# Big Data Techniques For Supporting Accurate Predictions of Energy Production From Renewable Sources

Michelangelo Ceci
UNIBA
Via Orabona, 4, Bari Italy
michelangelo.ceci@uniba.it

Roberto Corizzo
UNIBA
Via Orabona, 4, Bari Italy
roberto.corizzo@uniba.it

Fabio Fumarola
UNIBA
Via Orabona, 4, Bari Italy
fabio.fumarola@uniba.it

Michele Ianni
UNICAL
Via P. Bucci, Rende, Italy
michele.ianni@unical.it

Donato Malerba
UNIBA
Via Orabona, 4, Bari Italy
donato.malerba@uniba.it

Gaspare Maria
GFM-Integration
Catania, Italy
gaspare.maria@gfmnet.it

Elio Masciari
ICAR-CNR
Via P. Bucci, Rende, Italy
masciari@icar.cnr.it

Marco Oliverio
ICAR-CNR
Via P. Bucci, Rende, Italy
oliverio@icar.cnr.it

Aleksandra Rashkovska
UNIBA
Via Orabona, 4, Bari Italy
aleksandra.rashkovska@uniba.it

## ABSTRACT

Predicting the output power of renewable energy production plants distributed on a wide territory is a really valuable goal, both for marketing and energy management purposes. *Vi-POC* (Virtual Power Operating Center) project aims at designing and implementing a prototype which is able to achieve this goal. Due to the heterogeneity and the high volume of data, it is necessary to exploit suitable Big Data analysis techniques in order to perform a quick and secure access to data that cannot be obtained with traditional approaches for data management. In this paper, we describe Vi-POC – a distributed system for storing huge amounts of data, gathered from energy production plants and weather prediction services. We use HBase over Hadoop framework on a cluster of commodity servers in order to provide a system that can be used as a basis for running machine learning algorithms. Indeed, we perform one-day ahead forecast of PV energy production based on Artificial Neural Networks in two learning settings, that is, structured and non-structured output prediction. Preliminary experimental results confirm the validity of the approach, also when compared with a baseline approach.

## 1. INTRODUCTION

Recently, renewable energy research is gathering a lot of attention due to the strategic and urgent need of reducing pollution emission and finding new revenue streams for utility companies. Indeed, wholesale vendors lead the utility sector because those companies provide energy to the most relevant share of private and industrial users. Due to their dominant position, they are able to gather huge amount of valuable information. In particular, they have access to both external and internal data, including sensor data from producing assets, real-time or end-of-day price data from a multitude of related markets, counterparty credit data, position management information, and many others. However, due to the availability of new (low cost) technologies, also small producers are able to collect data about their business. Indeed, data coming from small production plants are quite heterogeneous, they arrive continuously (fast) and their volume increases at an unprecedented growth rate. These features pose several challenges that can be solved using Big Data techniques [1, 2, 3, 5, 25, 26, 28, 29, 24].

In a sense, this abundance of data calls for re-thinking traditional techniques to effectively store and efficiently analyze Big Data. Moreover, these challenges are crucial for achieving several business objectives, such as reducing enterprise risk and shortening decision response times, thus enabling traders and decision makers to quickly react to sudden changes of market quotations. Furthermore, Big Data techniques can help management staff to maximize company returns both in the short and long time horizon.

Unfortunately, achieving these goals in a strategic market as the energy one can be quite difficult due to the inherent complexity and variety of systems that should be integrated. Indeed, as a general consideration, we can observe that a complete data storage will slow the analysis tasks, on the contrary discarding some information will cause an incomplete analysis to be performed.

In this perspective, Vi-POC project has been developed in order to support renewable energy providers with a framework for collecting, storing, analyzing, querying and retrieving data coming from heterogeneous renewable energy production plants (such as photovoltaic, wind, geothermal, Sterling engine, water running) distributed on a wide territory. Moreover, Vi-POC features an innovative system for real-time prediction of the energy production that integrates data coming from production plants and weather production services. Indeed, a key problem for low-business energy producers is the exact quantification of the amount of energy that can be pushed over the power supply network. This problem arises as energy cannot be stocked efficiently; thus, they are forced to give it for free if they produce more energy than the network can use or they will pay a huge penalty if they do not provide the expected

amount.

In this paper, we describe our end-to-end framework that, starting from the data cleaning step, allows a better engineering of data structures in order to support data analysis and prediction of energy production, thus offering a good trade-off between effective storage and efficient analysis of data. As for the prediction of the energy production, we propose a method for long-term forecast (one-day ahead) of photovoltaic energy production based on Artificial Neural Networks (ANN) and investigate the performance in two settings - structured and non-structured output prediction. While in non-structured output prediction a prediction model generates the forecast for a specific hour in the future, in structured output prediction, a prediction model generates the forecast for 24 hours in the future [23][7]. In principle, the main advantage of structured output prediction consists in the implicit consideration of the dependence of the predictions at two consecutive hours.

In [13, 12], we described the high level architecture of our system. In this paper, we will describe the actual implementation of the prototype along with the implementation issues that are crucial for building a system for Big Data management and analysis.

## 1.1 Plan of the paper

In Section 2, we discuss related work on Big Data systems and the main issues described in literature for dealing with Big Data in real life scenarios. In Section 3, we describe the building blocks of a Big Data systems that we exploited for our prototype. In Section 4, we describe our system for renewable energy production plant output prediction and we discuss experimental results on real data gathered by production sites located in south of Italy. Finally, in Section 5 we draw our conclusion.

