A **F**abric For **Re**producible **Co**mputation

FRESCO : Understanding the Provenance of Data

Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Lucian Carata, Andy Hopper, Ripduman Sohan

UNIVERSITY OF CAMBRIDGE

Introduction

- Do you believe these outrageous headlines? Maybe they're true? Wouldn't it be useful to be able to check both the **sources** and **calculations** that lead up to their publication?
- Data deluges us – each day (2.5) quintillion bytes of data ($2.5 \cdot 10^{18}$) are created and eventually incorporated, either directly or indirectly, into other data items.
- Our options for checking their sources and accuracy are primitive. ***We depend on data, but cannot easily verify if it is dependable.***
- In **FRESCO** we are building systems for automating data provenance capture, management and query, providing insights into the sources and computations that have produced a given data item.

Asking the right questions

- Where does this information come from?
- Under what conditions is it valid?
- How was it interpreted, transformed and processed?
- Does it depend on other pieces of data?

All of these are valid questions that allow us to reason about claims made using the data, whether they are published in a newspaper article or in a peer-reviewed journal.

Provenance Capture

Two fundamental approaches to provenance are explored in FRESCO

Disclosed Provenance

Provenance data is captured either from manual annotations or by explicit disclosure in the application source code.

Correctly disclosed provenance can lead to rich and descriptive provenance with low capture overhead.

The developer needs to make applications provenance aware by appropriatelly using a provenance API.

Observed Provenance

Provenance is captured by observing an application as it carries out computation and I/O.

The observation can be performed at one or more layers such as within the application (e.g. function call interposition), operating system (e.g. through tracing system calls) or hardware (e.g. branch tracing).

The level of provenance information captured is directly correlated to the overhead of capture.

IPAPI (An Improved Provenance API)

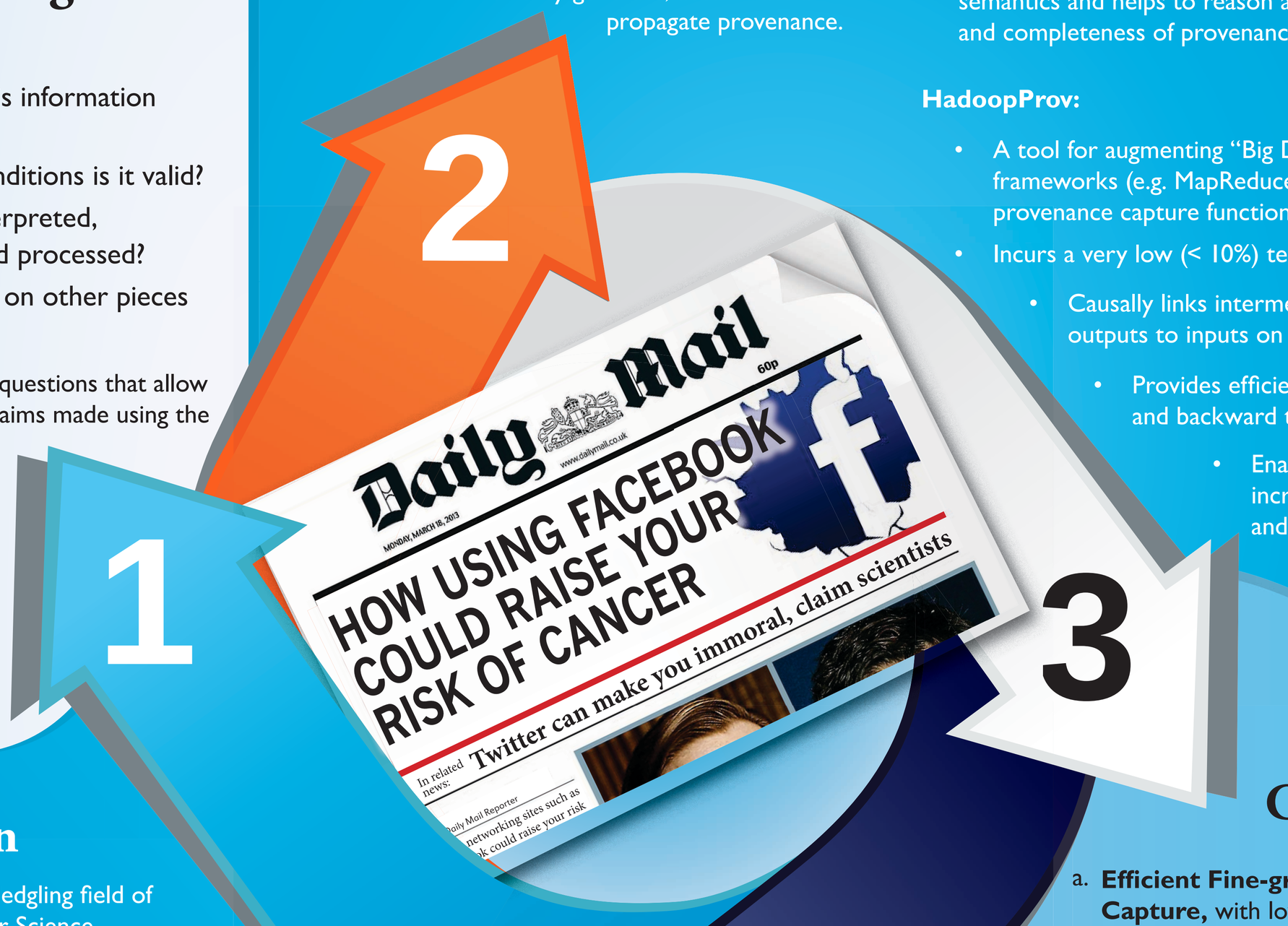
- Aims for completeness and flexibility, allowing for provenance capture at **multiple granularities** and in **distributed environments**.
- The API is the application-facing component of a decentralised architecture built around the notion of "provenance repositories"
- Similar to distributed version control systems like git, but for provenance.
- Defines a mapping between system-level entities (processes, files, pipes, shared memory) and the manner in which they generate, accumulate and propagate provenance.

