ShareInsights - An Unified Approach to Full-stack Data Processing

Mukund Deshpande
Persistent Systems Ltd.
'Bhageerath', Senapati Bapat Road,
Pune 411005, INDIA
mukund_deshpande@persistent.com

Dhruva Ray
Persistent Systems Ltd.
'Bhageerath', Senapati Bapat Road,
Pune 411005, INDIA
dhruva_ray@persistent.com

Sameer Dixit
Persistent Systems Ltd.
'Bhageerath', Senapati Bapat Road,
Pune 411005, INDIA
sameer_dixit@persistent.com

Avadhoot Agasti
Persistent Systems Ltd.
'Bhageerath', Senapati Bapat Road,
Pune 411005, INDIA
avadhoot_agasti@persistent.com

ABSTRACT

The field of data analysis seeks to extract value from data for either business or scientific benefit. This field has seen a renewed interest with the advent of big data technologies and a new organizational role called data scientist. Even with the new found focus, the task of analyzing large amounts of data is still challenging and time-consuming.

The essence of data analysis involves setting up data pipelines which consists of several operations that are chained together - starting from data collection, data quality checks, data integration, data analysis and data visualization (including the setting up of interaction paths in that visualization).

In our opinion, the challenges stem from from the technology diversity at each stage of the data pipeline as well as the lack of process around the analysis.

In this paper we present a platform that aims to significantly reduce the time it takes to build data pipelines. The platform attempts to achieve this in following ways.

1. Allow the user to describe the entire data pipeline with a single language and idioms - all the way from data ingestion to insight expression (via visualization and end-user interaction).

2. Provide a rich library of parts that allow users to quickly assemble a data analysis pipeline in the language.

3. Allow for a collaboration model that allows multiple users to work together on a data analysis pipeline as well as leverage and extend prior work with minimal effort.

We studied the efficacy of the platform for a data hackathon competition conducted in our organization. The hackathon provided us with a way to study the impact of the approach. Rich data pipelines which traditionally took weeks to build were constructed and deployed in hours. Consequently, we believe that the complexity of designing and running the data analysis pipeline can be significantly reduced; leading to a marked improvement in the productivity of data analysts/data scientists.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.3.2 [**Language Classifications**]: [Data-flow languages]

General Terms

Management, Design, Experimentation, Human Factors, Languages.

Keywords

Big data; Data analysis; Data visualization; Data pipeline

1. INTRODUCTION

The field of data analysis has seen dramatic changes in the last few years. These changes are seen primarily on the dimensions of data, evolving user needs and technology changes.

On the data side of the equation, the volume of data that needs to be analyzed has become extremely large. It is now routine to analyze terabytes of data and in some cases, petabytes as well. The nature of the data is also multi-structured with text, audio as well as visual data being analyzed side-by-side with structured tabular data. In line with an increased amount of public data being made available,

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2758-9/15/05 ...$15.00.
http://dx.doi.org/10.1145/2723372.2742800.

the need for integrating it with private organization data has never been higher.

On the consumption side of the equation, users are making increasing demands on the data visualization component of data analysis. With the advent of multiple devices and form factors, data needs to be visualized and analyzed at multiple resolutions, possibly in environments with varying network connectivity. Also, the bar for aesthetics and ease of use for data visualization has gone up significantly. Users are demanding richer and more interesting widgets for displaying data and intuitive ways of interacting with them. Finally, with users being increasingly connected on networks (enterprise and social), the ability to share and collaborate with their peers for building and running the data pipeline is also becoming crucial.

On the technology side of the equation, the last few years have seen several technologies reaching mainstream - both big data technologies as well as advanced use of JavaScript for data visualization. Though big data technologies allow analyzing large data-sets, they have also brought an additional complexity of multiple technologies and different programming paradigms. Additionally, at every boundary, there remain integration challenges. Concomitantly, data crunching libraries in Python and R are seeing a revived interest and there appears to be a renewed interest in using these libraries in the data pipeline.

In our opinion, all of these challenges have made the task of doing data analysis increasingly difficult and time consuming.

The essence of data analysis involves setting up data pipelines which consists of several operations that are chained together - starting from data collection, data quality checks, data integration, data analysis and data visualization. (includes setting up interaction paths in that visualization)

Since, data pipelines operates on electronic media, the task of running and changing a data analysis pipeline is relatively easy; Consequently, in most cases, the data analysis pipeline is built and stabilized by incrementally building the pipeline. This iterative process mandates that it be very easy to change and edit an existing data pipeline. Sadly, that is not the state of the affairs[4].

In this paper, we present a platform which attempts to significantly ease the process of building complete data pipelines for businesses and data scientists. The platform and the supposed benefits were tested by running a data hackathon. Data pipelines and dashboards which took weeks to implement were completed in under six hours.

The paper is organized as follows. In section 2, we explain current approaches and the challenges faced in building data pipelines. In section 3, we illustrate key concepts of the unified representation provided on the ShareInsights platform by building complex data pipelines. Section 4 describes the key building blocks and features provided by the platform. Section 5 reports the findings of the hackathon used to verify the utility of the platform, Finally we conclude the paper with observations and future areas of work.

2. BUILDING A DATA ANALYSIS PIPELINE

In this section, we outline the current approaches to building data pipelines and the challenges faced by businesses and data scientists.

2.1 Current Approaches

2.1.1 Data Warehouse stack

This stack uses an extract-transform-load (ETL) mechanism to ingest, clean, transform and ultimately load processed data into the data warehouse[8]. Reporting tools query the warehouse to give reports to the end-user. The typical architecture is shown in figure 1.

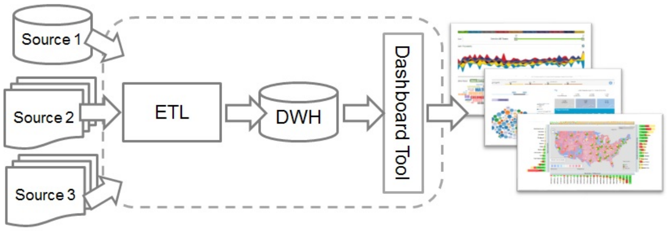


Figure 1: BI Architecture for data pipelines

This configuration is widely used and there is a relative maturity in the tooling and understanding of the data pipeline process. Almost all available tools provide a user interface for each step of the ETL process including reporting. Even so, building the pipeline requires investment in tool knowledge and relational data theory. Due to it's relational lineage, this stack was traditionally not adept at handling unstructured data.

2.1.2 Big Data stack

This stack leverages the Hadoop platform to do the traditional ETL processing described earlier. ETL is done with a map-reduce paradigm (as well as other programming styles) and uses a cluster of commodity hardware to scale[6]. Support for handling unstructured data is more natural on this platform.

With the advent of powerful client machines, fast JavaScript engines and rich widget libraries, JavaScript has become the de facto choice for creating interactive data visualizations.

The canonical architecture for this stack is shown in figure 2.

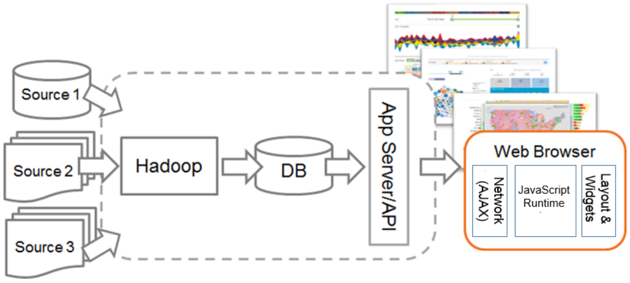


Figure 2: Big Data Architecture for data pipelines

2.2 Challenges

Both these approaches have serious challenges that limit the efficacy of developers building the data pipeline in organizations[14].

1. Multiple Technologies:
   Data visualization technologies/tooling and data transformation technologies/tooling are inherently different. Apart from requiring different mind-sets, multiple technology stacks bring their attendant problems of data serialization, interface design and the like. This problem is more acute for the Big Data stack where engineers need to glue reporting to the transformed data.

2. Unavailability of Skills:
   Given the diverse technology pieces present in the data pipeline, lack of skills is an acute problem that the industry faces. This translates to a lower team velocity to build data pipelines. The problem is made more acute as there is an explosion of technology options at each phase of the data pipeline.

