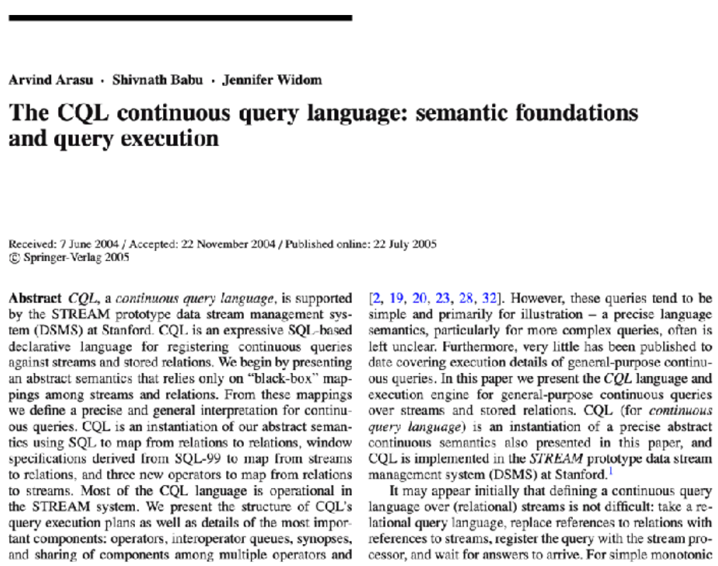


Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

Abstract *CQL*, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics that relies only on "black-box" mappings among streams and relations. From these mappings we define a precise and general interpretation for continuous queries. CQL is an instantiation of our abstract semantics using SQL to map from relations to relations, window specifications derived from SQL-99 to map from streams to relations, and three new operators to map from relations to streams. Most of the CQL language is operational in the STREAM system. We present the structure of CQL's query execution plans as well as details of the most important components: operators, interoperator queues, synopses, and sharing of components among multiple operators and

[2, 19, 20, 23, 28, 32]. However, these queries tend to be simple and primarily for illustration – a precise language semantics, particularly for more complex queries, often is left unclear. Furthermore, very little has been published to date covering execution details of general-purpose continuous queries. In this paper we present the *CQL* language and execution engine for general-purpose continuous queries over streams and stored relations. CQL (for *continuous query language*) is an instantiation of a precise abstract continuous semantics also presented in this paper, and CQL is implemented in the *STREAM* prototype data stream management system (DSMS) at Stanford.[1]

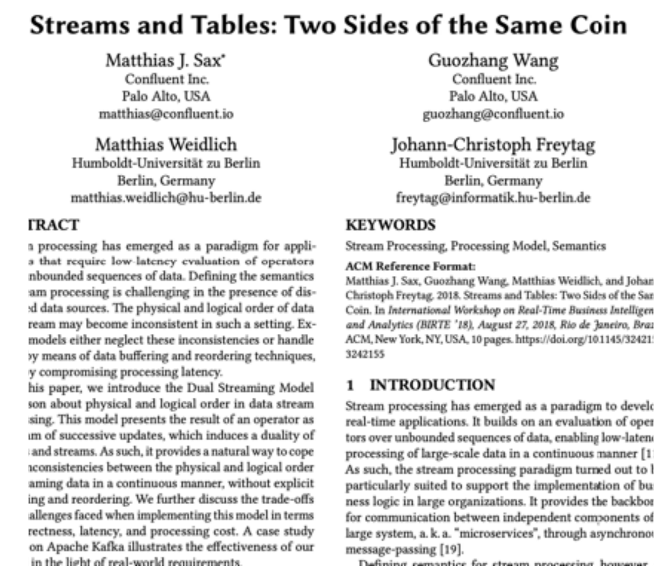
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

A (Query) Language Perspective

It addresses the problem of manipulating and managing data-streams**.** It stresses on the **operations** that are **necessary** to build **applications**. Some **Assumptions** are made on the underlying **system**(s).

A ~~new~~ hybrid ~~hope~~ perspective

It addresses the problems in between. How do we map the abstraction of an high-level language to the underlying system?



Streams and Tables: Two Sides of the Same Coin

Matthias J. Sax*
Confluent Inc.
Palo Alto, USA
matthias@confluent.io

Guozhang Wang
Confluent Inc.
Palo Alto, USA
guozhang@confluent.io

Matthias Weidlich
Humboldt-Universität zu Berlin
Berlin, Germany
matthias.weidlich@hu-berlin.de

Johann-Christoph Freytag
Humboldt-Universität zu Berlin
Berlin, Germany
freytag@informatik.hu-berlin.de

TRACT

n processing has emerged as a paradigm for appli- s that require low-latency evaluation of operators nbounded sequences of data. Defining the semantics am processing is challenging in the presence of dis- d data sources. The physical and logical order of data ream may become inconsistent in such a setting. Ex- models either neglect these inconsistencies or handle y means of data buffering and reordering techniques, y compromising processing latency.

his paper, we introduce the Dual Streaming Model son about physical and logical order in data stream sing. This model presents the result of an operator as m of successive updates, which induces a duality of and streams. As such, it provides a natural way to cope nconsistencies between the physical and logical order aming data in a continuous manner, without explicit ing and reordering. We further discuss the trade-offs allenges faced when implementing this model in terms rectness, latency, and processing cost. A case study on Apache Kafka illustrates the effectiveness of our in the light of real-world requirements.