OPUS (Observational Provenance In User-Space)

- A general purpose process-level provenance capture system for POSIX environments.
- Designed to be simple, easy to deploy and use and lightweight in terms of resource consumption.
- Comprises of a front-end that captures library level, function call and argument information and a back-end that converts this information to provenance data.
- Implements a semi-formal **Provenance Versioning Model** (PVM).
- PVM abstracts away the underlying file system semantics and helps to reason about the properties and completeness of provenance capture semantics.

HadoopProv:

- A tool for augmenting "Big Data" programming frameworks (e.g. MapReduce) with fined-grained provenance capture functionality.
- Incurs a very low (< 10%) temporal overhead.
- Causally links intermediate and final outputs to inputs on a per-record level.
- Provides efficient means for forward and backward tracing of data items.
- Enables more efficient incremental computation and log analysis.



Challenges

a. **Efficient Fine-grained Provenance Capture,** with low temporal overhead and efficient storage representations.

b. **Useful Query Functionality:** Not only should it be possible to quickly carry out simple queries but it should also be possible to construct "what-if" scenarios by replaying a set of system events on existing entities.

c. **Distributed Provenance Capture:** In order to be useful in production environments, it is necessary to capture and identify related provenance information across hosts.

d. **Consistent View Across Systems:** We anticipate provenance will be simultaneously captured by both observational and declarative tools, possibly at multiple levels. In order to gain a holistic view of the provenance information of a given event, provenance captured between sources should be related in a system agnostic manner.

Conclusion

Provenance is still a fledgling field of research in Computer Science.
We believe the work of the FRESCO project will result in general purpose systems that allow users to more confidently reason about the sources and accuracy of their data.

Our ultimate goal is to make **provenance support** a *fundamental aspect of all general purpose computing systems.*

FRESCO has wider implications beyond the project. Support for data provenance as a first class construct of modern computing systems will enable users to reason more confidently about the origin of data and verify the correctness of the computations carried out to derive it. This, in turn, should allow users to categorise data according to trustworthiness and ultimately result in an increase in the quality of data on which computations, and by extension, decisions are based.