3. Lack of Collaboration Model:
   Although most technologies (whether it is ETL or reporting) provide lot of libraries to work with, the abstraction model is not necessarily enforced. It is quite common for the complexity to spread both in the library as well as the glue code that puts systems together. This problem is more acute for the Big Data stack - that uses JavaScript based interactive visualization - where the user needs to resort to significant custom programming. This hampers collaboration and it is very hard to leverage prior work. More importantly, since there is no systemic enforcement of abstractions, existing code needs to be understood before re-use. This leads to significant overheads.

3. SHAREINSIGHTS PLATFORM

The platform's unique contribution is to present an unified representation across the data stack so that common idioms can be used for the **entire** data pipeline. Since data pipeline architects are dealing with one logical model for the whole pipeline, we believe that it would translate into a faster turnaround.

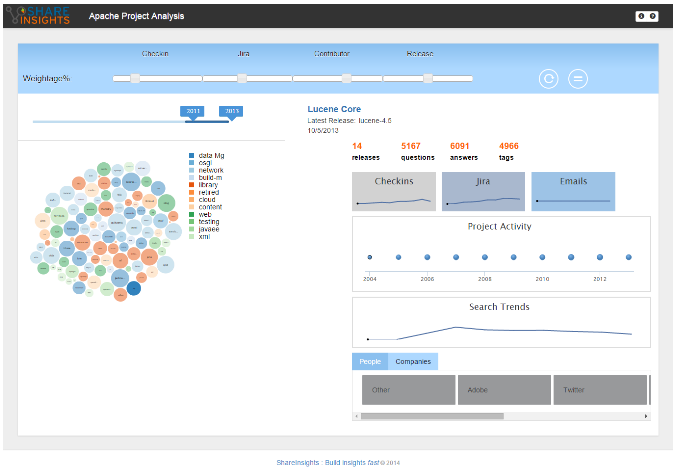
In the subsequent sections, we explain the key abstractions and techniques provided by the platform using examples.

Use Case : Analysis of Apache Open Source Projects.

The Apache activity analysis dashboard analyzes open source projects on http://www.apache.org. It computes a project activity index to allow for relative comparison of project activity in the same (and/or complementary) technology category . This dashboard is shown in figure 3.

The raw data for this dashboard comes from bug tickets, project commit history, stack overflow traffic and project collaborators information. We allow dashboard users to tweak the weightage given to each of the four parameters - check-ins, bug issues, contributors and releases (the four sliders at the top of the dashboard) to determine the overall project activity (and thereby relevance) in the project cloud. Once the weight factors have been decided, users can change the date range of their analysis using the provided date slider. Finally, clicking on a project bubble will reveal statistics about the selected project.(figure 13).

In the following sections, we demonstrate the key abstractions of our system using this example.



Lucene Core
Latest Release: lucene-4.5
10/5/2013

14 releases
5167 questions
6091 answers
4966 tags

Figure 3: Apache Open Source Project Analysis

3.1 Flow File

All data pipelines are expressed in a flow file. Each flow file has a name and contains the data pipelines associated with it. Each flow file has a Data (D) configuration section, a Widget (W) configuration section, a Task (T) configuration section, a Flow (F) configuration section - to process data objects through tasks (T) - and a Layout (L) configuration section to position widgets (W) in the dashboard.

These sections are elaborated below.

3.2 Data Section

This section contains the data objects (sources and sinks) used by the data pipeline. Every dashboard needs data objects to work off from. The data section allows the dashboard to configure these data objects. Data object configuration involves specifying the protocol and protocol parameters used to fetch the data as well as parameters to understand the payload.

The platform provides popular protocol connectors - such as File (local, remote), HTTP/S, FTP, JDBC, ad-hoc queries over JDBC - and recognizes popular data payload formats such as CSV, AVRO, XML and JSON documents.

A sample data object configuration described in figure 4 indicates that the data source is a CSV file. (present in the data directory of the dashboard)

```
D.stack_summary :
     separator : ','
     source : 'stackoverflow.csv'
     format : 'csv'
```

Figure 4: Data source configuration

Additionally, users need to explicitly call out the schema of the payload as shown in figure 5.

```
stack_summary:
   [project,question,answer,tags]
```

Figure 5: Data source schema

These column names may be used to configure tasks which transform data sources into other forms. In this example, the data object columns are questions, answers and tags related to the projects being analyzed by this data pipeline. Even though, this example shows data read from a local file, the data object can be configured to directly talk to the provider APIs. This is shown in figure 6. The => notation maps JSON paths in the payload to column names.

```
D:
   stack_questions : [
       question => title ,
       tags => tags ,
   ]
D.stack_questions:
   source : https://api.stackexchange.com/
   2.2/questions?order=desc&
   sort=activity&site=stackoverflow

   protocol : http
   format : json
   request_type : get
   http_headers :
     X-Access-Key : XXX
```

Figure 6: Configure data source with provider APIs

3.3 Task Section

The task section contains all the task configurations used by the data pipeline.

Tasks operate on data objects and are consumed in data flows (shown in section 3.4). The platform comes pre-loaded with a set of useful transformations. The input data to a task is determined contextually; the preceding data object or task (tasks also output temporary data objects). Tasks are generally configured by specifying column names to operate on.

For example a filter task would use a column name in a filter expression as shown in figure 7. Note that the task configuration assumes that it will be used in a context where the data source has a **rating** column.

```
classification :
     type : filterby
     filter_expression :rating < 3
```

Figure 7: Filter task

Tasks can be reused in multiple flows as long as the preceding data source has the column the task consumes.

Task transformations can add columns (e.g. join operation) or reduce columns (e.g. group operation) or preserve columns (e.g. filter operation) to the transformed data source.

3.4 Flow Section

The flow section contains a collection of flows. Each flow allows data-sets to be transformed into other forms. We leverage the Unix pipe notation to express data transformation semantics.

A sample flow is shown in the F section of figure 8. The flow transforms **svn_jira_summary** to **checkin_jira_emails data sink** with a groupby task (using a composite key).

```
D:
 ...
   svn_jira_summary:[project ,year ,noOfBugs
     noOfCheckins ,noOfEmailsTotal  ...]

   checkin_jira_emails:[project ,year ,
       total_checkins ,total_jira ,
       total_emails  ]
F:
 ...
D.checkin_jira_emails :
  D.svn_jira_summary  | T.get_svnjira_count
T:
 ...
get_svnjira_count :
     type : groupby
     groupby : [project ,year]
     aggregates :
          - operator : sum
            apply_on : noOfCheckins
            out_field : total_checkins
          - operator : sum
            apply_on : noOfBugs
            out_field : total_jira
          - operator : sum
            apply_on : noOfEmailsTotal
            out_field : total_emails
```

Figure 8: A data transformation flow

We constrain each flow to start with (optionally) multiple data sources (i.e fan-in) and end with (optionally) multiple data outputs (fan-out). The input data objects are transformed to output data objects via transform (or task) operators configured in the task section. Data sinks are automatically configured by the system. (As outlined in section 3.2). Note that data-sink is purely a concept for clarity of explanation. The system internally makes no differentiation between a data source and a data sink.

3.4.1 *Sharing processed data objects*

Data objects (sources or sinks) can be shared in the following ways.

Enable Remote Access.

To make the data object available to the dashboard (over a network connection), specify **endpoint** to true as shown in figure 9. Data objects which hold aggregated or summarized data for visualization purposes would set this property to true.

Enable Group Access.

To make the data object available to other dashboards, specify a name by which this data object will be referenced by other dashboards. In figure 10, **checkin_jira_emails** is published as a named data source - **project_chatter**. Other dashboards can use this data object by name without having to configure it in their own dashboards. (The platform

```
D.checkin_jira_emails:
    endpoint : true
       OR
#+ is an alias for endpoint :true
+D.checkin_jira_emails:
   D.svn_jira_summary|T.get_svnjira_count
```

Figure 9: A data sink configured as an endpoint

searches for this data object - in the shared objects list - when referenced in another dashboard)

This concept allows for creation of dashboards with the sole intention of building flows to transform raw data sources into a shared set of data entities on the platform.

```
D.checkin_jira_emails:
       publish : project_chatter
       endpoint : true
```

Figure 10: A published data sink

Data sources and sinks can be configured with both publish and endpoint values. If none of these keywords are used, it is assumed to be a throw-away data source/sink.

