# Cleanix: a Parallel Big Data Cleaning System

Hongzhi Wang
Harbin Institute of Technology
wangzh@hit.edu.cn

Mingda Li
Harbin Institute of Technology
limingda@hit.edu.cn

Yingyi Bu
University of California, Irvine
yingyib@ics.uci.edu

Jianzhong Li
Harbin Institute of Technology
lijzh@hit.edu.cn

Hong Gao
Harbin Institute of Technology
honggao@hit.edu.cn

Jiacheng Zhang
Tsinghua University
chinahitzjc@gmail.com

## ABSTRACT

For big data, data quality problem is more serious. Big data cleaning system requires scalability and the ability of handling mixed errors. Motivated by this, we develop Cleanix, a prototype system for cleaning relational Big Data. Cleanix takes data integrated from multiple data sources and cleans them on a shared-nothing machine cluster. The backend system is built on-top-of an extensible and flexible data-parallel substrate— the Hyracks framework. Cleanix supports various data cleaning tasks such as abnormal value detection and correction, incomplete data filling, de-duplication, and conflict resolution. In this paper, we show the organization, data cleaning algorithms as well as the design of Cleanix.

## 1. INTRODUCTION

Recent popular Big Data analytics applications are motivating both industry and academia to design and implement highly scalable data management tools. However, the value of data not only depends on the quantity but also relies on the quality. On one side, due to the high volume and variation, those Big Data applications suffer more data quality issues than traditional applications. On the other side, efficiently cleaning a huge amount of data in a shared-nothing architecture has not been well studied yet. Therefore, to improve the data quality is an important yet challenging task.

Many data cleaning tools [1] have been proposed to help users to detect and repair errors in data. Although these systems could clean data effectively for many datasets, they are not suitable for cleaning Big Data due to the following three reasons. First, none of the existing systems can scale out to thousands of machines in a shared-nothing manner. Second, various error types such as incompleteness, inconsistency, duplication, and value conflicting may co-exist in the Big Data while most existing systems are ad-hoc. As examples, CerFix [2] focuses on inconsistency while AJAX [3] is for de-duplication and conflict resolution. The last but not least, existing systems often requires users to have specific data cleaning expertise. For example, CerFix [2] requires users to understand the concept of conditional functional dependency (CFD), while AJAX [3] needs users to express data cleaning tasks with a declarative language. However, many real-world users do not have a solid data cleaning background nor understand the semantics of a specific data cleaning language.

In order to address the fundamental issues in existing systems and support data cleaning at a large scale, we design and implement a new system called Cleanix. The key features of Cleanix are listed as follows.

- **Scalability**. Cleanix performs data quality reporting tasks and data cleaning tasks in parallel on a shared-nothing cluster. The backend system is built on-top-of Hyracks [4], an extensible, flexible, scalable and general-purpose data parallel execution engine, with our user-defined data cleaning second-order operators and first-order functions.
- **Unification**. Cleanix unifies various automated data repairing tasks for errors by integrating them into a single parallel dataflow. New cleaning functionalities for newly discovered data quality issues could be easily added to the Cleanix dataflow as either user-defined second-order operators or first-order functions.
- **Usability**. Cleanix does not require users to be data cleaning experts. It provides a simple and friendly graphical user interface for users to select rules with intuitive meanings and high-level descriptions. Cleanix also provides a bunch of visualization utilities for users to better understand error statistics, easily locate the errors and fix them.

The main goal of this demonstration is to present the Cleanix system architecture and execution process by performing a series of data integration and cleaning tasks. We show how the data cleaning operators are used to clean data integrated from multiple data sources.

## 2. RELATED WORK

Big data has been very popular among academical and industrial fields. However, due to the features of

four *V*s, which means Volume, Velocity, Variety and Value, we may face the problems of data quality easily and need to detect and solve the errors in data. The detection of error means to find dirty items. According to the difference among methods, there are three types of methods.

