Visualization of Data Flow Graphs for In Situ Data Analysis

Jacob Edwards

A thesis presented for the degree of
Master of Science

Database Systems and Information Management Group
Technische Universität Berlin
Berlin, Germany

31/07/2015

Author:
Jacob Edwards
Technische Universität Berlin
Berlin, 2015.

Abstract

TODO-> write abstract <-TODO

Acknowledgements

Contents

List of Figures

List of Abbreviations

DAG	directed acyclic graph
KPI	key performance indicator

CHAPTER 1

Introduction

1.1 Motivation

IN-SITU data processing is currently extremely popular. In this approach, in order to achieve the minimum possible time in which results are returned, very little preprocessing of any kind is performed. This means that users do not have a very comprehensive understanding of the nuances and problems which may exist in the data beforehand. Any potential pitfalls are likely to only be discovered at a later time, after much time and effort will already have been invested.

Intermediate data sets

Standard statistics such as minimum, maximum, average, or median may help for simple numeric data. However, text data or (semi-) structured data call for different approaches. Aside from knowing what your raw data looks like at the input stage it is also crucial to understand intermediate data sets, i.e. how the different operations affect the data within the data flow.

Directed Acyclic Graphs

It is typical for large scale analysis systems such as Flink [BEHK10], Pig [ADD$^+$11], or IBMs System S [**?**] to represent analysis jobs as a series of individual tasks. These tasks are connected into a data flow which generally takes the form of an directed acyclic graph (DAG), which provides a useful visual metaphor for the ordering and dependencies of each task within a job. While this is adequate for describing the process by which data is analyzed, it leaves much to be desired in terms of describing the data itself. In particular, in cases where execution times are particularly long. Thus far, few systems making use of data flow graphs have invested significantly in the area of visual feedback within these graphs. System S provides basic feedback indicating the status of dataset processing without real feedback regarding data features [PLGA10], and Lipstick has evolved from a method of providing provenance models for pig latin queries [ADD$^+$11] to providing rudimentary DAG visualization capabilities for Apache Pig in its current development state [ADD$^+$11].

1.2 Structure of this Thesis

Chapter 2 contains a survey of related work
Chapter 3 provides an overview of data types and models
Chapter 4 details the implementation
Chapter 5 results and conclusions

CHAPTER 2

Related Work

THE FIELD OF DATA VISUALIZATION has existed in some form for as long as data analysis has taken place. The primary purpose of data visualization is of course the effective communication of information through the use of graphics. Across varying fields and time periods, different approaches have been applied to varying degrees of success. Most are familiar with basic forms of information graphics, such as tables or basic charts, but as more data is generated and the economy becomes increasingly information-driven we have seen data visualization expand as a field of study in and of itself.

2.1 Visualization of Data

GENERAL PURPOSE VISUALIZATION TECHNIQUES have evolved over the past several decades, but often simple techniques still provide the most effective solution. One of the most seminal works in information display is Edward Tufte's "The Visual Display of Quantitative Information"[Tuf83]. This work provided a summary of several different types of visualizations applied in many fields, but more importantly it set guidelines as to what makes an effective visualization.

Chart Junk

Many of the key concepts of Tufte's work revolve around the idea of limiting what he called *chart junk*. Chart junk refers to "useless, non-informative, or information-obscuring elements of information displays"[Tuf83]. While Tufte acknowledges that using non-data graphics can help to editorialize or provide context for the information being displayed, it is more important to ensure that data is not distorted in order to fit an aesthetic.

Data-rich Visualizations

In addition to limiting non-data information in visualizations, Tufte makes a strong case for the value of data-rich visualizations. Data-rich visualizations are those which include all available information, providing a comprehensive view from which macro trends may emerge. In essence, perhaps at the expense of being able to read individual data points, viewing a complete data set visually may provide insight without need for mathematical analysis. One of many examples of this given in the work is the famous map of central London used by Dr. John Snow to determine the root cause of a cholera outbreak, shown in Figure 2.1. By marking the location of cholera deaths with dots and water pumps with crosses it became immediately clear that deaths were clustered around a central pump on Broad Street. Dismantling this pump quickly stopped the deaths. This provides a clear case where a simple graphical analysis proved far more efficient than mathematical computation would have been in determining a causal link.

Dashboards

A more contemporary area of work which is directly connected to digital display is the concept of a *dashboard*. As defined by Stephen Few, a pre-eminent expert in this area, a dashboard is a single-screen visual display of the information required to achieve a specific set of goals. In a business context, this generally refers to key performance Indicators (KPIs). Such a dashboard is typically generated dynamically, allowing for real-time display of data trends as they occur.