3.4.2 Linear flow specification

Data sinks can also be input to other flows as demonstrated in figure 11.

```
D.temp_release_count : D.releases
                | T.calculate_total_release

D.rel_qa_tags : (
          D.temp_release_count ,
          D.stack_summary
) | T.combine_stack_summary
```

Figure 11: A flow using intermediate data object

Note that the user can only specify simple (as in linear) flows. Since data sinks can become data sources for other flows, it is possible to build up arbitrarily complicated transformation paths.

On submission, the platform internally builds a directed acyclic graph (DAG) from the collection of flows specified by the user.

3.5 Widget Section

The widget section contains a collection of widget configurations. Widgets show transformed (endpoint) data and needs to be minimally configured with a data source and the type of widget (Line, Pie, Bubble etc). The platform comes pre-loaded with a set of commonly used widgets. Every widget has a set of attributes which associate (or bind) with data source columns. These attributes are called data attributes or widget columns[9]. The remaining attributes of a widget are visual attributes. Visual attributes are available in various categories such as legend, axis, dimension configurations and the like.

```
D:
 ...

 project_data:[project,year,total_wt...]

W:
 ...

 project_technology_bubble :
   # Data Attributes
   type : BubbleChart
   source : D.project_data | T.get_date
            | T.aggregate_project_bubbles
   text : project
   size : total_wt
   legend_text : technology
   # Visual Attributes
   default_selection : True
   default_selection_key : text
   default_selection_value : 'pig'
   ...
   legend :
     show_legends : true
     ...
```

Figure 12: Widget configuration

Figure 12 shows how the project bubble widget is configured using project_data (endpoint data) as a source. The widget columns are **text** and **size** and are bound to the **project** and **total_wt** columns of the data schema respectively.

Note that multiple widgets can be configured with the same data source.

Most of the widgets used by the Apache dashboard are provided by the platform. The widget which lets the user decide the relative weight of each parameter to compute project activity index is a custom widget - written using the platform extension APIs.(Section 4.2)

3.5.1 Widget to Widget Interaction

Dashboards allow users to interact with widgets to explore data patterns. Common examples include filter and drill-down operations. For example, a pie wedge selection may be used to filter rows in a data grid. In general, the visual information-seeking mantra: ”overview first, zoom and filter, then details on demand” [13] has never been more relevant.

In the Apache dashboard, selection of a project in the bubble widget reflects the project statistics at the right as shown in figure 13.

Figure 14 demonstrates how this interaction is represented as a flow in the widget source.

Notice that the interaction specification is represented as a data transformation flow - identical in all respects to flows in the Flow (F) section. These transformations also use tasks defined in the task (T) section. Popular widget flow tasks provided by the platform are group, filter and map.

The task **filter_projects** configuration is shown in figure 15. This task filters by the **project** column of the data source with values retrieved from widget **project category bubble**'s **text** widget column property. Note that we

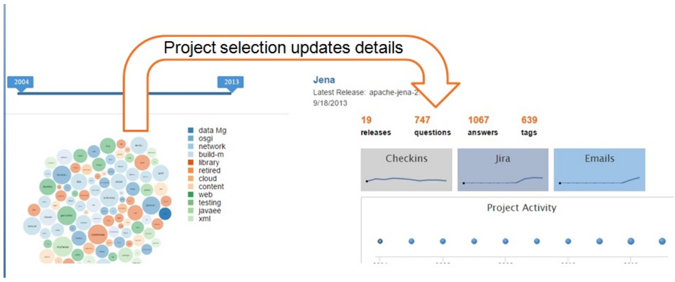


Figure 13: Project selection updates project details

```
W:

...

project_name :
   # Data Attributes
   type : HTML
   tag : section
   source : D.project_latest_release
                  | T.filter_projects
```

Figure 14: Widget interaction modeling

treat widgets as data objects and widget columns as data columns(attributes).

Note the **absence** of event handlers and traditional imperative programming constructs to handle widget - widget interaction.

```
T:

...

filter_projects :
  type : filterby
  filterby : [project]

  #widget data source
  filter_source :W.project_category_bubble

  #widget column
  filter_val : [text]
```

Figure 15: A filter task configured for use in interaction flows

3.6 Layout Section

The layout section arranges the widgets on the dashboard screen. The platform models any dashboard as a grid of widgets. Every cell in the grid holds a reference to a widget name or can itself be a layout. Every row in the grid is broken into twelve columns (arbitrary) of equal width. Each cell specifies how many columns it will span. A sample layout is specified in figure 16.

```
L:
  description : Apache Project Analysis
  rows :
   #Row 1 has a cell spans 12 columns
   − [span12 : W.apache_custom_widget]

   #Row 2 has two cells
   #Cell 1 spans 4 cols
   #Cell 2 spans 8 cols
   − [span4 :W.year_slider_layout ,
      span8 :W.right_project_info_layout]

  ...

   − [span5 :W.project_category_bubble ,
      span7 :W.right_sliders_layout]
```

Figure 16: Apache dashboard layout

To explain the key concepts, we have only used few configuration items from the complete Apache project analysis dashboard. The complete listing of this dashboard can be availed on request from any of the authors.

The flow file grammar (simplified) is presented in appendix B for reference.

3.7 Data Sharing

Use Case : Tweet Analysis .

In this section, we demonstrate data sharing by creating two data pipelines. The first data pipeline transforms source tweet data into shared data objects. The second data pipeline consumes these shared data objects to create an interactive dashboard. We also illustrate the ability to handle unstructured data and the notion of custom tasks. The dashboard we want to build is shown in figure 17.

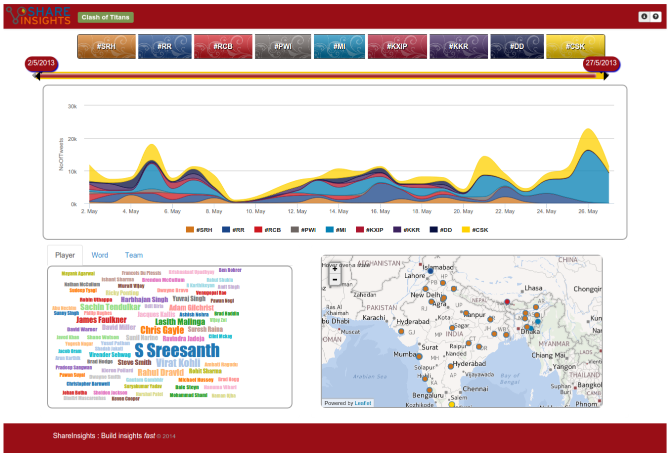


Figure 17: Tweet analysis for IPL

The raw data for this dashboard comes from the tweets collected during the Indian Premier League (IPL) competition. The dashboard allows users to select teams and a date range. The remaining widgets are filtered for the selected team and date range. The streamgraph [3] depicts the relative team tweet volumes, the word clouds show popular words(players, team names) and the map shows the favorite team for major cities in India. (The diameter of the marker is proportional to total tweets for that team and location)

3.7.1 Data-Processing Mode

The platform allows dashboards to be created in a data-processing mode only. In this mode, the flow file has only a data section (D), flow section (F) and a task section (T). The (transformed) aggregated data from the flows are made 'available' (as defined in section 3.4.1) for consumption in other dashboards. We demonstrate using one representative flow. The data section is shown in figure 18.

```
D:
ipltweets : [
    postedTime => created_at ,
    body => text ,
    location => user.location
]
players_tweets : [
    date ,
    player ,#normalized name of player
    count #count of tweets for a player
]
```

Figure 18: IPL Data Section

In this case, source data are tweets retrieved from Gnip provided APIs [7]. Each tweet is a document with hierarchical data items. Users need to map paths in this document (in a similar fashion to XPath or JSONPath queries) to column names. In figure 18, **user.location** path is mapped to a schema attribute - **location**. The flow in figure 19 takes raw tweets (from **ipltweets**) and transforms it to a schema represented by **players_tweets**.

```
F:
 ...
D.players_tweets : D.ipltweets |
        T.players_pipeline |
        T.players_count
D.players_tweets :
  endpoint : true
  publish : players_tweets
```

