Janiform Intra-Document Analytics for Reproducible Research

Jens Dittrich   Patrick Bender

Saarland University
infosys.cs.uni-saarland.de

ABSTRACT

SELECT * FROM data1   Peer-reviewed publication of research papers is a corner stone of science. However, one of the many issues of our publication culture is that our publications only publish a snapshot of the final result of a long project. This means, we put well-polished graphs describing (some) of our experimental results into our publications. However, the algorithms, input datasets, benchmarks, raw result datasets, as well as scripts that were used to produce the graphs in the first place are rarely published and typically not available to other researchers. Often they are only available when personally asking the authors. In many cases, however, they are not available at all. This means from a long workflow that led to producing a graph for a research paper, we only publish the final result rather than the entire workflow. This is unfortunate and has been lamented upon in various scientific communities. In this demo we argue that one part of the problem is our dated view on what a "document" and hence "a publication" *is*, *should*, and *can be*. As a remedy, we introduce portable database files (PDbF). These files are janiform, i.e. they are at the same time a standard static pdf as well as a highly dynamic (offline) HTML-document. PDbFs allow you to access the raw data behind a file, perform OLAP-style analysis, and reproduce your own graphs from the raw data — all of this *within* a portable document. We demo a tool allowing you to create PDbFs smoothly from within LaTeX. This tool allows you to preserve the connection of raw data to its final graphical output through all stages of the workflow. Notice that this pdf already showcases our technology: rename this file to ".html" and see what happens (currently we support the desktop versions of Firefox, Chrome, and Safari).

1. INTRODUCTION

Irreproducibility is a problem frequently lamented upon in various scientific communities [11, 14]. In the context of computer science it has recently been coined "The Real Software Crisis" [9]. The database community has identified it more than ten years ago and is attacking it through repeatability committees, e.g. [10]. These committees rerun the experiments of accepted papers using the datasets and code provided by the authors. Obviously, given the sheer size and complexity of some projects, in many cases these boils down to black box testing, i.e. it can neither be tested if the code actually implements the algorithms presented in the paper nor whether the code measures and reports results in a proper way. Another problem of repeatability committees is that they cannot remedy inherent publication bias: "reviewers don't like negative results". Hence, for an experimental evaluation you need the "right" queries, the "right" datasets and the "right" baselines. The results then need to be visualized, presented, and interpreted in the "right" way (e.g. log vs linear scale, offset on y-axis). This naturally leads to a flood of papers with positive results. And to papers where the "improvements don't add up" [3]. Publication bias was attacked by the inauguration of Experiments&Analysis papers at (P)VLDB. These kind of papers reevaluate existing work in a uniform setting and may also publish negative results. These experimental evaluations may then serve as landmarks in the flood of papers with (overly) positive results giving clear advice on the strength and weaknesses of a particular method.

This small demo is neither the place to even summarize nor defend the different arguments in the debate on repeatability and our experimental culture. It is an emotional topic where the esteemed reader of these lines probably has strong opinions in one way or the other. This is just fine. In the following, we will simply accept that there is a problem [14, 9][1]. And that this problem is calling "for a new model for the way how we publish our results" [11]. Handling this problem can be regarded an instance of "small data" [5, 6][2].

Obviously, we cannot solve all of the world's problems with reproducible research like baseline, dataset, query, and presentation bias. In this demo we will attack the latter and show how to preserve the connectivity from raw data to graphical display in the research workflow. We believe that our demo is an important step towards making access and analysis of raw data more transparent.

2. SIGNIFICANCE OF THE CONTRIBUTION

We believe that our contribution is significant for the following reasons:

1. **Portable DataBase Files.** We provide Portable DataBase Files (PDbF), a general model to combine a static pdf document with additional highly dynamic content. In order to specify a PDbF, we simply require access to a static document S, dynamic content D, and a PDbF-configuration file

---

[1]Just read the "ten simple rules for reproducible results" [14] and then ask yourself how little of this we implement in our papers.

