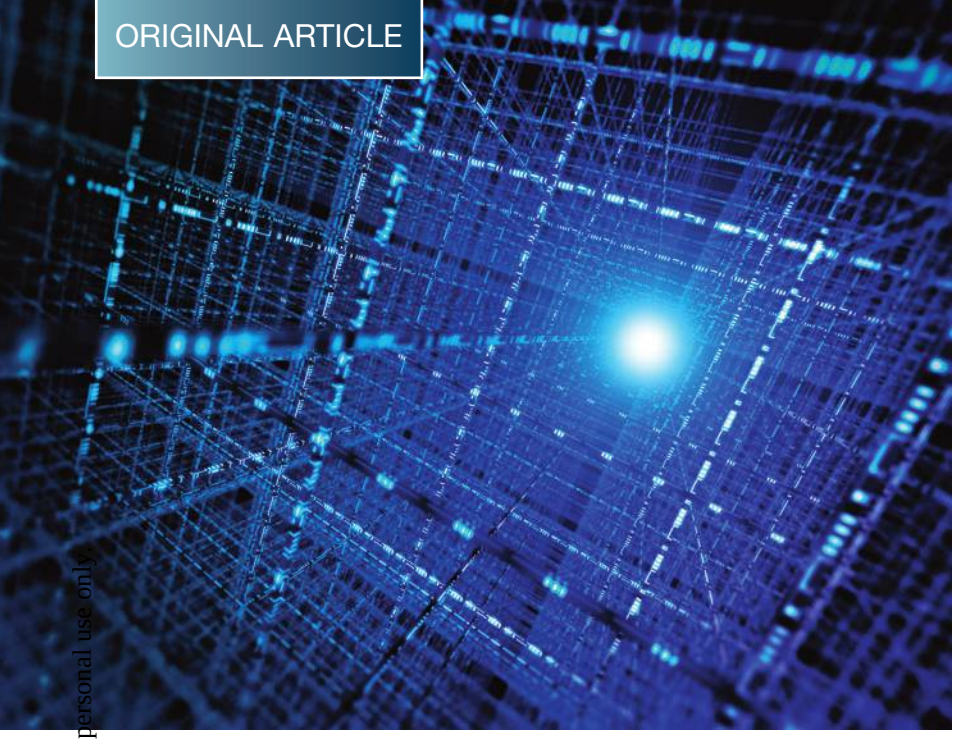
ORIGINAL ARTICLE



APACHE DRILL:

Interactive Ad-Hoc Analysis at Scale

Michael Hausenblas and Jacques Nadeau
MapR Technologies

Abstract

Apache Drill is a distributed system for interactive ad-hoc analysis of large-scale datasets. Designed to handle up to petabytes of data spread across thousands of servers, the goal of Drill is to respond to ad-hoc queries in a low-latency manner. In this article, we introduce Drill's architecture, discuss its extensibility points, and put it into the context of the emerging offerings in the interactive analytics realm.

Introduction

When it comes down to large-scale data processing in the enterprise, we encounter a variety of workloads. In order to achieve a business goal, we often see a combination of said workloads deployed:

- batch-oriented processing, for example, MapReduce-based[1] frameworks like Hadoop, for recurring tasks such as large-scale data mining or aggregation;
- OLTP, such as user-facing e-commerce transactions, where NoSQL data-stores shine, including Apache HBase (http://hbase.apache.org/) or Apache Cassandra (http://cassandra.apache.org/);
- stream processing, to handle stream sources such as social media feeds or sensor data, with Storm (http://storm-project.net/) being a representative framework;
- search over semistructured data items and documents—a widely used platform in this category is Apache Solr (http://lucene.apache.org/solr/);
- and last but not least, interactive ad-hoc query and analysis.

In this article, we focus on the last category: workloads that involve a human, interacting online with a system, demanding low-latency answers and support for formulating ad-hoc queries.

From Design to Implementation

Given a human in the loop who sits, say, in front of a business analytics application such as Tableau Desktop, clearly, a query should take only seconds or less to execute—even at scale. Further, allowing the user to issue ad-hoc queries is essential; often, the user might not necessarily know ahead of time what queries to issue. Also, one may need to react to changing circumstances. The lack of tools to perform interactive ad-hoc analysis at scale is a gap that Apache Drill fills.