Figure 19: Transform raw tweet data

The first step transforms (in parallel) the date to a standard format and extracts player names from the tweet by looking at an user provided dictionary (which maps the multitude of player names - abbreviations, nick names etc - to a standardized player name). This is shown (and explained with comments) in figure 20 and figure 21.

```
players_pipeline :
    parallel:[
        T.norm_ipldate ,T.extract_players
    ]
```

Figure 20: Transform 1 : Normalize date and extract player in parallel

```
norm_ipldate :
  type : map
  operator : date
  transform:postedTime #attribute to map
  input_format:'E MMM dd HH:mm:ss Z yyyy'
  output_format : yyyy-MM-dd
  output : date # output column
extract_players :
  type : map
  operator : extract
  transform : body #attribute to transform
  dict : players.txt #standard names map
  output : player #output column
```

Figure 21: Transform 1 : Normalize date and extract player tasks

The first step of the transformation thus produces the schema shown in figure 22.

```
players_tweets : [
  postedTime, body, location # ipltweets
  date,    #normalized date
  player   #normalized player name
]
```

Figure 22: Intermediate schema

The second step of the transformation takes the intermediate schema and groups by date and player to give the count of tweets for the composite key (date + player). This is shown in figure 23. This produces data in the scheme described in figure 18.

```
players_count :
  type : groupby
  groupby : [date, player]
```

Figure 23: Transform 2 : Group by date and player

Some of the important transformation tasks are provided by the platform. Users however can extend the capability of the platform by providing their own transformation tasks. User defined tasks are treated on par with system provided tasks and are represented in the flow file in an identical fashion. The complete listing for this dashboard is available in appendix A.1.

3.7.2 Data-consumption Mode

The shared data object (**player_tweets**) is now available to other consumers (dashboards) on the platform. The consumption dashboard typically has a layout section (L), widget section (W) and a task section (T) for modeling widget interaction. When widgets use the shared data object as their source, the platform automatically searches the shared data objects on the platform for matches. Breaking up the data processing and data consumption activities results in benefits which are outlined in greater detail in section 4.5.3. The complete listing for this dashboard is available in appendix A.2.

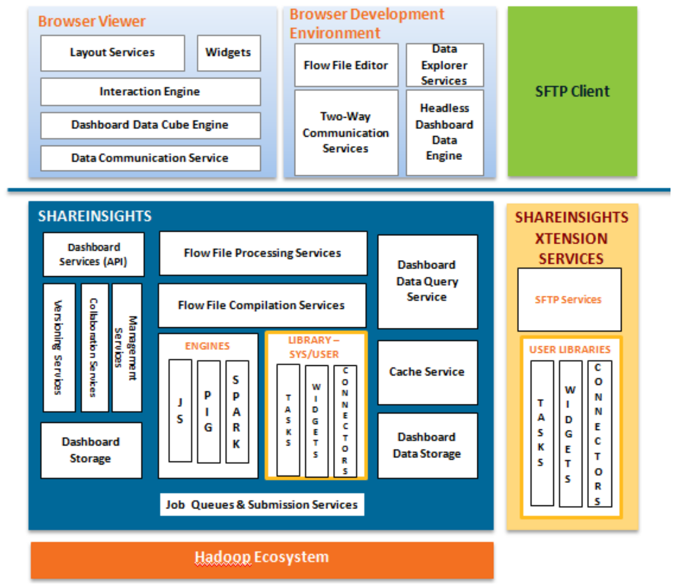


Figure 24: ShareInsights Platform Blocks

4. THE SHAREINSIGHTS PLATFORM

In this section, we provide details on the building blocks of the ShareInsights platform (See figure 24).

4.1 Flow File Compilation Services

The flow file compilation module is the heart of the platform. The compilation module builds an abstract syntax tree (AST) from the flows specified in the flow and widget section. The AST eventually gets converted into either a Pig[12]/Spark[15] job - for data processing - and a data cube (in JavaScript) - for ad-hoc widget interaction (group, filter etc). This is shown diagrammatically in figure 25.

The AST provides opportunities to optimize the complete flow. For example, tasks can be re-arranged to minimize data transfers to the browser.

Additionally, the generated output needs to be cognizant of the operating environment settings (constraints) such as screen resolution and client computing resources.

Screen Resolution : At one end of the spectrum, mobile devices have limited screen space - further limiting interaction methods - and at the other end, power analysts have large monitors that affords richer visualization and interaction methods.

Client Computing Resources : It is not guaranteed that the user will have a powerful device for accessing the visualization. Additionally, JavaScript support may be varied across devices and in the worst case even turned off.

These constraints influence what analysis can be displayed meaningfully and the platform needs to choose the appropriate representation and execution engine.

4.2 ShareInsights Extension Services

The ShareInsights platform comes with a library of data connectors, data formats, tasks and widgets. Through the platform extension services, users can bring their own extensions to the platform. The section lists the possible ways to extend the platform.

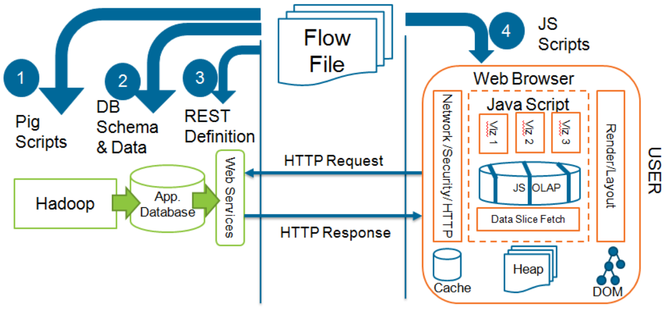


Figure 25: Flow file compilation

Connectors : The Connectors API allows to connect data sources via new protocols to the platform.

Data formats : The Data formats API allows for recognizing and processing new data formats.

Tasks : The Tasks API allows for introducing new kinds of tasks on the platform. Tasks can be broadly categorized as

1. Transforming a column value into another value. These kinds of tasks are called operators and work off a given data point.

2. Transforming a bag of values into a point value. These kinds of tasks are called user defined aggregates.

3. Transforming a data object via the underlying engine (e.g. Spark) APIs. For example, the platform **group** operator generates calls to the engine group operator.

4. Transforming a data object via a native map reduce job. This is very useful as many organizations have existing map reduce jobs and they can be part of the platform through this route.

Since, we support multiple execution contexts, each task (potentially) has an implementation for each run-time environment - Hadoop and JavaScript (browser) run-times. To encourage adoption and lower barriers of entry, tasks can be written in either Java, JavaScript, Python or R and can use existing libraries on those environments. For example Python natural language processing libraries can be wrapped with our APIs and used in the flow file.

Widgets : The Widgets API allows new widgets to be introduced to the platform. Commercial and open source widgets can easily be made part of the platform by implementing this interface.

Styling : The dashboard look and feel can be changed or enhanced using Cascading Style Sheets (CSS). Stylesheet authors can use widget names specified in the flow file as style targets in the CSS file.

4.3 ShareInsights Development Services

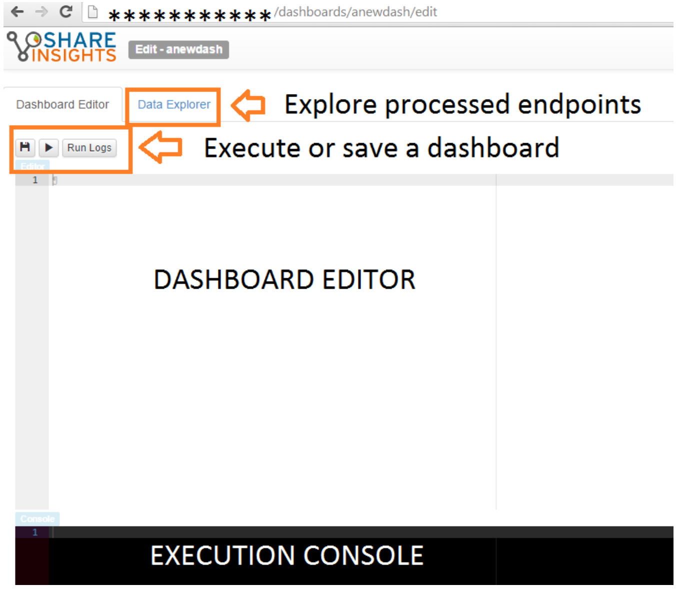
ShareInsights uses regular desktop software for contributors to be productive and requires no additional installs. Data pipeline analysts interact with ShareInsights via the development interface or the extension services interface. These interfaces are described in the subsequent sections.

