It’s a StreamER World

A Journey Through Processing Flows of Data

Paper We Love ⋈ Kafka Meetup

Who I Am

Emanuele Della Valle
Professor @ Polimi
Inventor of Stream Reasoning
Kafka Enthusiast

Who I Am

Riccardo Tommasini
PhD Candidate @ Polimi
Thesis: Velocity on the Web
Future AssistantProfessor @
University of Tartu, Estonia
Enthusiast!

Timeline

Precision not Recall

2002 2005 2006 2007 2008 2010 2014 2015 2018

The Origin

Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

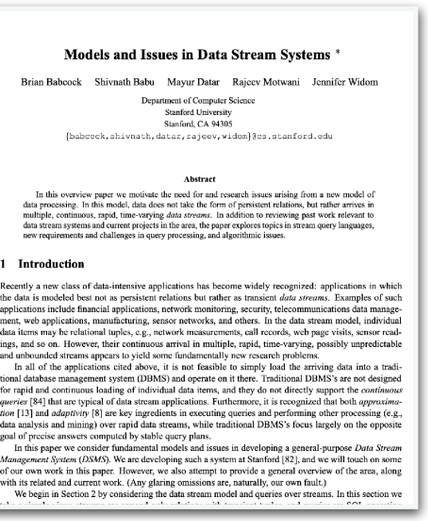
Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

1 Introduction

Timeline

Precision not Recall



Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we

The 8 Requirements of Real-Time Stream Processing

ABSTRACT

It's a Streaming World! Reasoning upon Rapidly Changing Information

MOA: Massive Online Analysis

Abstract

1. Introduction

Dynamically Scaling

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

ABSTRACT

1. INTRODUCTION

Apache Samza

Apache Flink™: Stream and Batch Processing in a Single Engine

Abstract

1 Introduction

2002 2005 2006 2007 2008 2010 2014 2015 2018

The Vision

The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data steams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap micro-sensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This sensorization of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluation alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to corn flakes factories, will also move to such "firehose" data volumes and sub-second latency requirements.

There is a "sea change" arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., mote [1] and Lojack [2]). Over time, this sea change will cause everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting vital-signs monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost

Timeline

Precision not Recall

Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we

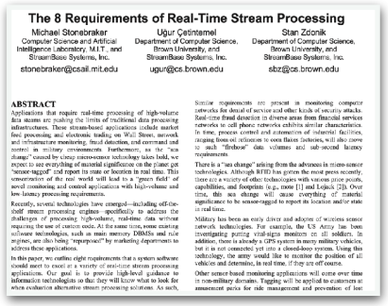
MOA: Massive Online Analysis

Abstract

Dynamically Scaling

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

Apache Samza



The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap microsensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This sensorization of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluation alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to own fusion factories, will also move to such "firehose" data volumes and sub-second latency requirements.

There is a "sea change" arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., mote [1] and Lojack [2]). Over time, this sea change will entail everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting viral-signs monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost

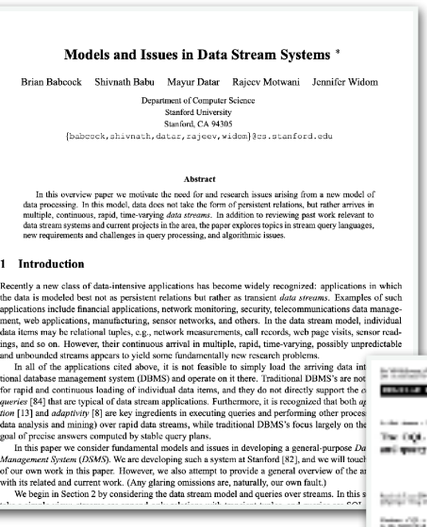
It's a Streaming World! Reasoning upon Rapidly Changing Information

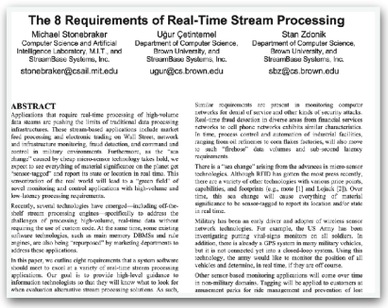
Apache Flink™: Stream and Batch Processing in a Single Engine

