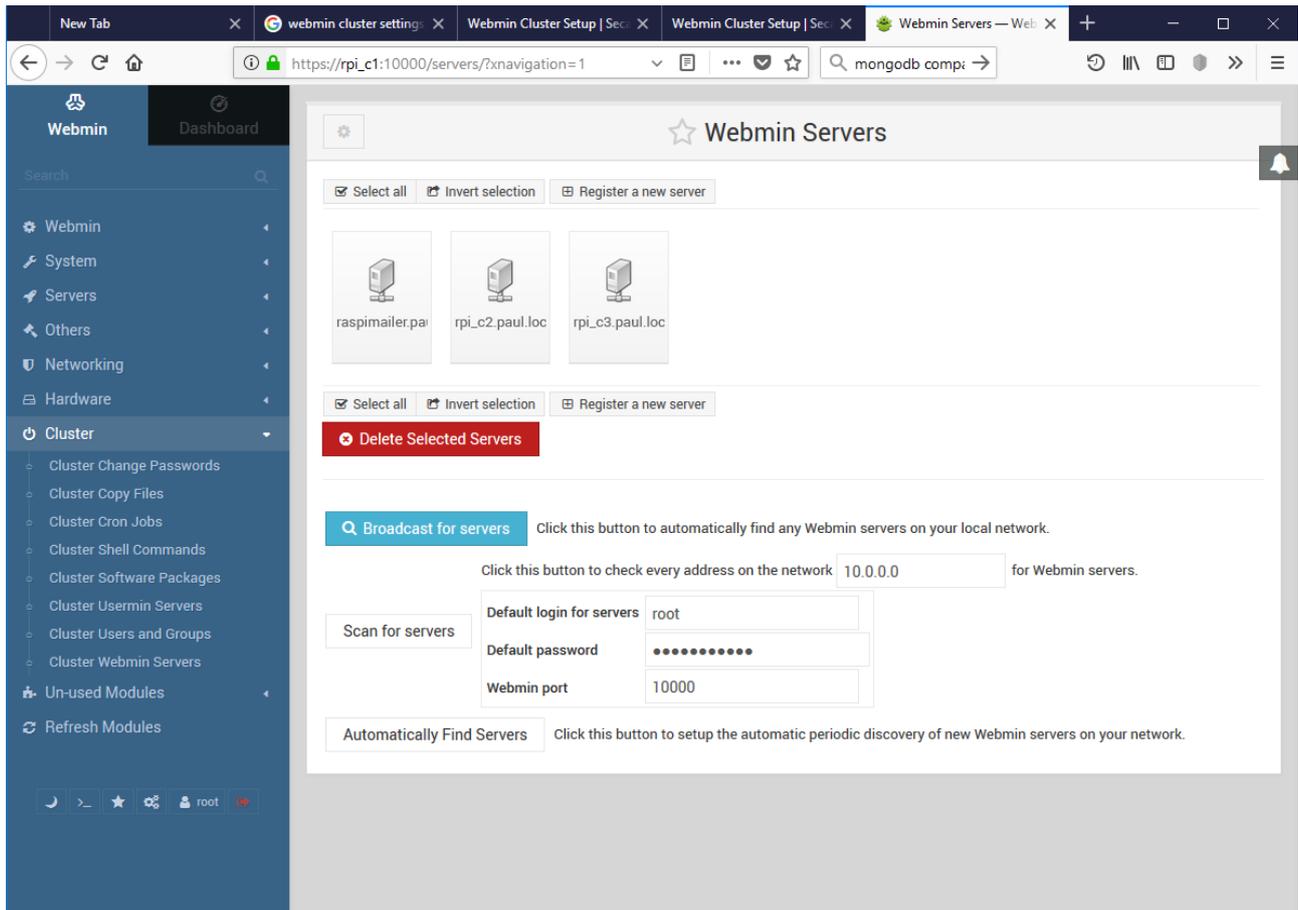


Figure 19 – Webmin cluster operation

4.2.4 MongoDB computer cluster

The Big Data platform for WiseGRID applications should be scalable in order to be able to withstand to increasing requirements from the applications in the development phase and also in the deployment phase. The requirements from the applications can be only estimated in the beginning, the real life deployment should define the amount of resources needed.

Scalability of platform will be the defining characteristic of this platform.

There are two broad categories of scaling strategies for data.

- **Vertical scaling** involves adding more resources to a server so that it can handle larger datasets. The upside is that the process is usually as simple as migrating the database, but it often involves downtime and is difficult to automate.
- **Horizontal scaling** involves adding more servers to increase the resources, and is generally preferred in configurations that use fast-growing, dynamic datasets. Because it is based on the concept of adding more servers, not more resources on one server, datasets often need to be broken into parts and distributed across the servers. Sharding refers to the breaking up of data into subsets so that it can be stored on separate database servers (a sharded cluster).

The solution for scalability in this case is a computer cluster hosting the MongoDB database management system.

For a MongoDB cluster architecture a recommended structure is defined by the developers. There are three types of servers:

- **Config Server** - This stores metadata and configuration settings for the rest of the cluster. There can be used only one config server for simplicity but in production environments, this should be a replica set of at least three cluster nodes.
- **Query Router** - The mongos daemon of MongoDB installation acts as an interface between the client application and the cluster shards. Since data is distributed among multiple servers, queries need to be routed to the shard where a given piece of information is stored. The query router is run on the application server. To the limit only one Query Router can be used, although is recommended to use one on each application server in the cluster. Considering the application structure of WiseGRID project there should be considered a least one Query Router server for each application
- **Shard** - A shard is simply a database server that holds a portion of data. Items in the database are divided among shards either by range or hashing, which we'll explain later. The number of shards will be defined based on the resources requested by the applications. For increased availability of cluster there should be considered that a shard can be a replica set. For production cluster a recommended number of replica nodes on each shard is of 3 members.

A diagram of the cluster structure can be seen in Figure 12 – Sharding Diagram

There is a Primary shard for each database that holds all the unsharded collections for that database. Each database has its own primary shard. The primary shard has no relation to the primary in a replica set. The collections from a database can be distributed on more than one shard.

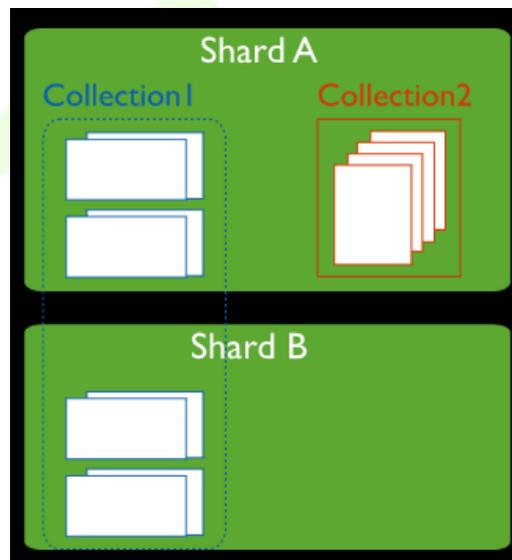


Figure 20 – Data distribution over multiple shards

When it is deployed a new sharded cluster with shards that were previously used as replica sets, all existing databases continue to reside on their original replica sets. Databases created subsequently may reside on any shard in the cluster.

4.2.5 Setting a MongoDB computer cluster

A MongoDB computer cluster is a cluster that are providing the services for a MongoDB database management system.

The first step of setting a MongoDB computer cluster is to install on all nodes the MongoDB database manager.

The next step is to allocate to each cluster node the role into the cluster that it will play.

The cluster nodes will have allocated a private IP behind the router and the cluster network will become accessible from the Internet through a router that will filter the access only to the application router. The communication between the cluster nodes will be done over the internal network and not through internet. Through internet only the application communication will take place.

In order to deploy a sharded cluster the following steps should be executed:

- Start the Config Server Database Instances
- Start the mongos Instances
- Add Shards to the Cluster
- Enable Sharding for a Database
- Shard a Collection

4.2.5.1 Start the config Server Database Instances

As declared previously for a production cluster, the recommended number of configuration server cluster nodes will be three. On the config servers node will run config server processes that are mongod instances that store the cluster's metadata. A mongod is designated as a config server using the `--configsvr` option. Each config server stores a complete copy of the cluster's metadata.

In production deployments, exactly three config server instances, each running on different servers to assure good uptime and data safety. In test environments, all three instances can run on a single server.

Create data directories for each of the three config server instances. By default, a config server stores its data files in the `/data/configdb` directory. Choose a different location even on a storage server. To create a data directory, issue a command similar to the following:

```
mkdir /data/configdb
```

Start the three config server instances. Start each by issuing a command using the following syntax:

```
mongod --configsvr --dbpath <path> --port <port>
```

The default port for config servers is 27019. Specifying a different port is an alternative. The following example starts a config server using the default port and default data directory:

```
mongod --configsvr --dbpath /data/configdb --port 27019
```

For additional command options, see `mongod` or Configuration File Options in the MongoDB manuals [8].

4.2.5.2 Start the mongos Instances

The mongos Instances are application routers. They will route the information from user applications to the sharded cluster containing the databases.

The mongos instances are lightweight and do not require data directories since they will not participate directly to data storage and retrieval. A mongos instance can be run on a system that runs other cluster components, such as on an application server or a server running a mongod process. By default, a mongos instance runs on port 27017.

WiseGRID is proposing to use separate cluster nodes for each WiseGRID application in order to provide the requested processing power and isolation between applications.

To start a mongos instance, issue a command on corresponding cluster node using the following syntax:

```
mongos --configdb <config server hostnames>
```

For example, to start a mongos that connects to config server instance running on the following hosts and on the default ports:

```
cfg0.paul.local  
cfg1.paul.local  
cfg2.paul.local
```

The following command would be issued:

```
mongos --configdb cfg0.paul.local:27019,cfg1.paul.local:27019,cfg2.paul.local:27019
```

Each mongos in a sharded cluster must use the same configDB string, with identical host names listed in identical order.

4.2.5.3 Add Shards to the Cluster

The shards are cluster nodes that are containing and processing directly the databases. A shard can be a standalone mongod instance on a cluster node or a replica set running on multiple cluster nodes. In a production environment, each shard should be a replica set in order to increase the availability of the database. Use the procedure described in MongoDB manual in Deploy a Replica Set to deploy replica sets for each shard.

Up to this moment, the settings were applied through the OS shell. From now on, we are directly working to the mongoDB database and for a mongo shell we will connect to the mongos instance. Issue a command using the following syntax:

```
mongo --host <hostname of machine running mongos> --port <port mongos listens on>
```

For example, if a mongos is accessible at mongos0.paul.local on port 27017, issue the following command:

```
mongo --host mongos0.example.net --port 27017
```

Add each shard to the cluster using the `sh.addShard()` method, as shown in the examples below. Issue `sh.addShard()` separately for each shard. If the shard is a replica set, specify the name of the replica set and specify a member of the set. In production deployments, all shards should be replica sets.