4.3.1 Development interface

ShareInsights uses the browser exclusively for data-pipeline development. All ShareInsights dashboard operations (create, edit, run, save etc) as well as data operations can be accessed via intuitive REST APIs.

For example, to create a dashboard from scratch, users need to navigate to the following URL : https://<server-base-path>/dashboards/<dashboard-name>/create.

Running this command would take the user to a web editor to edit the dashboard as shown in figure 26. The key components of the editor are also highlighted in the same figure. The data explorer will be covered in section 4.4.



Explore processed endpoints

Execute or save a dashboard

DASHBOARD EDITOR

EXECUTION CONSOLE

Figure 26: Dashboard Editor

4.3.2 Extension Services interface

The ShareInsights platform provides a secure file transfer protocol (SFTP) interface to upload the various types of extensions - widgets, connectors, tasks and stylesheets. The interface is file based and each dashboard has appropriately named folders for task, widgets etc. Additionally, users can upload dashboard data to a 'data' folder. All data files in this folder can be referred in the data object configuration (using relative paths from this data folder).

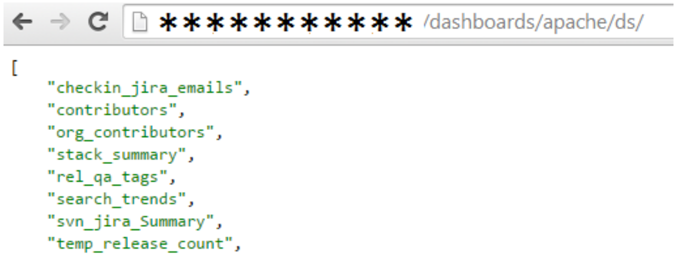
4.4 ShareInsights Data API

The generated interactive dashboard code from the flow file follows a Single Page Architecture style [10].

Dashboard data is retrieved via a REST API provided by the platform(for endpoint data - see section 3.4.1). This is described in the following sections.

Dashboard endpoint data.
To see all endpoint data associated with a dashboard, the following URL (/<dashboard-name>/ds) can be accessed via the browser or curl - as shown in figure 27.

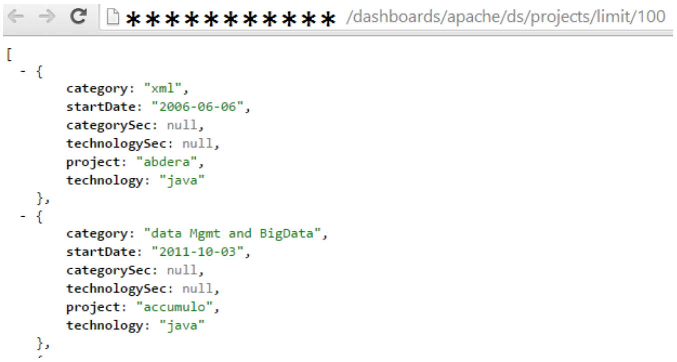
Once the endpoint data names are known, they can be used to further browse data and perform ad-hoc queries as demonstrated in the following sections.



```
[
    "checkin_jira_emails",
    "contributors",
    "org_contributors",
    "stack_summary",
    "rel_qa_tags",
    "search_trends",
    "svn_jira_Summary",
    "temp_release_count",
```

Figure 27: Endpoint data for dashboard

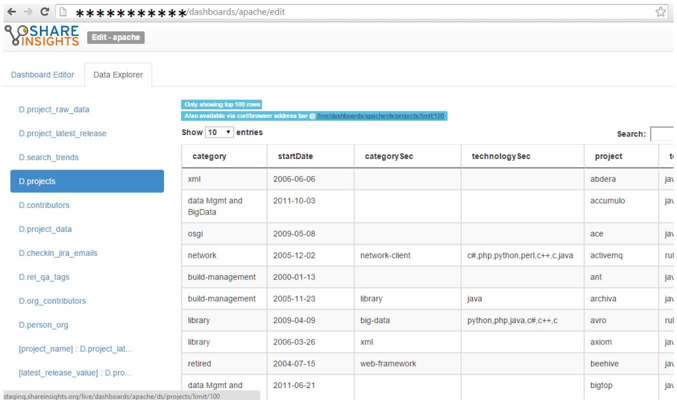
Browsing dashboard endpoint data.
Data browsing can be achieved either from the raw REST APIs as shown in figure 28 or from the platform provided data explorer.



```
[
 - {
       category: "xml",
       startDate: "2006-06-06",
       categorySec: null,
       technologySec: null,
       project: "abdera",
       technology: "java"
   },
 - {
       category: "data Mgmt and BigData",
       startDate: "2011-10-03",
       categorySec: null,
       technologySec: null,
       project: "accumulo",
       technology: "java"
   },
```

Figure 28: Query endpoint data

The data explorer effectively runs the dashboard in a headless mode and displays the data in a tabular format. The data explorer for the Apache dashboard is shown in figure 29.



Only showing top 100 rows
Also available via curl/browser address bar @ /live/dashboards/apache/ds/projects/limit/100

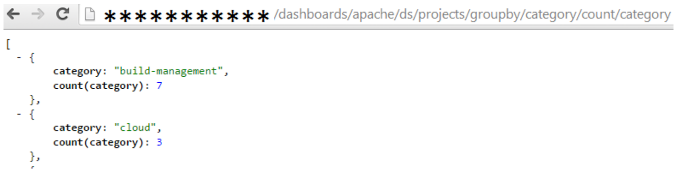
Show 10 entries

category	startDate	categorySec	technologySec	project	t
xml	2006-06-06			abdera	ja
data Mgmt and BigData	2011-10-03			accumulo	ja
osgi	2009-05-08			ace	ja
network	2005-12-02	network-client	c#,php,python,perl,c++,c,java	activemq	ru
build-management	2000-01-13			ant	ja
build-management	2005-11-23	library	java	archiva	ja
library	2009-04-09	big-data	python,php,java,c#,c++,c	avro	ru
library	2006-03-26	xml		axiom	ja
retired	2004-07-15	web-framework		beehive	ja
data Mgmt and	2011-06-21			bigtop	ja

staging.shareinsights.org/live/dashboards/apache/ds/projects/limit/100

Figure 29: Data Explorer (UI for endpoint data)

Ad-hoc queries on dashboard endpoint data.
The platform also exposes a simplified query language for ad-hoc queries. In figure 30, the count of items in each category is returned by querying the data endpoint - **projects**. (/ds/<dataset-name>/groupby/<column>/ <aggregate-function>/<column>)



```
*********** /dashboards/apache/ds/projects/groupby/category/count/category
[
  - {
        category: "build-management",
        count(category): 7
    },
  - {
        category: "cloud",
        count(category): 3
    },
```

Figure 30: Ad-hoc query on endpoint data

4.5 Collaboration Services

ShareInsights encourages collaboration in the data analysis ecosystem via the following mechanisms.

4.5.1 *Branch and Merge Model*

The ShareInsights platform leverages the collaboration model found in distributed version control systems (DVCS), like Git. Since the entire data pipeline is represented as a single text file, it makes it very amenable to manage via a source control system. CRUD operations on flow files map to source commits. A DVCS encourages collaboration via branching and repeated merging. This model is mature and is the preferred way to manage code assets [2]. ShareInsights leverages this model for flow files.

Additionally, since the flow file has clearly demarcated sections, the anxieties with merging and repeated branching should be significantly lower.