- **Entity identification** The entity identification means to find the different items representing the same thing in the real world. By entity identification, we can detect the phenomenon of duplication. There has already been several methods for entity identification [5].
- **Error detection according to rules** To utilize rules during the detection we can use variable kinds of rules such as the functional dependencies [6], conditional functional dependencies [7], and so on. [7] designs an auto-detection algorithm based on the SQL language to find the items going against the conditional functional dependencies and extending inclusion dependencies.
- **Error detection based on master data** The main data is a high-quality data set to provide a synchronous consistent view. For example, [8] gives a relatively complete theory to describe the integrity of main data and the complexity of the users' queries.

The repairing for error means the modification or supplement to the data with error to improve the quality. According to different thoughts, the methods for repairing can be divided into three parts.

- **Repairing by rules** Repairing by rules mainly means to modify the data and make it satisfy the rules provided by managers. The [9] provides a repairing algorithm GREEDY_REPAIR. The algorithm is expended from the above method and uses the conditional functional dependencies for repairing. The [10] repairs the inconsistent data by the graph theory.
- **Truth Discovery** To solve the conflict data during entity resolution, we use truth discovery algorithm. The [11] uses the iteration method to calculate the truth degree of the source and the self-confidence degree of the value. [12] considers dependency among data sources, which is calculated from the self-confidence of the value.
- **Machine learning** Machine learning methods are mainly used for repairing incomplete data. The methods based on machine learning include decision tree, Bayesian network and Neural Network.

[13] is the demo plan of this paper. This paper introduces the detail of design and techniques in Cleanix.

# 3. SYSTEM OVERVIEW

## 3.1 Data Cleaning Tasks

Cleanix aims to handle four types of data quality issues in a unified way:

- *Abnormal value detection and correcting* is to find the anomalies according to users' options of rules and modify them to a near value that coincides with the rules.
- *Incomplete data filling* is to find the empty attributes in the data and fill them with proper values.
- *De-duplication* is to merge and remove duplicated data.
- *Conflict resolution* is to find conflicting attributes in the tuples referring to the same real-world entity and find the true values for these attributes. For example, tuples referring to the same person may have different values in age, but only one value should be chosen.

We believe that these four data cleaning tasks cover most data quality issues. Note that even though some data errors could not be processed directly such as nonconcurrency and inconsistency, one can take care of them by dynamically deploying new first-order user-defined functions into our system. For example, nonconcurrency can be processed as conflict resolutions among the data referring to the same real-world entity.

## 3.2 The Hyracks Execution Engine

We use Hyracks as backend to accomplish the above tasks efficiently at large scales. Hyracks is a data-parallel execution engine for Big Data computations on shared-nothing commodity machine clusters. Compared to MapReduce, Hyracks has following advantages:

- **Extensibility**. It allows users to add data processing operators and connectors, and orchestrate them into whatever DAGs. However, in the MapReduce world, we need to cast the data cleaning semantics into a scan (map)—group-by(reduce) framework.
- **Flexibility**. Hyracks supports a variety of materialization policies for repartitioning connectors, while MapReduce only has local file system blocking-materialization policy and HDFS materialization policy. This allows Hyracks to be elastic to different cluster configurations., e.g., behaving like a parallel database style optimistic engine for small clusters (e.g., 200 machines) but a MapReduce style pessimistic engine for large clusters (e.g., 2000 machines).
- **Efficiency**. The extensibility and flexibility together lead to significant efficiency potentials., e.g., one can implement the hybrid-hash style conflict resolution on Hyracks but not on MapReduce.

Several cloud computing vendors are developing non-MapReduce parallel SQL engines such as Impala [1] and Stinger [2] to support fast Big Data analytics. However, these systems are like "onions" [14]—one cannot directly use their internal Hyracks-like engines under the SQL skin for data cleaning. However, the Hyracks software

[1] https://github.com/cloudera/impala
[2] http://hortonworks.com/blog/100x-faster-hive/

stack is like a layered "parfait" [14] and Cleanix is yet-another parfait layer on-top-of the core Hyracks layer.