2002 2005 2006 2007 2008 2010 2014 2015 2018

Timeline

Precision not Recall





2002 2005 2006 2007 2008 2010 2014 2015 2018

The VLDB Journal (2006) 15(2): 121–142
DOI 10.1007/s00778-004-0147-z

REGULAR PAPER

Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

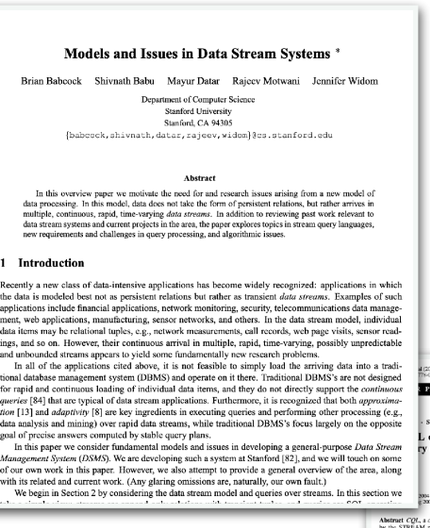
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

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 queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL. The relative ease of capturing these applications in CQL is [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 queries over complete stream histories, indeed this approach is nearly sufficient. However, as queries get more complex – when we add aggregation, subqueries, windowing constructs, relations mixed with streams, etc. – the situation becomes much murkier. Consider the following simple query:

Timeline

Precision not Recall



Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

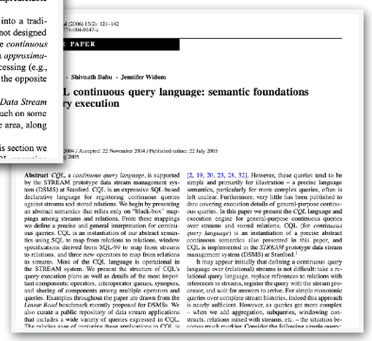
1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we



PAPER

Shivnath Babu · Jennifer Widom

L continuous query language: semantic foundations
ry execution

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 queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL. The relative ease of formulating these applications in CQL is

[2, 19, 20, 23, 24, 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 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 queries over complete stream histories, indeed this approach is nearly sufficient. However, as queries get more complex – when we add aggregation, subqueries, windowing constructs, relations mixed with streams, etc. – the situation becomes much trickier. Consider the following simple monoto-

MOA: Massive Online Analysis

Albert Bifet
Geoff Holmes
Richard Kirkby
Bernhard Pfahringer
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Mikio Braun

Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naive Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning methods.

A data stream environment has different requirements from the traditional batch learning setting. The most significant are the following:

Requirement 1 Process an example at a time, and inspect it only once (at most)

Requirement 2 Use a limited amount of memory

Dynamically Scaling
of

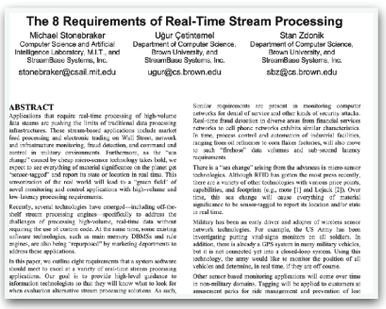
The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing

ABSTRACT

1. INTRODUCTION

Apache Samza

Sides of the Same C



The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap microsensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This sensorization of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluation alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to even fusion factories, will also move to such "firehose" data volumes and sub-second latency requirements.

There is a "sea change" arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., motes [1] and Lojack 2.0). Over time, this sea change will cause everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting viral-sigma monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost

THE SEMANTIC WEB

It's a Streaming World!
Reasoning upon Rapidly
Changing Information

Apache Flink™: Stream and Batch Processing in a Single Engine

Paris Carbone
Asterios Katsifodimos

Stephan Ewen
Volker Markl

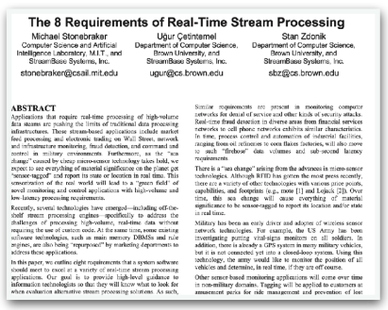
Seif Haridi
Kostas Tzoumas

Abstract

2002 2005 2006 2007 2008 2010 2014 2015 2018

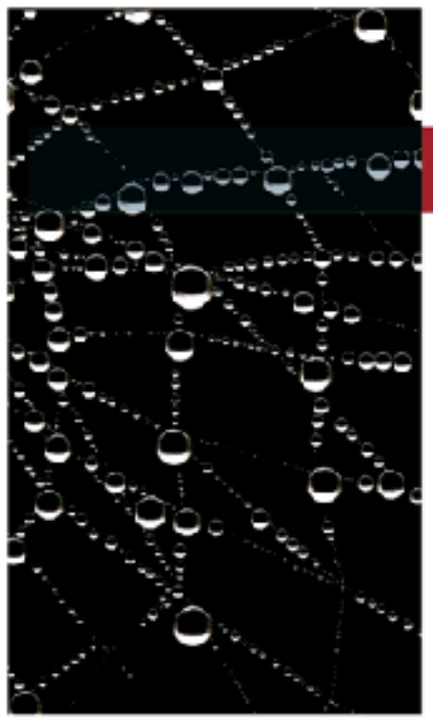
Timeline

Precision not Recall



2002 2005 2006 2007 2008 2010 2014 2015 2018

The Title



THE SEMANTIC WEB

Editor: **Steffen Staab,** University of Koblenz-Landau, staab@uni-koblenz.de

It's a Streaming World! Reasoning upon Rapidly Changing Information

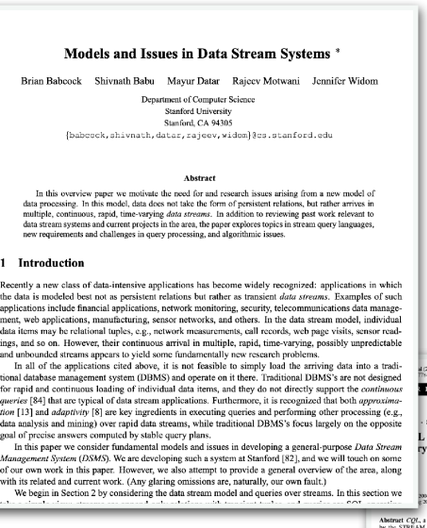
Emanuele Della Valle and **Stefano Ceri**, *Politecnico di Milano*
Frank van Harmelen, *Vrije Universiteit Amsterdam*
Dieter Fensel, *University of Innsbruck*

Will there be a traffic jam on this highway? Can we reroute travelers on the basis of the forecast? By examining the clickstream from a given IP, can we discover shifts in interest of the person behind the computer? Which content on the news Web portal is attracting the most attention? Which navigation pattern would lead readers to other news related to that content? Do trends in medical records indicate any new disease spreading in a given part of the world? Where are all my

The state of the art in reasoning over changing worlds is based on temporal logic and belief revision; these are heavyweight tools, suitable for data that changes in low volumes at low frequency. Similarly, the problem of changing vocabularies and evolving ontologies has undergone thorough investigation, but here the standard practice relies on configuration management techniques taken from software engineering, such as vocabulary and ontology versioning. These are suitable for ontologies that change on a weekly or monthly basis, but not

Timeline

Precision not Recall



Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

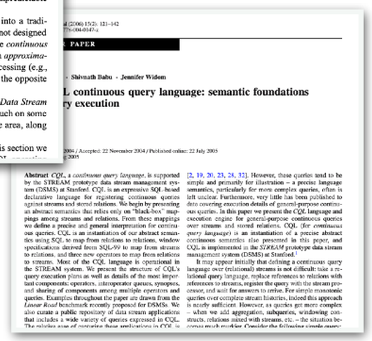
1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we



PAPER

Shivnath Babu · Jennifer Widom

L continuous query language: semantic foundations
y execution

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 queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL.

MOA: Massive Online Analysis

Albert Bifet
Geoff Holmes
Richard Kirkby
Bernhard Pfahringer
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Mikio Braun

Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naive Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning methods.