Dashboard constraints

In Stephen Few's "Information Dashboard Design" [Few06] a comprehensive guide to the development of dashboards is given. In particular, specific charts and graphics are matched to appropriate use cases and perhaps more importantly, areas in which some visualizations are inappropriate are defined. Beyond being a discussion simply on visual design, interactivity is discussed. The author notes that although the capability to explore data and perform analysis is available, for monitoring purposes it is more appropriate to not allow such features. Though these analyses are often important, it is more crucial to the purpose of a dashboard to display the data in the form that the dashboard was originally designed for. To do otherwise would risk undermining the purpose, which is a focus on optimal display of key metrics.

Evaluation of Visualizations

2.2 In-Situ Processing

Processing large quantities of data has become a common task within many organizations. Data sources such as sensor networks or click streams necessitate handling both massive quantities of information and rapid rates of change. The size of

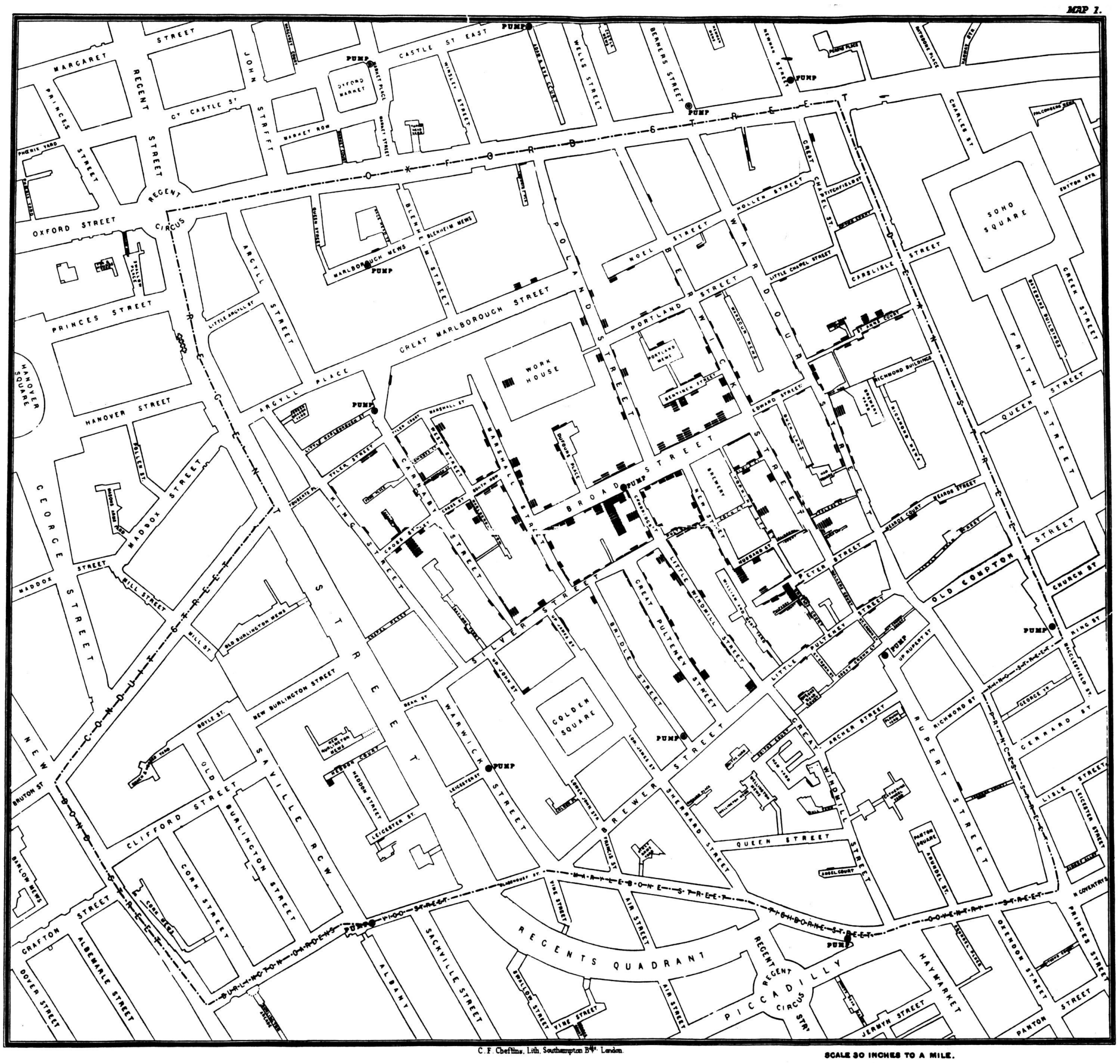

Figure 2.1: The map used by John Snow to determine the source of a cholera outbreak [Tuf83]