## 3.3 Cleanix Architecture

Cleanix provides web interfaces for users to input the information of data sources, parameters and rule selections. Data from multiple data sources are preprocessed and loaded into a distributed file system—HDFS[3]. Then each slave machine reads part of data to start cleaning. The data cleaning dataflow containing second-order operators and connectors is executed on slaves according to the user specified parameters and rules (e.g., first-order functions). At end of dataflow, the cleaned data are written to HDFS. Finally, cleaned data are extracted from HDFS and loaded into desired target database.

# 4. THE SYSTEM INTERNALS

In this section, we discuss the details of the Cleanix data cleaning tasks, pipeline, the algorithmic operators and the profiling mechanism.

## 4.1 Integration of Data

Before cleaning data, we need to download data to our system. We can support downloading from different kinds of databases including MySQL and MSSQL. While integrating items from different source databases, there will be a problem of conflicts among primary keys. To solve this, we add a new column as new primary key and a new column to show source of an item.

## 4.2 Data Cleaning Algorithms

The four data cleaning tasks including abnormal value detection and correcting, incomplete data filling, de-duplication and conflict resolution all have their own implementation algorithms. These algorithms are developed for parallel platform. We will introduce solutions to the four tasks respectively as following.

### *4.2.1 Abnormal Value Detection and Correcting*

Before executing the part, users can select data type for each attribute from types provided by Cleanix. Meanwhile, they can set cleaning rules for attributes. The rules include the detection rule for data type, legal range and the correcting rule after detecting abnormal data. According to the rules, we can detect and do simple filling in this cleaning part. For example, to the Numeric(integer or float) Attribute: *Detection Rule*: Set the maximum number and minimum number to detect abnormal value. *Filling Rule*: **a**. Simple Filling: Fill with fixed number or date. **b**. Intelligent Filling: According to attributes provided by users, which are relevant to the abnormal attribute, we find similar items. Using the items' attributes relevant to the abnormal value, we get suitable result by choosing the biggest, smallest, most frequent, least frequent, average number of them.

[3] `http://en.wikipedia.org/wiki/Apache_Hadoop`

### *4.2.2 Data Imputation Algorithm*

To fill incomplete data, the system needs to find items similar to the incomplete items. For big data, we need an efficient algorithm to obtain the similarity. There are two common algorithms called Edit Distance and 2-Gram Table. We choose the 2-Gram Table [15] and give up the Edit Distance due to its high complexity ($O(n^2)$), working process not suitable for parallelization(compare one string with all others) and drawback for some common phenomenons such as reversal of names. To build the 2-Gram table, we firstly build an empty Hash Table with two columns called Key and Number. Each item of Key Column is a String and each item of the Number Column's is an Integer Array. Then, for all the items, we read strings and take each string's 2-character substring as Hash's key. If there has already been such a key in the table, we add the number to the end of corresponding array. Otherwise, we add a new item into the table with the key and number of the string.

With the 2-Gram table, the similarity between strings can be calculated by the following process: **Firstly**, build another Hash table. There are two columns in the table called Key and Amount. Each item of Key Column is an integer to show the number of a string. Each item of Amount Column an integer to record the number of the same substrings. **Then**, traverse all 2-character substrings. Regard substring *s* as Key and inquire it in the 2-Gram table we have got. **Thirdly**, if there is such a Key in 2-Gram table, we regard the items in the corresponding array as key and insert them into Hash table. We initialize the corresponding amount as 1. If there is already such key, we add the corresponding amount with 1. **Finally**, after traverse to strings, we traverse the Hash table we build this time. Key represents string and Amount represents the number of the same substrings. So the bigger Amount represents more similar string.

After getting the similarities, if it is larger than a threshold, we regard the item similar to the incomplete item. Put all similar items into an ArrayList. According to filling rule set by users, we complete the imputation.

### *4.2.3 De-duplication Algorithm*

De-duplication is also called entity identification. We divide this duty into two parts. One part is the Grouping Algorithm and the other part is the Merging In A Group.