Keywords: data streams, classification, ensemble methods, java, machine learning software

1. Introduction

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

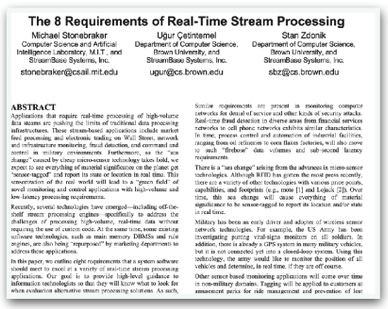
Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle
Google
{takidau, slava, chambers, chernyak, rfernand, relax, sgmc, millsd, fjp, eschmidt, samuelw}@google.com

ABSTRACT

1. INTRODUCTION

Apache Samza

Slides of the Same C



The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap microsensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This sensorization of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

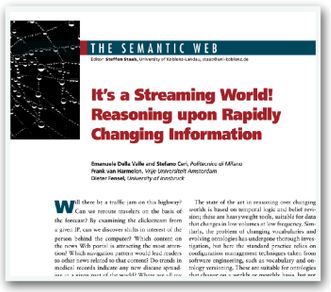
In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluating alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to corn flakes factories, will also move to such "firehose" data volumes and sub-second latency requirements.

There is a "sea change" arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., motes [1] and Lojack 2.0). Over time, this sea change will cause everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting viral-signs monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost



THE SEMANTIC WEB

Editor: Steffen Staab, University of Koblenz-Landau, staab@uni-koblenz.de

It's a Streaming World! Reasoning upon Rapidly Changing Information

Emanuele Della Valle and Stefano Ceri, *Politecnico di Milano*
Frank van Harmelen, *Vrije Universiteit Amsterdam*
Dieter Fensel, *University of Innsbruck*

Will there be a traffic jam on this highway? Can we execute travelers on the basis of the forecast? By examining the clickstream from a given IP, can we discover skills in interest of the person behind the computer? Which content on the news Web portal is attracting the most attention? Which navigation pattern would lead readers to other news related to that content? Do trends in medical records indicate any new disease spread

The state of the art in assessing over changing worlds is based on temporal logic and belief revision; these are heavyweight tools, suitable for data that changes in low volumes or low frequency. Similarly, the problem of changing vocabularies and evolving ontologies has undergone thorough investigation, but here the standard practice relies on configuration management techniques taken from software engineering, such as vocabulary and ontology versioning. These are suitable for ontologies

Apache Flink[TM]: Stream and Batch Processing in a Single Engine

Paris Carbone[*]
Asterios Katsifodimos[†]

Stephan Ewen[†]
Volker Markl[†]

Seif Haridi[*]
Kostas Tzoumas[†]

[*]KTH/SICS Sweden

[†]Data Artisans

[†]TU Berlin & DFKI

Abstract

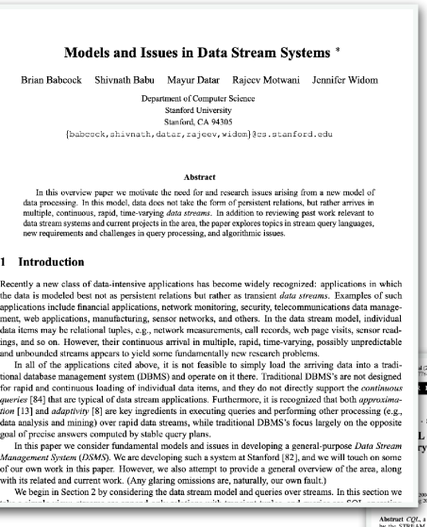
Apache Flink[1] is an open-source system for processing streaming and batch data. Flink is built on the philosophy that many classes of data processing applications, including real-time analytics, continuous data pipelines, historic data processing (batch), and iterative algorithms (machine learning, graph analysis) can be expressed and executed as pipelined fault-tolerant dataflows. In this paper, we present Flink's architecture and expand on how a (seemingly diverse) set of use cases can be unified under a single execution model.

