BigDansing

A System for **<u>Big</u>** **<u>Da</u>**ta Clea**<u>nsing</u>**

Table of Content

- Introduction
- Data Cleansing Process
- Architecture of BigDansing
- Experiment
- Discussion

Background

- "Inaccurate data has a direct impact ... the average company losing 12% of its revenue" -- Ben Davis (Econsultancy)

- "More than 25 Percent of Critical Data Used in Large Corporations is Flawed" – Gartner Inc.

- "This is the digital universe. It is growing 40% a year into the next decade" -- EMC2

Concepts

Data Cleansing

Data cleansing, data cleaning, or data scrubbing is the process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database.

Data Quality

Data are of high quality “if they are fit for their intended uses in operations, decision making and planning ".

An Intuitive Example

Quality Rule

Two customers having the same zipcode cannot be in different cities

D(zipcode -> city)

Dirty Dataset

Name	Zipcode	City
Winnie	91340	San Francisco
Robbert	91340	New York
Emma	91340	San Francisco

An Intuitive Example

Quality Rule

Two customers having the same zipcode cannot be in different cities

D(zipcode -> city)

Dirty Dataset

Name	Zipcode	City
Winnie	91340	San Francisco
Robbert	91340	New York
Emma	91340	San Francisco

Challenges

In the context of big data

- **Scalability**

Can not scale to large datasets

- **Abstraction**

Need to understand both the quality rules

and the distributed platform

Challenges

In the context of big data

- **Scalability**

Can not scale to large datasets

- **Abstraction**

Need to understand both the quality rules

and the distributed platform

BigDansing

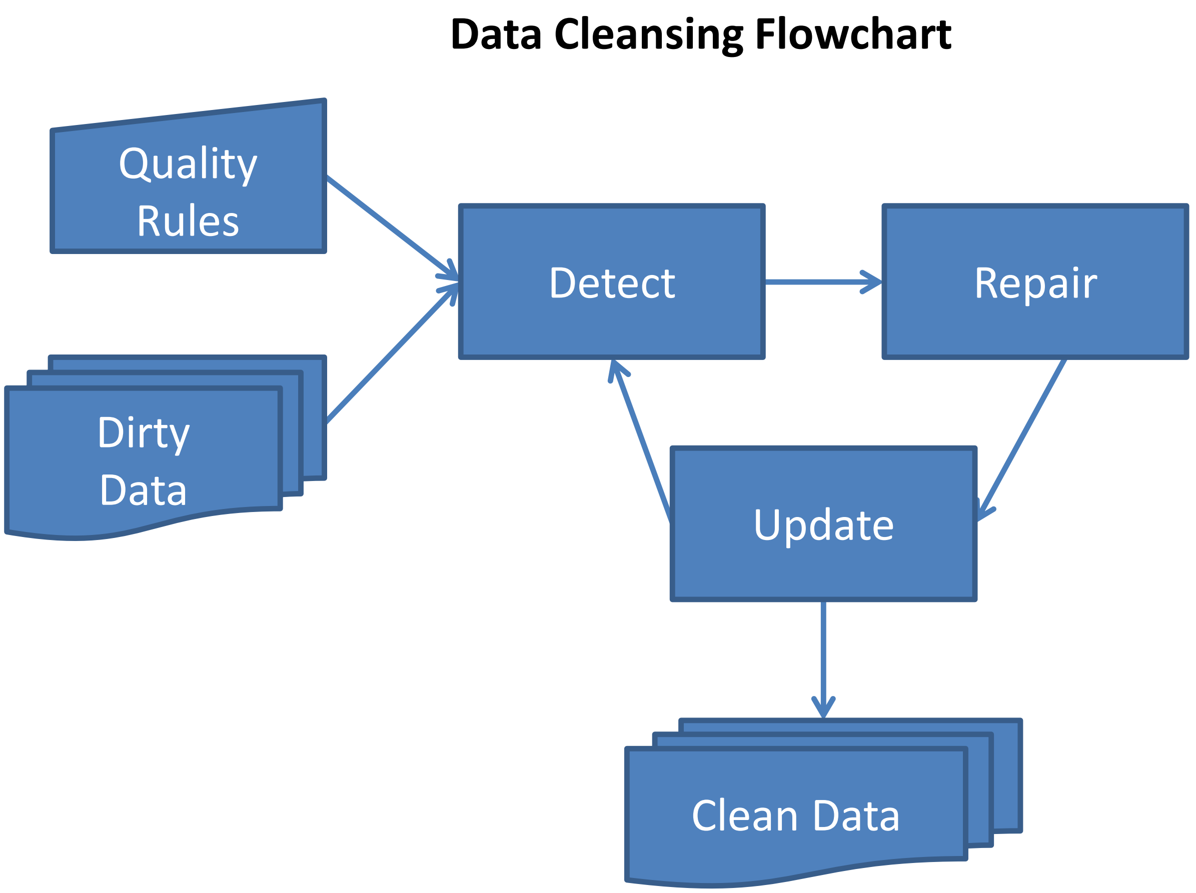
A Big Data Cleansing system to tackle efficiency, scalability, and ease-of-use issues in data cleansing

BigDansing

To address the challenges

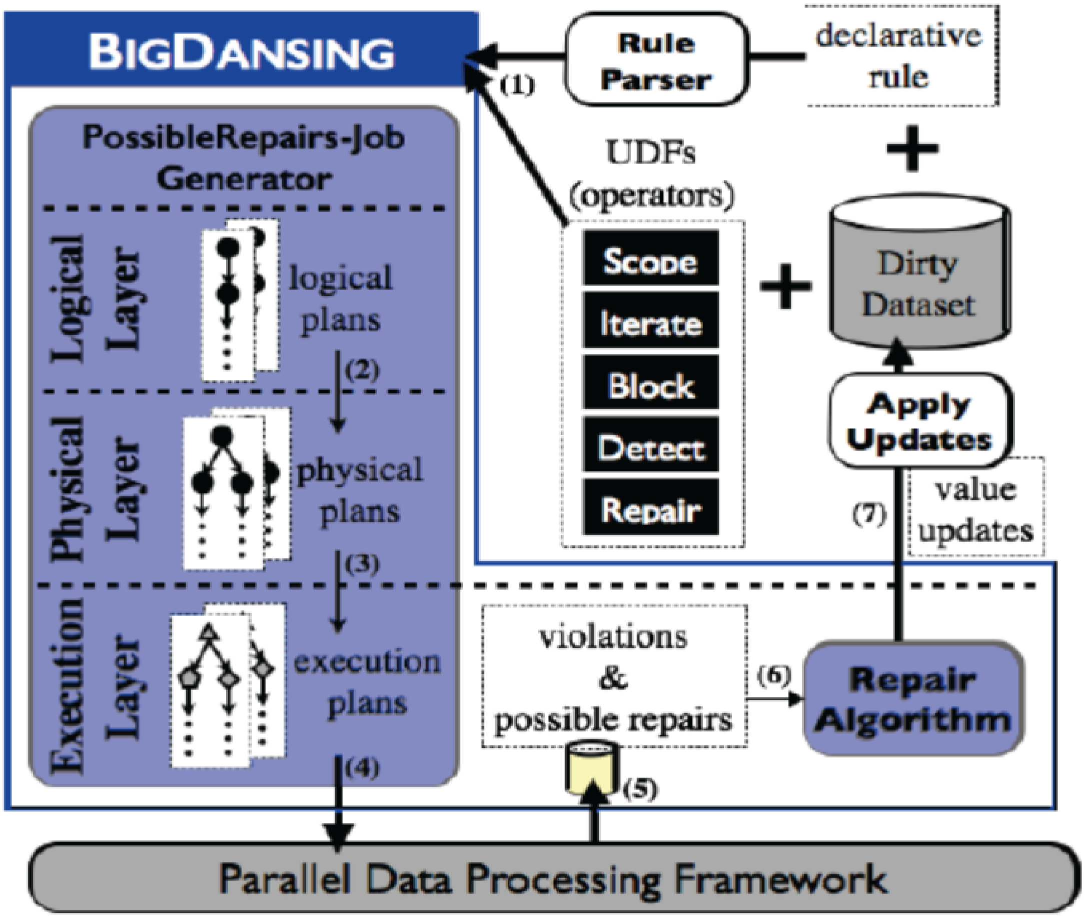
- Generic **abstraction**
  - define customized rules together with traditional rules in a simple way
  - the underlying distributed platform is transparent
- Fast and **scalable** detection, repair and updates
  - 1.9B rows → 13B violations < 3 hours on 16 small machines