To add a shard for a replica set named `rs1` with a member running on port 27017 on `mongodb0.paul.local`, issue the following command:

```
sh.addShard( "rs1/mongodb0.paul.local:27017" )
```

To add a shard for a standalone mongod on port 27017 of `mongodb0.paul.local`, issue the following command:

```
sh.addShard( "mongodb0.example.net:27017" )
```

4.2.5.4 Enable Sharding for a Database

Before shard a collection, it must enabled sharding for the collection's database. Enabling sharding for a database does not redistribute data but make it possible to shard the collections in that database. All databases in WiseGRID project will have sharding enabled from the beginning.

Once sharding is enabled for a database, MongoDB assigns a primary shard for that database where MongoDB stores all data before sharding begins.

From a mongo shell, connect to the mongos instance. Issue a command using the following syntax:

```
mongo --host <hostname of machine running mongos> --port <port mongos listens on>
```

Issue the `sh.enableSharding()` method, specifying the name of the database for which to enable sharding. Use the following syntax:

```
sh.enableSharding("<database>")
```

Optionally, sharding can be enabled for a database using the `enableSharding` command, which uses the following syntax:

```
db.runCommand( { enableSharding: <database> } )
```

4.2.5.5 Shard a Collection

On a database that has sharding enabled, set collections sharded. The user shards on a per-collection basis.

It is possible to determine what has to be used for the shard key. The selection of the shard key affects the efficiency of sharding. See the selection considerations listed MongoDB manual in the Considerations for Selecting Shard Key.

If the collection already contains data, an index on the shard key using `createIndex()` must be created. If the collection is empty then MongoDB will create the index as part of the `sh.shardCollection()` step.

Shard a collection by issuing the `sh.shardCollection()` method in the mongo shell. The method uses the following syntax:

```
sh.shardCollection("<database>.<collection>", shard-key-pattern)
```

Replace the `<database>.<collection>` string with the full namespace of database, which consists of the name of the database, a dot (e.g. `.`), and the full name of the collection. The `shard-key-pattern` represents the shard key, which is specified in the same form as an index key pattern.

Example

The following sequence of commands shards four collections:

```
sh.shardCollection("records.people", { "zipcode": 1, "name": 1 } )
```

```
sh.shardCollection("people.addresses", { "state": 1, "_id": 1 } )
```

```
sh.shardCollection("assets.chairs", { "type": 1, "_id": 1 } )
```

```
sh.shardCollection("events.alerts", { "_id": "hashed" } )
```

4.2.6 Administration of a MongoDB computer cluster

The administration documentation addresses the ongoing operation and maintenance of MongoDB instances and deployments. This documentation includes both high level overviews of these concerns as well as tutorials that cover specific procedures and processes for operating MongoDB.

A popular tool for mongoDB administration is MongoDB Compass

MongoDB Compass is a tool developed by the same team as the MongoDB database. The developer team declares the following characteristics and features [13]:

The Easiest Way to Explore and Manipulate MongoDB Data.

The GUI for MongoDB. Visually explore your data. Run ad hoc queries in seconds. Interact with your data with full CRUD functionality. View and optimize your query performance. Available on Linux, Mac, or Windows. Compass empowers you to make smarter decisions about indexing, document validation, and more.

Know your data with built-in schema visualization

MongoDB Compass analyzes your documents and displays rich structures within your collections through an intuitive GUI. It allows you to quickly visualize and explore your schema to understand the frequency, types and ranges of fields in your data set.

Get immediate insight into server status and query performance

Real-time server statistics let you view key server metrics and database operations. Drill down into database operations easily and understand your most active collections.

Visualize, understand, and work with geospatial data

Point and click to construct sophisticated queries, execute them with the push of a button and Compass will display your results both graphically and as set of JSON documents.

A better approach to CRUD makes it easier to interact with data

Modify existing documents with greater confidence using the intuitive visual editor, or insert new documents and clone or delete existing ones in just a few clicks.

Understand performance issues with visual explain plans

Know how queries are running through an easy-to-understand GUI that helps you identify and resolve performance issues.

View utilization and manage indexes

Understand the type and size of your indexes, their utilization and special properties. Add and remove indexes at the click of a button.

A simpler way to validate data

Create and modify rules that validate your data using a simple point and click interface. CRUD support lets you fix data quality issues easily in individual documents.

Extensible via plugins

The Compass Plugin Framework is exposed as an API, making it extensible by users. Looking for other functionality? Install a plugin or build your own [14].

Mongo compass can be used for managing standalone MongoDB instances or cluster MongoDB instances.

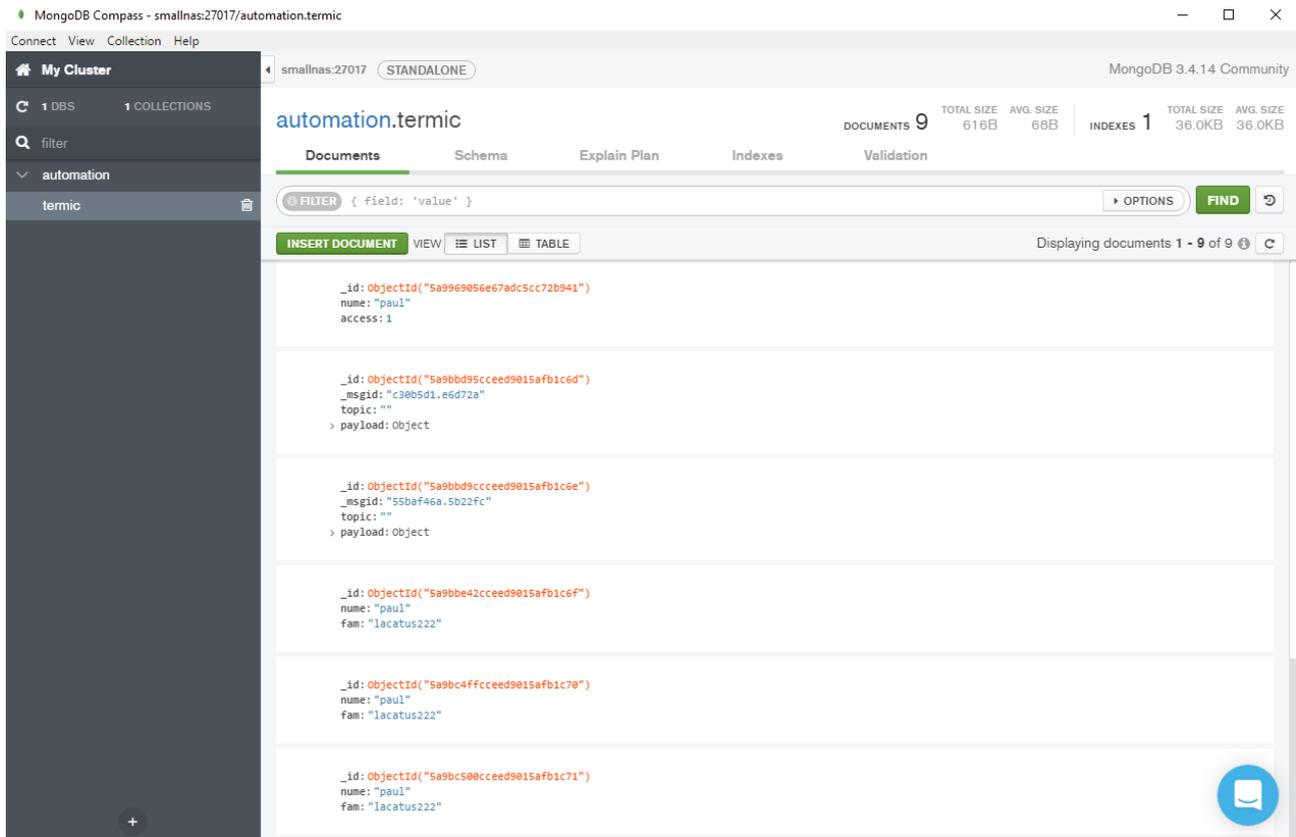


Figure 21– MongoDB Compass screenshot

4.2.7 Horizontally scaling a MongoDB computer cluster

The WiseGRID applications are now in the development phase. Right now only an estimation of needed resources can be done. MongoDB provides the flexibility to start the production cluster with a minimal set of resources and after deploying them. When the real resources requirements will be calculated, it will be possible to scale the computer cluster in order to fulfil the application requirements. The Horizontally scaling will be done by adding new shards to the cluster.

To add shards to a sharded cluster after the creation of the cluster or any time when needed to add capacity to the cluster.

When scaling the cluster some considerations will be used.

4.2.7.1 Balancing after adding a shard to a cluster

When a shard is added to a sharded cluster, this affect the balance of chunks among the shards of a cluster for all existing sharded collections. The balancer will begin migrating chunks so that the cluster will achieve balance. See *Sharded Collection Balancing in MongoDB manual* for more information [8].

Chunk migrations can have an impact on disk space. *Starting in MongoDB 2.6, the source shard automatically archives the migrated documents by default. For details, see moveChunk directory* [8].

4.2.7.2 Capacity Planning when adding a shard to a cluster

When adding a shard to a cluster, always ensure that the cluster has enough capacity to support the migration required for balancing the cluster without affecting legitimate production traffic. The adding of the new shard can de bone on a production running database

4.2.7.3 Add a Shard to a Cluster

For adding the shard to a cluster has to interact with a sharded cluster by connecting to a mongos instance.

From a mongo shell, connect to the mongos instance. For example, if a mongos is accessible at mongos0.paul.local on port 27017, issues the following command:

```
mongo --host mongos0.paul.local --port 27017
```

Adding a shard to the cluster will be done using the *sh.addShard()* method, as shown in the examples below. Issue *sh.addShard()* separately for each shard. If the shard is a replica set, specify the name of the replica set and specify a member of the set. In production deployments, all shards should be replica sets.

The following are examples of adding a shard with *sh.addShard()*:

To add a shard for a replica set named rs1 with a member running on port 27017 on mongodb0.oaul.local, issue the following command:

```
sh.addShard( "rs1/mongodb0.paul.local:27017" )
```

To add a shard for a standalone mongod on port 27017 of mongodb0.paul.local, issue the following command:

```
sh.addShard( "mongodb0.paul.local:27017" )
```

After issuing the commands chunks of data will migrate on a new shard that can take a time on a production cluster

5 PRIVACY AND DATA PROTECTION IN WISEGRID BIG DATA INFRASTRUCTURE

5.1 BIG DATA CLOUD BASED ARCHITECTURE LOCATION OF SENSITIVE DATA

The privacy and data protection are treated in the special deliverable of the project D3.2 “WiseGRID architecture, data models, standards and data protection (v2)”.