# 2. RELATED WORKS

Nowadays, Big Data systems are key components for addressing practical problems in several application areas. Thus, a comprehensive study of the data features (volume, arrival rate and information types) is the preliminary step for building effective Big Data systems as we will discuss in detail in next sections. As an example, it is widely recognized that Big Data can help in prevention, preparedness, response, and recovery of natural disasters [20]. Moreover, new technologies help biologist (especially the ones working on genome sequencing) to produce massive quantity of data that have to be handled by Big Data systems in order to perform fruitful analysis. In [38], a survey of Big Data systems for genomics developed upon the Hadoop Framework is reported. In [18], several studies about the use of Big Data for e-health are discussed. To the best of our knowledge, there is no description in literature of a Big Data system devoted to renewable energy production plants analysis. However, main issues and challenges posed by such a system are instead widely discussed by big players on the market searching for a satisfactory solution. Indeed, the required skills for setting up such a system, are interdisciplinary, spanning from deep domain knowledge of renewable energy production patterns, to low level software performance analysis. Furthermore, in a Big Data system, data undergo to many transformation before they can lead to knowledge discovery. Each of these transformation usually should be performed by a specific tool. Thus, building an end-to-end Big Data system requires the knowledge of a plethora of different tools and a way to make them interact with each other. Indeed, it is important to exploit a wide range of different tools in order to provide the most suited tool for each analysis task. In [4], good design principles are shown for building high quality Big Data end-to-end systems. These design rules take into account the above consideration and summarize the key requirements for a Big Data system. These design rules are listed below:

1. Support for a variety of analysis tools;
2. Use appropriate tool in every stage of the data manipulation pipeline;
3. Make data accessible trough using opensource tools and web-api services for retrieving data.

In [17], two additional design principles are added to the ones discussed above:

1. *Timely analysis*: as Big Data arise in a streaming way, at high velocity and high variety, the system should be able to analyze new data in near real time without being overwhelmed;
2. *Ensure Security in Big Data*: even if security represents an essential feature of enterprise information systems, tools exploited in Big Data system are not usually shipped with security out of the box.

On the contrary, in [19], the focus is posed on building a Big Data system based on Hadoop, which will auto-tune itself for getting better performances trough the overall data life-cycle.

Finally, as noted in [2, 5], the availability of a system that can achieve good performance can be really valuable in context where there is a lack of technical skills regarding system's internals and where a system is used on pay-as-you-go infrastructure.

# 3. BACKGROUND

Nowadays, dealing with a big volume of data is very challenging, since traditional technologies, like RDBMS or classical object oriented programming, are not well suited for this purpose. In this section, we will discuss some issues related to Big Data management which heavily affect the design of every system tailored for dealing with huge amounts of data. As Big Data also arise at high speed and variety, we need to cope with all these features. In this respect, *Scalability* is a crucial issue to be addressed.

Scalability refers to the ability of a system to handle a growing amount of information in an efficient and effective way. Obviously, the system must be able to provide a proper data storage as new data are available [11]. First of all, we must distinguish between horizontal and vertical scalability. More in detail, vertical scaling (or *scaling up*) refers to the upgrade of a system obtained by adding hardware and/or other computing facilities to a single machine. Horizontal scaling (or *scaling out*), instead, is performed by distributing the computational work load across several machines. Both solutions offer some advantages and suffer some drawbacks, thus, the choice of the right solution is influenced by the application scenario being analyzed. Indeed, in order to deal with the huge workload required to process Big Data calls for a highly scalable system design. Most systems developed for Big Data analysis exploits horizontal scaling, as there is virtually no limit in process scaling and it is less expensive than vertical scaling.

From an architectural point of view, distributed computing systems are composed of many layers. This choice allows a great flexibility

as different technologies can be exploited for task execution within each layer. The choice of the most suitable technology for each layer depends on the kind of application and the amount of data that the system will manage.

A further property of distributed systems that must be guaranteed is *fault-tolerance*. The latter is defined as the ability of a system to properly work despite the malfunction of some of its components. This property is crucial especially for those distributed systems composed of several computers located in different (geographical) places. Indeed, the fact that a system can work despite the crash of one of the machines in the computing pool is mandatory in order to avoid overall system malfunction that can cause data loss or wrong analysis results. Developing such distributed systems is a challenging task for system designers, even though the availability of suitable software modules (e.g. Message Passing Interface (MPI)) providing useful abstractions for communication across machines. Indeed, machine coordination and resource sharing among nodes, which play an essential role in Big Data systems, are very hard to be developed properly. Finally, the implementation of data analysis algorithms applied to big energy production datasets is difficult, since the most of these algorithms are not designed for distributed computing.

Many open source technologies were developed in order to effectively handle massive amounts of data. The majority of these technologies are based on the MapReduce programming model. This paradigm makes easier to implement solutions based on the use of distributed systems for executing analysis tasks. The MapReduce framework is based on the following steps:

- *Map*: Each node executes the *map* function on its local data, creating a set of pairs $\langle key, value \rangle$, and stores the results in a temporary storage.
- *Shuffle*: Pairs $\langle key, value \rangle$ are redistributed among nodes, in such a way that all the pairs with the same key are assigned to the same node.
- *Reduce*: Each node processes its group of pairs, independently of other nodes.

It is worth noticing that, since each mapping operation does not depend on the others, mapping operations can be parallelized. In a similar way, also the reduce step can be performed by multiple nodes at the same time, if the reduction function is associative.

The most widespread implementation of the MapReduce programming model is Hadoop MapReduce, part of the Hadoop framework [35]. Although Hadoop is a really pervasive technology, it has some drawbacks, especially when dealing with algorithms based on iterative operations. The latter limitation occurs because Hadoop MapReduce stores the results of intermediate computations on secondary storage, thus the overhead to launch each job, is very high. Indeed, MapReduce is well suited for large distributed data processing where fast performance is not an issue. Its high-latency batch model, instead, is not effective for fast computations or real data analysis.

Latest advances in Big Data processing lead to new technologies to overcome the shortcomings of Hadoop Map Reduce implementation. One the most successful opensource framework in this new generation of big data processing tools is Apache Spark [37], optimized for low-latency tasks. Spark caches data sets in memory and has a very low overhead in launching distributed computations. As stated in Spark website, Spark can "run programs up to 100 times faster than Hadoop MapReduce in memory, or 10 times faster on disk." In multi-step jobs, moreover, Hadoop MapReduce blocks each job from beginning until all the preceding jobs halt. This can lead to long computation times, even with small data sets. There are other ways to schedule tasks, one of which is *Directed Acyclic Graphs (DAG)*. A graph is used, where the vertex represent the jobs and the edges specify the order of execution of the jobs themselves. Since the graph is acyclic, independent nodes can run in parallel, resulting in a much lower overhead compared to the traditional MapReduce. Spark offers capabilities for building highly interactive, real-time computing systems using DAGs and so is very suitable for implementing applications that require a high level of parallelism. Spark is built against Hadoop in order to access *HDFS (Hadoop Distributed Filesystem)*.