Data Cleansing Process



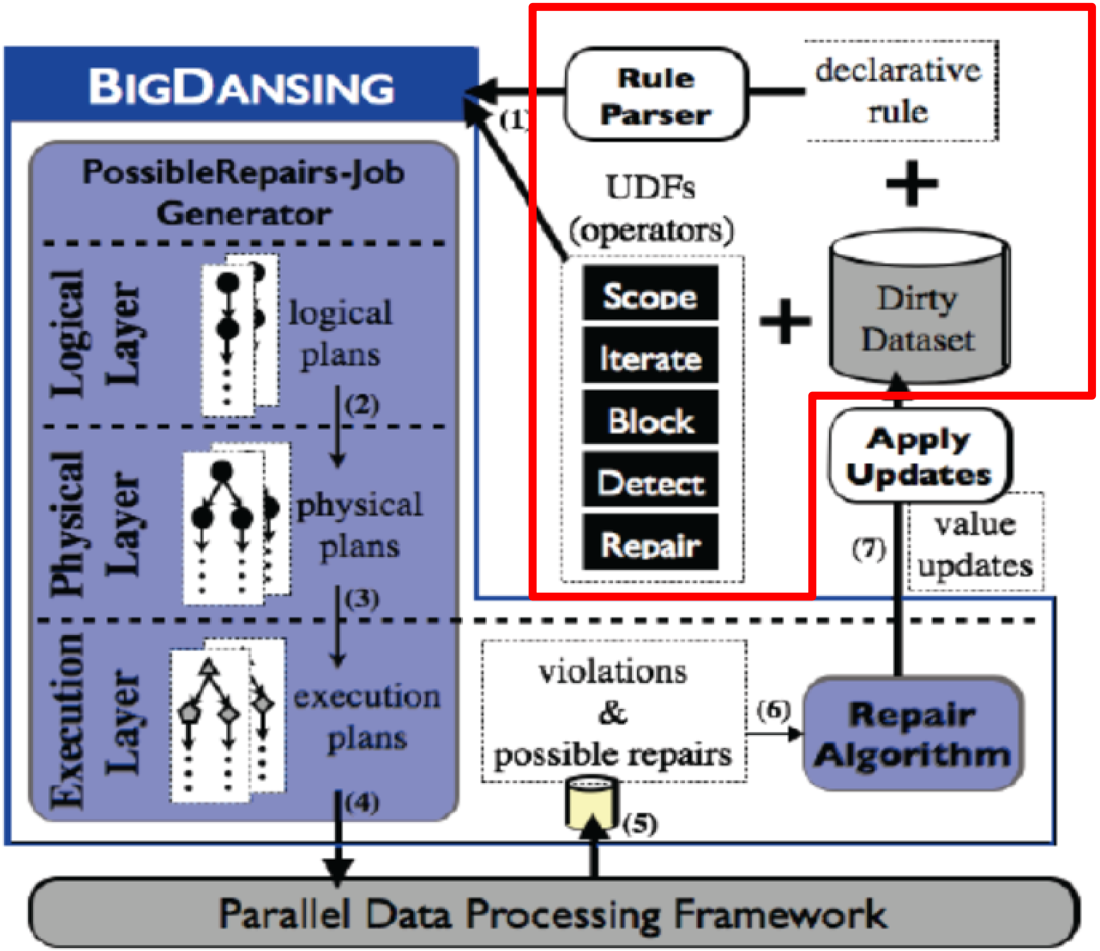
Source: https://www.slideshare.net/Zuhairkhayyat/bigdansing-presentation-slides-for-kaust

Architecture



Source: Zuhair Khayyat et al: BigDansing: A System for Big Data Cleansing. SIGMOD Conference 2015: 1215-1230

Architecture



Input:
quality rules
a dirty dataset

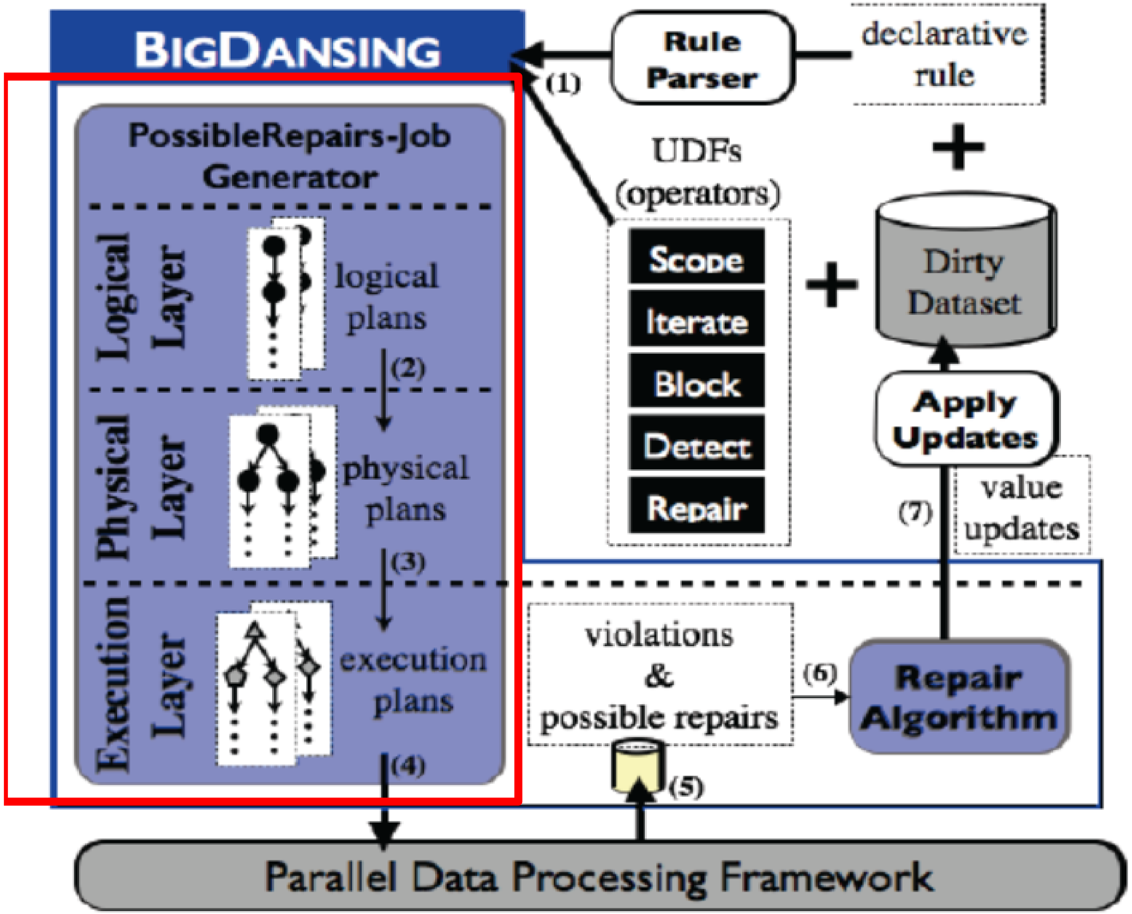
Architecture

RuleEngine:

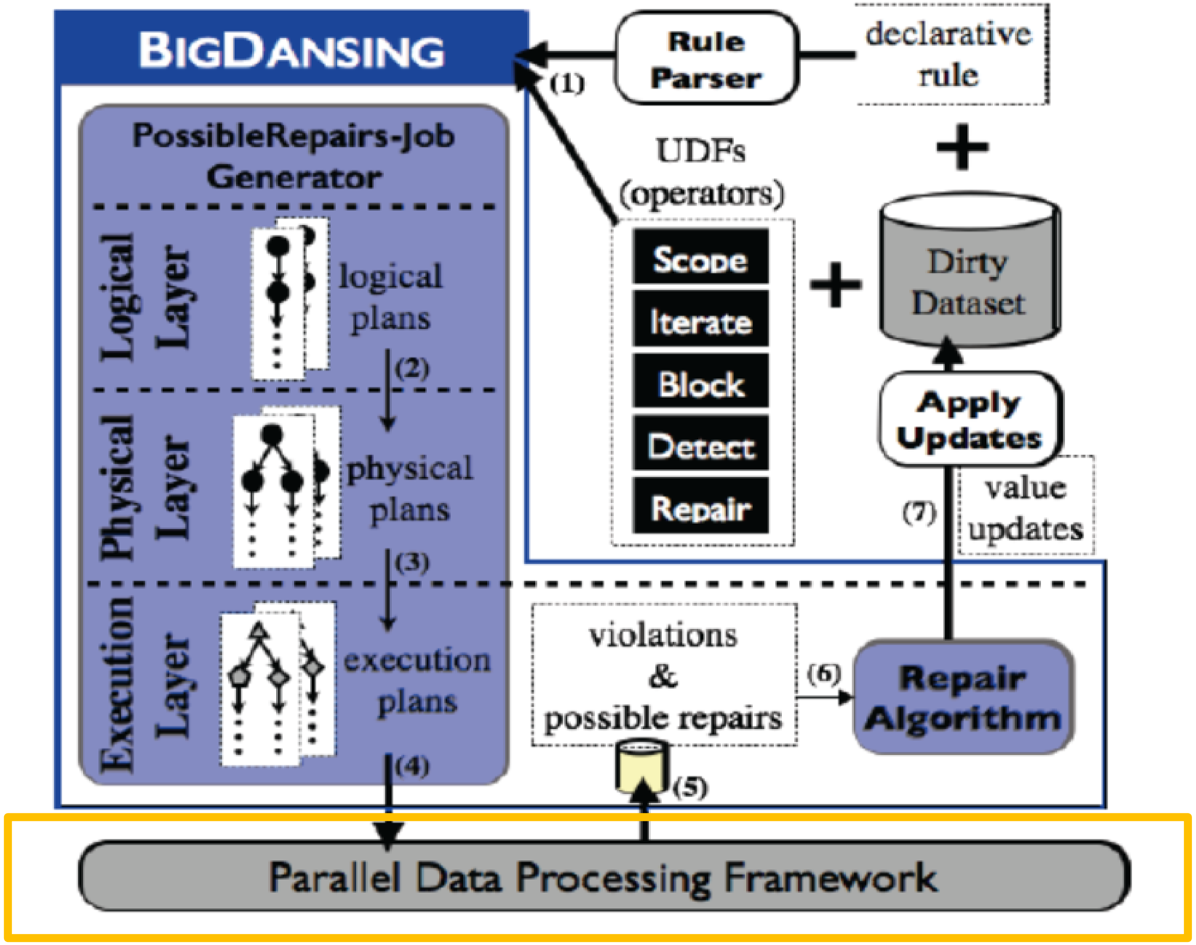
logical layer

physical layer

execution layer

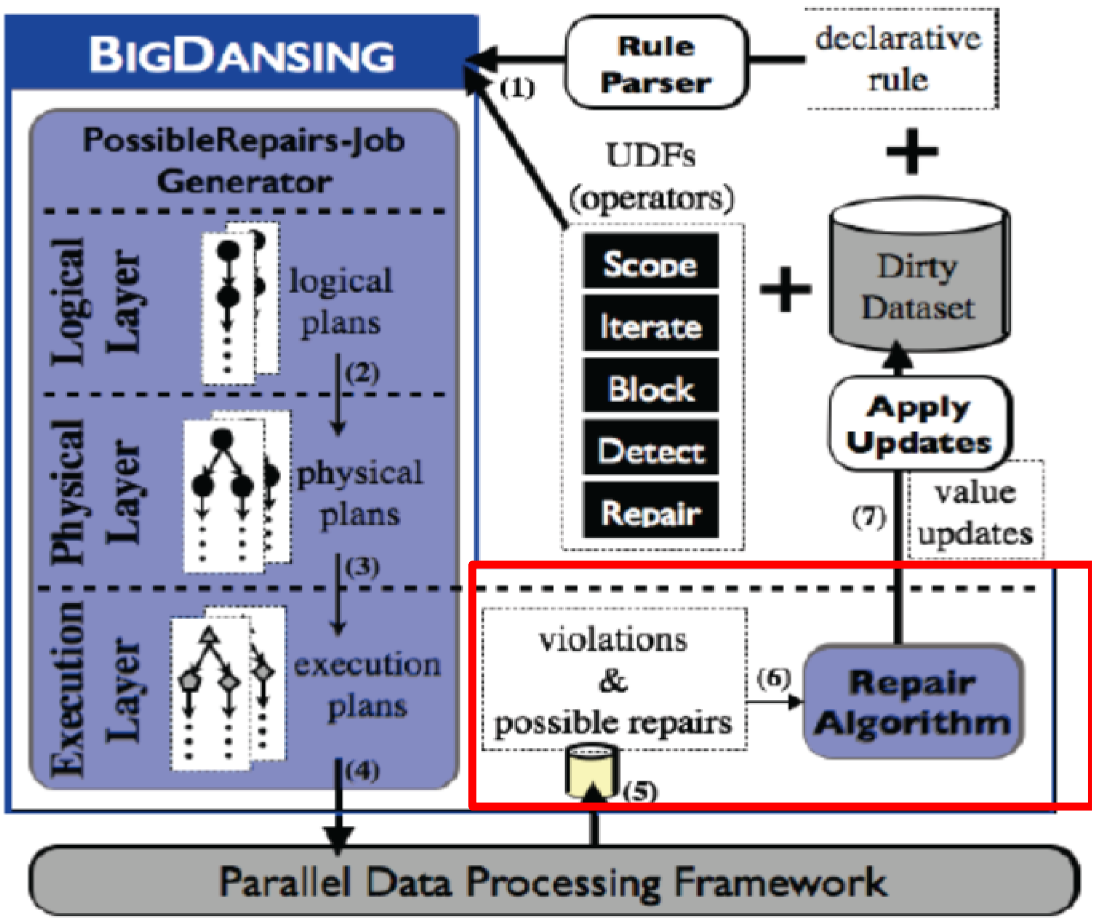


Architecture



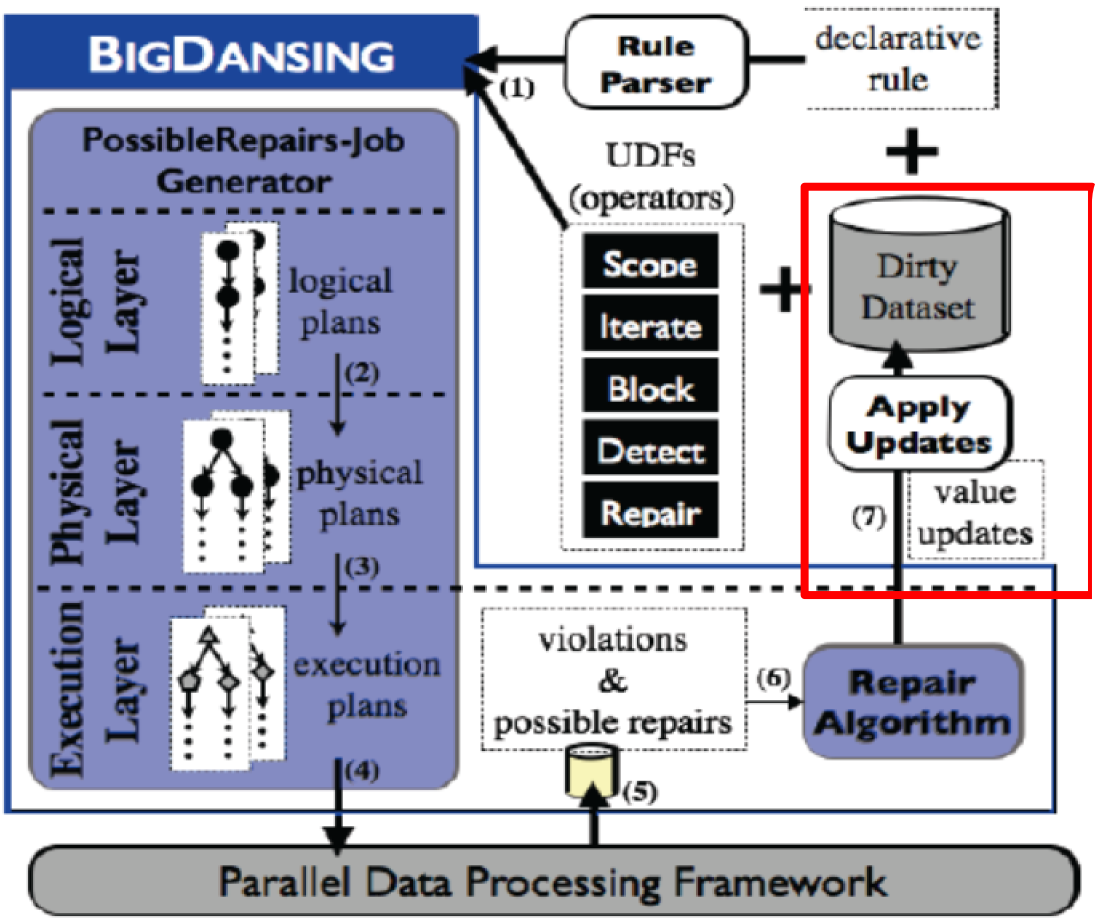
general purpose distributed platform like Hadoop or Spark

Architecture



Repair Algorithm : run existing centralized algorithm as a black box in a distributed way