1 Introduction

2002 2005 2006 2007 2008 2010 2014 2015 2018

Timeline

Precision not Recall


Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

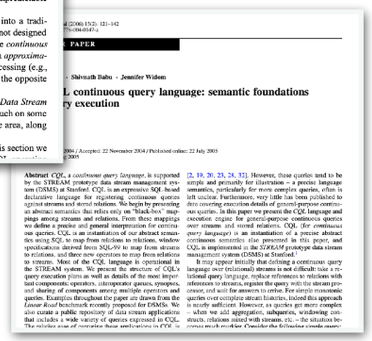
1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

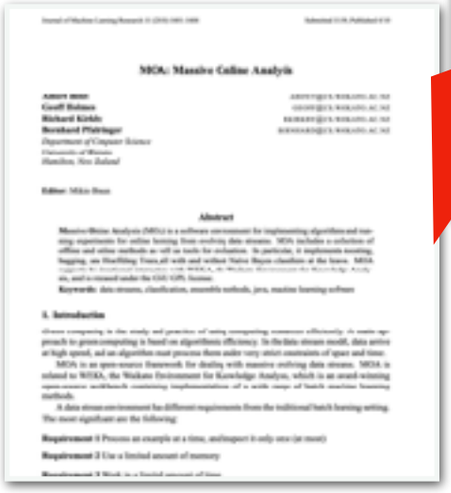
We begin in Section 2 by considering the data stream model and queries over streams. In this section we


PAPER

Shivnath Babu · Jennifer Widom

L continuous query language: semantic foundations
ry execution

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 queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL.


MOA: Massive Online Analysis

Abstract


Dynamically Scaling Apach
of Streami


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

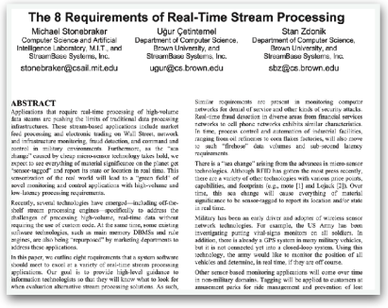
ABSTRACT

1. INTRODUCTION

Apache Samza


Sides of the Same C

Stream Processing + AI
(Inductive)


The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

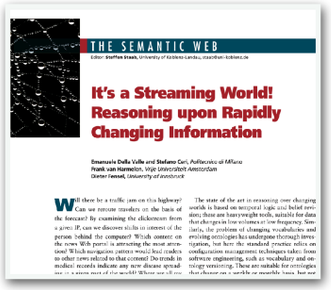
Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap microsensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This *sensorization* of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluating alternative stream processing solutions. As such,

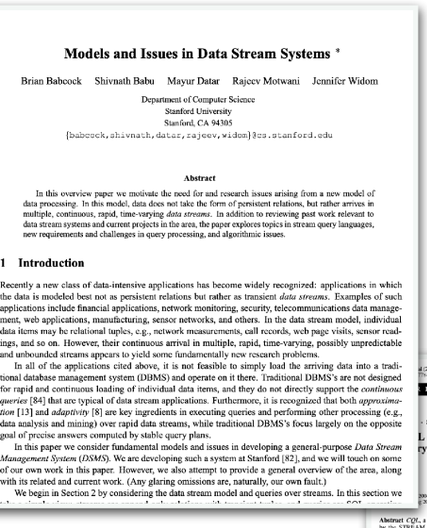

THE SEMANTIC WEB

It's a Streaming World! Reasoning upon Rapidly Changing Information

2002 2005 2006 2007 2008 2010 2014 2015 2018

Timeline

Precision not Recall



Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

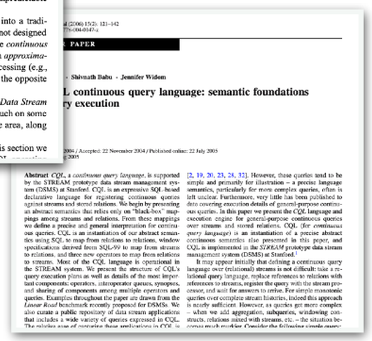
1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we



PAPER

Shivnath Babu · Jennifer Widom

L continuous query language: semantic foundations
y execution

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 queries. Examples throughout the paper are drawn from the *Linear Road* benchmark recently proposed for DSMSs. We also curate a public repository of data stream applications that includes a wide variety of queries expressed in CQL.

MOA: Massive Online Analysis

Albert Bifet
Geoff Holmes
Richard Kirkby
Bernhard Pfahringer
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Mikio Braun

Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naive Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning methods.