6 Interface with IOP and applications

6.1 APPLICATION REQUIREMENTS FROM WISEGRID APPLICATIONS TO THE BIG DATA PLATFORM

6.1.1 Big data requirements from WG IOP application

6.1.1.1 Short application description

The WiseGRID Interoperable Platform (WG IOP) is a scalable, secure and open ICT platform, with interoperable interfaces, for real-time monitoring and decentralized control to support effective operation of the energy network. The objective of the platform is to manage and process the heterogeneous and massive data streams coming from the distributed energy infrastructure deployed. This platform enable new services and reduce ICT costs for prosumers and smaller players, whilst it will facilitate cross-network and cross-entity interoperability. In order to increase adoption and speed up deployment, this platform will have open interfaces to the relevant energy, IoT and Smart City standards. It enable the cooperation and synergies among the different actors targeted by the different WiseGRID technological solutions. The core of the architecture is the RabbitMQ message broker by which all message are flowing, the broker is agnostic about the content of the message and its structure.

6.1.1.2 Interface with Big Data platform

As detailed in the D4.2 “WiseGRID interoperable Integrated Process (WG IOP)”, the WG IOP does not directly access the big data platform but uses external services (DB interaction services) and WiseGRID tools like the WG RESCO or WG Cockpit and so on to perform DB readings and writings. The next image show the WG IOP architecture in which it is possible to identify the interaction with the Big Data platform via external services. Another example of interface with Big Data platform is also available in the section dedicated to the WG RESCO tool in which is also better described the technologies used by the tool and DB interaction services to interface with the big data platform.

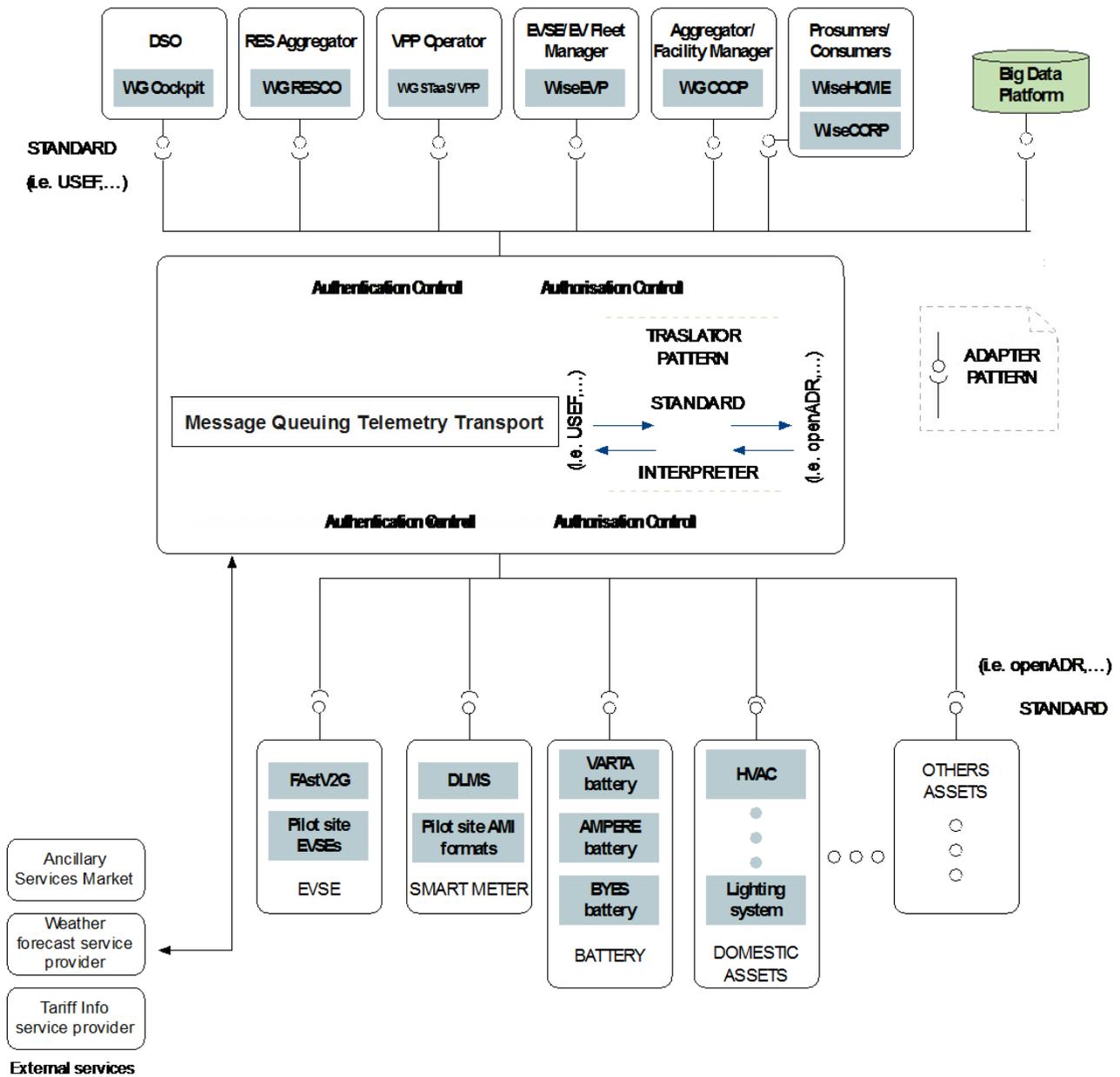


Figure 22 – WG IOP Architecture

6.1.2 Big data requirements from WG Cockpit application

6.1.2.1 Short application description

WiseGRID Cockpit is the WiseGRID technological solution targeting DSOs and microgrid operators, allowing them to control, manage and monitor their own grid, improving flexibility, stability and security of their network. Taking into account the goals of the project, the features to be implemented within WiseGRID cockpit consider a scenario of increasing share of distributed renewable resources and services provided by communities of prosumers (aggregated in the form of VPPs or cooperatives in order to achieve higher participation and environmental, social and economic benefits).

The main purpose of the WiseGRID Cockpit is to enable DSOs to manage the fundamental changes that

distribution grids are facing nowadays, some remarkable ones of those being the transition towards a grid with high penetration of distributed renewable energy resources and the presence of additional significant loads coming from electric vehicles among others. In addition, this particular outcome of the WiseGRID project aims at approaching the benefits that new technologies (such as big data or unbundled smart meters) and algorithms (such as state estimation or fault detection) bring to the operation of the grid. Finally, since one of the objectives of the project is the empowerment of the citizens in the energy field, the WiseGRID Cockpit will also demonstrate how that empowerment can be beneficial for several actors - including DSOs - , and how the whole ecosystem of actors can contribute to reach an environmentally and economically sustainable energy system.



Figure 23 - WiseGRID Cockpit

6.1.2.2 Interface with Big Data platform

The big data platform will be used by the WiseGRID Cockpit in order to hold the history of data provided by the sensors of interest of the application. As depicted in the architecture of the WG Cockpit, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds)
- Advanced Metering Infrastructure systems, retrieving data from already deployed Smart Meters – usually hourly curves retrieved once a day
- SCADA systems, retrieving real-time electric measurements from monitored infrastructure under control of the DSO (buses of the distribution grid) and status of safety elements

This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseGRID Cockpit application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

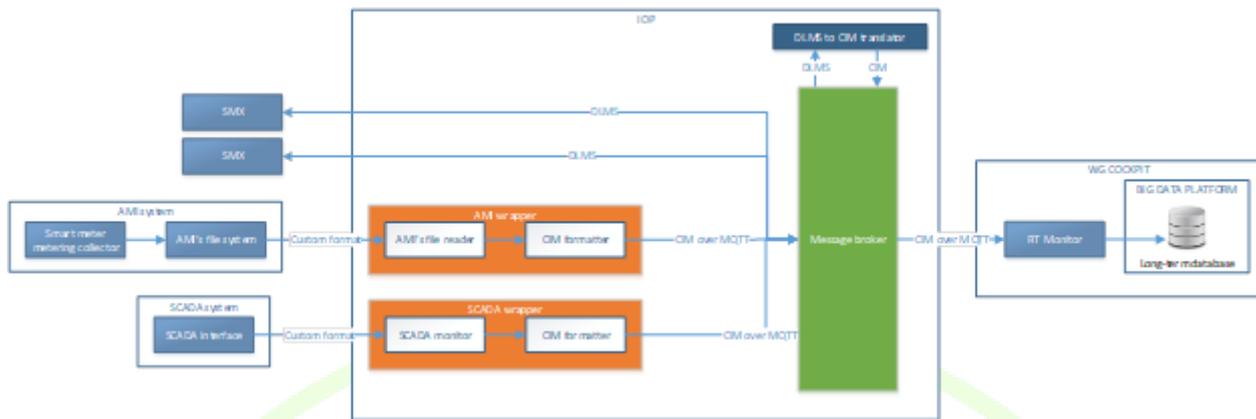


Figure 24 – WG Cockpit interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WG Cockpit application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseGRID Cockpit will use the Data mining and analytics module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, which will process the information to obtain:

- Hourly/Daily/Monthly aggregated data to produce summaries of interest to the DSO operator (energy imported from HV grid, energy produced)
- Statistical data (average, normal deviation) on electric parameters affecting quality of the service (frequency, voltage)

These jobs appear in the architecture of the WiseGRID Cockpit as KPI engine module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data) and published to the internal Enterprise Service Bus if required by other modules.

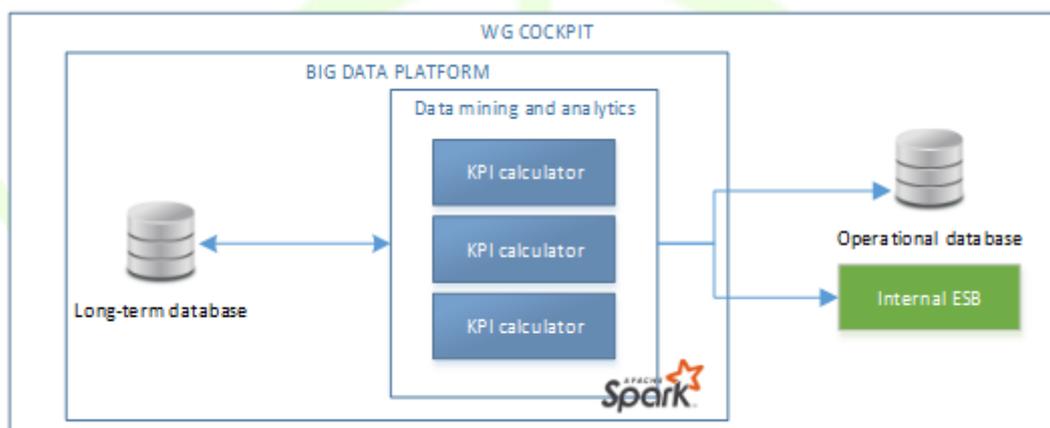


Figure 25 – WG Cockpit data mining

As part of the design phase of the application, a first estimation of the big data storage size required to deploy the application in the different pilot sites has been taken. Results are presented in the following table:

Table 1 – Resource estimation for WG Cockpit

Collection	Comments	Avg. doc. size (bytes)	Interval (s)	Retention time (months)	#Sensors Crevillent	#Sensors Terni	#Sensors Mesogia	#Sensors Kythnos	Expected size Crevillent (GB)	Expected size Terni (GB)	Expected size Mesogia (GB)	Expected size Kythnos (GB)
Meas. - Real-time	1 document per sensor	1000	10	24	200	200	200	200	1244.16	1244.16	1244.16	1244.16
Meas. – hourly	Hourly aggregations of measurements	1000	3600	24	200	200	200	200	3.456	3.456	3.456	3.456
Meas. - daily	Daily aggregations of measurements	1000	86400	24	200	200	200	200	0.144	0.144	0.144	0.144
								Total expected size (GB):	<u>1247.76</u>	<u>1247.76</u>	<u>1247.76</u>	<u>1247.76</u>

6.1.3 Big data requirements from WG CORP application

6.1.3.1 Short application description

WiseCORP is the WiseGRID technological solution targeting businesses, industries, ESCOs and public facility consumers and prosumers, with the objective of providing them the necessary mechanisms to become smarter energy players. By means of energy usage monitoring and analysis, proper information can be given to facility managers helping them to reduce energy costs and environmental impact.