4.5.2 *Configuration Driven Development*

The flow file has been designed to let users configure tasks, data, widgets and even flows. Configuration is the activity of setting property values.(as seen from earlier examples)

The only **active** elements in the flow file then are the flows in the flow(F) section and widget sources (for widget to widget interaction). Here too, we have restricted the representation to linear flows. There are no control structures other than the notion of chaining via pipes. Consequently, it is easier to pick up paths which matter when users modify existing flows (users 'fork' existing dashboards to modify). We believe that this will reduce barriers to experimentation.

4.5.3 *Flow file Groups*

The canonical form of the flow file is to have all sections. However, this is optional. The system allows for creation of flow files which have only the data, flow and task sections. Processed data objects can be shared - either by publishing and/or making it a visualization endpoint - for consumption by other dashboards. The set of dashboards which share data objects and the set of dashboards which consumes them form a natural flow file group.

This technique has the following benefits

1. It addresses the problem of allowing other users to benefit and leverage prior work.(an important aspect of collaboration). It is a common experience that data cleaning takes a significant percentage of the total time [5]. The platform ecosystem encourages that this effort is not repeated across data pipelines.

2. It allows for teams to be deployed in their area of strength. To glean insights effectively, it is a common practice to divide and conquer; each dashboard attempts to answer specific questions or guide users towards a certain dimension of the problem space. Consequently, there could be multiple non-intersecting dashboards in the problem space - each consuming only a subset of data flows. The team which designs dashboards have possibly a different mindset from the team which focuses on publishing processed data for the problem space. This technique then allows each subteam to focus on their strengths and work fairly independently.

3. It allows for efficient processing of raw data sources. In this configuration, long running data flows are executed only by the dashboard which shares the data objects.

4. As a corollary to the above point, teams building interactive dashboards on processed data can get extremely quick feedback to changes in the flow file.(as long running data pipelines will not be executed when the flow file is saved) This is vital to encourage rapid changes during dashboard design.

5. RACE2INSIGHTS

To test the value provided by the unified representation, we organized an internal competition called Race2Insights where untrained users were challenged to create interactive dashboards within six hours.

5.1 Competition setup

This section describes the format and the setup of the competition.

Data-sets:
We identified seven interesting data-sets that contained both public and enterprise data. Each data-set had multiple files that contained both transaction as well as reference data about business entities. Teams were randomly allocated a data-set and were encouraged to augment with other interesting data-sets.

Teams and Team Composition:
Fifty two teams participated in the competition. Each team had five members. Participants were encouraged to form teams with known colleagues. Teams had varying skill level ranging from zero to little programming background at one end of the spectrum to significant skills in data processing at the other end.

Training:
Participants were exposed to the platform five days prior to the competition. Training videos and hands-on lab sessions were conducted for participants to familiarize them with the nuances of flow file creation. A day before the competition, participants were assigned the data-set (chosen by a lottery system) and only schema details were shared. Teams were encouraged to paper-prototype and design the data pipelines. Access to systems were switched off.

Competition day:
On the day of the competition, the actual data-sets were published and teams were given six hours to build interactive dashboards to demonstrate business insights for their problem space.

Judging:

Two rounds of panel-style judging were performed. Teams had to present their dashboard to an internal and an external committee. Teams had to present the flow file to the internal committee for review as well as highlight any extensions they may have written or additional data-sets used. The external committee purely evaluated the dashboard in terms of business value for that domain.

5.2 Learnings

The section describes some of the findings from the hackathon.

5.2.1 *Dashboards from competition data*

The data generated during the competition as well as the practice sessions - application logs, flow file growth, error messages, execution logs - were used to build dashboards (using the platform) to illustrate usage of the platform during the competition hours.

The dashboard shown in figure 31 highlights the popular operators and widgets.

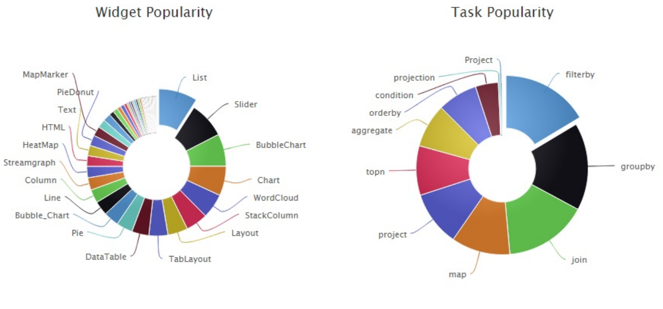


Figure 31: Platform usage

Another interesting plot shows the correlation between practice and success. Practice runs versus competition runs (a run = a dashboard execution) for the various teams are shown in figure 35. The finalists are teams{5,9,12,18,33,35,41}. The winning teams are teams{12, 18, 33}.

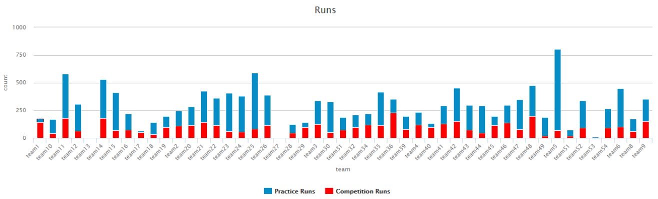
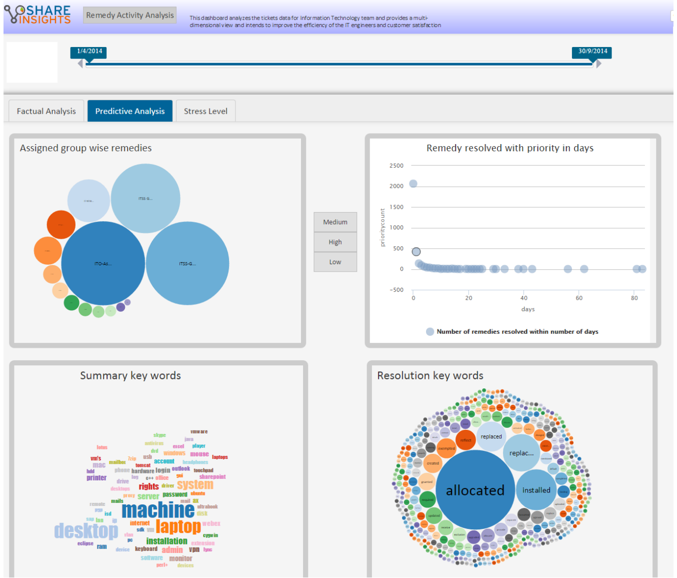


Figure 32: Does practice matter?

5.2.2 *Observations*

1. Teams produced extremely rich dashboards in six hours. Prior to building this platform, equivalent dashboards took four to six weeks to develop. A few dashboards are shown in figure 33 and 34.

2. Some of the winning teams wrote custom tasks to process the data. For example, one team wrote a task to predict resolution dates of service tickets based on keywords present in the ticket. The custom task looks no different from a platform provided task and was used by other team members as a black box.



This dashboard analyzes the tickets data for Information Technology team and provides a multi-dimensional view and intends to improve the efficiency of the IT engineers and customer satisfaction

Figure 33: Service Desk Ticket Analysis

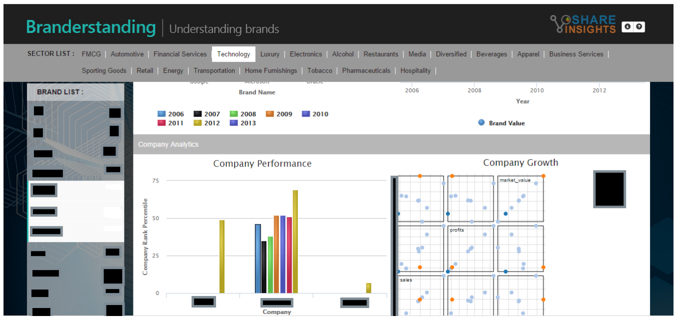


Figure 34: 'Branderstanding'

The extension points provided by the platform are crucial as it is not possible to anticipate all possibilities. The ability to not differentiate between a platform task and a user defined one is also critical for a consistent conceptual model.

3. Teams 'forked' off existing (help or sample) dashboards to get started. Figure 35 shows the flow file size for each team at the start of the competition.

4. Data cleaning is non trivial task. Teams prepared synthetic data for practice runs. During the actual competition, the real data provided forced teams to define more elaborate pipelines to cleanse the data.