Keywords: data streams, classification, ensemble methods, java, machine learning software

1. Introduction

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

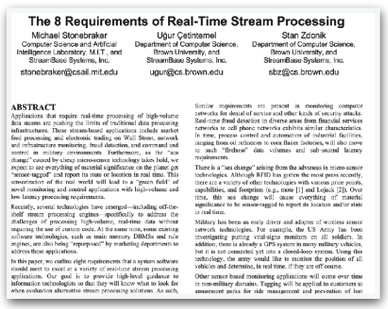
Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, Sam Whittle
Google
{takidau, slava, chambers, chernyak, rfernand, relax, sgmc, millsd, fjp, eschmidt, samuelw}@google.com

ABSTRACT

1. INTRODUCTION

Apache Samza

Slides of the Same C



The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

Applications that require real-time processing of high-volume data streams are pushing the limits of traditional data processing infrastructures. These stream-based applications include market feed processing and electronic trading on Wall Street, network and infrastructure monitoring, fraud detection, and command and control in military environments. Furthermore, as the "sea change" caused by cheap microsensor technology takes hold, we expect to see everything of material significance on the planet get "sensor-tagged" and report its state or location in real time. This sensorization of the real world will lead to a "green field" of novel monitoring and control applications with high-volume and low-latency processing requirements.

Recently, several technologies have emerged—including off-the-shelf stream processing engines—specifically to address the challenges of processing high-volume, real-time data without requiring the use of custom code. At the same time, some existing software technologies, such as main memory DBMSs and rule engines, are also being "repurposed" by marketing departments to address these applications.

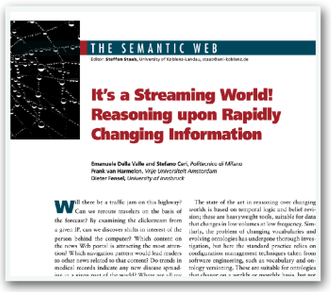
In this paper, we outline eight requirements that a system software should meet to excel at a variety of real-time stream processing applications. Our goal is to provide high-level guidance to information technologists so that they will know what to look for when evaluating alternative stream processing solutions. As such,

Similar requirements are present in monitoring computer networks for denial of service and other kinds of security attacks. Real-time fraud detection in diverse areas from financial services networks to cell phone networks exhibits similar characteristics. In time, process control and automation of industrial facilities, ranging from oil refineries to corn flakes factories, will also move to such "firehose" data volumes and sub-second latency requirements.

There is a "sea change" arising from the advances in micro-sensor technologies. Although RFID has gotten the most press recently, there are a variety of other technologies with various price points, capabilities, and footprints (e.g., motes [1] and Lojack 2.0). Over time, this sea change will cause everything of material significance to be sensor-tagged to report its location and/or state in real time.

Military has been an early driver and adopter of wireless sensor network technologies. For example, the US Army has been investigating putting viral-signs monitors on all soldiers. In addition, there is already a GPS system in many military vehicles, but it is not connected yet into a closed-loop system. Using this technology, the army would like to monitor the position of all vehicles and determine, in real time, if they are off course.

Other sensor-based monitoring applications will come over time in non-military domains. Tagging will be applied to customers at amusement parks for ride management and prevention of lost



THE SEMANTIC WEB

Editor: Steffen Staab, University of Koblenz-Landau, staab@uni-koblenz.de

It's a Streaming World! Reasoning upon Rapidly Changing Information

Emanuele Della Valle and Stefano Ceri, *Politecnico di Milano*
Frank van Harmelen, *Vrije Universiteit Amsterdam*
Dieter Fensel, *University of Innsbruck*

Will there be a traffic jam on this highway? Can we execute travelers on the basis of the forecast? By examining the clickstream from a given IP, can we discover skills in interest of the person behind the computer? Which content on the news Web portal is attracting the most attention? Which navigation pattern would lead readers to other news related to that content? Do trends in medical records indicate any new disease spread

The state of the art in assessing over changing worlds is based on temporal logic and belief revision; these are heavyweight tools, suitable for data that changes in low volumes or low frequency. Similarly, the problem of changing vocabularies and evolving ontologies has undergone thorough investigation, but here the standard practice relies on configuration management techniques taken from software engineering, such as vocabulary and ontology versioning. These are suitable for ontologies

Apache Flink[TM]: Stream and Batch Processing in a Single Engine

Paris Carbone[*]
Asterios Katsifodimos[†]