*Grouping Algorithm* The kernel idea is to group the whole data. We put the similar items into the same group. Meanwhile, we promise different items are in different groups. So the First Problem is to separate the items which are similar but represent different things.

The Second Problem is to put the items, which are not similar but represent the same thing, together. For the First Problem, we firstly use the Grouping Algorithm to put them together. For the Second Problem, we set a formula to calculate the similarity. We add other attributes and let users to set the weight. Thus, we can ensure the items, which are not similar but represent the same thing, in one group. In our algorithm, users can set many attributes as relevant attributes in De-duplication. By setting weight for each relevant attribute and a threshold, if the similarity calculated is larger than the threshold, we think the two items are the same and put them in the same group.

*Merging In One Group* After grouping, master node will get an array whose elements represent a set for similar items.Then we do the Merging In One Group. **Firstly**, Master Node traverses the sets in the array generated from the Grouping Algorithm. **Then**, Master Node use the greedy method to send each set to different Slave Node and try to make each node deal the same amount of data. **Finally**, each Slave Node uses the effective clustering approach introduced in [16]. We can know the complexity of the Merging In One Group algorithm is $O(n)$. The main theory is that if there are two items' similar substrings are more than a threshold in a group, we will regard them as duplicate items.

#### 4.2.4 Conflict Resolution Algorithm

Conflict Resolution is to solve the conflicts while merging many duplicate items to one item. During the process, some attribute may be different.The system will automatically choose the strings which appear most frequently. Meanwhile, the users can also set their own rules to solve the problem. For example, to date attribute, users can choose the earliest, latest, most frequent and least frequent date to fill in. The working process of the Conflict Resolution is as following: **Firstly**, Master Node sends the users' rules to each slave node. **Then**, Slave Nodes traverse the repeated items set. If there is any conflict among the items, we solve the conflicts by the rules set by users. If there is no user rule, we can automatically choose the most frequent one. **Finally**, each Slave Node sends back the items to Master Node and Master Node collects the items from Slave Nodes and get the final result.

#### 4.2.5 Share Information and Output Result

*Share information among cluster* In the working process for parallel cluster, we always need to share information. The easiest method shown in Figure 1 is to let all slave nodes send their own information to the master node. Then, the master node sends the integrated information back. This method is easy, but has some problems. For example, when many slave nodes send data to master node in the meantime, there may be blocking for the network and the overlap of memory in master node. Therefore, we design an annular transmission algorithm to solve the above problem as the Figure 2. We suppose there are $m$ nodes called $NC_1$, $NC_2$....$NC_m$. Suppose the data they store is $n_1$,....$n_m$. We only need to circle for $m$-1 times to finish sharing information. In each circling, each Slave Node only needs to send his own data.

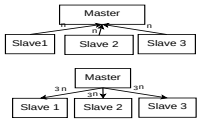


Figure 1: Primary Method to Send Data

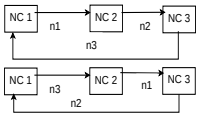


Figure 2: Improved Method to Send Data

*Output Result* During the Conflict Resolution, we have finished the data collection. But when we collect items from different sources, we will face a problem of conflict to the Key. So we build a new ID column as a new key to solve the conflict. Meanwhile, we will add the column describing the data source.

### 4.3 Data Processing Ordering

To make the discussion brief, we use A, I, D and C to represent the modules of the process of abnormal value detection and correcting, incomplete data filling, de-duplication and conflict resolution, respectively. The order of four tasks of data cleaning in Cleanix is determined with the consideration of effectiveness and efficiency. These four modules could be divided into two groups. Module A and I are in the same group (Group 1) sharing the same detection phase since the detection of abnormal values and empty attributes can be accomplished in a single scan of the data. Module D and C are in the same group (Group 2) since the identifications of entities with the entity resolution operator are required for both de-duplication and conflict resolution. De-duplication merges tuples with the same entity identification while conflict resolution is to find true values for conflicting attributes for the different tuples referring the same entity identification. The reason why Group 1 is executed before Group 2 is that the repairation of abnormal values and empty attributed will increase the accuracy of entity resolution. In Group 1, Module A is before I since abnormal values interfere the incomplete attribute filing and lead to incorrect fillings. In Group 2, Module D is before C since only when different tuples referring to the same entity are found and grouped, the true values of conflicting attributes could be found.