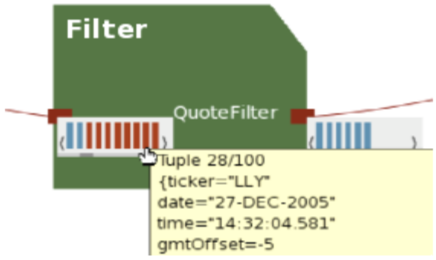

Figure 2.2: An executing operator as visualized in IBM's System S [PLGA10]

this data presents issues in the efficiency of storage solutions and there are many options for handling such problems [KAL+11]. Beyond storage, when analysis occurs on large data stores it is often necessary to apply in-situ processing rather than a more thoroughly controlled approach. In-situ analysis allows for results to be obtained quickly by ignoring much, or all, of the preprocessing that may be involved in an analysis performed on a more controlled data source. Removing preprocessing steps of course increases speed while introducing a number of potential unknown factors.

Pig

Flink

2.3 Visualization of Data Flow Graphs

DATA FLOW GRAPH VISUALIZATIONS have existed in some form for as long as data flow graphs have been used in analysis systems. However, their use is almost exclusively applied to examining meta-information such as optimization plans. Relatively little work has been done in generating visualizations which help in the understanding of data, as a supplement to the analyses themselves.

IBM System S

IBM research has developed a stream processing system known as *System S*, which builds processing graphs using predefined operators [**?**] and has included basic visualization of these graphs [PLGA10]. The visualizations show the DAG of analysis operators and indicate whether the operations have completed through colour coding. Additionally, each operator has a small widget which identifies the tuples which have been passed to or from the operator, as seen in Figure 2.3. These tuples can be highlighted in order to show specific data values, and to highlight data dependencies which exist downstream.

Retrospective Debugging

This type of visualization exists primarily to support debugging after some failure has been detected post-analysis. It can be seen in Figure 2.3 that there are only ten tuples visible at a single time. Though this number can be expanded, this limitation is here because the envisioned use-case consists of a user scrolling through tuples to identify a single suspected problem tuple. While this is very useful for repairing a problem which is found post-analysis, in cases where this computation is very expensive or the problem is particularly unclear after a failure it may not be efficient.

Lipstick

Lipstick [ADD$^+$11], a workflow provenance model framework built for use with Pig takes a similar approach to that of IBM. Lipstick examines the internals of modules within a data flow in order to determine dependencies between parts of a flow. This approach is used for very much the same debugging cases which are expected within System S, with the addition of an added feature allowing developers to query a dependency graph. These queries allow developers to change parameters of the tuples in the graph in order to undertake "what-if" style analyses. Beyond the analysis options introduced through the querying capabilities of Lipstick however, the added visualization features are relatively simple. Like in System S, single operations change colour to indicate status and the tuples being passed to and from operations are identified. In this case the key difference is that the widget for selecting single tuples from System S is replaced with a simple integer iindicating the quantity of tuples moving through a flow. The exploratory capabilities here are left for queries made against the graphs generated in Lipstick.

CHAPTER 3

Implementation

THE PROPOSED METHOD for implementing an in-situ visualization system is comprised of several vital parts. Although the output visualization is key from a user perspective, there are important factors to be considered in the way that data is collected and how this method fits into the overarching analysis system.

3.1 Overview

THIS PORTION will eventually include models etc.

3.2 Data Collection

DATA TYPES IN FLINK are analyzed by the optimizer to determine the most efficient execution strategies. In order to make this process simpler, Flink places limits on the types of data which can be used. There are four categories of types: General objects and POJOs, Tuples, Values, and Hadoop writeables. The handling of each of these types must of course be considered when data is being collected from an analysis graph.

Tuples

Tuples are used to represent composite data sets, and are composed of a set length list of fields of various types. Tuples can include any valid Flink type as an element, including further tuples. One of the major benefits of using tuple types is the ability to use built-in functions to navigate through the tuple values. Specifically, these functions allow the selection of specific fields as the key for operations and more generally allow the navigation of tuple fields using string expressions.

Values Values are types which have their serialization and de-serialization specified manually.

Hadoop Writeables Objects which implement the Hadoop writeable interface.

Data Collector The data collector class acts as a simple addition to a pre-existing analysis program in Flink which collects data as it passes through operations. A single collector object exists for a given analysis flow, and collects data at a specific point with a single added line of code calling the collect method.

Collect Method Each time the collect method is called, it sends a new dataset to the central visualizer class. This method accepts a dataset as its sole argument and writes this dataset to memory in a format which can be read by the data collector. The data collector then reads this data into a new dataset outside of the original analysis flow's execution environment.