A key factor towards achieving these objectives is a proper retrieval and analysis of energy usage data, and visualization of meaningful information extracted from it. This information may include:

- Detailed visualization of energy demand at different areas of the building, helping facility managers to identify opportunities for enhancing energy efficiency.
- Energy tariff comparison, enabling a direct economic cost reduction by shifting to a more adequate tariff.
- Energy demand forecast, enabling medium to long term cost estimations and supporting operative decisions about the usage of the facilities.
- Demand flexibility estimation, allowing the execution of optimization algorithms that will - either automatically or by providing advices - shift demand in order to minimize economic costs - by maximizing self-consumption or moving demand to off-peak periods - or minimize environmental impact - by shifting demand to periods where green energy is available.



Figure 26 – WiseCORP

6.1.3.2 Interface with Big Data platform

The big data platform will be used by the WiseCORP in order to hold the history of data provided by the different elements of interest of the application. As depicted in the architecture of the WiseCORP, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds).
- Advanced Metering Infrastructure systems from the DSO, retrieving data from already deployed

Smart Meters – usually hourly curves retrieved once a day.

- Storage systems.
- CHP devices, publishing their operation setpoint.
- HVAC devices, publishing their operation setpoint.
- Sensors for measuring indoor conditions, mainly temperature.

This data will flow from the source devices/systems, through the WG IOP Message Broker into the WiseCORP application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

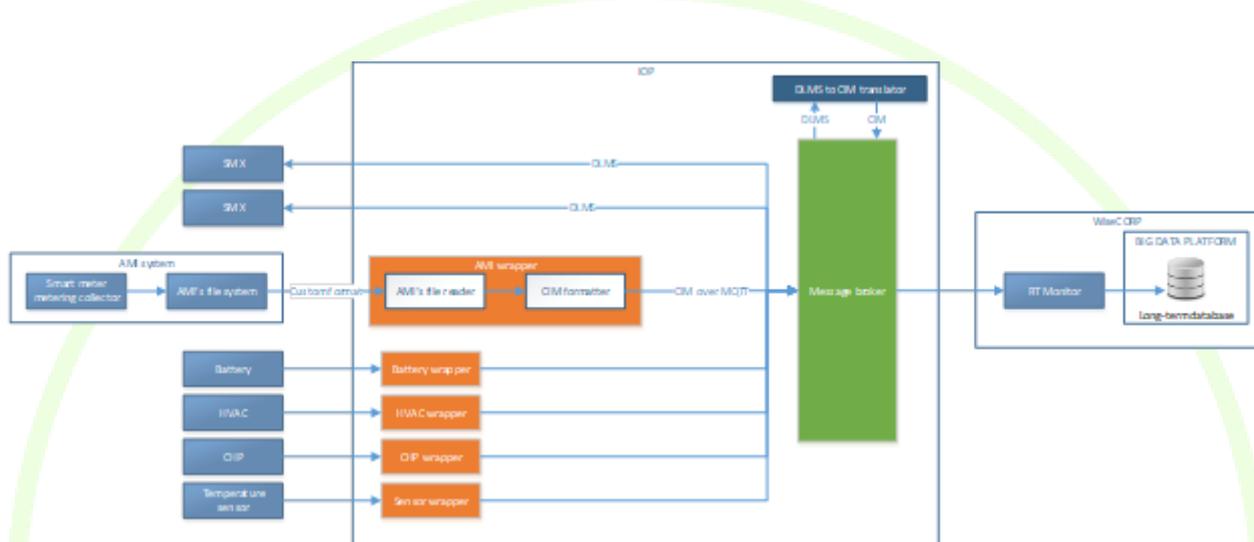


Figure 27 – WG CORP interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WiseCORP application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseCORP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, which will process the information to obtain:

- Hourly/Daily/Monthly aggregated data to produce summaries of interest to the ESCO or facility manager (demand, production).
- Analysis of the demand (according to source, tariff period, self-consumption capabilities)
- Analysis of the used capacity (histograms of active power).

These jobs appear in the architecture of the WiseCORP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

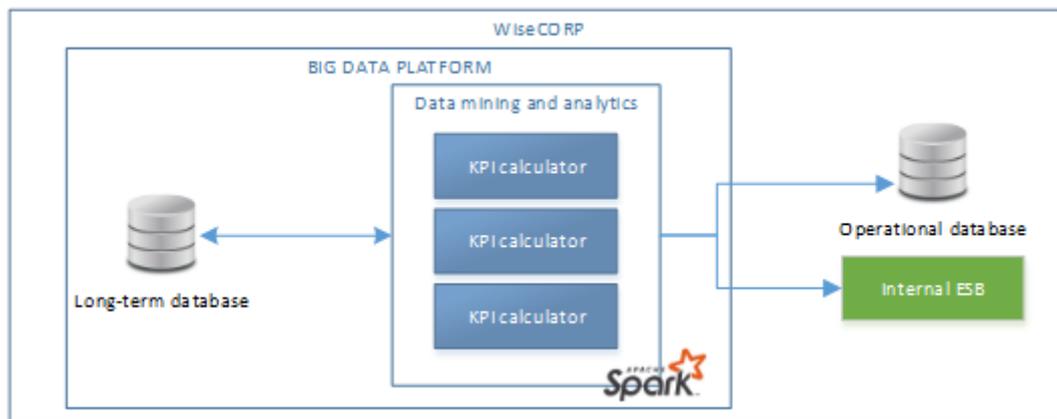


Figure 28 — WiseCORP data mining

6.1.4 Big data requirements from WG COOP application

6.1.4.1 Short application description

WiseCOOP is the WiseGRID technological solution targeting aggregators of consumers and prosumers - particularly focused on domestic and small businesses -, supporting them in their roles of energy retailers, local communities and cooperatives - which may have different objectives.

The main goal of the solution is helping consumers and prosumers to work together in order to achieve better energy deals while relieving them from administrative procedures and cumbersome research. In the particular scenario of increasing share of distributed renewable resources, this goal can be achieved by pursuing several objectives:

- Net-metering: supporting the operation of communities of prosumers that invest in renewable energy sources aiming at reducing their environmental impact
- Member profiling: clusters of consumers and prosumers with common energy usage patterns may be identified, allowing the aggregator to negotiate special terms (as for instance energy tariffs) particularly beneficial for those groups
- Demand forecasting: by allowing aggregator (in its retailer role) to forecast the demand of its customers, optimized purchase of energy at the wholesale market is enabled
- Tariff comparison: by offering members a tool for comparing their particular consumption with different available tariffs, those will have access to very valuable information to reduce their energy bills
- Implicit price-based demand response towards modulating the overall demand of the group to achieve a common objective (as, for instance, maximize usage of renewable energy sources produced within the group or minimize deviations between actual demand and energy purchased at the wholesale market)
- Providing clear information to members to raise awareness on efficient energy usage and environmental awareness



Figure 29 – WiseCOOP

6.1.4.2 Interface with Big Data platform

The big data platform will be used by the WiseCOOP mainly with the objective of storing the history of energy demand and production of the members of the portfolio aggregated by the organization using the application. As depicted in the architecture of the WiseCOOP, the main data sources for this information are:

- Unbundled Smart Meters, providing readings with high frequency (up to every 10 seconds).
- Advanced Metering Infrastructure systems from the DSO, retrieving data from already deployed Smart Meters – usually hourly curves retrieved once a day.

This data will flow from the source devices/systems, through the WG IOP Message Broker into the WiseCOOP application. The Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

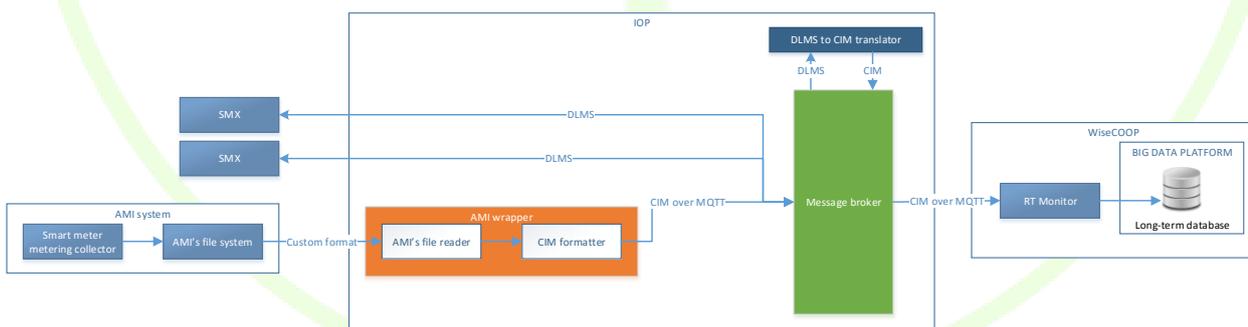


Figure 30 – WG WiseCOOP Big Data Platform interface

The RT monitor module is, therefore, the module which, inside the WiseCOOP application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseCOOP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, and will take advantage of the machine learning capabilities of this framework, in order to process the following required information:

- Aggregated data of interest to the manager of the aggregation of prosumers and to individual prosumers
 - Cumulative daily consumption in real-time
 - Cumulative weekly consumption in real-time
 - Cumulative monthly consumption in real-time
 - Cumulative yearly consumption in real-time
 - Cumulative energy consumption for the same calendar day of previous year
 - Cumulative weekly energy consumption for the same week of previous year until corresponding day
 - Cumulative energy consumption for the same month of previous year until corresponding day
 - Cumulative energy consumption during the same previous year until corresponding day
 - Cumulative energy consumption for the same day (24h) of previous year
 - Cumulative energy consumption for the whole week (7 full days) of previous year
 - Cumulative energy consumption for the full month of previous year
 - Cumulative energy consumption during the entire previous year
- Profiling (classification) of prosumer portfolio according to characteristics of the demand (time distribution, amount of demand, associated economic cost, associated environmental impact)
- Breakdown of portfolio demand/production according to contract type (domestic, tertiary)