### 4.4 Dataflow

The dataflow graph is shown in Figure 3. The dataflow has 8 algorithmic operators and 4 stages, where the computation of each stage is "local" to each single

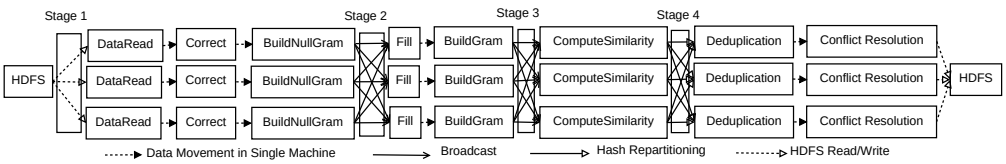


Figure 3: The Cleanix Dataflow Graph

machine and the data exchange (e.g., broadcast or hash repartitioning) happens at the stage boundaries. In following part, we illustrate the algorithmic operators and the rules for each stage in topological order in Figure 3.
**Stage 1**.This is performed on each slave machine.

- *DataRead*. It scans incoming file splits from the HDFS. The data are parsed and translated into the Cleanix internal format.
- *Correct*. This is blocking operator—data are checked according to the rules selected by users to detect the abnormal values and incomplete tuples. When an abnormal value is detected, it is corrected according to corresponding revision rules (first-order functions). When an incomplete tuple is encountered, it is identified for further processing.
- *BuildNullGram*. This operator builds an inverted list for all incomplete tuples for the imputation based on similar tuples. The inverted list is called *gram table*. It is a hash table in which the $k$-gram is the key and the id set of tuples containing such a $k$-gram is the value.

**Stage 2**. The incoming broadcast connector to this stage broadcasts the gram tables such that all slaves share the same global gram table.

- *Fill*. For each tuple with incomplete attribute, similar tuples are found according to the gram table. The incomplete attribute is filled with the aggregated value of the corresponding attribute in similar tuples according to the imputation rules (first-order functions) selected by users such as average, max or the most frequent.
- *BuildGram*. A local gram table is built for the local data for the attributes potentially containing duplications or conflicts, which are chosen by users. Since a local gram table has been built with BuildNullGram operator, only the newly filled values of corresponding attributes are scanned in this step.

**Stage 3**.The local gram tables are broadcast to make all slaves share the same global gram table. Note: only updated values in local gram tables are broadcast in Stage3.

- *ComputeSimilarity*. The similarities between each local tuple and other tuples are computed according to the global gram table. When the similarity between two tuples is larger than a threshold, they are added to the same group and form many groups finally.

**Stage 4**.Groups are partitioned according to hashing value of bloom filter of the union of gram sets in group.

- *De-duplication*. A weighted graph $G$ is built to describe the similarity between tuples in each group. Similar vertices are merged iteratively in $G$ until no pairs of vertices can be merged [16]. This step is executed iteratively until the ratio between the number of shared connected vertices and the number of the adjacent vertices of each vertex is smaller than a threshold. The tuples corresponding to all merged vertices are considered as duplications.
- *Conflict Resolution*. Tuples corresponding to the merged vertices are merged. During merging, when an attribute with conflicting values is detected, it is resolved with voting according to selected rules chosen by users. Options (first-order functions) include max, min, average and the most frequent.

## 4.5 Data Cleaning Result

In the final part of the demonstration, we illustrate the exploration of data cleaning results and interaction of user and the system. More specifically, Cleanix will compare the repaired data with the original ones. The original and modified data are distinguished in different colors. When the user selects a modified value, the modifications are shown. Additionally, the user could modify the data. The modifications are merged when the cleaned data is transmitted from HDFS to the target database.