A cluster is made up of two types of processes: (i) a *driver program*, which most of the times runs on a master node, and (ii) *executors*, which run on worker nodes and execute the tasks specified by the driver. The key concept beyond Spark is called *Resilient Distributed Dataset (RDD)* [36]. An RDD is a read-only, partitioned collection of records. Data are partitioned across many nodes in the cluster. Fault tolerance techniques are used to avoid data loss due to node failures. Given an RDD, we can manipulate the distributed data through operations called *transformations* and *actions*. Transformations consist in the creation of new data set from an existing one, and actions in running a computation on the data set and returning the results to the driver program. For instance, implementing MapReduce paradigm in Spark Framework is done using flatMap transformation and reduceByKey action.

As mentioned above, traditional RBDMS are not suitable for the typical size and scalability requirements of Big Data. In order to meet these requirements, column oriented databases have been proposed. In our project, we exploited HBase whose features will be described in the next section.

To summarize the above discussion, a system dealing with big data should implement the technologies reported below:

- A distributed file-system with replication;
- A distributed resource manager;
- A Map Reduce Framework implementation;
- A distributed database storage system;
- A layer of visualization tools.

## 3.1 Column Oriented DBMS

Using the row oriented approach, the data storage layer contains records (i.e. rows), while in a column oriented system it contains families of rows (i.e. columns). The widespread use of the relational approach is mainly due to its flexibility and sound theoretical foundation. Moreover, RDBMS users are able to access and manipulate data without being involved in any technical aspects concerning data storage and access. This is a simple model but not particularly suitable for data analysis. Indeed, row-oriented

databases are not adequate to deal with complex analysis of massive datasets because they are designed for transactional processing and is very hard to make them scale horizontally. Thus, this approach is not suitable in an analytic systems (for large scale processing), because a lot of read operations are executed in order to access a small subset of attributes in a big volume of data. In fact, transactional queries are answered by (typically) scanning all the database records, but processing only few elements of them. On the contrary, in a column-oriented database, all instances of a single data element, such as account number, are stored together so they can be accessed sequentially. Therefore, aggregate operations such as *MIN, MAX, SUM, COUNT, AVG* can be performed very quickly.

Recently, *NoSQL* (*Not Only SQL*) approaches are being used to solve the efficiency problems discussed above. The rationale to develop and use NoSQL data stores can be summarized as follows:

- *Avoidance of Unneeded Complexity*: Relational databases provide a variety of features that however must obey strict data consistency constraints. This rich feature set and the ACID properties implemented by RDBMSs are mandatory while for some application scenarios they could be disregarded;
- *High Throughput*: NoSQL databases provide a significantly higher data throughput with respect to traditional RDBMSs;
- *Horizontal Scalability and Possible Running on Commodity Hardware*: In contrast to relational database management systems, most NoSQL databases are designed to scale well in horizontal way and not rely on the hardware features;
- *Avoidance of Expensive Object-Relational Mapping*: Most of the NoSQL databases are designed to store data structures that are either simple or more similar to the ones of object-oriented programming languages compared to relational data structures. They do not make expensive object-relational mapping that are no longer needed.

Non-relational data stores are usually grouped according to their data model:

1. *Key-value Stores*: These systems store values along with an index based on the key defined by users;
2. *Document Stores*: These systems store documents. A “document" can contain values that are nested documents or list of values as well as scalar values. Attribute names are dynamically defined for each document at runtime. Documents are indexed and a simple query mechanism is provided through Javascript.
3. *Column Family Stores*: These systems store extensible records that can be partitioned vertically and horizontally (eventually simultaneously on the same table) across nodes, but generally do not support secondary indexes. Rows are split across nodes through sharding on the primary key, while columns of a table are distributed over multiple nodes by using the so called “column groups".
4. *Graph Stores*: Provide efficient storage and querying of a graph exploiting references among nodes. Like for relational DBMS, these systems usually support ACID transactions.

Systems belonging to categories 1 ,2 and 3 achieve scalability by reading (potentially) out-of-date replicas obeying the constraints fixed by *CAP* theorem [16]. Indeed, CAP theorem states that a system can exhibits only two out of three of the following properties: *Consistency*, *Availability*, and *Partition-tolerance*.

As usual in a distributed system, it is *Consistent* if update operations performed by a writer can be seen by all users on the shared data source. *Availability* refers to the property of a system to provide the proper answer for any request. Finally, *Partition-tolerance* is the ability of a system to properly work even if some node in the cluster fails or some hardware or software components are out of work due to maintenance operations. Due to the complexity of a satisfactory trade-off, usually NoSQL systems smooth consistency constraints.

### *3.1.1 HBase*

*HBase* is an open source, non-relational, distributed database modeled as Google BigTable, and developed in Java. More in detail, it is an Apache project and runs on top of HDFS, providing BigTable-like capabilities for Hadoop, i.e., it provides a fault-tolerant way of storing large quantities of sparse data. HBase main features are:

- good compression performances;
- in-memory execution of operation;
- bloom filters on a per-column basis as in BigTable specification.

Tables in HBase are used to perform Input and Output for MapReduce jobs running on Hadoop, and may be accessed through the Java API but also through REST, Avro or Thrift gateway APIs. It is worth noting that HBase is not a column-oriented database in the typical RDBMS sense, but utilizes an on-disk column storage format. Rows are composed of columns, and those, in turn, are grouped into column families in order to build semantical or topical boundaries between the data as shown in Figure 1. Furthermore, the latter data organization makes it possible to improve compression or specific in-memory operation.