These jobs appear in the architecture of the WiseCOOP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

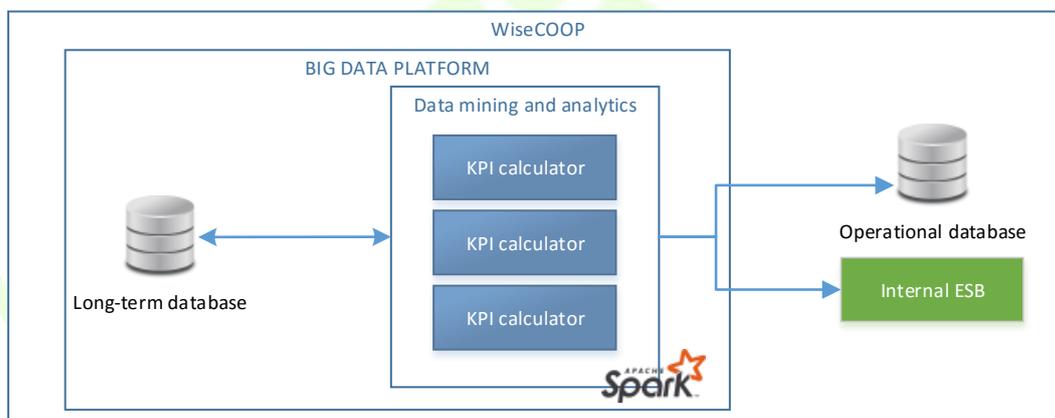


Figure 31 – WG WiseCOOP Big Data mining

6.1.5 Big data requirements from WiseHOME application

6.1.5.1 Short application description

WiseHome is an application used by the home users that are in the WiseCOOP framework structure in order to get information of the status of their participation in WiseCOOP. WiseHOME interacts with other application such as WiseCOOP and WG Staas/VPP through WG IOP sending data requests and generation simple and comprehensive graphic interface for the user accessible by a large type of devices as computers and mobile devices.

6.1.5.2 Interface with Big Data platform

WG Home does not interact directly to the Big Data platform and is not storing any data in the long term database from the Big Data Platform. WiseHOME is using a small local SQL database in order to process the user authentication data and data got from WG COOP and WG Staas/VPP. The interaction diagram of WiseHOME to the other WiseGRID application is defined in the bellow diagram:

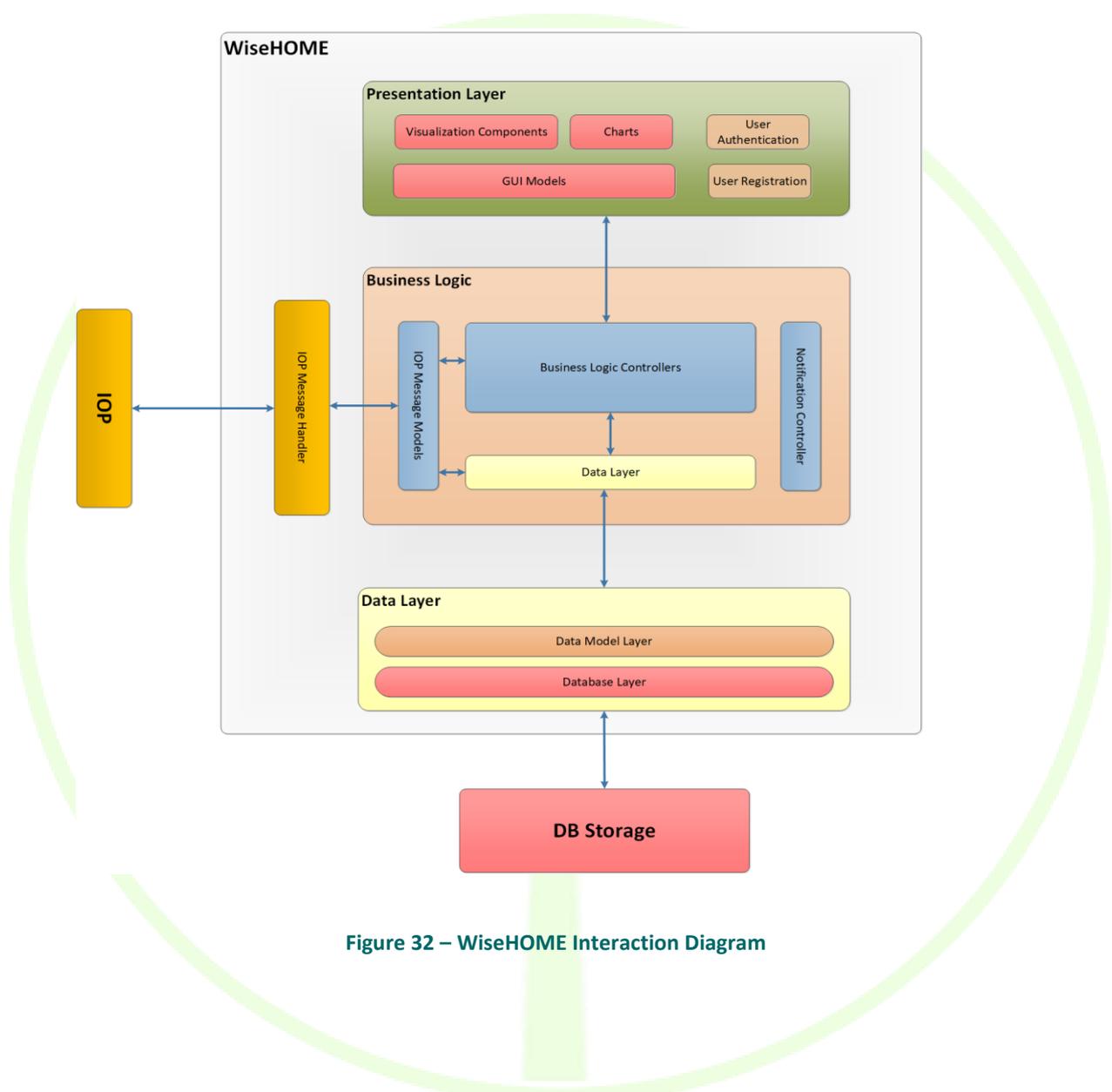


Figure 32 – WiseHOME Interaction Diagram

6.1.6 Big data requirements from WG EVP application

6.1.6.1 Short application description

WiseEVP is the WiseGRID technological solution for

- Vehicle-sharing companies or electric vehicle fleet managers and
- Electric vehicle infrastructure (EVSE) operators

In order to optimize the activities related with smart charging and discharging of the EVs including V2G (vehicle to grid, energy injection in the distribution network) and V2B (vehicle to building).

The management of the EVSEs charging and discharging processes will meet the following objectives:

- Reduce the EV charging energy bill.
- Follow flexibility requests from DSO to help the electric distribution network operation in exchange for an economic compensation.
- Follow flexibility requests to increase injection of RES production reducing curtailment in exchange for an economic consideration compensation.
- Contribute in the building energy management system with the main objectives of reducing the energy bill and maximize local RES production.

All the aforementioned objectives will be subordinated to the EV user preferences: desired state of charge (SOC) at the time of unplugging the EV.



Figure 33 – WG Wise EVP

6.1.6.2 Interface with Big Data platform

The big data platform will be used by the WiseEVP for storing the history of energy supplied or injected by the different EVSEs, and hold the history of the parameters read out from the vehicles of the fleet, which will be used for analyzing the usage of those and optimize the charging schemes. As depicted in the architecture of the WiseEVP, the main data sources for this information are:

- Electric Vehicle Supply Equipment (EVSEs), providing readings of energy supplied (or injected under V2G schemes) and the identification of the vehicle associated to these energy flows.
- Electric Vehicles (EVs), which will be monitored to regularly extract information to model their usage

(State of Charge, travelled distances, location, energy charged).

This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker into the WiseEVP application. Similarly to other applications, the Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform.

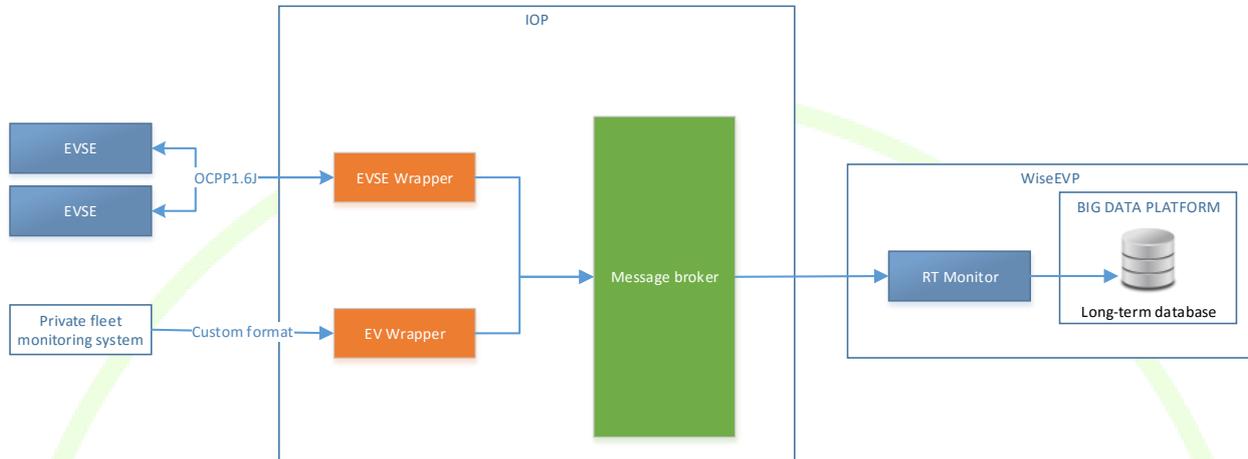


Figure 34 – WiseEVP interface with Big Data platform

The RT monitor module is, therefore, the module which, inside the WiseEVP application, implements the bridge between the live data flow started by the field devices and the big-data platform.

On the other hand, the WiseEVP will use the *Data mining and analytics* module of the big data platform to process the historic data in order to calculate KPIs. These calculations will be scheduled as regular Spark jobs to be executed by the big data platform, and will process the following required information:

- Analysis of energy supplied by EVSEs accordingly to their source
- Hourly/Daily/Monthly aggregations of energy demanded by the fleet
- Hourly/Daily/Monthly aggregations of energy supplied or injected back to the grid by the EVSEs
- Estimation of costs associated to energy demand (both economic and environmental impact)
- Comparison of costs with previous periods
- Analysis of battery health of the EVs of the fleet (trends in energy/distance component)

These jobs appear in the architecture of the WiseEVP as *KPI engine* module. Results of this calculation will be stored back to different collections of the long-term database, but may also be stored into the operational database (which holds information about the current status of the grid and a short-term window of data), or republished to the internal ESB with the objective of making certain KPIs accessible to other modules of the application.