For a concrete motivation, imagine a marketing analyst trying to experiment with ways to target user segments for an upcoming campaign. Let us further assume that the necessary data for the analysis resides in different data sources, as is often found in the enterprise: the web logs, containing the customers' click behavior, may be stored in Hadoop Distributed File System (HDFS), while user profiles might come from a MongoDB instance as well as transaction data stemming from a conventional relational database management system (RDBMS) such as Oracle.

Currently, in order to realize an interactive analysis, one would need to Extract, Transform and Load (ETL) data between the different systems and deal with the issues on a one-off basis, including impedance mismatches and delays in the actual analysis due to data preparation tasks. What is needed is an

BIG DATA JUNE 2013 • DOI: 10.1089/big.2013.0011

in-situ data processing method that can deal with different data sources in a flexible and scalable way. Enter Apache Drill.

High-level architecture

Back in 2010, Google published the seminal Dremel[2] research paper, introducing two main innovations: generically handling nested data with column-striped representations (including record assembly) and multilevel query execution trees, allowing for parallel processing of data spread over thousands of computing nodes. In mid 2012, these innovations were taken to The Apache Software Foundation, forming the core of a new incubator, Apache Drill.

At a high level, Apache Drill's architecture (Fig. 1) comprises the following layers:

- **User**: providing interfaces such as a command line interface (CLI), a REST interface, JDBC/ODBC, etc., for human or application-driven interaction.
- **Processing**: allowing for pluggable query languages as well as the query planner, execution, and storage engines.
- **Data sources**: pluggable data sources either local or in a cluster setup, providing *in-situ* data processing.

> "NOTE THAT APACHE DRILL IS NOT A DATABASE BUT RATHER A QUERY LAYER THAT WORKS WITH A NUMBER OF UNDERLYING DATA SOURCES."

Note that Apache Drill is not a database but rather a query layer that works with a number of underlying data sources. It is primarily designed to do full table scans of relevant data as opposed to, say, maintaining indices. Not unlike the MapReduce part of Hadoop provides a framework for parallel processing, Apache Drill provides for a flexible query execution framework, enabling a number of use cases from quick aggregation of statistics to explorative data analysis.

The workers in Apache Drill, suitably called drillbits, run on each processing node in order to maximize data locality. The coordination of the drillbits, the query planning, as well as the optimization, scheduling, and execution are performed and distributed.

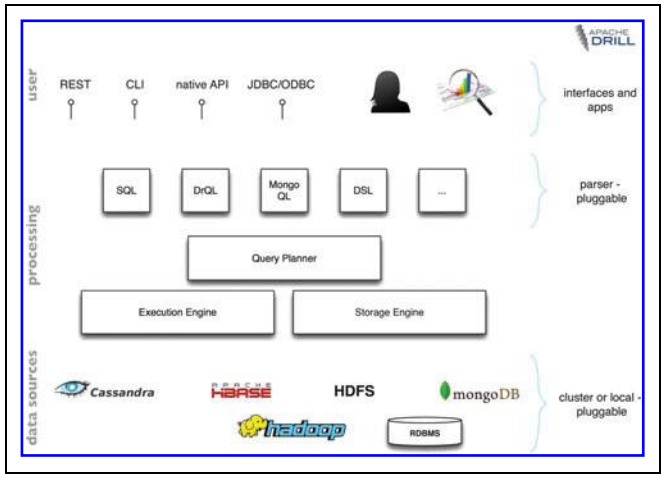


FIG. 1. Apache Drill's high-level architecture.

Key features

In the following, we highlight key features of Apache Drill and argue how the design of Apache Drill has been influenced by observations made in practical settings as well as requirements stated by the community.

- **Extensibility rules**: Apache Drill is designed for extensibility, with well-defined application programming interface (API) and interfaces. This includes, starting from the user layer, pluggable query languages via the query API as well as support for user-defined functions (UDF). Also, custom operators can be implemented, which is already being done for Apache Mahout, for example. Further, the default cost-based optimizer can be replaced or extended. Last but not least, custom scanners for data sources (such as new NoSQL datastores) or file formats can be implemented via the API. The latter can be handy for cases in which the data is embedded in container formats, for example, metadata contained in video or audio documents.
- **Full Structured Query Language (SQL) please**: many settings require the integration with deployed Business intelligence (BI) tools, including but not limited to Tableau, Excel, SAP Crystal Reports, etc. For this, SQL-like is not sufficient and consequently Apache Drill supports standard American National Standards Institute (ANSI) SQL 2003 along with allowing for standard Open Database Connectivity (ODBC)/Java Database Connectivity (JDBC) drivers.
- **Nested data as a first-class citizen**: Apache Drill is rather flexible concerning supported data shapes.[3] As nested data is becoming prevalent (think JSON/BSON in document stores, XML, ProtoBuf, etc.) and flattening of nested data is error-prone, we support nested data directly in Apache Drill, effectively establishing an extension to above-mentioned SQL support.
- **Use but don't abuse schema**: data sources increasingly do not have rigid schemas; the schema might change rapidly or differ on a per-record level (for example, the case with HBase). Apache Drill hence supports queries against unknown schemas. The user is free to define a schema upfront or let Apache Drill discover it.