[2]Also notice an upcoming Dagstuhl perspectives workshop on Artifact Evaluation for Publications [4] where the first author participates.

C defining where to place the dynamic content. An example of *D* could be an embedded relational database and an appropriate visualization of some of its content, e.g. a bar chart visualizing measurements collected in a table.

2. **Alternative Dynamic Views on the Data.** Our technology allows you to perform **offline** OLAP-style analytics on the **data shipped within the document**. All you need is a Web Browser (currently Firefox, Chrome or Safari on a desktop machine). We support a rich feature list. See Section 4 for our currently supported features (as of March 31, 2015).

3. **PDbF-Compiler.** We provide a compiler taking as its input the triple (S, D, C). Our compiler outputs a janiform document. That document is **at the same time** a valid pdf **and** a valid HTML-document. Thus, if you open the file with a pdf-viewer, you will see the static content, i.e. only the *S*-part. However, if you rename the file to ".html" and open it with a Web browser, you will be able to inspect the dynamic part, i.e. *D* and *S*.

4. **Full LaTeX integration.** We instrument LaTeX to output not only the static pdf-file *S*, but also the necessary data to create a valid PDbF-configuration file *C*. In addition, we provide an extension to LaTeX allowing users to create graphs directly from within LaTeX — without requiring the user to invoke multiple tools manually. In addition, this process creates dynamic variants of the graphs as a side-effect, i.e. the *D*-part. This means, the user simply defines the initial display of the graph (or table). Everything else is generated automatically. The result of this instrumentation is again a triple (S, D, C) which can be fed into our PDbF-Compiler (see Contribution 3) to create a janiform PDbF-document.

5. **Preservation of Raw Data and Graph-Connectivity.** Our compilers preserve the connection between raw data and the graphs and/or tables produced from that raw data *through the LaTeX compiler*. In addition, we are able to ship that part of the workflow, i.e. data, graphs, and the code producing the graphs within a single "document".

6. **Longterm Preservation of Raw Data.** As our tools embed the raw data within the publication, PDbF-documents naturally archive the raw data with the document. Therefore, the raw data may be "downloaded" directly from within the PDbF-document. In addition, there is no need anymore to only publish a subset of the measurements (as graphs). Embedding all raw data does not require much space and allows other researchers to see the whole picture. And all of this works by simply shipping a single "pdf" to the publisher of the research.

7. **Impact On Research in General.** We believe that our technology may not only be interesting to the database community. Our tool may be interesting for all research communities working with experimental data. Therefore, we are planning to open source our tool upon publication of this paper.

3. THE PDBF FRAMEWORK

3.1 Janiform File Format

How is it possible to create a single file that may both be interpreted as a valid pdf document and a valid HTML document? The core idea is to create a document where complementary parts of the file are ignored by the different applications. The core structure of a

PDbF is shown in Figure 1. Its core structure is inspired by "funky files" [2].

(a) PDF perspective: the pdf reader interprets the green part, everything else is treated as a comment

(b) HTML perspective: the HTML browser interprets the green part, everything else is treated as a comment

Figure 1: Structure of the janiform PDbF format

A **PDF reader** has the following **perspective** on this file: It reads the *magic number* "%PDF-1.5%" (line 1). Then it locates the Xref table at the end of the file. That Xref contains a dictionary of all objects in this PDF file except for the dummy object (lines 3 to 8) which contains the HTML part. As the dummy object is not referenced by another object, it is never read by PDF viewers. Hence, the file is displayed as a valid PDF.

An **HTML browser** has the following **perspective** on this file: It reads lines 1 and 2. The text "%PDF-1.5%" is displayed at this point. Lines 2–5 are ignored as they are an HTML comment. The same happens for lines 6 and all following lines until the end of the document. The HTML comment is actually never closed, however, browsers are not very strict with such things. The same happens for line 1, because normally there should not be any content before the HTML tag. Hence, the HTML content is displayed.