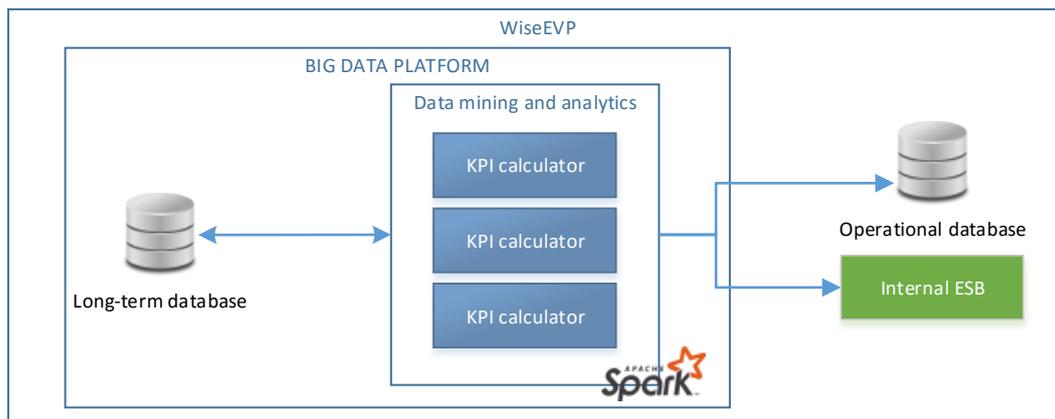


Figure 35 – WiseEVP Big Data mining

6.1.7 Big data requirements from WG FastV2G application

6.1.7.1 Interface with Big Data platform

FastV2G is the Charging Station for Electric Vehicles developed within the WiseGRID project with the capability of transferring energy stored in EVs battery to grid. This asset will be managed by the WiseEVP application, and will not make use of the big data platform.

6.1.8 Big data requirements from WG StaaS/VPP application

6.1.8.1 Short application description

WG StaaS/VPP (“Storage as a Service / Virtual Power Plant”) is WiseGrid’s solution to clustering distributed storage systems and making them usable for the grid in general and for DSOs in particular. This will be achieved by:

- enabling forecasting of the flexibility of the connected distributed batteries,
- enabling collective contribution to various markets (wholesale, ancillary services) with this flexibility, and
- enabling collective scheduling of distributed storage systems in order to fulfil market requests in an optimized way.

This is done by separating WG StaaS/VPP into several sub-components as sketched in Figure 36 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools.

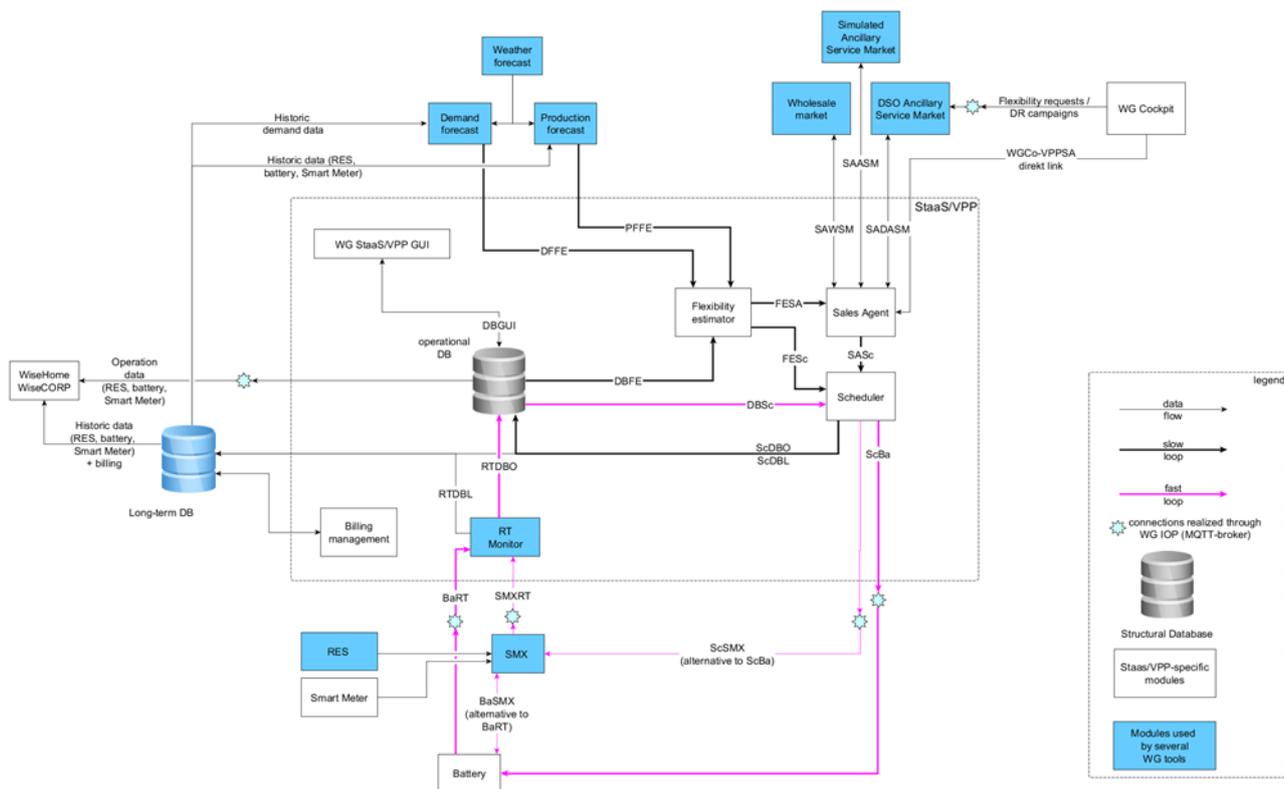


Figure 36 – Sketch of data flow between WG StaaS/VPP components and neighbouring WG tools

There are two main loops of data flow within WG StaaS/VPP:

Slow loop (executed every 15 minutes):

1. Flexibility Estimator determines remaining flexibility within the next 24 hours.
2. Sales Agent tries to sell this flexibility to the markets.
3. If some flexibility could be sold to the markets: Scheduler schedules fulfilling of required services by the batteries and communicates this schedule back to the Flexibility Estimator.
4. back to step 1.

Fast loop (executed every couple of seconds):

1. If services need to be served to the grid right now: Scheduler determines if, e.g., required active power is currently fulfilled. If not: Scheduler re-calculates power to be delivered by each individual battery.
2. Batteries try to follow Scheduler's requests.
3. back to step 1.

6.1.8.2 Interface with Big Data platform

WG StaaS/VPP uses two databases fulfilling different purposes:

- Operational DB: WG-internal DB that holds short-term data, contractual data and data intended to be passed from one sub-component, e.g., the "Scheduler" to another one, e.g., the "Flexibility Estimator".

- Long-term DB: Big data platform as described in this document to store data required for longer periods of time.

The following data are intended to be stored in the long-term DB:

- Monitoring data from the batteries as specified in **¡Error! No se encuentra el origen de la referencia.**
- Battery schedules as determined by the Scheduler sub-component: required metering point active power, required battery system reactive power, required battery system frequency control set-points, ...

Table 2 – List of WG StaaS/VPP monitoring data

Type	Variable	Explanation	Unit	Range
Double	Meter_Active_Power	at the grid connection point of the household as measured by the battery controller	W	[-Pmax, Pmax]
Double	Meter_Reactive_Power	at the grid connection point of the household as measured by the battery controller	VAr	[-Qmax, Qmax]
Double	Inverter_Active_Power	(AC) active power of the battery inverter	W	[-Pmax, Pmax]
Double	Inverter_Reactive_Power	of the battery inverter	VAr	[-Qmax, Qmax]
Double	Inverter_PV_Power	of the PV inverter included in a battery system, if it is such a “hybrid” system	W	[-Pmax, Pmax]
Double	External_PV	of a stand-alone PV inverter, which is somehow measured by or in communication with the battery controller	W	[-Pmax, Pmax]
Double	Inverter_Battery_Power	DC power of the battery inverter	W	[-Pmax, Pmax]
Double	Battery_SOC	usable energy presently in the battery divided by actually usable energy capacity	%	[0, 100]
Double	Battery_SOH	actually usable energy capacity divided by rated capacity	%	[0, 100]
Double	Battery_Voltage		V	[0, 500]
Double	Meter_Grid_Voltage	at the grid connection point of the household as measured by the battery controller	V	[0, 500]
Double	Meter_Grid_Frequency	at the grid connection point of the household as measured by the battery controller	Hz	[0, 500]
Array	Temperatures: Batt_Cell_Max_T, Batt_Cell_Min_T, Inverter_T	maximum temperature of the battery cells, minimum temperature of the battery cells, inverter IC temperature	°C	[-500, 500]

Double	Charge_Available	actual maximum available active charge power (AC) of the battery	W	[0, Pmax]
Double	Discharge_Available	actual maximum available active discharge power (AC) of the battery	W	[0, Pmax]
Int	Status	0: disconnected, 1: connected, 2: charge, 3: discharge, 4: standby, 5: error, 6: busy, 7: islanding	---	---
Array	Alarms	Error numbers	---	---
Double	Inverter_PV_Voltage		V	[0, 1000]
Int	Working mode	as defined in sheet 'Operation'	---	---
Double	Demand	power consumed at the house	W	[-Pmax, Pmax]

Based on these datasets, the “Billing management” sub-component ensures proper remuneration of contribution battery systems by retrieving back historical data from the long-term DB, e.g. active powers actually delivered by individual batteries and other individual contributions to the overall VPP-services. Additionally, the forecast modules can generate production and demand forecasts based on these historical data, which are required to estimate the battery system flexibilities within the Flexibility Estimator module.

6.1.9 Big data requirements from WG RESCO application (ENG)

6.1.9.1 Short application description

The WG RESCO will be a tool conceived for RESCOs - Renewable Energy Service Companies and ESCOs that want to provide RES services to end-users (households or businesses) that do not own nor wish to maintain the necessary equipment. In this perspective, three potential scenarios are envisaged:

- RESCO pays a fee to end-users using their premises (e.g. for installing PVs on their roof), installs and maintains the RES assets and markets all produced energy;
- RESCO provides to customers the supply of energy coming from RES owned by the RESCO (i.e. allowing self-consumption) and markets the production surplus;
- RESCO provides to customers the installation of RES equipment (e.g. PV panels) which are owned and maintained by the RESCO but fully exploited by the end customers (renting business model).