Besides, users can also check the data quality in high level. We can see how the violations are distributed among the data by different histograms and statistical categorization for both attribute and tuple level.

# 5. INTERFACE

The system allows users to add several machines to clean data from different data sources. You can input the name of new Slave Nodes in the NodeController Name as shown in Figure 4. And if we are cleaning data stored in different machine, we can input the IP address of the machine, Port, Username and Password to get access to the data at the bottom of the same page shown in Figure 4. After setting the database connection information, we can set the cleaning rules to find abnormal value and do filling as Figure 5 shows. Users can also set the weight and threshold to do de-duplication like Figure 6.

After setting the basic information for data cleaning, we can click the button *See the status of the working system and you need to start it to work here* and open the page like Figure 7. We can start working there and find the status of system. When system finishes cleaning, we can check the data cleaning result by inputting the range of item's ID. Cleaning results include four types of dirty items including *abnormal value*, *duplication*, *incomplete value* and *conflict*. Part of the four types of cleaning results is shown in Figure 8.

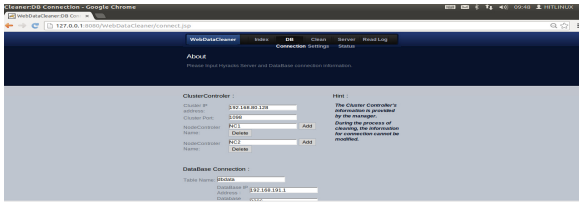

Figure 4: Add New Slave Nodes and Data Source

Figure 5: Set Data Cleaning Rules

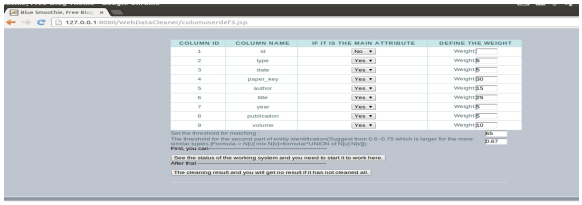

Figure 6: De-duplication

Figure 7: See the Working Status of the System

*Acknowledgements.* This paper was supported by NGFR 973 grant 2012CB316200 and NSFC grant 61472099,61133002.

Figure 8: *Part of Cleaning result* : Incomplete Value and Conflict

## 6. REFERENCES

[1] Thomas N. Herzog, Fritz J. Scheuren, and William E. Winkler. *Data quality and record linkage techniques*. Springer, 2007.

[2] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. CerFix: A system for cleaning data with certain fixes. *PVLDB*, 4(12):1375–1378, 2011.

[3] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[4] Vinayak R. Borkar, Michael J. Carey, Raman Grover, Nicola Onose, and Rares Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In *ICDE*, pages 1151–1162, 2011.

[5] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[6] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[7] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[8] Wenfei Fan and Floris Geerts. Relative information completeness. *ACM Trans. Database Syst.*, 35(4):27, 2010.

[9] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 143–154, 2005.

[10] Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: Consistency and accuracy. In *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 315–326, 2007.

[11] Amélie Marian and Minji Wu. Corroborating information from web sources. *IEEE Data Eng. Bull.*, 34(3):11–17, 2011.

[12] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. Integrating conflicting data: The role of source dependence. *PVLDB*, 2(1):550–561, 2009.

[13] Hongzhi Wang, Mingda Li, Yingyi Bu, Jianzhong Li, Hong Gao, and Jiacheng Zhang. Cleanix: A big data cleaning parfait. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 2024–2026, 2014.

[14] Vinayak R. Borkar, Michael J. Carey, and Chen Li. Inside "Big Data management": ogres, onions, or parfaits? In *EDBT*, pages 3–14, 2012.

[15] Esko Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–211, 1992.

[16] Lingli Li, Hongzhi Wang, Hong Gao, and Jianzhong Li. EIF: A framework of effective entity identification. In *WAIM*, pages 717–728, 2010.