5. More training was required to grasp widget to widget interaction specification.

6. Browser based development and deployment - a zero install footprint - helped the competition logistics during the actual competition hours as well as during training sessions by allowing teams to participate remotely.

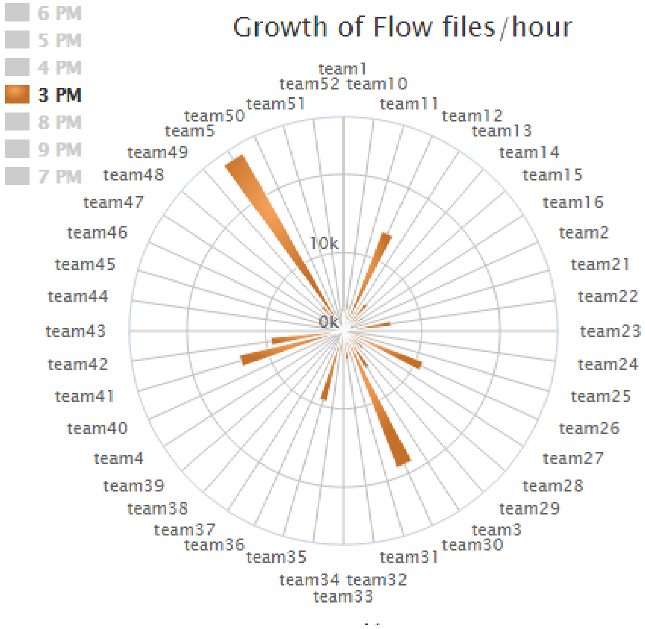


Figure 35: Fork to go (Size in bytes)

7. Error reporting (during dashboard execution) leaked the abstraction. The most popular debugging strategy was to go to a stable version and then incrementally add till the error resurfaced.

6. FUTURE DIRECTIONS

This section outlines some of the key enhancements to the platform.

Future platform versions will focus on exploiting the AST generated from the flow file to do execution optimization. For example one can conceive of rearranging user flows to minimize data transfers to the client. The uniform interface for widget and flow tasks will facilitate this feature.

Also, since the flow file is an abstraction layer, more work needs to be done to enable users to pin-point errors quickly. (Without leaking the underlying engine errors or debug logs). Also, tools to identify performance bottlenecks need to be provided.

We want to auto-construct meta-dashboards which provide statistics and analysis of all the data columns used in the data pipeline. Since, data cleaning is a non-trivial activity[5], we believe this feature would be of immense help for huge data sizes.

Since data is published on the platform, it potentially allows for discovery of data-sets to enrich an existing data pipeline. This is an important feature [11] [1].

7. CONCLUSION

Having an unified representation to model all stages of the data pipeline from data ingestion to interactive visualization design has significant benefits in terms of reduced time to build. Configuration driven development reduces the complexities associated with control structures and cleanly segregated sections allow for reuse and easy modification. The abstraction hides technology differences at every layer of the data stack and empowers the data analyst/data scientist to focus on building value for end-users. The notion of allowing transformed data to be a shared entity allows for re-use (by other data pipelines) of the non trivial effort which goes behind curating and cleansing data.

8. ACKNOWLEDGMENTS

We thank the following individuals for their involvement with the ShareInsights platform. Reshma Godse, Kalyani Zomte and Leena Rajendran for their involvement as the lead engineers of the platform, Vinayak Datar and Kartik Vyas for their customer facing roles and Nilesh Joshi for systems infrastructure support.

9. REFERENCES

[1] C. Bizer, P. Boncz, M. L. Brodie, and O. Erling. The meaningful use of big data: Four perspectives – four challenges. *SIGMOD Rec.*, 40(4):56–60, Jan. 2012.

[2] C. Brindescu, M. Codoban, S. Shmarkatiuk, and D. Dig. How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 322–333, New York, NY, USA, 2014. ACM.

[3] L. Byron and M. Wattenberg. Stacked graphs ; geometry & aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1245–1252, Nov. 2008.

[4] S. Chaudhuri. What next?: A half-dozen data management research goals for big data and the cloud. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS '12, pages 1–4, New York, NY, USA, 2012. ACM.

[5] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

[6] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[7] Gnip.com. Twitter data - gnip : https://gnip.com/sources/twitter/, 2015.

[8] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

[9] R. Krishnamurthy and M. M. Zloof. RBE: rendering by example. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 288–297, 1995.

[10] M. Mikowski and J. Powell. *Single Page Web Applications: JavaScript End-to-end*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2013.

[11] K. Morton, M. Balazinska, D. Grossman, and J. D. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *PVLDB*, 7(6):453–456, 2014.

[12] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of*

Data, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[13] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, pages 336–, Washington, DC, USA, 1996. IEEE Computer Society.

[14] M. Winslett. Databases in virtual organizations: A collective interview and call for researchers. *SIGMOD Rec.*, 34(1):86–89, Mar. 2005.

[15] 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.

APPENDIX

A. IPL FLOW GROUP

A.1 Data Processing Dashboard

```
D:
    ipltweets : [
       postedTime => created_at ,
       body => text ,
       displayName => user.location
    ]
    players_tweets : [
       date , player , count
    ]
    teams_tweets : [
       date , team , count
    ]
    dim_teams : [
       team_number , team ,
       team_fullName , sort_order ,
       color , noOfTweets
    ]
    team_players : [
       player , team_fullName ,
       team , player_id , noOfTweets
    ]
    latlong : [
       state , point_one , point_two ,
       point_three
    ]
    player_tweets : [player ,
       team , date , player_id ,
       team_fullName , noOfTweets
    ]
    team_tweets : [
        sort_order , date , color ,
        team , team_fullName , noOfTweets
    ]
    tm_rgn_raw_cnt : [
        date , team , state , count
    ]
    tm_rgn_tm_dtls : [
        sort_order , noOfTweets , color ,
        state , team , date , team_fullName
    ]
    team_region_tweets : [
       point_one , point_two ,
       point_three , state ,
       team_fullName , team ,
       color , sort_order ,
       date , noOfTweets
    ]
    tagcloud_tweets_raw : [
       date , word , count
    ]
    tagcloud_tweets : [
       date , word , count
    ]
# ------------------------------------
F:
    D.players_tweets : D.ipltweets |
        T.players_pipeline |
        T.players_count

    D.player_tweets : (
        D.players_tweets ,
        D.team_players
    ) | T.join_player_team

    D.teams_tweets :  D.ipltweets |
        T.teams_pipeline |
        T.teams_count

    D.team_tweets : (
        D.teams_tweets ,
        D.dim_teams
    ) | T.join_dim_teams

    D.tm_rgn_raw_cnt : D.ipltweets |
        T.teams_pipeline_region |
        T.teams_regions_count

    D.tm_rgn_tm_dtls : (
        D.tm_rgn_raw_cnt ,
        D.dim_teams
    ) | T.join_dim_teams_two

    D.team_region_tweets : (
        D.tm_rgn_tm_dtls ,
        D.latlong
    ) | T.join_latlong

    D.tagcloud_tweets_raw :
        D.ipltweets |
        T.word_date_extraction |
        T.words_count

    D.tagcloud_tweets :
        D.tagcloud_tweets_raw |
        T.topwords

# ------------------------------------
T:
  players_pipeline :
    parallel : [
        T.norm_ipldate ,
        T.extract_players
    ]
```

```
teams_pipeline :
  parallel : [
      T.norm_ipldate,
      T.extract_teams
  ]
teams_pipeline_region :
  parallel : [
      T.norm_ipldate,
      T.extract_location,
      T.extract_teams
  ]
word_date_extraction :
  parallel : [
       T.norm_ipldate,
       T.extract_words
  ]
norm_ipldate :
  type : map
  operator : date
  transform : postedTime
  input_format:'E MMM dd HH:mm: ss Z yyyy'
  output_format : yyyy-MM-dd
  output : date
extract_players :
  type : map
  operator : extract
  transform : body
  dict : players.txt
  output : player
extract_teams :
  type : map
  operator : extract
  transform : body
  dict : teams.csv
  output : team
extract_location :
  type : map
  operator : extract_location
  transform : displayName
  match : city
  country : IND
  output : state
extract_words :
  type : map
  operator : extract_words
  transform : body
  output : word

join_player_team :
  type : join
  left : players_tweets by player
  right : team_players by player
  join_condition : left outer
  project :
    players_tweets_date : date
    players_tweets_player : player
    players_tweets_state : state
    players_tweets_count : noOfTweets
    team_players_team : team
    team_players_team_fullName:
                         team_fullName
    team_players_player_id :
                         player_id
```

```
join_dim_teams :
  type : join
  left : teams_tweets by team
  right : dim_teams by team_fullName
  join_condition : left outer
  project :
    teams_tweets_date : date
    teams_tweets_team : team_fullName
    teams_tweets_count : noOfTweets
    dim_teams_team : team
    dim_teams_sort_order : sort_order
    dim_teams_color : color

join_dim_teams_two :
  type : join
  left:tm_rgn_raw_cnt by team
  right : dim_teams by team_fullName
  join_condition : left outer
  project :
    tm_rgn_raw_cnt_date : date
    tm_rgn_raw_cnt_team : team_fullName
    tm_rgn_raw_cnt_state : state
    tm_rgn_raw_cnt_count : noOfTweets
    dim_teams_Team : team
    dim_teams_sort_order : sort_order
    dim_teams_color : color

join_latlong :
 type : join
 left : tm_rgn_tm_dtls by state
 right : latlong by state
 join_condition : left outer
 project :
  tm_rgn_tm_dtls_team_fullName :
                        team_fullName
  tm_rgn_tm_dtls_state: state
  tm_rgn_tm_dtls_date: date
  tm_rgn_tm_dtls_noOfTweets: noOfTweets
  tm_rgn_tm_dtls_team: team
  tm_rgn_tm_dtls_sort_order: sort_order
  tm_rgn_tm_dtls_color: color
  latlong_point_one : point_one
  latlong_point_two : point_two
  latlong_point_three : point_three

players_count :
    type : groupby
    groupby : [date, player]
teams_count :
    type : groupby
    groupby : [date, team]
teams_regions_count :
    type : groupby
    groupby : [date, team, state]
words_count :
    type : groupby
    groupby : [date, word]
topwords :
    type : topn
    groupby : [date]
    orderby_column : [count DESC]
    limit : 20
```