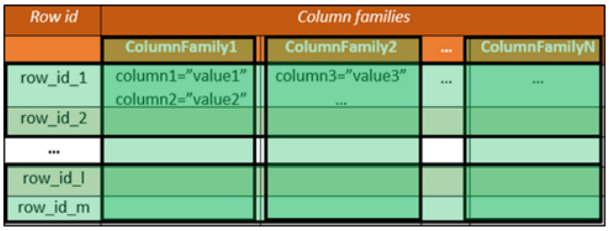



**Figure 1: HBase storage organization**

Columns are referenced as family having a qualifier represented as an array of bytes. Each column value (or cell) is either implicitly timestamped by the system or can be set explicitly by the user. Rows in the tables are sorted by a *row key* and this key provides access to information contained in the row. On the other side, columns are grouped into *column families* and can be updated at runtime (by specifying the column family through a prefix). Indeed, this model turns to be efficient and scalable, thus well suited for Big Data management as in this context a row based approach is inefficient, simple column based approaches are efficient but not

scalable, while column family based approaches achieve both efficiency and scalability. Figure 2 summarizes the features and the difference among the approaches.

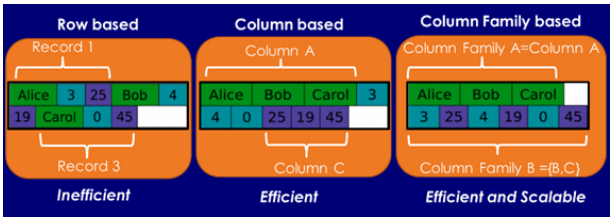



**Figure 2: Features of several storage models**

At the physical level, all columns in a column family are stored together in the same low level storage file, called an HFile. In addition to the notion of the column, table and row, HBase uses the so called "region". In fact, the HBase tables are automatically partitioned horizontally into regions that are distributed in the cluster. Each region consists of a subset of rows of a table and in this way a table that is too large to be contained in a server can be distributed in different servers in the cluster.

## 3.2 HBase data model

HBase data model is "sparse, distributed, persistent, multi-dimensional sorted map". More in detail, data are *sparse* as they do not explicitly represent *null* values. HBase *distributed* and *persistent* features are guaranteed by automatically storing data in a redundant way through exploiting a specialized distributed file system as HDFS, that spreads data across different machines usually representing different nodes of a given cluster. Moreover, data are stored in a *multi-dimensional* map for fast indexing by row key, column and version. Finally, data are lexicographically *sorted* by row key. Row-key and column-qualifier can be of arbitrary type (i.e. raw bytes) while column family qualifier must be composed only of standard characters. Version identifier is represented as a long integer, usually representing the time stamp of value insertion in the map.

However, this data model lacks some useful operations available for classical RDBMS solutions, like joins, foreign keys, referential integrity and transaction support. If the application being implemented requires these features, they need to be implemented ad-hoc. As for transaction support, although the CAP theorem holds, that is, it is not possible to guarantee both consistency and availability while partitioning data in a distributed system, HBase is partition-tolerant and consistent (*CP*).

## 3.3 Design Issues

HBase offers a useful set of APIs for data management. Indeed, besides the typical CRUD operations available for querying and inserting values, there exist some advanced features, like efficient range scan of rows or atomic increment of counters. Regarding data access efficiency, the map is sorted lexicographically on row key values for quick tuple retrieval (i.e. comparable to primary indexes on classical RDBMS). Furthermore, data are also partitioned across nodes of the cluster in *regions* composed of a contiguous range of row. Every region is served by only one machine, denoted as *Region Server*. The above considerations clarify how row key design is the most important issue when designing an HBase data store.

To further understand this requirement, we briefly recall the physical representation of data. Indeed, values belonging to a column family are stored in one or more files called *HFile*s. Every cell stored in these files brings all information required for retrieving the cell itself, i.e., its coordinates: *row key* and *column key*. As each cell value is stored together with its row key, we can move information from columns to row key without increasing overall storage space required. The operation performed for storing more information in a single key is referred as *mashing* and can be performed in several ways, as value concatenation or by formatting data using a suitable delimiter.

Mashing has several important advantages such as: i) Row key ordering allows fastest query answering compared w.r.t. alternative query patters as timestamp based querying or column qualifier based querying; ii) It allows partial scan of HBase data based on mash ordering. For example, suppose that the row key is composed of three attributes $a_1, a_2, a_3$. Then, it is possible to fetch rows based on a portion of the key, e.g we may want to retrieve all data having a specific $a_1$ value. In order to profitably exploit this feature, we need to design keys in a suitable way; iii) As tables are partitioned at region boundaries among nodes, tables should not contain a huge number of columns w.r.t. the number of rows in order to make the partitioning more effective. Indeed, properly exploiting row keys allows to define tables exhibiting the above mentioned feature as mashing causes column values to collapse in a single row key identifier.

Finally, row key design plays a crucial role for load balancing of region servers. For example, when tuples are inserted in the data store, if they share a row key prefix, they will be stored in the same region server causing an unbalanced cluster loading. Indeed, if all tuples in the upcoming stream share the same prefix, they will be loaded through the same cluster node. This drawback is referred as *hot spotting*. However, it is possible to avoid this bad behavior by exploiting some strategies such as *salting* (adding a random value as row key prefix in order to make the value distribution uniform) or *hashing* (the hash code of the key is used instead of the row key identifier). It is also possible to apply row key identifier reversing in order to obtain the least significant digit as prefix. The rationale of this choice relies in the fact that often last digit changes more frequently than the ones preceding it (e.g. the timestamp).

# 4. RENEWABLE ENERGY CASE STUDY

The Vi-POC project aims at designing and implementing a prototype able to manage renewable energy production plants distributed over national territory. Vi-POC implements an innovative system for real-time prediction of the energy production. It exploits Big Data techniques explained above in order to deal with the heterogeneity of data coming from different sources such as photovoltaic (PV), wind, geothermal, Sterling engine, water running. efficient, effective and reliable. Vi-POC is intended to predict real-time energy production with higher precision as it exploit historical information about production and weather conditions. The high accuracy and efficiency we can achieve, will allow energy market operators to implement a more effective purchasing strategy.

We exploited a HBase storage system designed for storing weather information and plant sensor data. The data is exploited by clients running data mining algorithms to predict output power of plants. Every plant sends periodically all the data collected by installed sensors. The time granularity is set based on the type and the dimension of the plant.