Architecture

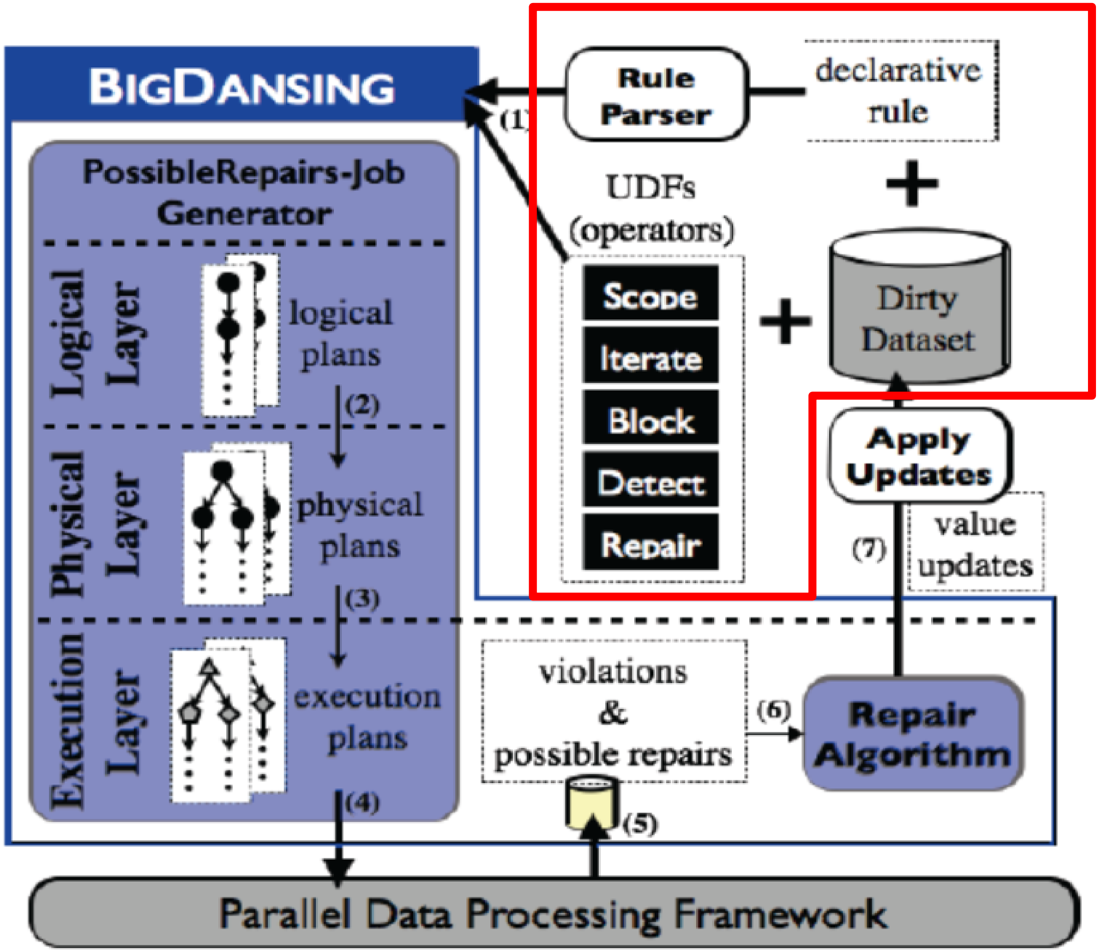


Output:
a clean dataset

Architecture

Let’s dig into the components one by one…

Architecture



Input:
quality rules
a dirty dataset

Quality Rule

Dirty data is detected by

- **Declarative Rule**
- Functional dependencies (FD)

  e.g. Zipcode → City
- Conditional functional dependency (CFD)