Query Execution

In order to appreciate Apache Drill's flexibility, let us now have a closer look at the query execution in greater detail. As depicted in Figure 2, Apache Drill works by transforming a

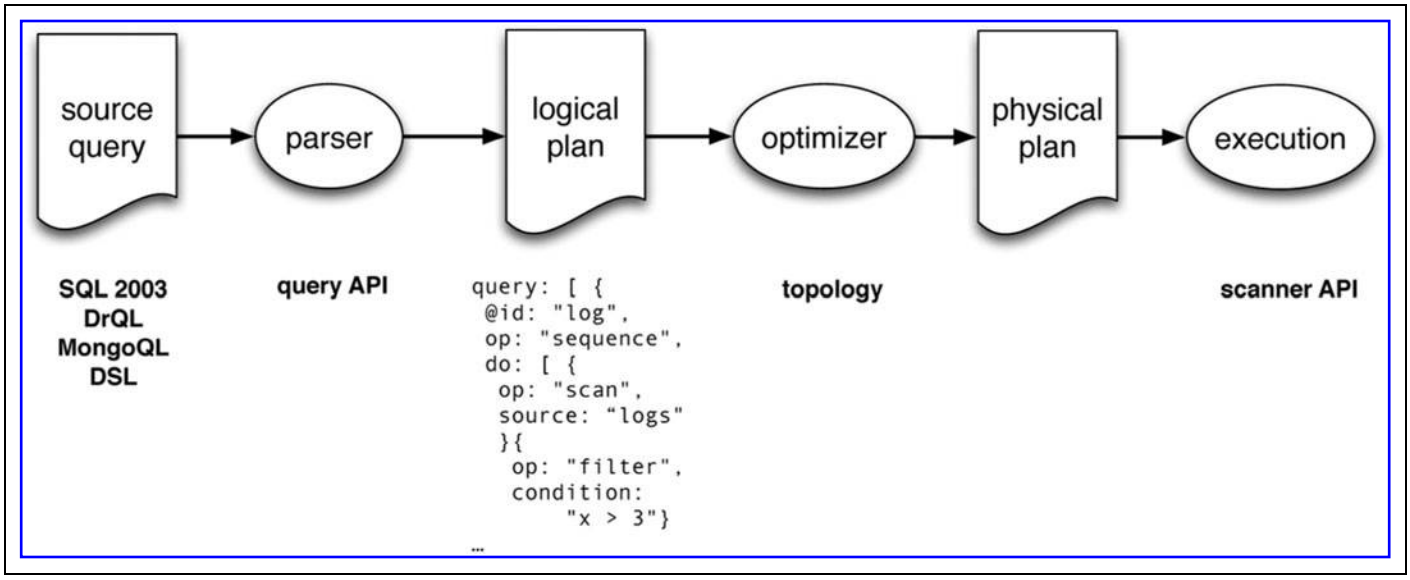


FIG. 2. Query Execution in Apache Drill.

query written in a human-readable syntax, such as SQL, into an internal form known as the logical plan. Then, Apache Drill transforms the logical plan into a physical plan, which is executed against the data source(s).

In Figure 2, an interface, which may be a JDBC/ODBC library, provides the query to Apache Drill. Once presented with the source query, it is parsed and transformed to generate the logical plan. Typically, the logical plan lives in memory in the form of Java objects, but it also possesses a textual form (an example snippet is shown in Fig. 2). Alternatively, one can directly inject the logical plan via a domain-specific language (DSL) stemming, for example, from Ruby, Python, or Scala. The logical query is then transformed and optimized into the physical plan, representing the actual structure of computation. The optimizer can, taking the topology into account, substantially restructure the logical plan as it produces the physical plan; one of the most important ones is the introduction of parallel computation. Another is the exploitation of columnar data to improve processing speed. Finally, in the execution, Apache Drill benefits from the flexible scanner API, allowing to push down the queries to the internals of the data source.