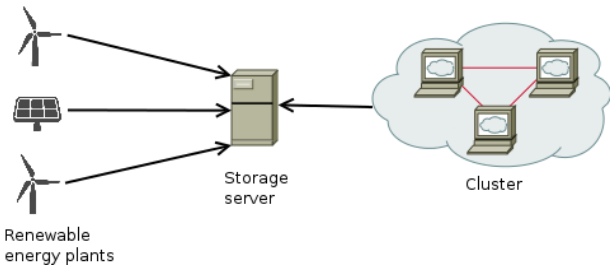



**Figure 3: System Architecture**

Data coming from plants usually consists of different measures, gathered from several sensors at a given timestamp. Indeed, the number and the type of sensors may differ among plants. Forecast data instead, consists of various predicted weather parameters forecasted for a given time and location.

Our architecture stores the data on a HBase system consisting of three tables: one for storing plants information, one for storing measurements from plants and one for storing forecasting information. To store data regarding a location, we use Geohash[1]. It is a standard way to represent latitude/longitude information as a string of characters having very useful properties. As an example, sites close to each another share the same prefix in the string.

HBase performances heavily decrease when more than three column families are used. This is exhibited because flushing and compaction are performed on a per-region basis, thus, if a column family is carrying the bulk of the data being flushed, the adjacent families will also be flushed even though the amount of data they carry is small. As a consequence, when many column families are exploited, the flushing and compaction interaction can heavily decrease system performances. In this respect, we designed column family schemes having at most two column families.

## 4.1 System Architecture

As depicted in Figure 3, we can see the interactions between the different subsystems in our architecture. As stated before, there are many renewable energy plants that send data periodically to our system. Separation between the plants and the computation cluster is a key concept. The plants, in fact, do not send their measurements directly to the computation cluster, but to a separated storage level, made of several file servers. Different fault tolerance strategies are applied among these levels in order to avoid the block of the entire system due to the failure of one of the components. Data is then taken by computation cluster's *Extract, Transform and Load (ETL)* tool and stored in a non-relational distributed database across the nodes of the cluster itself.

The independence between computation cluster and storage level make easier the management of the cluster, since storage level acts as a frontend to it. The cluster can then be modified without changing the configuration of the different plants. This solution introduces, moreover, the isolation of the cluster, since it has only to communicate with the storage level, that is located in the same subnet.

Figure 4 shows the software architecture implemented on a cluster

[1]www.geohash.org

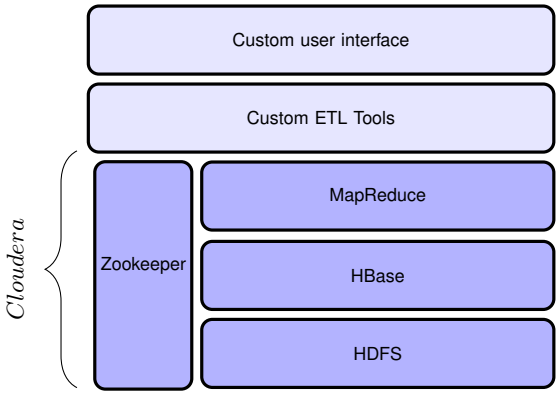



**Figure 4: Architecture layers**

of computing nodes. It is composed of several levels. As stated before, we use HDFS as distributed file system. Data is stored on different commodity machines of our computation cluster. We plan to simplify the software setup of the computation cluster through the use of software containers (in particular Dockers [2]), thus providing platform as a service (PaaS) style deployment.

On top of HDFS we run HBase, which provides BigTable-like capabilities for Hadoop. The large quantities of data, coming from renewable energy plants, are stored in a fault-tolerant way across the nodes of the cluster. Tables in HBase serve as the input and output for MapReduce jobs. We use *Apache ZooKeeper* that provides services like distributed configuration, synchronization and naming registry. Cloudera Distribution including Apache Hadoop [3](*CDH*) offers a quick way to deploy all of the above components.

We wrote a custom ETL tool which manages the interaction between the storage level and the computation cluster. The tool periodically downloads the new data from the storage servers. This data is in csv format and needs to be transformed in order to be stored, according to the schema discussed in Section 4.2. The ETL tool provides this transformation and the subsequent upload to HBase tables. The definition of queries on data and the visualization of results are made by another custom tool that stands on top of our architecture.

## 4.2 Table schemas

Based on the above considerations, we designed tables described below (we do not report the actual name of each attribute as they are coded by the plant owner and they are not easily understandable).

Table *Plants*:

- *RowKey*: concatenation of the type of the plant (solar, wind, hydroelectric) and a plant identifier;
- *Column family 1*: contains as many attributes as the cardinality of data. Every attribute represents raw information as the configuration parameters or the coordinates of the plant;

[2]https://www.docker.com/
[3]http://www.cloudera.com/content/cloudera/en/home.html

| Table: Weather Data | | | |
|---|---|---|---|
| row key: | geohash + Reversed Timestamp + Measurament Type +Server Id | | |
| Family: | c | column | collected data |
| | p | column | predicted data |

| Table: Plants Info | | | | |
|---|---|---|---|---|
| Row key: | Type+PlantID | | | |
| Family: | c | columns: | <attribute> | <value> |
| | i | columns: | <timestamp> | maintenance_description |

| Table: Predicted Measures | | | | |
|---|---|---|---|---|
| row key: | plantID+reverse_timestamp+MeasuramentType | | | |
| family: | c | columns: | <uid> | <value> |

| Table: Measures | | | | |
|---|---|---|---|---|
| row key: | plantID+reverse_timestamp+MeasuramentType | | | |
| family: | c | columns: | <uid> | <value> |

**Figure 5: HBase table schemas**

- *Column family 2*: stores log information about maintenance operations for the specific plant.

Table *Measure*:

- *RowKey*: concatenation of the identifier of the plant, the reverse time stamp and the measurement type;
- *Column family*: stores all collected measures. The number of attributes is equal to the cardinality of counters being collected.

Table *Predicted Measure*:

- *RowKey*: the same structure as the *Measure* table;
- *Column family*: stores the measures predicted by mining algorithms. The number of attributes is equal to the cardinality of predicted data.

