Poor Usability in Data Processing

Adam Marcus
Locu/GoDaddy
marcua@marcua.net

1. INTRODUCTION

The databases community works hard on the scale, performance, and correctness of the storage and query processing systems that our users depend on. Researchers are therefore frustrated to see less principled, and often incorrect 2010s implementations of concepts that were introduced in the 1970s. The lens of *usability* can help us understand how certain systems see adoption, regardless of the soundness of their implementation. Usability deficiencies in best-of-class systems can explain the success of systems with poor transactional semantics or unideal query languages, and even the use of spreadsheets for large-scale data management.

2. FROM DISCOVERY TO DEPLOYMENT

In a world where small teams can read about, prototype, and deploy reasonable open source data processing systems in a few hours, it's not uncommon for important system deficiencies to slip through the cracks. Here are some questions developers ask themselves as they explore a new system that might explain how a usable, approachable system might be adopted in place of a more ideal one.

Understanding the README. Is there a single document a developer can look at that contains the problem, solution, copy/pasteable examples, steps to deployment, and how to interact with the community? That's tablestakes on github, but not for research contributions.

A 15-minute sniff test. Can the developer walk through parts of the codebase and get a development system up and running in 15 minutes? Nothing beats seeing a system solve your problem, especially with some low-effort prototyping.

Multi-environment deployment. Deploying a system in development, staging, and production is the next common painful experience on the path to adoption.

Edge cases that stress deployments. Once deployed, a bunch of issues pop up that teach you which performance, scale, and correctness guarantees you wish your deployed system had provided.

The research community often focuses on the deployment-stressing edge cases that pop up in the last phase. This last phase provides lots of interesting research nuggets in areas like performance, transactional semantics, and coordination tradeoffs, but it's the last thing a developer is thinking of when they think of adopting your system. The first three phases might be less intellectually stimulating to the research community, but usability failures in the first three phases can explain much of the reason why good systems aren't adopted over ones that focused on the first three phases of the developer experience.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2015.
7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.
CIDR 2015

3. USABILITY CHALLENGES

Here are a few meaningful usability challenges that developers run into in adopting various systems and APIs.

Transactional APIs. Failure cases dictate that RDBMS transactions must be wrapped in retry loops and multiple *try/catch* statements. We need APIs to help developers avoid thinking about this[1].

Transactions across systems. Developers savvy enough to use database transactions run into new issues when they use systems like messaging queues that aren't aware of the transactional semantics of the database. We need to make it easier to usably manage transactions across systems.

Schema awareness. Data lives outside the database: on the wire for communication, on disk for logging, and in memory for caching. These layers should be aware of the same schema and migration logic for versioned serializations.

Logging. It's common to see logs in multiple text files across *YYYY/MM/DD/HH* folder trees that developers glob together to do log analysis. We need better APIs for collecting, retrieving, and loading the data into analytics systems.

External dependencies. Getting up and running often requires depending on other systems for storage, messaging, coordination, logging, transactions, and analysis. These dependencies complicate the 15-minute adoption sniff test, and would ideally be stubbed out for prototyping and testing.

Multinode deployments. If you develop a system that can effectively scale to multiple nodes, make it easy for a developer to see this happen in practice. Providing tools to show off your new approach to sharding and failures will make it easier for developers to appreciate them.

Small-scale data cleaning, understanding, and management. Products like Trifacta and Tableau make it easier to prepare data and answer questions about it, but we're nowhere near a world where data cleaning and exploration tools are accessible, affordable, and deployable at the small scale. The open source world has a long way to go before that's the case.

[1]Evan Jones gave a great talk on this topic at New England Database Day 2014.