3.2 Compiler Architecture

A flow chart describing how our different processing steps are invoked and how the different compilers interact is shown in Figure 2. The figure shows the entire workflow to compile a tex-file into a PDbF.
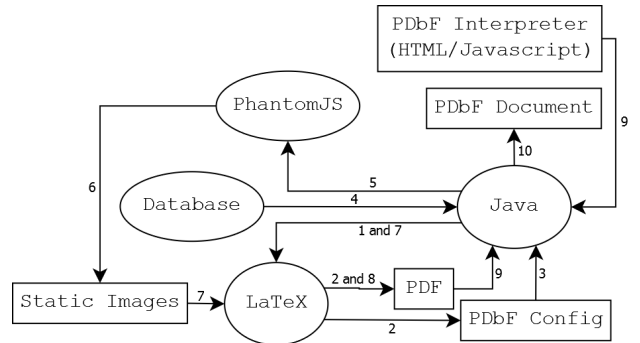

Figure 2: Compilation steps of the PDbF compiler framework

(1.) Our main controller is written in Java (abbreviated *Java* in the following). It invokes the LaTeX Compiler.

(2.) The LaTeX compiler outputs the PDbF configuration file and a draft version of the PDF document.

(3.) Java reads the PDbF configuration file and does some post-processing.

(4.) Java reads all tables from the database. (The database is specified as a source in the PDbF configuration file.)

(5.) Java invokes PhantomJS to create static snapshots of all graphs automatically.

(6.) PhantomJS outputs a static image for every dynamic object present in the PDbF configuration file.

(7.) Java invokes the LaTeX compiler a second time to obtain the final placements of all static snapshots.

(8.) The LaTeX compiler outputs the final PDF document with images from Step 6.

(9.) Java reads the final PDF document as well as the PDbF interpreter (see Section 3.3).

(10.) Java reads the final PDF document, the finalized PDbF configuration, and the PDbF interpreter and outputs the resulting PDbF.

Obviously, the entire process adds a few seconds to the standard LaTeX compilation time, however the overhead is not substantial and only required once for the final deployment of the PDbF. For instance, on an Intel Core i5-3230M CPU@2.60 GHz (2 cores) we require 21 seconds total compilation time for this submission. The total file size of the PDbF is roughly the sum of 2.7 MB (for the embedded tools) plus twice the size of the static pdf plus the size of the database. For instance, the file size of this demo submission is only ~9 MB in total (5 MB of which is the database).

3.3 PDbF Interpreter

In order to display the static pdf as well as the HTML-overlay in a Web browser we extended PDF.js [12]. We store the static pdf, the database, and the config file as base64-encoded javascript strings inside the HTML. When the PDbF is viewed as an HTML-file, our PDbF interpreter first decodes this data. Then, the queries of all visualizations are executed by the alasql [1], an in-memory javascript SQL-engine. After that, all visualizations are rendered and placed on top of their static image counterpart.

4. PDBF FEATURES

In this section we discuss the features currently supported by our framework.

4.1 Support for DESKTOP Browsers and PDF Viewers

We have tested our PDbF files with the following browsers: Chrome 41, Opera 28, Firefox 36, and Safari 8. We tested the following PDF-Viewers: Adobe Reader XI, PDF-XChange Viewer, QuickPDF (Android), Google Drive-PDF-Viewer (Android).

4.2 Dynamic Graphs

In the *PDF view*, a static snapshot of each graph will be generated showing the initial configuration of the graph. In the *HTML view*, click the ⤢ symbol in the upper left corner of a graph. A window will appear with additional options. You may change the display in many ways, e.g. from linear scale to logscale, adjust display ranges, and many other options.