Table *Weather Data*:

- *RowKey*: concatenation of Geohash, reverse time stamp, measurement type and server identifier, where server identifier is used to trace which server senst the prediction;
- *Column Family 1*: used to store collected weather data.
- *Column Family 2*: used to store predicted weather data (weather forecasts).

Figure 5 shows the implemented Hbase schema definition for representing the information described above.

## 4.3 Long-term forecast of PV energy production

During the last years, the forecast of PV energy production has received significant attention since photovoltaics are becoming a major source of renewable energy for the world [15]. Forecasting methods depend on the tools and information available, the forecast horizon, the number of plants considered and the size of the geographic area they cover [31]. Diverse resources are used to generate solar and PV forecasts, ranging from measured weather and PV system data, satellite and sky imagery cloud observations, to Numerical Weather Prediction (NWP) models [22]. The short-term forecasts typically use measured weather and PV system data, and satellite and sky imagery observations of clouds, while the long-term forecasts use numerical weather prediction (NWP) models. The best approaches make use of both measured data and NWP models.

In the literature, several data mining approaches have been proposed for renewable energy power forecasting. We typically distinguish between physical and statistical approaches. Physical approaches deal with refining NWP forecast with physical considerations, while statistical approaches deal with building models that establish a relationship between historical values and forecasted variables. Methodologically, there are approaches based on time series [14] and approaches that learn adaptive models [6][33].

It has been noted that physical (e.g. wind speed and solar irradiation) property behavior exhibits a trail called concept drift, i.e., they change characteristics over time [9]. In this respect, adaptive models are generally considered to produce more reliable predictions regarding concept drift, but require a continuous training phase. For example, in [27], the estimation of the model parameters is based on an exponential weighted adaptive recursive least squares controlled by a forgetting factor. A different solution is proposed in [32], where a recursive method for the estimation of the local model coefficients of a linear regression function is proposed. In this case, the time dependence of the cost function is ensured by exponential forgetting of past observations.

In [21], the author uses a stochastic gradient for online training of neural networks in wind power forecasting. Another work which uses neural networks is [8], where the authors train local recurrent neural networks of online learning algorithms based on the recursive prediction error. Bacher et al. [6] propose to forecast the average output power of rooftop PV systems by considering past measurements of the average power and NWP forecasts as inputs to an autoregressive model with exogenous input (ARX).

Sharma et al. [33] consider the impact of the weather conditions explicitly and used an SVM classifier in conjunction with a RBF kernel to predict solar irradiation. Bofinger et al. [10] propose an algorithm where the forecasts of an European weather prediction center (of midrange weathers) were refined by local statistical models to obtain a fine tuned forecast. Other works on temporal modeling with applications to sustainability focus on motif mining. For example, Patnaik et al. [30] proposed a novel approach to convert multivariate time-series data into a stream of symbols and mine frequent episodes in the stream to characterize sustainable regions of operation in a data center. Finally, Chakraborty et al. [14] propose a Bayesian ensemble which involves three diverse predictors, that is, naïve Bayes, $K$-NN and sequence prediction.

In this case study, we propose an adaptive method for long-term forecast (one-day ahead) of PV energy production based on ANNs. The proposed approach exploits NWP to benefit from uncontrollable factors (such as weather conditions). We investigate the predictive performance of two structured (all hours of the forecasted

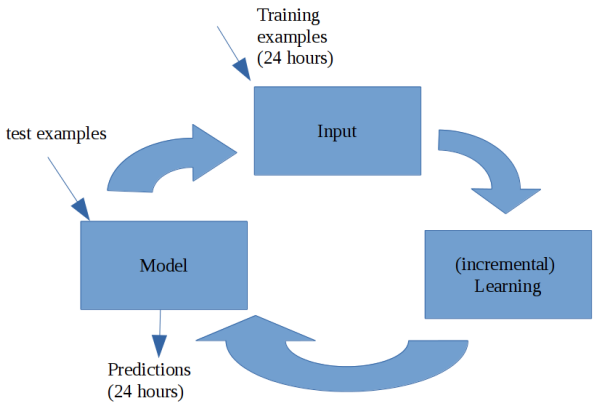



**Figure 6: Our learning scheme**

day are outputs from a single model) and non-structured output prediction models (each hour of the forecasted day is output from one model).

### 4.3.1 Method

The machine learning task is to predict the PV power generation using the following input attributes:

- the geographic coordinates of the plant: latitude and longitude,
- the sun positions at the location of the plant: altitude and azimuth, queried by SunPosition (`http://www.susdesign.com/sunposition/index.php`),
- the properties of the plant: site ID, brand ID, model ID, age in months,
- weather data: ambient temperature, irradiance, pressure, wind speed, wind bearing, humidity, dew point, cloud cover, descriptive weather summary.

Additionally, in the case of structured output prediction, also the day is passed as feature, while in the case of non-structured output prediction, besides the day, also the hour. In the training phase, we use historical weather information collected by sensors, while for prediction purposes, we use weather forecast data provided by NWP systems. The output is the prediction of the power production (KWh) for the next day at one hour intervals. The prediction models are updated on a daily basis as depicted in Figure 6.

### 4.3.2 Data preprocessing

Since the aim is to predict the energy production at a hourly granularity, the data was aggregated so that each row represents an hour. Additional, we addressed also the issue of missing data, irregularities and outliers, and performed normalization of the data before the learning process.

In order to fix completely missing hourly data points, we adopt the following approach. Missing production values in kWh are replaced by the average value observed by sensors in the same month at the same hour. Missing temperature values are replaced by historical data. Moreover, we also observed that sometimes the irradiance assumes a zero value while the plant is in a productive state (kWh>0). To correct irregularities of that type, we consider the average irradiance value in the same month of the same year at the same hour to replace the zero value. In any other case in which the irradiance is zero, we check if the average irradiance value in the same month of the same year at the same hour is zero too: if not, the resulting average value replaces the missing value.