Data Sets A custom data set class exists for the use of the collector and visualizer. This class is very similar to the data set class which is native to Flink, but allows for the tracking of additional metadata which may be useful for debugging. This information could include timestamps, tags referring to specific operations in the analysis flow, or other semantically relevant information. These datasets are always initialized to contain tuple type objects. As a tuple can of course include any item of a basic type, this implementation will create a tuple of any general object in order to simplify data set operations. For example, if a single integer field is passed through the initial analysis flow, the data set generated in the visualizer will consider this as a tuple of size one which contains an integer.

Type Erasure When analysis jobs are executed, the java compiler will erase types and operate exclusively with generics. This means that when this data is extracted, some additional work is needed in order to determine a sufficient approximation of the original type for storage in a custom data set. To handle this, as each record is read into a data collector they are parsed through a set of pattern matching checks which determine the number of fields and the fields' types. Firstly, a simple line split determines the size of the tuples which should exist in the data set based on the input record. Next, each field is checked individually using the java string utilities library to determine whether they are numeric or non-numeric. Fields in each of these categories are then passed through a cascading set of conditional checks which determine their specific basic type, from least to most complex. For example, this method will attempt to parse a numeric field as an integer, and upon failure attempt to parse the field as a long. This process continues until a match is found; in the case that one is not an exception is thrown.

3.3 Distribution

DISTRIBUTION IN ANALYSIS SYSTEMS following the general mapreduce model all operate very similarly in concept. This means that generally speaking, we can expect the dataset to be mapped into a set of key-value pairs which are then partitioned across a cluster in a uniformly distributed way. Because we may want to examine the intermediate dataset at a point prior to a reduce operation which would centralize the dataset, we must collect it piecemeal from each node in the cluster. This is achieved by sending the datasets from each node in the cluster to the visualizer for summary.

Message Passing

Message passing allows us to invoke a send message call from each in-situ data collector operating on a shard of the complete data set, and then receiving it in the visualizer. The visualizer can perform whichever operations are needed in order to merge the datasets considering the original locations and timing in order to generate useful output.

Patterns?

I'm not sure yet if a specific pattern will apply.

RabbitMQ

So far arbitrary.

Specifics

Implementation details such as server locale etc.

3.4 Visualization

Processing 2

Libraries

Swing

Presentation of Visualizations

CHAPTER 4

Evaluation

4.1 Accuracy

4.2 Performance

4.3 Usefulness

APPENDIX A

Implementation

A.1 My Algorithm

THE FOLLOWING FUNCTION computes something

```
#include <cv.h>
using namespace cv;
// your code goes here
```

Bibliography

[ADD+11] Y. Amsterdamer, S. B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, "Putting Lipstick on Pig : Enabling Database-style Workflow Provenance," *Proceedings of the VLDB Endowment*, vol. 5, no. 4, pp. 346–357, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2095693

[BEHK10] D. Battré, S. Ewen, F. Hueske, and O. Kao, "Nephele / PACTs : A Programming Model and Execution Framework for Web-Scale Analytical Processing Categories and Subject Descriptors," *ACM Symposium on Cloud Computing*, pp. 119–130, 2010. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807148$\delimiter"026E30F$nhttp://dl.acm.org/citation.cfm?id=1807148

[Few06] S. Few, *Information Dashboard Design*, 2006. [Online]. Available: http://proquest.safaribooksonline.com/0596100167?suggested=top

[KAL+11] S. Klasky, H. Abbasi, J. Logan, M. Parashar, K. Schwan, A. Shoshani, M. Wolf, A. Sean, I. Altintas, W. Bethel, C. Luis, C. Chang, J. Chen, H. Childs, J. Cummings, C. Docan, G. Eisenhauer, S. Ethier, R. Grout, S. Lakshminarasimhan, Z. Lin, Q. Liu, X. Ma, K. Moreland, V. Pascucci, N. Podhorszki, N. Samatova, W. Schroeder, R. Tchoua, Y. Tian, R. Vatsavai, J. Wu, W. Yu, and F. Zheng, "In Situ Data Processing for Extreme-Scale Computing," in *SciDAC Conference*, 2011. [Online]. Available: http://pasl.eng.auburn.edu/pubs/scidac11-adios-insitu.pdf

[PLGA10] W. D. Pauw, M. Leţia, B. Gedik, and H. Andrade, "Visual debugging for stream processing applications," *Runtime Verification*, pp. 18–35, 2010. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-16612-9_3

[Tuf83] E. Tufle, "The visual display of quantitative information," *CT Graphics, Cheshire*, 1983. [Online]. Available: http://www.colorado.edu/UCB/AcademicAffairs/ArtsSciences/geography/foote/maps/assign/reading/TufteCoversheet.pdf

Declaration of Authorship

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Formulations and ideas taken from other sources are cited as such. This work has not been published.

Berlin, 31 July 2015

Jacob A. Edwards