Pivotal to Apache Drill's query execution is the so called logical plan. We hence provide some more details in the following text. The logical plan describes a dataflow program using a JavaScript Object Notation (JSON)-based syntax (http://j.mp/apache-drill-plan-syntax) consisting of a directed acyclic graph (DAG). The DAG is composed of operators—the nodes in the graph—operating on sequences of records that traverse the edges of the graph. Structurally, the logical plan consists of a top-level object with three components:

1. A *header* providing metadata about the query (version, software used to generate the query, etc.).
2. The *data sources*, specifying where data resides.
3. A *query component*, defining the dataflow DAG.

To obtain a better understanding of the concrete internals, we discuss an exemplary logical plan in the following (see Supplementary Material, available online at www.liebertonline.com/big, for details on how to set it up and execute it). The exemplary logical plan defines a JSON document in the local files system (donuts.json) as the one and only data source and specifies a series of operations (filter, aggregation, ordering) on it. Finally, it defines the console as the data sink, that is, as output destination.

```
{
 head:{
  type:"apache_drill_logical_plan",
  version:"1",
  generator:{
   type:"manual",
   info:"na"
  }
 },
 storage:[
  {
   type:"console",
   name:"console"
  },
  {
   type:"fs",
   name:"fs1",
   root:"file:///"
  },
  {
   type:"classpath",
   name:"cp"
  }
 ],
 query:[
  {
   op:"sequence",
   do:[
       {
        op: "scan",
        memo: "initial_scan",
```

```
        ref: "donuts",
        storageengine: "cp",
        selection: {
         path: "/donuts.json",
         type: "JSON"
        }
       },
    {
        op: "transform",
        transforms: [
         { ref: "quantity", expr: "donuts.sales"}
        ]
       },
       {
        op: "filter",
        expr: "donuts.ppu<1.00"
       },
       {
        op: "segment",
        ref: "ppusegment",
        exprs: ["donuts.ppu"]
       },
       {
        op: "collapsingaggregate",
        within: "ppusegment",
        carryovers: ["donuts.ppu"],
        aggregations: [
         { ref: "donuts.typeCount", expr: "count(1)" },
         { ref: "donuts.quantity", expr: "sum(quantity)"},
         { ref: "donuts.sales", expr: "sum(donuts.ppu *
           quantity)" }
        ]
       },
       {
        op: "order",
        orderings: [
         {order: "desc", expr: "donuts.ppu" }
        ]
       },
       {
        op: "project",
        projections: [
         { ref: "output", expr: "donuts" }
        ]
       },
       {
      op: "limit",
      first: 0,
      last: 100
     },
       {
        op: "store",
        memo: "output sink",
        storageengine: "console",
        target: {pipe: "STD_OUT"}
       }
    ]
   }
  ]
}
```

Emerging Offerings

The work that the open source community has put into Apache Drill since mid 2012 has since been validated by commercial offerings. Only recently, right before and during the Strata Conference 2013, a number of new announcements have been made. We will review the emerging offerings in this paragraph and discuss their relation to Apache Drill.

The contenders listed in Table 1 are as follows:

- Apache Drill (http://incubator.apache.org/drill/), subject of this article
- Apache Hive (http://hive.apache.org/), Hadoop-based data warehouse
- BigQuery (https://developers.google.com/bigquery/), Google's hosted offering of Dremel
- CitusDB (http://citusdata.com/docs), hybrid analytics database
- Hadapt (http://hadapt.com/product/), analytical platform based on hybrid storage layer
- HAWQ (www.greenplum.com/blog/topics/hadoop/introducing-pivotal-hd), relational database running atop HDFS
- Impala (https://github.com/cloudera/impala), distributed query execution engine on top of HDFS
- Phoenix (https://github.com/forcedotcom/phoenix), SQL layer over HBase

We note that the Stinger initiative (http://hortonworks.com/blog/100x-faster-hive/) has recently been announced as well, with the goal to "enhance Hive with more SQL and better performance for these human-time use cases." However, when writing this article, too little details were known in order to include it in the above review.