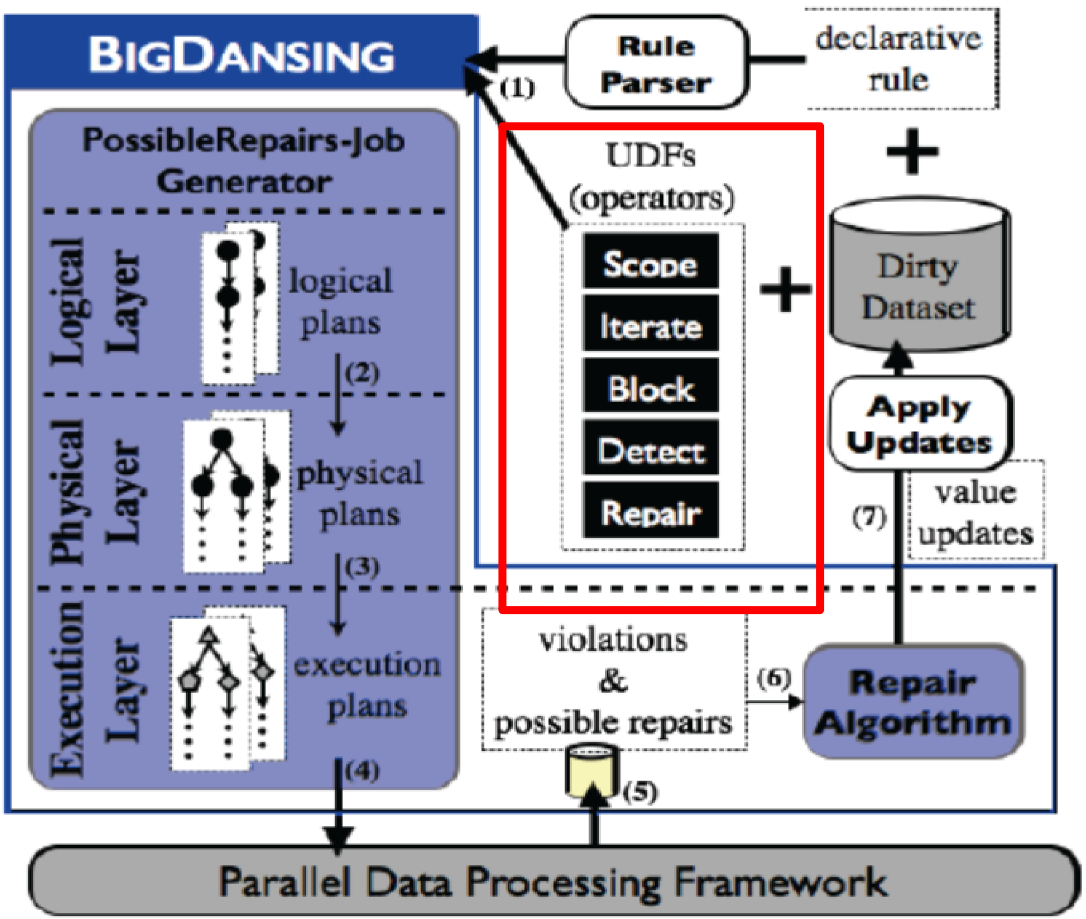
  e.g. Country = 'Saudi Arabia', Zipcode → City
- Denial constraints (DC)

  e.g. $\forall$ t1, t2 $\in$ D, $\neg$(t1.Salary > t2.Salary ^ t1.Rate < t2.Rate)

- **User Defined Function (UDF)**

Duplicates, statistical errors

Abstraction
UDF-based Approach



Abstraction
UDF-based Approach

- Quality rules are represented by 5 functions

Scope, Block, Iterate, Detect, and GenFix

- declarative rules

-automatically be translated into logical operators

- UDFs

-can be implemented using logical operators

Logic Operators
Scope

Scope

Input: data units

Output: data units

Example: Zipcode → City

Input: t1 – t6

Output:

t1 – t6 (Zipcode,City)

	Name	Zipcode	City
t1	Annie	60601	NY
t2	Laure	90210	LA
t3	John	60601	CH
t4	Mark	90210	SF
t5	Robert	60601	CH
t6	Mary	90210	LA

	Zipcode	City
t1	60601	NY
t2	90210	LA
t3	60601	CH
t4	90210	SF
t5	60601	CH
t6	90210	LA

Logic Operators
Block

Block

Input: data units

Output: grouping key

Example: Zipcode → City

Input: t1 – t6

Output:

<60601, (t1,t3,t5)>,

<90210, (t2,t4,t6)>

	Zipcode	City
t1	60601	NY
t2	90210	LA
t3	60601	CH
t4	90210	SF
t5	60601	CH
t6	90210	LA

Logic Operators

Iterate

Iterate

Input: a group of data units

Output: single tuple, tuple pair

Example: Zipcode → City

Input: <60601, (t1,t3,t5)>

Output:

<t1,t3>, <t1,t5>, <t3,t5>

	Zipcode	City
t1	60601	NY
t3	60601	CH
t5	60601	CH

Logic Operators

Detect

Detect

Input: data units

Output: Violation(s)

Example: Zipcode → City

Input: <t1,t3>, <t1,t5>, <t3,t5>

Output:

<t1.City ≠ t3.City>,

<t1.City ≠ t5.City>

	Zipcode	City
t1	60601	NY
t3	60601	CH
t5	60601	CH

Logic Operators
GenFix

GenFix

Input: Violation

Output: possible fix(es)

Example: Zipcode → City

Input: <t1.City ≠ t3.City>,

<t1.City ≠ t5.City>

Output:

<t1.City = t3.City>,

<t1.City = t5.City>

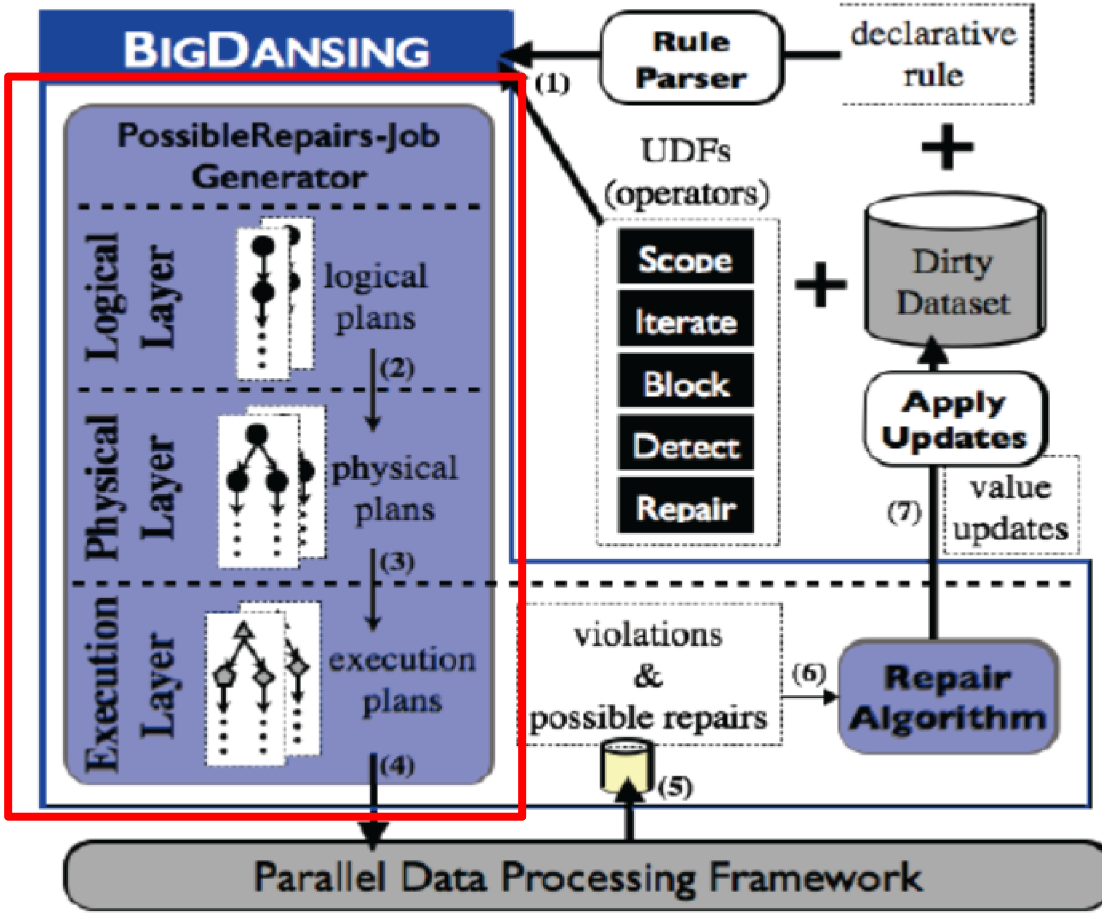
	Zipcode	City
t1	60601	NY
t3	60601	CH
t5	60601	CH

	Zipcode	City
t1	60601	CH
t3	60601	CH
t5	60601	CH

Rule Engine

RuleEngine

- three-layer
- on top of general purpose data processing framework such as MapReduce-like with execution abstraction