A.2 Data consumption Dashboard

For brevity of the source listing, it is assumed that all the data objects used by widgets in this listing have been published (with identical names) and end-pointed in listing A.1.

```
# ------------------------------------------
L:
  description : Clash of Titans
  rows :
    - [span12 : W.teams ]
    - [span11 :  W.iplduration  ]
    - [span11 : W.relativeteamtweets  ]
    - [span6  : W.word_team_player_tweets ,
       span5  : W.regiontweets  ]

# ------------------------------------------
W:

iplduration :
    type : Slider
    source : ['2013-05-02','2013-05-27']
    static : true
    range : true
    slidertype : date

relativeteamtweets:
    type : Streamgraph
    source : D.team_tweets |
          T.filter_by_date |
          T.filter_by_team
    #widget columns
    x : date
    y : noOfTweets
    color : color
    serie : team
    xAxis:
        type: 'datetime'
    yAxis:
        allowDecimals: false
        min: 0
        max: 25000

teams :
    type : List
    source : D.dim_teams
    #widget columns
    text : team
    #Visual attributes
    image_position : right

playertweets:
    type : WordCloud
    source : D.player_tweets |
            T.filter_by_date |
            T.filter_by_team |
            T.aggregate_by_player
    #widget columns
    text : player
    size : noOfTweets
    #visual attributes
    show_tooltip : true
    tooltip_text : [player,noOfTweets]
```

```
teamtweets:
    type : WordCloud
    source : D.team_tweets |
              T.filter_by_date |
              T.aggregate_by_team
    #widget columns
    text : team
    size : noOfTweets
    #visual attributes
    show_tooltip : true
    tooltip_text : [team, noOfTweets ]

wordtweets:
    type : WordCloud
    source : D.tagcloud_tweets |
              T.filter_by_date |
                T.aggregate_by_word
    #widget columns
    text : word
    size : count
    #visual attributes
    show_tooltip : true
    tooltip_text : [word, count ]

regiontweets :
    type : MapMarker
    source : D.team_region_tweets |
           T.filter_by_date |
           T.filter_by_team |
           T.aggregate_by_team_region
    country : IND
    markers :
      - marker1 :
          type : circle_marker
          #widget columns
          latlong_value : point_one
          markersize : noOfTweets
          fill_color : color
          tooltip_text : [
            state,
            team,
            noOfTweets
         ]

#sub layouts

teamtweetstab:
    type : Layout
    rows :
      - [span11 : W.teamtweets]

playertweetstab:
    type : Layout
    rows :
      - [span11 : W.playertweets]

wordtweetstab:
    type : Layout
    rows :
      - [span11 : W.wordtweets]
```

```
word_team_player_tweets :
   type : TabLayout
   tabs :
      -    name : 'Player'
           body : W.playertweetstab
      -    name : 'Word'
           body : W.wordtweetstab
      -    name : 'Team'
           body : W.teamtweetstab

# ----------------------------------------

T:


aggregate_by_player :
   type : groupby
   groupby : [player]
   aggregates :
      - operator : sum
        apply_on : noOfTweets
        out_field : noOfTweets


aggregate_by_team :
   type : groupby
   groupby : [team]
   aggregates :
      - operator : sum
        apply_on : noOfTweets
        out_field : noOfTweets

aggregate_by_word :
   type : groupby
   groupby : [word]
   aggregates :
      - operator : sum
        apply_on : count
        out_field : count
        orderby_aggregates : true


filter_by_date :
   type : filterby
   filterby : [date]
   filter_source : W.iplduration


filter_by_team :
   type : filterby
   filterby : [team]
   filter_source : W.teams
   filter_val : [text]


aggregate_by_team_region :
   type : groupby
   groupby : [team,point_one,state,color]
   aggregates :
      - operator : sum
        apply_on : noOfTweets
        out_field : noOfTweets
```

B. FLOW FILE GRAMMAR

```
/* ---------- Parser rules -------------- */
flowFile :
   dataSection? flowSection?
   taskSection?
   dataDetailsSection? EOF ;

/* ------------ data section ---------- */
dataSection : 'D' ':' dataItem+ ;
dataItem : dataItemName ':' '[' hdrs ']';
dataItemName : IDENTIFIER ;
hdrs : hdrField (',' hdrField)* ;
hdrField : IDENTIFIER ;

/* ------------ flow section ---------- */
flowSection : 'F' ':' flow+ ;
flow:'D.' outputData ':' multiDataStart*
      'D.' inputDataInFlow
      (',' 'D.' inputDataInFlow)*
      multiDataEnd
      ( '|' 'T.' taskInDataFlow )+;
multiDataStart : ROUND_BRACKET_OPEN ;
multiDataEnd : ROUND_BRACKET_CLOSE ;
outputData : IDENTIFIER ;
inputDataInFlow : IDENTIFIER ;
taskInDataFlow : IDENTIFIER ;

/* ------------ task section ---------- */
taskSection : 'T' ':' task+ ;
task
  :
  taskName ':'
  'type' ':' taskType
  (paramName ':'
     '['*
     '('* paramValue (',' paramValue)* ')'*
     ']'* )+
  ;
taskName : IDENTIFIER ;
taskType : IDENTIFIER ;
paramName : IDENTIFIER ;
paramValue : IDENTIFIER;

/* ---- data details section ---- */
dataDetailsSection : dataItemInDetail+ ;
dataItemInDetail :
   'D.' dataItemNameInDetail ':' prop+ ;
prop :
      (dd_paramName ':' dd_paramValue )+;
dataItemNameInDetail : IDENTIFIER ;
dd_paramName : IDENTIFIER ;
dd_paramValue : IDENTIFIER;

/* --------- Lexer rules ----------- */
ROUND_BRACKET_OPEN : '(' ;
ROUND_BRACKET_CLOSE : ')' ;
IDENTIFIER : [a-zA-Z_]+[a-zA-Z0-9_]* ;
WS : [ \t\r\n\u000C]+ -> skip ;
LINE_COMMENT : '#' ~[\r\n]* -> skip ;
```