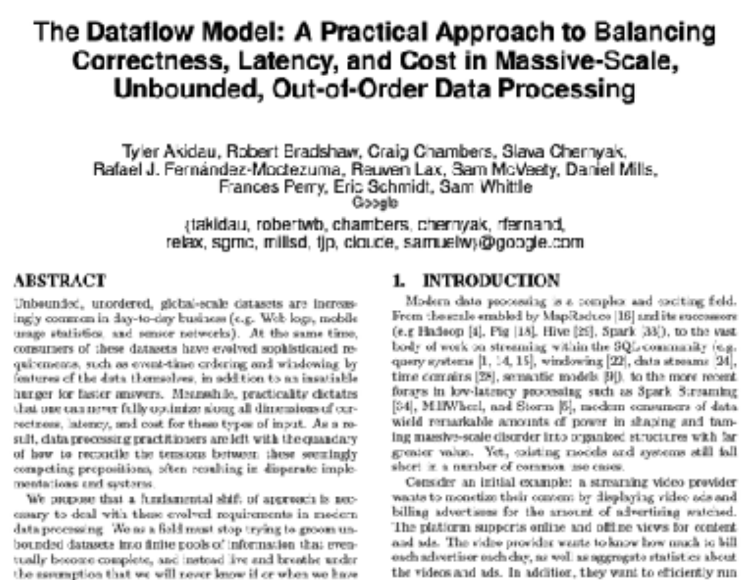
KEYWORDS

Stream Processing, Processing Model, Semantics

ACM Reference Format:
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johan Christoph Freytag. 2018. Streams and Tables: Two Sides of the Sar Coin. In *International Workshop on Real-Time Business Intelligen and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Bra* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/32421 3242155

1 INTRODUCTION

Stream processing has emerged as a paradigm to devel real-time applications. It builds on an evaluation of oper tors over unbounded sequences of data, enabling low-laten processing of large-scale data in a continuous manner [1 As such, the stream processing paradigm turned out to l particularly suited to support the implementation of bu ness logic in large organizations. It provides the backbo for communication between independent components of large system, a. k. a. "microservices", through asynchrono message-passing [19].



The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle
Google
{takidau, robertwb, chambers, chernyak, rfernand, relax, sgmc, millsd, fjp, cloude, samuelw}@google.com

ABSTRACT

Unbounded, unordered, global-scale datasets are increasingly common in day-to-day business (e.g. Web logs, mobile usage statistics, and sensor networks). At the same time, consumers of these datasets have evolved sophisticated requirements, such as event-time ordering and windowing by features of the data themselves, in addition to an insatiable hunger for faster answers. Meanwhile, practicality dictates that one can never fully optimize along all dimensions of correctness, latency, and cost for these types of input. As a result, data processing practitioners are left with the quandary of how to reconcile the tensions between these seemingly competing propositions, often resulting in disparate implementations and systems.

We propose that a fundamental shift of approach is necessary to deal with these evolved requirements in modern data processing. We as a field must stop trying to groom unbounded datasets into finite pools of information that eventually become complete, and instead live and breathe under the assumption that we will never know if or when we have

1. INTRODUCTION

Modern data processing is a complex and exciting field. From the scale enabled by MapReduce [16] and its successors (e.g Hadoop [4], Pig [18], Hive [29], Spark [33]), to the vast body of work on streaming within the SQL community (e.g. query systems [1, 14, 15], windowing [22], data streams [24], time domains [28], semantic models [9]), to the more recent forays in low-latency processing such as Spark Streaming [34], MillWheel, and Storm [5], modern consumers of data wield remarkable amounts of power in shaping and taming massive-scale disorder into organized structures with far greater value. Yet, existing models and systems still fall short in a number of common use cases.

Consider an initial example: a streaming video provider wants to monetize their content by displaying video ads and billing advertisers for the amount of advertising watched. The platform supports online and offline views for content and ads. The video provider wants to know how much to bill each advertiser each day, as well as aggregate statistics about the videos and ads. In addition, they want to efficiently run

A System Perspective

It address the problem of dealing with **unbounded data**. Discuss the systems' **primitives** that are necessary to guarantee **low-latency** and **fault-tolerance in** presence of *uncertainty*, e.g., late arrivals.