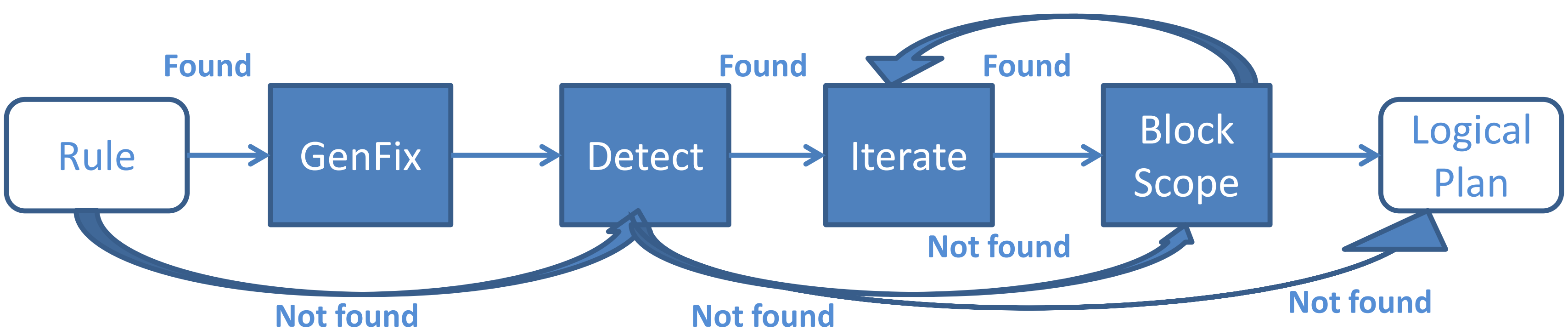
Rule Engine
Logical Plan

- Define the data unit flow

- Validating the plan:

At least one input dataset
For UDF: at least one detect
For Rules: at least one rule

- Support simple and bushy plans



Rule Engine
Physical Plan

- Physical operators are system specific
MPI, Hadoop, Spark

- Each physical operator is an independent execution unit

- Each logical operator → one physical operator

- BigDansing consolidate logical plans to improve I/O

- More physical operators can be added with different optimizations to improve logical plans

Rule Engine
Execution Plan

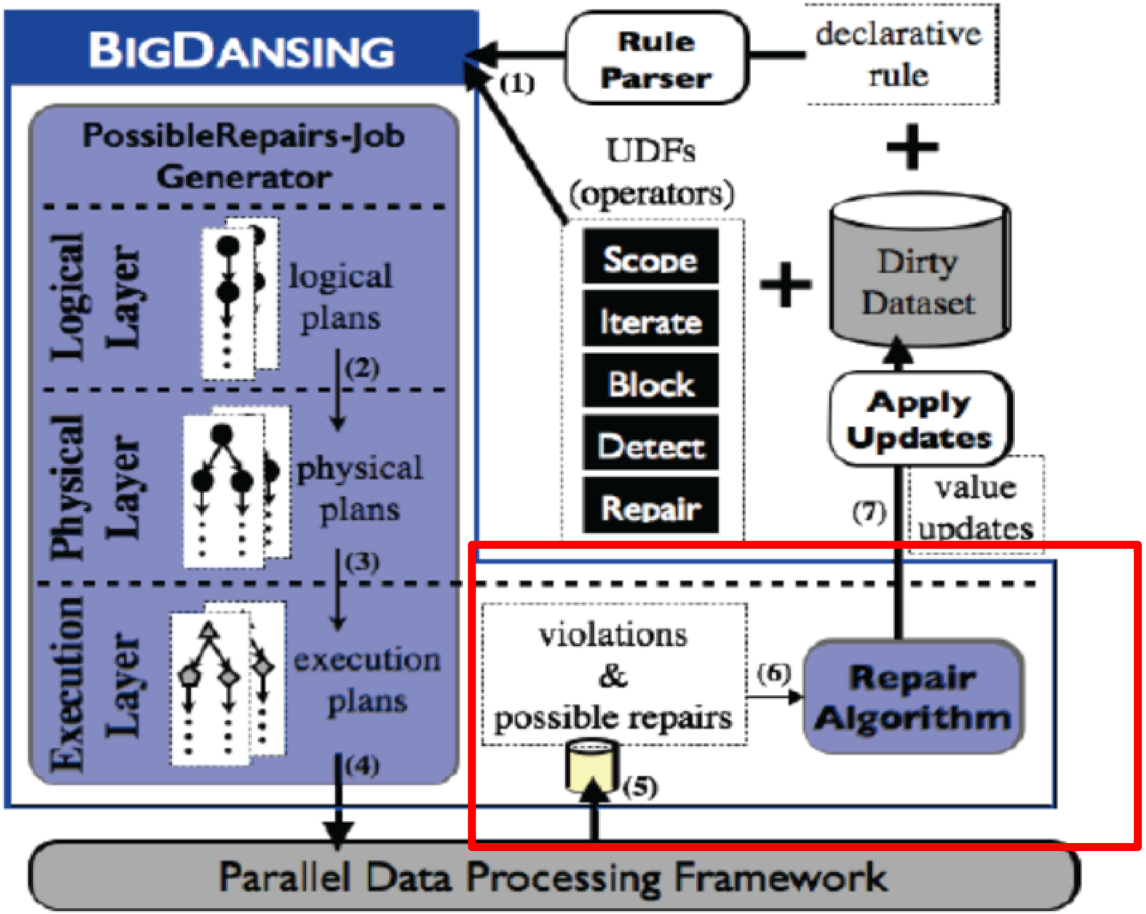
- Run on the parallel general purpose data processing framework such as MapReduce-like

- User don't need to care about how the distributed computation performs

Repair Algorithm

Repair Algorithm

- equivalence class
- hypergraph-based



Repair Algorithm

- Implement two existing **serial** repair algorithms to run in distributed mode
  - Equivalence class algorithm
  - Hypergraph algorithm
- Design a **distributed** version of the seminal equivalence class algorithm

Equivalence Class Algorithm

- Fix errors based on (=,≠)

- Based on heuristics:

  Partition the possible fixes into different groups

  Assign the highest frequency value to group

- Example:

  Group 1: Zipcode = 60601

  Highest frequency = CH

  Group 2: Zipcode = 90210

  Highest frequency = LA

	Name	Zipcode	City
t1	Annie	60601	NY
t2	Laure	90210	LA
t3	John	60601	CH
t4	Mark	90210	SF
t5	Robert	60601	CH
t6	Mary	90210	LA
t7	Jon	60601	CH