Data for dynamic content can be specified in three ways: inline in LaTeX, using a file with sql queries, or via JDBC connector that imports arbitrary SQL results. We support several types of graphs including multi-column bar plots (Figure 3), line plots (Figure 4), and stacked bar plots (not displayed due to space constraints). They may be specified directly in LaTeX. For instance, in order to specify a line chart you simply write:

```
\lineChart[width=\textwidth, height=0.8\textwidth,
          xunit=Date, yunit={Runtime [in sec]}]{
    SELECT date, runtimeA AS engine_A,
                 runtimeB AS engine_B FROM data2;
}
```

The dynamic graphs are built using dygraph [7]. Arbitrary options may be specified in LaTeX that are then passed to dygraph.

4.3 Dynamic Pivot Tables

We also support dynamic pivot tables. See Figure 5 for an example. This visualization is based on HTML pivot tables [13] and jQuery [8]. You may group on arbitrary keys and display grouping keys in rows and columns. This will in turn change the underlying database query on the fly.

4.4 Raw Data Access

Raw data may be "downloaded" directly from the dynamic visualizations as CSV. This may also be extended to "download" source code and/or binaries from within the PDbF.

4.5 Future Work

We only started full time development on this project in February 2015. Hence, as of March 2015 there are many things left to do. For instance, grouped stacked bar plots, change a query that produced the plot, add filtering functions, native LaTeX tables, and vectorized static graphs. In addition, we would like to add automatic support to compute measurement outliers and confidence intervals.

5. THE DEMO

Major part of the demo (Section 5.1) are already in your hands, the other parts (Sections 5.2 and 5.3) will be shown at VLDB.

5.1 The PDbF File Format

This is Contributions 1, 5, 2, and 6. This part of the demo is already contained in this document. We invite the reader and (also the audience at VLDB) to change the suffix of this file from ".pdf" to ".html" (on your desktop computer). Your desktop browser will open and you will be able to see the dynamic features explained in Section 4. Notice that the dynamic content is completely offline and runs entirely in your Web browser's sandbox (Firefox, Chrome or Safari).

5.2 The PDbF Compiler

This is Contribution 3. We will invite the audience to create PDbF-files themselves on arbitrary input documents. These documents can then be enriched with dynamic content using our compiler. The final result may be viewed with a desktop Web browser.

5.3 LaTeX Integration

This is Contribution 4, see also Section 3.2. We invite readers to bring their own LaTeX files. We will show them how to prepare their tex-documents in order to be able to define graphs directly on the raw data. We will demonstrate the seamless integration of our technology into existing LaTeX compilers.

6. CONCLUSIONS

This demo opened the book for portable database files (PDbF). PDbFs allow you to embed raw data into a document while preserving the entire data-pipeline from raw data to graphs. This allows readers to perform in-document OLAP-style analytics on the published document. PDbFs are janiform documents that are at the same time a valid pdf and a valid HTML document. Thus they can
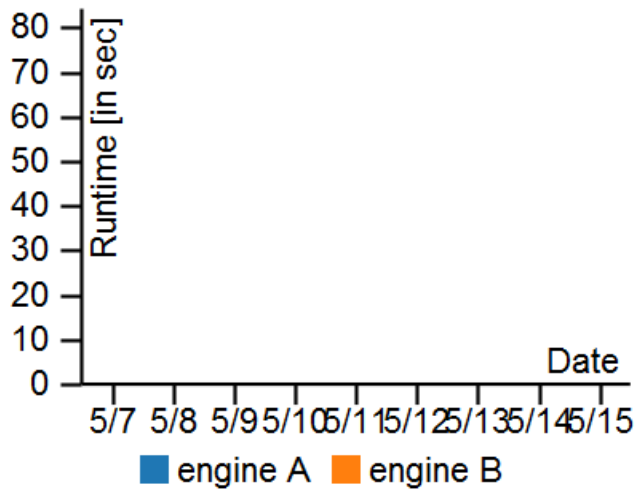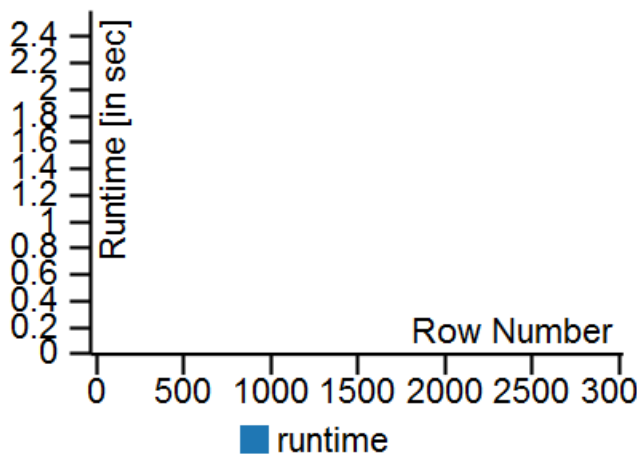
Figure 3: Multi-column Bar Chart