Looking at Table 1 suggests that Apache Drill has some architectural commonalities with other systems that have been inspired by Dremel, including columnar storage support and the query dispatching. Beside the community consensus on the APIs, there are a number of unique technological points to Apache Drill: the extensibility regarding both data sources and query languages, vectorized columnar execution, nested data support, as well as the capability to benefit from an upfront defined schema without forcing it upon the user.

Conclusions and the Road Ahead

Dealing with latencies in large-scale systems[4] is subject to active development and research. With Apache Drill, the open source community has taken an important step to make the innovations introduced by Google's Dremel available to a large audience, under a free license. Apache Drill's design is motivated by requirements from a range of use cases in the interactive analysis realm as well as informed by the "experience of the crowd," addressing extensibility, support for full SQL, and nested data, as well as optional schema handling.

As the community is growing, more and more people start engaging via the mailing lists (http://incubator.apache.org/drill/mailing-lists.html). At the time that writing contributions from multiple organizations and individuals are provided, a reference implementation including a single-node demo is available (https://cwiki.apache.org/confluence/display/DRILL/). The work-in-progress spans from a SQL interpreter to storage engine implementations (Accumolo, Cassandra, HBase, etc.) with an alpha release of the entire system expected for Q2 of 2013.

Table 1. Offerings in the Interactive Large-Scale Data Processing Space in Early 2013

	Apache Drill	*Apache Hive*	*BigQuery*	*CitusDB*	*Hadapt*	*HAWQ*	*Impala*	*Phoenix*
Owner	Community	Community	Google	CitusData	Hadapt	Greenplum	Cloudera	Salesforce.com
Low-latency	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Operational mode	On-premise	On-premise	Hosted, SaaS offering	On-premise	On-premise	Part of Pivotal HD appliance	On-premise	On-premise
Data shapes	Nested, tabular	Nested, tabular	Nested, tabular	Nested, tabular	Tabular	Tabular	Tabular	Tabular
Data sources	Extensible, incl. HDFS, HBase, Cassandra, MongoDB, RDBMS, etc.	HDFS, HBase	N/A	PostgreSQL, MongoDB, HDFS	HDFS/ RDBMS	HDFS, HBase	HDFS, HBase	HDFS, HBase
Hadoop dependent	No	Yes	No	No	Yes	No	Yes	Yes
Schema	Optional	Required	Required	Required	Required	Required	Required	Required
License	Apache 2.0	Apache 2.0	ToS/SLA	Commercial	Commercial	Commercial	Apache 2.0/ Open Source	Proprietary
Source code	Open	Open	Closed	Closed	Closed	Closed	Open	Open
Query languages	Extensible, incl. SQL 2003, MongoQL, DSL, etc.	HiveQL	SQL subset	SQL	SQL subset	SQL subset	SQL/HiveQL subset	SQL subset
Columnar storage	Yes	Possible	Yes	No	No	Yes	Yes	No

Acknowledgments

The authors would like to thank the Apache Drill community for their contributions, be it concerning the design of Apache Drill or in terms of code commits. Further, the authors are grateful for MapR Technologies sponsoring their work on Apache Drill.

Author Disclosure Statement

No competing financial interests exist.

References

1. Sakr S, Liu A, Fayoumi AG. The family of MapReduce and large scale data processing systems. arXiv:1302.2966, 2013.
2. Melnik S, Gubarev A, Long JJ, et al. Dremel: interactive analysis of web-scale datasets. Proceedings of the 36th International Conference on Very Large Data Bases 2010, pp. 330–339.
3. Hausenblas M, Villazon-Terrazas B, Cyganiak R. Data shapes and data transformations. arXiv:1211.1565, 2012.
4. Dean J, Barroso LA. The tail at scale. Communications of the ACM 2013; 56:74–80.

Address correspondence to:

Michael Hausenblas
MapR Technologies, Europe
32 Bushypark Lawn
Galway
Ireland

E-mail: mhausenblas@maprtech.com

This article has been cited by:

1. P. Coetzee, S.A. Jarvis. 2017. Goal-based composition of scalable hybrid analytics for heterogeneous architectures. *Journal of Parallel and Distributed Computing* **108**, 59-73. [CrossRef]
2. Astrid Rheinl?nder, Ulf Leser, Goetz Graefe. 2017. Optimization of Complex Dataflows with User-Defined Functions. *ACM Computing Surveys* **50**:3, 1-39. [CrossRef]
3. Ahmad Ghazal, Todor Ivanov, Pekka Kostamaa, Alain Crolotte, Ryan Voong, Mohammed Al-Kateb, Waleed Ghazal, Roberto V. Zicari. BigBench V2: The New and Improved BigBench 1225-1236. [CrossRef]
4. Pietro Colombo, Elena Ferrari. Towards a Unifying Attribute Based Access Control Approach for NoSQL Datastores 709-720. [CrossRef]
5. Herodotos Herodotou. Business Intelligence and Analytics: Big Systems for Big Data 7-49. [CrossRef]
6. Thomas Vanhove, Merlijn Sebrechts, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, Filip De Turck. 2017. Data transformation as a means towards dynamic data storage and polyglot persistence. *International Journal of Network Management* **39**, e1976. [CrossRef]
7. Pradeeban Kathiravelu, Ashish Sharma. A Dynamic Data Warehousing Platform for Creating and Accessing Biomedical Data Lakes 101-120. [CrossRef]
8. Neda Abolhassani, Teresa Tung, Karthik Gomadam, Lakshmish Ramaswamy. Knowledge Graph-Based Query Rewriting in a Relational Data Harmonization Framework 433-438. [CrossRef]
9. Jinsong Wu, Song Guo, Jie Li, Deze Zeng. 2016. Big Data Meet Green Challenges: Greening Big Data. *IEEE Systems Journal* **10**:3, 873-887. [CrossRef]
10. Yang Liu, Yukun Zeng, Xuefeng Piao. High-Responsive Scheduling with MapReduce Performance Prediction on Hadoop YARN 238-247. [CrossRef]
11. Unai Aguilera, Diego López-de-Ipiña, Jorge Pérez. 2016. Collaboration-Centred Cities through Urban Apps Based on Open and User-Generated Data. *Sensors* **16**:7, 1022. [CrossRef]
12. Edmon Begoli, Ted Dunning, Charlie Frasure. Real-Time Discovery Services over Large, Heterogeneous and Complex Healthcare Datasets Using Schema-Less, Column-Oriented Methods 257-264. [CrossRef]
13. Linda Eggert, Yingjie Fan, Stefan Voß. Data-Intensive Analytics for Cat Bonds by Considering Supply Chain Risks 136-147. [CrossRef]
14. Parke Godfrey, Jarek Gryz, Piotr Lasek, Nasim Razavi. Interactive Visualization of Big Data 3-22. [CrossRef]
15. Gwendal Daniel, Gerson Sunyé, Jordi Cabot. UMLtoGraphDB: Mapping Conceptual Schemas to Graph Databases 430-444. [CrossRef]
16. Bao Rong Chang, Hsiu-Fen Tsai, Yo-Ai Wang, Chia-Yen Chen. Realization of secondary indexing to NoSQL database with intelligent adaptation 449-452. [CrossRef]
17. Dominik Moritz, Daniel Halperin, Bill Howe, Jeffrey Heer. 2015. Perfopticon: Visual Query Analysis for Distributed Databases. *Computer Graphics Forum* **34**:3, 71-80. [CrossRef]
18. Chris Douglas, Carlo Curino. Blind men and an elephant coalescing open-source, academic, and industrial perspectives on BigData 1523-1526. [CrossRef]
19. Prabha Susy Mathew, Anitha S. Pillai. Big Data solutions in Healthcare: Problems and perspectives 1-6. [CrossRef]
20. Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, Carlo Curino. Apache Tez 1357-1369. [CrossRef]
21. Bao Rong Chang, Hsiu-Fen Tsai, Chia-Yen Chen, Chien-Feng Huang, Hung-Ta Hsu. 2015. Implementation of Secondary Index on Cloud Computing NoSQL Database in Big Data Environment. *Scientific Programming* **2015**, 1-10. [CrossRef]
22. P. Coetzee, M. Leeke, S. Jarvis. 2014. Towards unified secure on- and off-line analytics at scale. *Parallel Computing* **40**:10, 738-753. [CrossRef]
23. Alberto Fernández, Sara del Río, Victoria López, Abdullah Bawakid, María J. del Jesus, José M. Benítez, Francisco Herrera. 2014. Big Data with Cloud Computing: an insight on the computing environment, MapReduce, and programming frameworks. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **4**:5, 380-409. [CrossRef]
24. Todor Ivanov, Sead Izberovic, Nikolaos Korfiatis. The Heterogeneity Paradigm in Big Data Architectures 218-245. [CrossRef]