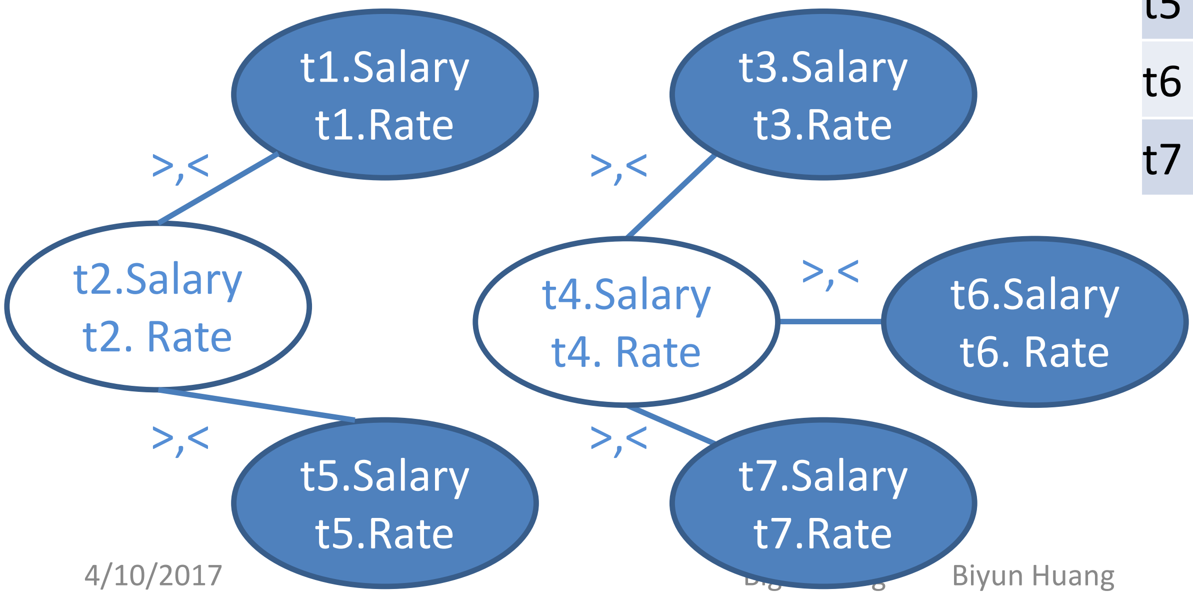
Hypergraph Algorithm

- Fix errors based on (<,>,≤, and ≥)

- Based on linear optimization and greedy MVC:

Select hyper-graph node with highest edges
Change its value depending on edge conditions

	Name	Salary	Rate
t1	Annie	24000	15
t2	Laure	25000	10
t3	John	40000	25
t4	Mark	88000	24
t5	Robert	15000	15
t6	Mary	81000	28
t7	Jon	40000	25



Experiment

Dataset

Dataset	Rows	Dataset	Rows
$TaxA_1$–$TaxA_5$	100K – 40M	customer2	32M
$TaxB_1$–$TaxB_3$	100K – 3M	NCVoter	9M
$TPCH_1$–$TPCH_{10}$	100K – 1907M	HAI	166k
customer1	19M		

More intuitively, TPCH datasets with 959M, 1271M, 1583M, and 1907M rows are of sizes **150GB**, **200GB**, **250GB**, and **300GB** respectively.

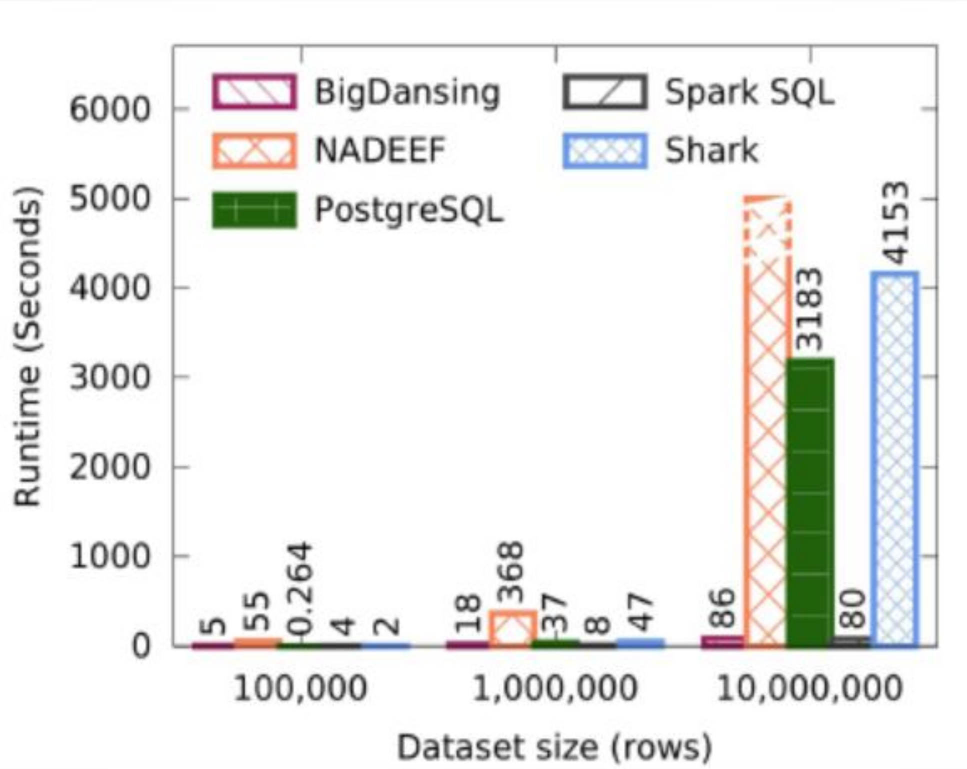
Experiment

Quality Rules

Identifier	Rule
φ_1 (FD):	Zipcode $\rightarrow$ City
φ_2 (DC):	$\forall t_1, t_2 \in TaxB, \neg(t_1.\text{Salary} > t2.\text{Salary} \wedge t_1.\text{Rate} < t_2.\text{Rate})$
φ_3 (FD):	o_custkey $\rightarrow$ c_address
φ_4 (UDF):	Two rows in Customer are duplicates
φ_5 (UDF):	Two rows in NCVoter are duplicates
φ_6 (FD):	Zipcode $\rightarrow$ State
φ_7 (FD):	PhoneNumber $\rightarrow$ Zipcode
φ_8 (FD):	ProviderID $\rightarrow$ City, PhoneNumber

Results

Single-node experiments with Φ1 (Functional Dependencies)

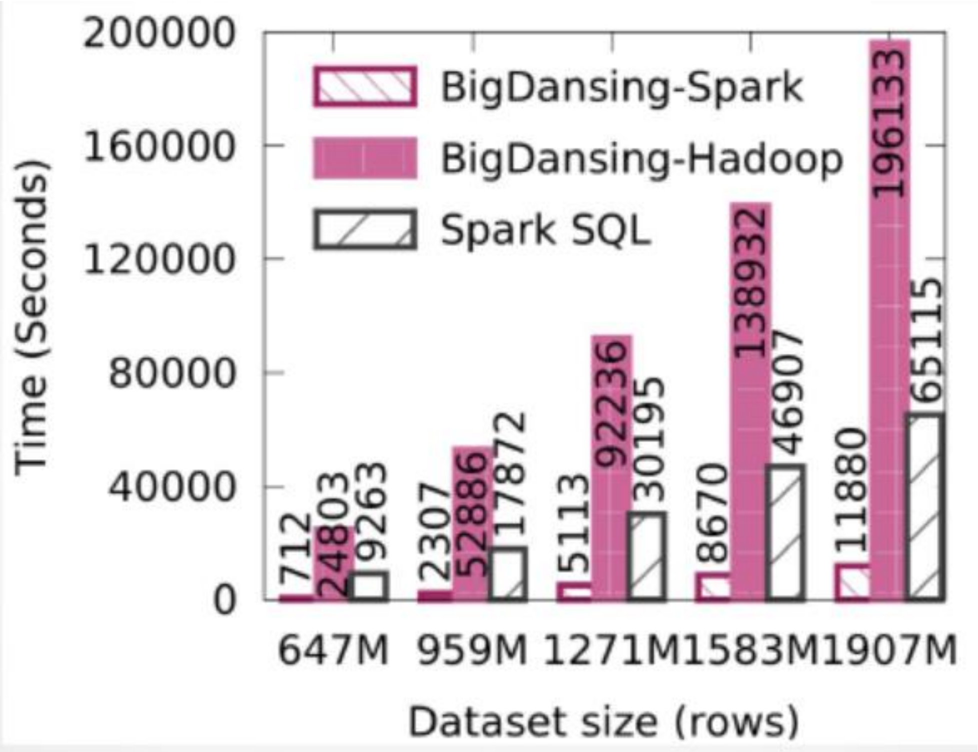


TaxA dataset:
FD: Zipcode → City
FD: Zipcode → State

provides a finer granular abstraction allowing users to specify rules more efficiently