According to that, the WG RESCO tool will support RESCOs in managing the relationship with their customers and the provision of energy to the consumers from renewable energy sources, usually PV, wind power or micro hydro. Since the generation equipment will be owned, serviced and operated by the RESCO itself, the WG RESCO will have a central feature in supporting the maintenance management of those assets



Figure 37 – WG RESCO

WiseGRID RESCO Tool was designed and developed based on the previously described perspective and understanding for the business model of a RESCO. Fundamentally, this tool needs to provide all the necessary functionalities that support and facilitate the activities of this business actor. The main functionalities can be summarized as follow:

1. Performs all the necessary functionalities required for the business operations like:
 - Manage contracts with customers
 - Manage portfolio of installed assets
 - Manage assets maintenance
 - Manage energy flows
 - Manage economic flows
 - RES production and consumption forecast
 - Market participation
 - potential provision of services to DSO.
2. Supports the Interoperability with external tools
3. Incorporates Functional User Interface

6.1.9.2 Interface with Big Data platform

The WG RESCO tool has been developed using the MEAN.JS full-stack JavaScript solution, this choice has been made to build fast, robust, and maintainable production web applications using MongoDB which is the one adopted for the Big Data platform, AngularJS and adopting Node.js as engine.

Currently the RESCO tool store its data in two different data base provided by the Big Data Platform.

The first one the “Local DB” for the Customer data, Contract data, Asset data, Configuration data and real time meter observation coming from RES source and stored by the RT Monitor tool.

The second one the “Long term DB” to store historical meter observation coming from RES source. This data will flow from the source devices/systems, through the WiseGRID IOP Message Broker platform into the RESCO tool application. Similarly to other applications, the Real-Time Monitor module is in charge of subscribing to the corresponding flows of data and store it to the long-term database, hosted by the big data platform. RESCO tool can also have direct access to the long-term DB to store data needed for future analysis and use specific DB interaction services to divide the computational load.

WG Resco tool access directly to Big Data platform via java script and use Mongoose that is an Object Data Modelling (ODM) library for MongoDB and Node.js. It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

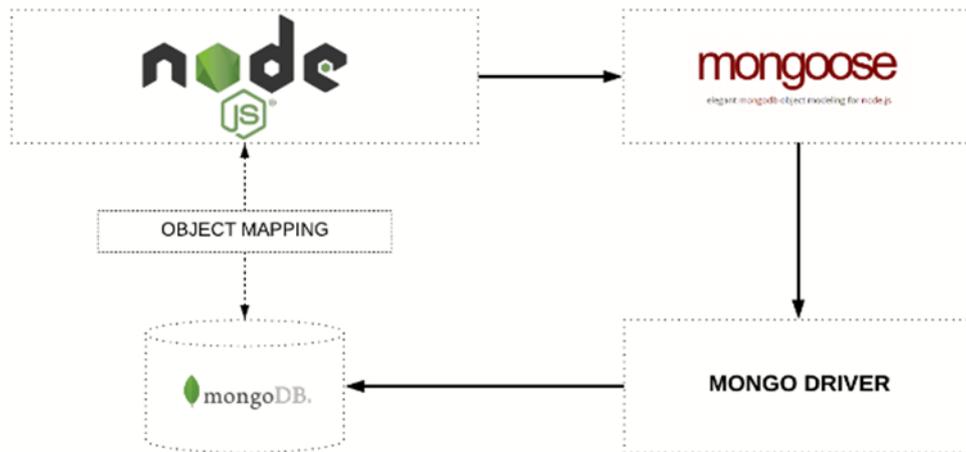


Figure 38 – Object Mapping between Node and MongoDB managed via Mongoose

The next image shows a high level architecture about the interface between the WG RESCO tool and the Big Data Platform. As shown, the WG RESCO tool can read and write directly into the Local DB and Long term DB by sending a request to the DB interaction services.. The DB interaction service read the request, perform an action into the Long Term DB and send the response via WG IOP to the WG RESCO tool that consume it.

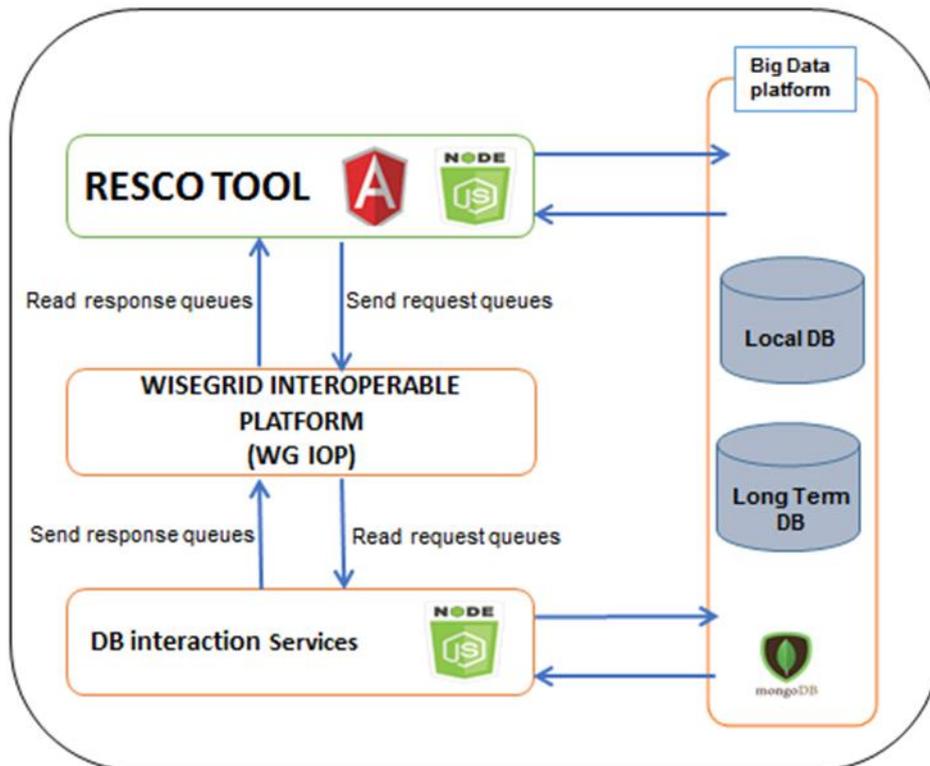


Figure 39 – WG RESCO Tool interface with Big Data Platform

7 BIG DATA PROCESSING MODULES

Big data is not only storing huge amount of data and retrieving it, but also processing stored data in order to obtain new information, trends, and insights.

There are two type of processes:

- **Data mining**
- **Data analytics**

7.1 DATA MINING CONCEPT

Data mining is a process to structure the raw data and formulate or recognise the various patterns in the data through the mathematical and computational algorithms, data mining helps to generate new information and unlock the various insights. The data is first placed into a data warehouse to do the required extraction of data to produce meaningful relationships and patterns. There is two type of data mining. One is descriptive, which gives information about existing data of the organisation, while the other is predictive: which makes forecasts based on the data.

Data mining is a pattern discovery task against a pool of data; therefore it requires classical and advanced components of artificial intelligence, pattern distribution and traditional statistics. The point to be noted is that data mining is done without any preconceived hypothesis, hence the information that comes from the data is not to answer specific questions of the organisation.

Data mining also helps in exploring trends from the data.

7.2 DATA ANALYTICS CONCEPT

Data analytics is the art of exploring the facts from the data with specific to answer specific questions, i.e. there is a test hypothesis framework for data analytics. The techniques used in analytics also are same as used in business analytics & business intelligence.

One needs various tools to get right data analytics, like data visualization tool and need to know languages like Python or R to perform robust data analytics.

To sum up, it is possible to see that the data analytics has its rooted from business analytics or business intelligence models while data mining uses more of scientific and mathematical techniques to come up with patterns and trends.

Data mining is basically close to machine learning.

7.3 DATA MINING AND ANALYTICS PLATFORMS

For Data mining and data analytics an important need of computation power should be considered. Large amount of data has to be processed so the answer for this request is also a computer cluster. As indicated in 5.2.2 there are two types of computer cluster:

- Load balancing cluster
- High performance cluster

The database cluster is mainly a load balancing cluster but now, for data mining and analytics the type of cluster to be used is a high performance cluster. This type of cluster is sharing the resources of the computers that are composing the cluster so the applications will run on a high performance machine with higher resources than each of the components of the cluster.

For reaching this goal special software will run on each computer in the cluster enabling the distributed processing and resources sharing.

For data mining and analytics Apache Software Foundation is developing several projects of open source software.

Apache Software Foundation is declaring through his Research Vice President Mark Driver:

"The Apache Software Foundation is a cornerstone of the modern Open Source software ecosystem – supporting some of the most widely used and important software solutions powering today's Internet economy." – Mark Driver, Research Vice President, Gartner [15]

The best known project of Apache is the Open source Http server but Apache is constantly developing over 200 projects also including distributed computing

7.3.1 Apache™ Hadoop® Project

The project description that is on web page declares:

The **Apache™ Hadoop®** project develops open-source software for reliable, scalable, distributed computing.

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures [16].

The Hadoop project is specially developed for providing tools for distributed computing.



Figure 40 – Apache Hadoop Logo

The project includes the following modules:

- **Hadoop Common:** contains a set of utilities used for supporting all other modules.
- **Hadoop Distributed File System (HDFS™):** To be able to process big amount of data in computer cluster it is necessary to have also a distributed file system on all nodes in computer cluster that will become able to provide high-throughput access to applications data.
Hadoop YARN: provides a framework for scheduling the jobs of a distributed application on the nodes of computer cluster based on nodes resources management.
- **Hadoop MapReduce:** a set of tools based on YARN for parallel processing on large datasets provided by Big data engines.

One of the main resource on cluster computing is the file system. For providing a file system to the nodes of the cluster the file system has to be shared between the nodes of the cluster thus being necessary to use a distributed file system.

The developers are defining the Hadoop distributed file system in the following manner:

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from

other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is part of the Apache Hadoop Core project.

[16]

The main Goals of Apache Hadoop is to manage in a performant way the following concepts:

Hardware Failure

Considering a cluster with hundreds or maybe thousands of nodes running an instance of HDFS, it is necessary to consider a quite high probability of failure of some nodes. Detection of faults and the possibility of automatic recovery from the faults in a quick way is one of the goals of the Hadoop HDFS architecture.

Streaming data access

The applications designed to run on a Big Data datasets needs streaming access. This applications are designed more for batch processing then interactive uses. High throughput of data access is a goal of Hadoop.

Large data sets

Applications designed to run on HDFS will use large data sets. In HDFS a file can be from gigabytes to terabytes of size. HDFS is tuned to support large files.

Simple Coherency Model

When using HDFS in Big Data Processing applications in the most of situations the access is Write-once-read-many model. The files are created, written and they do no need to be changed. The only changing methods are appends and truncates. This assumption simplifies data coherency issues and enables high throughput data access. A MapReduce application or a web crawler application fits perfectly with this model.

“Moving Computation is Cheaper than Moving Data”

When a data set is huge the computation needed by the application has to be near the data it operates on. Moving huge datasets can generate the network congestion so the assumption is that better to migrate the computation closer to where the data is located rather than moving the data to where the application is running.