After replacing missing values, we check the presence of outliers in the irradiance and temperature data. For example, if the irradiance (irr) observed by sensors is out of the range defined by [avg(irr) - 4* stddev(irr), avg(irr) + 4* stddev(irr)], this value is replaced by the average of the irradiance observed in the same month of the same year at the same hour. The same approach applies also fro handling outliers in the temperature data. Furthermore, we observed that irradiance measured locally by sensors has often lower values compared to irradiance extracted by NWP models, possibly because sensors located on plants can be covered by obstacles or dirt. Training a model by means of sensors data and using it to extract predictions with NWP data can lead to inaccurate predictions. To overcome this issue, we calculate the percentage of change between monthly NWP irradiance (extracted by PVGIS) and irradiance detected by sensors on historical data (same month at the same hour), and we normalize the latter.

In order to train the neural network, data was normalized in the range between 0 and 1. Hence, we applied a min-max normalization for each feature, considering the min and max of observed values. Actually, we considered the max increased by 30 percent, to handle future situations in which observed values of each feature might exceed the current maximum.

### 4.3.3 Experiments

In our empirical evaluation, we consider a real dataset collected at regular intervals of 15 minutes (measurements start at 2:00 AM and stop at 8:00 PM every day) by sensors located on 18 plants in Italy. The time period spans from January 1$^{st}$, 2012 to May 4$^{th}$, 2014. The weather data is queried from Forecast.io (`http://forecast.io/`), while the irradiance is queried from PVGIS (`http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php`). As anticipated before, the raw data are preprocessed and normalized according to the z-score normalization, before using them for learning, in order to resolve measurement errors.

In this paper, we use the *encog* implementation of the Resilient Propagation (RPROP+) algorithm for training neural networks (`http://www.heatonresearch.com/wiki/Resilient_Propagation#Implementing_RPROP.2B`). RPROP+ is one of the best general-purpose neural network training methods implementing the back-propagation technique. We use RPROP+ since it has been proven effective for renewable energy prediction [9]. For the evaluation, the dataset is randomly split into training days (85%) and testing days (15%). Experiments are run three times and average results are collected. For each run, the network is trained incrementally on the training dataset until a testing day is found. Then, it is repeatedly first tested on the testing day and after that it is re-trained with the sample added to the training set, together with all the training days before the next testing day. At the end, the average performance over all the test samples is reported as a result.

We distinguish between hourly (non-structured) and daily (struc-

**Table 1: Performance results for one-day ahead PV power forecast for hourly (non-structured) and daily (structured output) settings. No spatial (Lat Lon) indicate results without (with) geographic coordinates of the plant.**

| | RMSE | MAE | Impr. [%] |
|---|---|---|---|
| No Spatial Hourly | 0,120 | 0,079 | 17,410 |
| No Spatial Daily | 0,109 | 0,068 | 24,810 |
| Lat Lon Hourly | 0,120 | 0,078 | 17,443 |
| Lat Lon Daily | 0,111 | 0,069 | 23,915 |
| Persistence model | 0,146 | 0,085 | |

tured output) settings. In the hourly setting, we investigate non-structured models with single output - the production of the plant at a specified day and specified hour. In the daily setting, we investigate structured models with 19 outputs - the productions of the plant for the hours from 2:00 AM to 8:00 PM on a specified day. Furthermore, we consider scenarios with and without the latitude and the longitude of the plant taken as descriptive variables. The later will investigate whether the geographic coordinates play an important role for the prediction performance.

### *4.3.4 Results and discussion*

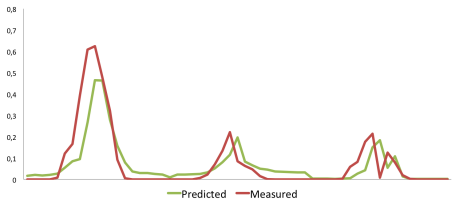
The results for the investigated hourly and daily scenarios are reported in Table 1. We consider three indicators of the predictive performance, namely the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE) and the improvement with respect to the persistence model (i.e., the model that forecasts the same production observed 24 hours before).

The results clearly show improvement of the predictive performance over the persistence model, with the structured-output prediction model clearly outperforming the non-structured one. From Table 1, we can also notice that geographic coordinates improve the prediction effectiveness, suggesting that data are subject to spatial autocorrelation phenomena. [34].

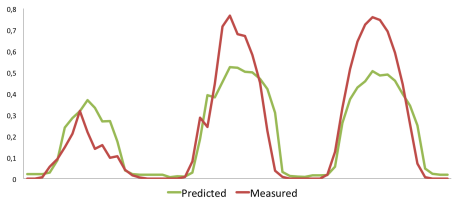
The predictive performance of the model can also be graphically inspected from Figure 7, where the predicted vs. measured power production are presented for three consecutive typical cold (in January) and warm (in May) days. In both cases, we report predictions for partially cloudy days. The predictions are obtained using the best performing model, i.e. structured output considering the latitude and longitude as input attributes.

## 5. CONCLUSIONS AND FUTURE WORK

Big Data analysis is a challenging task as we need to take into account the velocity, variety and volume of information to be analyzed. Indeed, such features heavily influence the design of a system for Big Data analysis. In this respect, we analyzed several design options in order to implement a prototype for accurate prediction of renewable energy production plant output. In this paper, we have presented the project Vi-POC – a distributed system for storing, querying and analyzing data collected from renewable energy production plants. In particular, we have described its data model and its forecasting capabilities. As for this last aspects, we have empirically shown its predictive capabilities and compared cases with structured output prediction and non-structured output prediction. Results confirm that predictive capabilities are better in case of structured output prediction, probably because of the implicit consideration of the dependence of the predictions at consecutive hours.




(a) January $4^{\text{th}}$, $5^{\text{th}}$, $6^{\text{th}}$.




(b) May $4^{\text{th}}$, $5^{\text{th}}$, $6^{\text{th}}$.

**Figure 7: Predictions (green) and measurements (red) of the productions for three consecutive days of a single plant. The three consecutive days are taken from January and May. Results are obtained with the daily (structured) setting. We recall that the time intervals considered are 2:00 AM - 8:00 PM. Results are obtained including geographic coordinates.**

As future work, we plan to explore further prediction techniques based on clustering along with the integration of additional data sources in our system in order to achieve more accurate results. More in detail, we plan to test our system in different regions having different weather condition w.r.t. south of Italy in order to generalize our technique for a widespread commercial use.

## 6. ADDITIONAL AUTHORS

## 7. REFERENCES


[1] Big data. *Nature*, September 2008.