Figure 4: Line chart

Machine	Table schema	Query	Compiler
Ivy Bridge	char	Q1	icpc
		Q2	g++
	int	Q1	icpc
		Q2	g++
	long	Q1	icpc
		Q2	g++
Nehalem	char	Q1	icpc
		Q2	g++
	int	Q1	clang++
		Q2	g++
	long	Q1	icpc
		Q2	clang++
Sandy Bridge	char	Q1	icpc
		Q2	g++
	int	Q1	icpc
		Q2	g++

Figure 5: Pivot table

be viewed in either way. This pdf is an example of such a janiform PDbF. In addition, the conference demo show-cases two compilers allowing you to seamlessly create PDbFs, also from within LaTeX.

As part of future work we want to research how to include other parts of the research pipeline as part of the pdf. For instance, in future, one might consider to embed a virtual machine emulator, an operating system image, and all software that is required to run the original code *within the pdf* (∼3GB of data). This may sound infeasible in 2015. But 4K video streaming over the Internet sounded equally silly in 1995.

7. REFERENCES

[1] https://github.com/agershun/alasql.
[2] A. Albertini. Funky File Formats. In *CCC*. 2014.
[3] T. G. Armstrong, A. Moffat, W. Webber, and J. Zobel. Improvements That Don't Add Up: Ad-hoc Retrieval Results Since 1998. CIKM '09, pages 601–610, New York, NY, USA, 2009. ACM.
[4] Dagstuhl Perspectives Workshop 15452: Artifact Evaluation for Publications. 11/2015. http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=15452.
[5] J. Dittrich. The Case for Small Data Management. In *CIDR*, 2015.
[6] J. Dittrich. The Case for Small Data Management. In *keynote at BTW*, 3/2015. extended version of [5] (youtube).
[7] http://dygraphs.com.
[8] https://jquery.com.
[9] S. Krishnamurthi and J. Vitek. The Real Software Crisis: Repeatability As a Core Value. *Commun. ACM*, 58(3):34–36, Feb. 2015.
[10] I. Manolescu, L. Afanasiev, A. Arion, J. Dittrich, S. Manegold, N. Polyzotis, K. Schnaitter, P. Senellart, S. Zoupanos, and D. Shasha. The Repeatability Experiment of SIGMOD 2008. *SIGMOD Rec.*, 37(1):39–45, Mar. 2008.
[11] J. P. Mesirov. COMPUTER SCIENCE. Accessible Reproducible Research. *Science (New York, N.Y.)*, 327(5964):10.1126/science.1179653, 01 2010.
[12] https://mozilla.github.io/pdf.js.
[13] https://github.com/nicolaskruchten/pivottable.
[14] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig. Ten Simple Rules for Reproducible Computational Research. *PLoS Comput Biol*, 9(10):e1003285, 10 2013. http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003285.

runtimeA AS [engine A]

runtimeB AS [engine B]

Runtime [in sec]

77
76
75
74
73
72
71
70

72.5
72
71.5
71
70.5
70
69.5

5/7   5/8   5/9   5/10 5/11   5/12   5/13   5/14   5/15

Date

■ engine A ■ engine B