Results

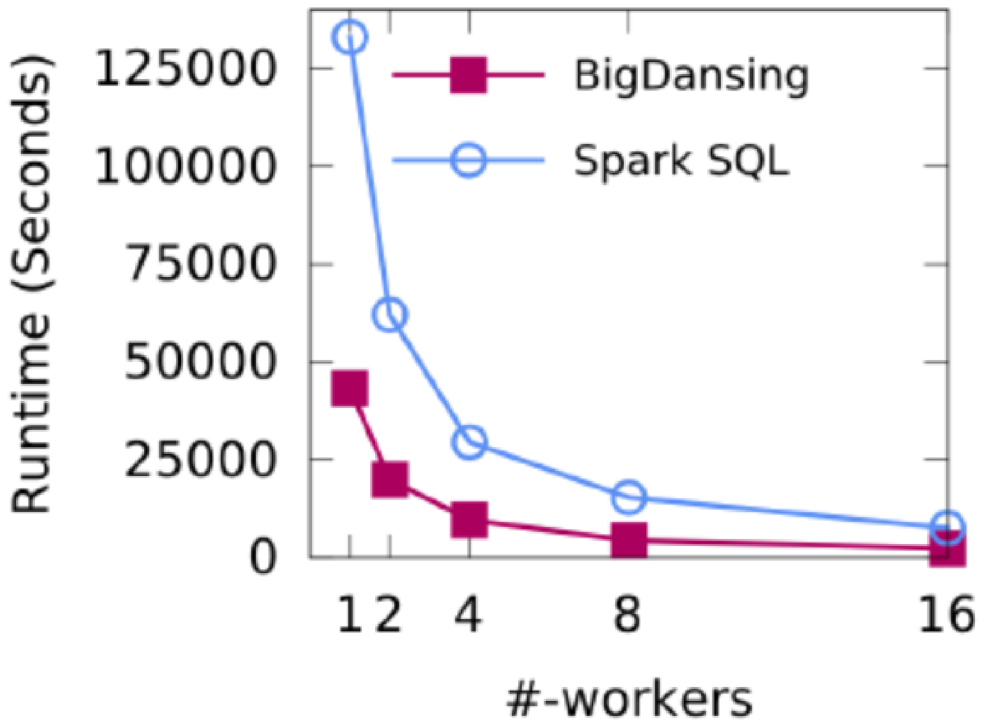
Multi-nodes experiments with Φ1(Functional Dependencies)



TPCH dataset:
FD: custkey →custAddress
16 Workers

Results

Scalability



TPCH Dataset:
FD: custkey → custAddress
Dataset: 500M rows

Results

Repair quality

Rule(s)	NADEEF precision	NADEEF recall	BigDansing precision	BigDansing recall	Itera-tions
φ_6	0.713	0.776	0.714	0.777	1
φ_6&φ_7	0.861	0.875	0.861	0.875	2
φ_6–φ_8	0.923	0.928	0.924	0.929	3
	|R,G|/e	**|R,G|**	**|R,G|/e**	**|R,G|**	**Iter.**
ϕ_D	17.1	8183	17.1	8221	5

TaxA/HAI Dataset:
Dataset: 100K--40M/166k rows

Results

Outperform existing baseline systems up to more than two orders of magnitude without sacrificing the repair quality!

Discussion
Contributions

- A data cleansing framework rather than a data cleansing algorithm

- very useful as the emerging of big data

- There have been extensive studies on data cleansing algorithm for decades.

But now the demand is to run these algorithms on distributed system.

Discussion
Contributions

- Flexibility and scalability

- **Logic Abstraction**
define customized rules together with traditional rules in a simple way with an UDF-based approach

- **Portable and Easy-to-use**
run on various platforms ranging from DBMS to MapReduce-like platforms

- **Flexibility of Repair Algorithm**
adopt existing algorithms or user-defined ones by treating the repair algorithm as a black box

- **Scalability and Fast Computation**
integrate on general purpose distributed framework

Discussion

Limitations

- When treating UDFs as black boxes, it is hard to do static analysis, such as consistency and implication, for the given rules.

- Dataset must be static and quality rules must be predefined.

  stream data / real-time cleansing / modified quality rules

- Require domain expertise.

  hinder its adoption in industrial targeting non-expert users

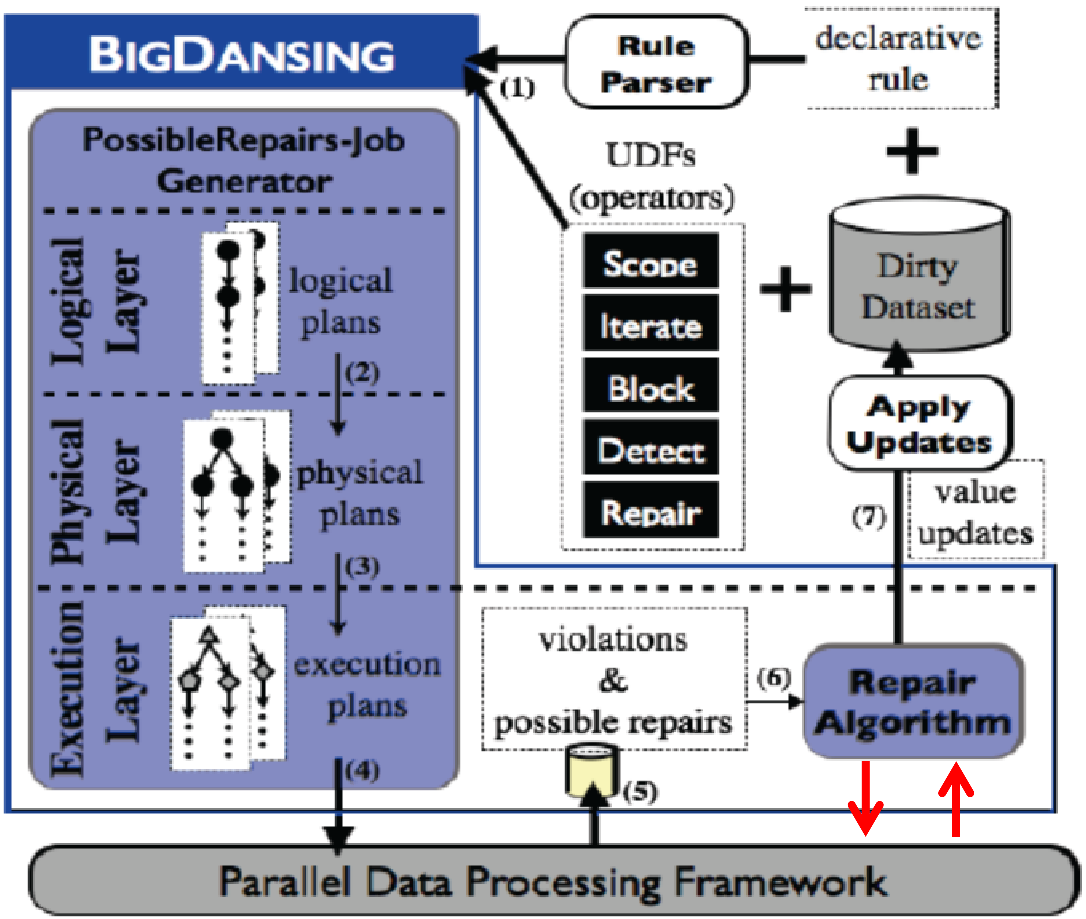
Discussion

Limitations

In the experiment, the authors added random text/numerical/duplicate errors and the system considers only key-based blockers, which is not accurate for many real-world data sets, due to dirty/missing data.

It is usually hard, if not impossible, to guarantee the accuracy of the data cleaning process without verifying it via experts or external sources.

Discussion
Limitations



Thanks!