Portability Across Heterogeneous Hardware and Software Platforms

The nodes of a big cluster can have a big degree of heterogeneity as in hardware also in running software so the HDFS should be easily portable form a platform to another

7.3.2 Hadoop compatibility with MongoDB

MongoDB and Hadoop are a powerful combination and can be used together to deliver complex analytics and data processing for data stored in MongoDB. For enabling this combination MongoDB developers produced a MongoDB Connector for Hadoop. In order to become familiar with the connector, the user can use it to pull MongoDB data into Hadoop Map-Reduce jobs, process the data and return results back to a MongoDB collection.

This combination requires a MongoDB cluster running the database management system and a separate cluster containing multiples nodes running Apache Hadoop.

For more information about using the MongoDB Hadoop connector is a Wiki connected on a GitHub development page [17] :

This page describes options that can be set on the Java Virtual Machine with -D to configure the MongoDB Hadoop Connector [18].

7.3.3 Hadoop Cluster setup

The installation process of a Hadoop instance is described extensively in Apache Hadoop documentation [19]

Installing a Hadoop cluster typically involves unpacking the software on all the machines in the cluster or installing it via a packaging system as appropriate for each operating system. It is important to divide up the hardware into functions.

Typically one machine in the cluster is designated as the NameNode and another machine as the ResourceManager, exclusively. These are the masters. Other services (such as Web App Proxy Server and MapReduce Job History server) are usually run either on dedicated hardware or on shared infrastructure, depending upon the load.

The rest of the machines in the cluster act as both DataNode and NodeManager. These are the workers [19].

A typical structure of a Hadoop Cluster is described in next figure:

Hadoop Cluster

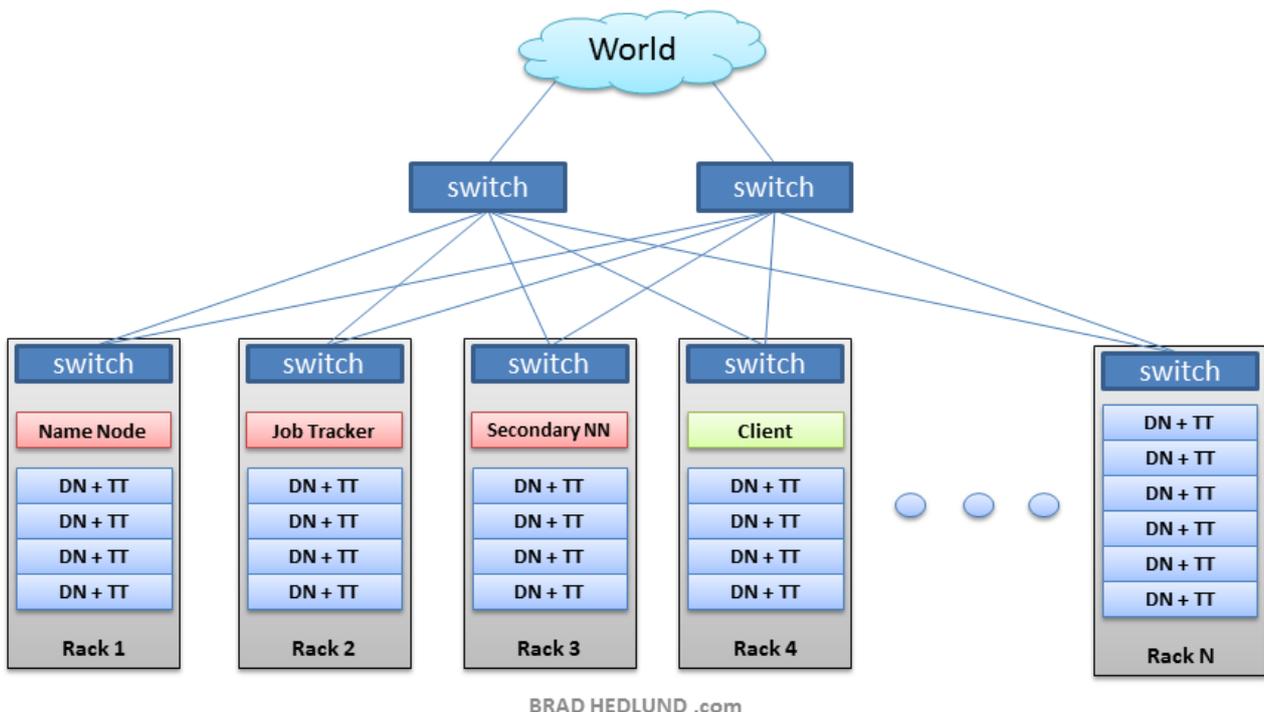


Figure 41 – Hadoop Cluster structure

The cluster nodes in a Hadoop cluster are separated in computer racks each with a switch in order to optimize the data transfer. The rack switches are managing the local data traffic and each rack is connected with two switches providing the client data traffic to the cloud clients

7.3.4 Apache Spark™ engine for large scale data processing

Apache is developing other tools that enables big datasets to be processed in a distributed processing cluster

Apache Spark™ is a fast and general engine for large-scale data processing. Apache Spark is providing a set of tool for applications written in Java, Scala, Python and R for building parallel processing apps.

Apache Spark is providing more of 80 High level operators.

Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. All these libraries can be combined seamlessly in the same application [20].



Figure 42 – Apache Spark Logo

7.4 ARCHITECTURE FOR WISEGRID OFFLINE BIG DATA PROCESSING

For providing the data mining and analytics service to the client applications developed in WiseGRID, a Hadoop cluster should be implemented. The Hadoop cluster will have a structure similar to the structure described in chapter Hadoop Cluster setup in this cluster will provide the Hadoop distributed file system and, on the cluster, will run applications designed by the application developers in WiseGRID written using Apache Spark operators.

8 CONCLUSIONS AND NEXT STEPS

8.1 CONCLUSIONS

This deliverable defined an infrastructure for implementing the Big Data services both online and offline. For providing the Big Data services needed in WiseGRID project are needed two horizontally scalable computer clusters.

1. The first cluster is providing online Big Data services as long term data storage and retrieving in separate databases for each applications.
 - A limited access database for each application containing data that are not under the restrictions of GDPR. The access will be granted by a user/password pair contained in each data base.
 - A limited access database for each application that is storing personal data, under the restrictions of GDPS. The access will be granted by user/password pair and also will be limited by the network identity of the client by the firewall installed before the application router
 - Each application will access the database through a dedicated application router installed on a separate cluster node.

The first cluster is running MongoDB NoSQL database management system.

2. The second cluster hardware separated from the database cluster is providing the offline services. This cluster is based by Apache Hadoop framework and will provide the remote processing of applications based on Apache Spark operators. The second cluster is connected to the MongoDB database cluster through the Hadoop MongoDB connector. Each application will have a separate account for each WiseGRID application.

Table 3 – Deliverable objectives

Objective	Achieved within WP
Defining the hardware infrastructure of a Big Data computer cluster providing cloud services to WiseGRID applications for online services like long term data storage and retrieving	WP5.1 WP5.2
Defining the hardware infrastructure of a Big Data computer cluster providing cloud services to WiseGRID applications for offline services line data mining and analytics.	WP5.1 WP 5.2
Defining the software structure for Big Data online services on MongoDB database management system	WP5.1 WP5.2
Defining the software structure for Big Data offline services using Apache Hadoop framework and Apache Spark distributed processing operators	WP5.1 WP5.2

8.2 NEXT STEPS TO BE IMPLEMENTED

The next step is to implement small scale demonstrator of both computer cluster, MongoDB computer cluster and Apache Hadoop computer cluster, scaled to the needs of Demo sites. This small scale demonstrator will be duplicated for each Demo Site providing the services for demo applications in each site. Based on the data collected from each demo site the computer clusters will be defined from the point of view of required resources.

9 REFERENCES AND ACRONYMS

9.1 REFERENCES

- [1] D. Laney, "3D Data Management: Controlling Data Volume, Velocity, and Variety," 2001.
- [2] MongoDB, "Big Data explained," [Online]. Available: <https://www.mongodb.com/big-data-explained>.
- [3] Cnet www.cnet.com, "Cnet- Facebook data processing," [Online]. Available: <https://www.cnet.com/news/facebook-processes-more-than-500-tb-of-data-daily/>.
- [4] VCloud news, "Vcloudnews - statistics of data produced," [Online]. Available: <http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion/>.
- [5] Key Web Metrics , "How Big Data impacts lives everyday - Infographic," [Online]. Available: <http://www.keywebmetrics.com/2015/04/how-big-data-impacts-lives-everyday-infographic/>.
- [6] B. Marr, "Big Data the 5 Vs everyone must know," [Online]. Available: <https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know/>.
- [7] Wikipedia, "Wikipedia BigData Definitions," [Online]. Available: https://en.wikipedia.org/wiki/Big_data.
- [8] MongoDB , "MongoDB Manual," [Online]. Available: <https://docs.mongodb.com/manual/>.
- [9] MongoDB, "MongoDB forGiant Ideas," [Online]. Available: <https://www.mongodb.com/>.
- [10] Tutorials point, "MongoDB Quick Guide," [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_quick_guide.htm.
- [11] Lawrence Livermore National laboratory, "Linux Cluster overview," [Online]. Available: https://computing.llnl.gov/tutorials/linux_clusters/.
- [12] Webmin site , "Webmin," [Online]. Available: <http://www.webmin.com/index.html>.
- [13] [Online]. Available: <https://www.mongodb.com/products/compass>.
- [14] MongoDB Compass, "MongoDB Compass," [Online]. Available: <https://www.mongodb.com/products/compass>.
- [15] Apache Software Foundation , "Apache Software foundation," Apache, [Online]. Available: <https://www.apache.org/>.
- [16] Apache , "Apache Hadoop," [Online]. Available: <http://hadoop.apache.org/>.
- [17] Mongo Hadoop Connector Wiki, "Mongo Hadoop Connector Wiki," [Online]. Available: <https://github.com/mongodb/mongo-hadoop/wiki>.
- [18] MongoDB, "Wiki of MongoDB Hadoop Connector," [Online]. Available: <https://github.com/mongodb/mongo-hadoop/wiki>.
- [19] Apache Hadoop, "Apache Hadoop Docs, Cluster setup," [Online]. Available: <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html>.

[20] Apache Spark , “Apache Spark™,” [Online]. Available: <https://spark.apache.org/>.

9.2 ACRONYMS

Table 4 – List of Acronyms

Acronyms List	
BSON	Binary JavaScript Object Notation
CHP	Combined Heat and Power
DB	Data base
DSO	Distribution service operator
ESB	Enterprise Service Bus
ESCO	Energy service company
EV	Electrical vehicle
EVSE	Electrical vehicle service equipment
HV	High Voltage
HVAC	Heat, Ventilation and Air Conditioning
IOP	Interoperability platform
JSON	JavaScript Object Notation
KPI	Key performance indicators
RDBMS	Relational Data Base Management system
RESCO	Renewable energy service company
RT	Real Time
SCADA	Supervisory Control And Data Acquisition
SoC	State of Charge
StaaS	Storage as a service
TM	Technological Manager
VPP	Virtual power plant