[2] Data, data everywhere. *The Economist*, Feb 2010.

[3] Drowning in numbers - digital data will flood the planet - and help us understand it better. *The Economist*, Nov 2011.

[4] *Design Principles for Effective Knowledge Discovery from Big Data*, Helsinki, Finland, 2012. August 20-24.

[5] D. Agrawal et al. Challenges and opportunities with big data. A community white paper developed by leading researchers across the United States. Mar 2012.

[6] Peder Bacher, Henrik Madsen, and Henrik Aalborg Nielsen. Online short-term solar power forecasting. *Solar Energy*, 83(10):1772 – 1783, 2009.

[7] Gökhan H. Bakır, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan, editors. *Predicting structured data*. The MIT Press, 2007.

[8] T. G. Barbounis and J. B. Theocharis. Locally recurrent neural networks for wind speed prediction using spatial correlation. *Inf. Sci.*, 177(24):5775–5797, December 2007.

[9] R.J. Bessa, V. Miranda, and J. Gama. Entropy and correntropy against minimum square error in offline and online three-day ahead wind power forecasting. *Power Systems, IEEE Transactions on*, 24(4):1657–1666, 2009.

[10] S. Bofinger and G. Heilscher. Solar electricity forecast - approaches and first results. In *20th Europ. PV conf.*, 2006.

[11] A. B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages

195–203, New York, NY, USA, 2000. ACM.

[12] M. Ceci, N. Cassavia, R. Corizzo, P. Dicosta, D. Malerba, G. Maria, E. Masciari, and C. Pastura. Innovative power operating center management exploiting big data techniques. In *18th International Database Engineering & Applications Symposium, IDEAS 2014, Porto, Portugal, July 7-9, 2014*, pages 326–329, 2014.

[13] Michelangelo Ceci, Nunziato Cassavia, Roberto Corizzo, Pietro Dicosta, Donato Malerba, Gaspare Maria, Elio Masciari, and Camillo Pastura. Big data techniques for renewable energy market. In Sergio Greco and Antonio Picariello, editors, *22nd Italian Symposium on Advanced Database Systems, SEBD 2014, Sorrento Coast, Italy, June 16-18, 2014.*, pages 369–377, 2014.

[14] Prithwish Chakraborty, Manish Marwah, Martin F. Arlitt, and Naren Ramakrishnan. Fine-grained photovoltaic output prediction using a bayesian ensemble. In *AAAI*, 2012.

[15] EPIA European Photovoltaic Industry Association. Global Market Outlook for Photovoltaics 2014-2018. http://www.epia.org/news/publications/global-market-outlook-for-photovoltaics-2014-2018, June 2014.

[16] A. Fox and E. A. Brewer. Harvest, yield, and scalable tolerant systems. In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, HOTOS '99, pages 174–, Washington, DC, USA, 1999. IEEE Computer Society.

[17] Anjana Gosain and Nikita Chugh. Article: New design principles for effective knowledge discovery from big data. *International Journal of Computer Applications*, 96(17):19–23, June 2014. Full text available.

[18] M. Herland, T. M. Khoshgoftaar, and R. Wa. A review of data mining using big data in health informatics. *Journal of Big Data*, 2014.

[19] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *In CIDR*, pages 261–272, 2011.

[20] C. K. Joseph and S. Kakad. Predicting impact of natural calamities in era of big data and data science. In *7th Intl. Congress on Env. Modelling and Software*, pages 90–98, 2014.

[21] George Kariniotakis. *Contribution to the development of an advanced control system for the optimal management of wind-diesel power systems*. PhD thesis, 1996.

[22] Jan Kleissl. *Solar Resource Assessment and Forecasting*. Elsevier, 2013.

[23] Dragi Kocev, Celine Vens, Jan Struyf, and Sašo Džeroski. Tree ensembles for predicting structured outputs. *Pattern Recognition*, 46(3):817–833, 2013.

[24] A. Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *PVLDB*, 5(12):2032–2033, 2012.

[25] S. Lohr. The age of big data. *nytimes.com*, Feb 2012.

[26] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey Global Institute*, May 2011.

[27] H. A. Nielsen, P. Pinson, L. E. Christiansen, T. S. Nielsen, H. Madsen, J. Badger, G. Giebel, and H. F. Ravn. Improvement and automation of tools for short term wind power forecasting. In *EWEC*, 2007.

[28] Y. Noguchi. Following digital breadcrumbs to big data gold. *National Public Radio*, Nov 2011.

[29] Y. Noguchi. The search for analysts to make sense of big data. *National Public Radio*, Nov 2011.

[30] Debprakash Patnaik, Manish Marwah, Ratnesh K. Sharma, and Naren Ramakrishnan. Temporal data mining approaches for sustainable chiller management in data centers. *ACM Trans. Intell. Syst. Technol.*, 2(4):34:1–34:29, July 2011.

[31] Sophie Pelland, Jan Remund, Jan Kleissl, Takashi Oozeki, and Karel De Brabandere. Photovoltaic and solar forecasting. Technical report, IEA PVPS, 2013.

[32] Pierre Pinson, Henrik Aa. Nielsen, Henrik Madsen, and Torben S. Nielsen. Local linear regression with adaptive orthogonal fitting for the wind power application. *Statistics and Computing*, 18(1):59–71, March 2008.

[33] Navin Sharma, Pranshu Sharma, David E. Irwin, and Prashant J. Shenoy. Predicting solar generation from weather forecasts using machine learning. In *SmartGridComm*, pages 528–533. IEEE, 2011.

[34] Daniela Stojanova, Michelangelo Ceci, Annalisa Appice, Donato Malerba, and Saso Dzeroski. Dealing with spatial autocorrelation when learning predictive clustering trees. *Ecological Informatics*, 13:22–39, 2013.

[35] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.

[36] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[37] M. Zaharia, M. Chowdhury, M.J Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[38] Quan Zou, Xu-Bin Li, Wen-Rui Jiang, Zi-Yu Lin, Gui-Lin Li, and Ke Chen. Survey of mapreduce frame operation in bioinformatics. *Briefings in Bioinformatics*, 2013.