Stephan Ewen[†]
Volker Markl[†]

Seif Haridi[*]
Kostas Tzoumas[†]

[*]KTH/SICS Sweden

[†]Data Artisans

[†]TU Berlin & DFKI

Abstract

Apache Flink[1] is an open-source system for processing streaming and batch data. Flink is built on the philosophy that many classes of data processing applications, including real-time analytics, continuous data pipelines, historic data processing (batch), and iterative algorithms (machine learning, graph analysis) can be expressed and executed as pipelined fault-tolerant dataflows. In this paper, we present Flink's architecture and expand on how a (seemingly diverse) set of use cases can be unified under a single execution model.

1 Introduction

2002 2005 2006 2007 2008 2010 2014 2015 2018

Journal of Machine Learning Research 11 (2010) 1601-1604 Submitted 11/09; Published 4/10

MOA: Massive Online Analysis

Albert Bifet ABIFET@CS.WAIKATO.AC.NZ
Geoff Holmes GEOFF@CS.WAIKATO.AC.NZ
Richard Kirkby RKIRKBY@CS.WAIKATO.AC.NZ
Bernhard Pfahringer BERNHARD@CS.WAIKATO.AC.NZ
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Mikio Braun

Abstract

Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams. MOA includes a collection of offline and online methods as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, all with and without Naïve Bayes classifiers at the leaves. MOA supports bi-directional interaction with WEKA, the Waikato Environment for Knowledge Analysis, and is released under the GNU GPL license.

Keywords: data streams, classification, ensemble methods, java, machine learning software

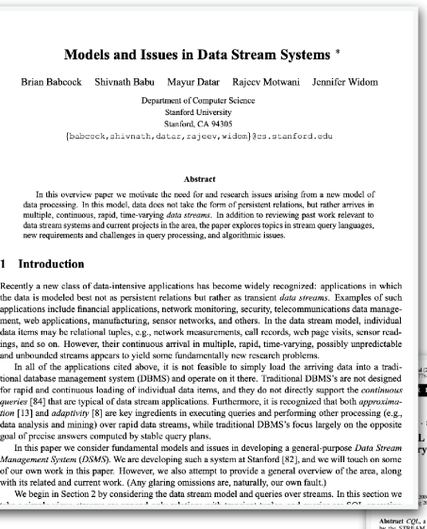
1. Introduction

Green computing is the study and practice of using computing resources efficiently. A main approach to green computing is based on algorithmic efficiency. In the data stream model, data arrive at high speed, and an algorithm must process them under very strict constraints of space and time.

MOA is an open-source framework for dealing with massive evolving data streams. MOA is related to WEKA, the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine learning

Timeline

Precision not Recall


Models and Issues in Data Stream Systems *

Brian Babcock Shivnath Babu Mayur Datar Rajeev Motwani Jennifer Widom

Department of Computer Science
Stanford University
Stanford, CA 94305
{babcock,shivnath,datar,rajeev,widom}@cs.stanford.edu

Abstract

In this overview paper we motivate the need for and research issues arising from a new model of data processing. In this model, data does not take the form of persistent relations, but rather arrives in multiple, continuous, rapid, time-varying *data streams*. In addition to reviewing past work relevant to data stream systems and current projects in the area, the paper explores topics in stream query languages, new requirements and challenges in query processing, and algorithmic issues.

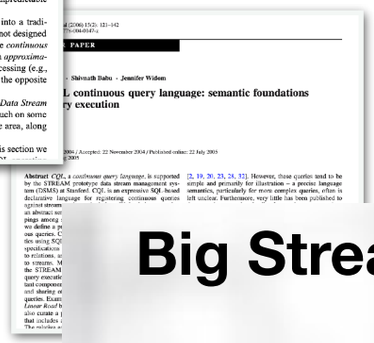
1 Introduction

Recently a new class of data-intensive applications has become widely recognized: applications in which the data is modeled best not as persistent relations but rather as transient *data streams*. Examples of such applications include financial applications, network monitoring, security, telecommunications data management, web applications, manufacturing, sensor networks, and others. In the data stream model, individual data items may be relational tuples, e.g., network measurements, call records, web page visits, sensor readings, and so on. However, their continuous arrival in multiple, rapid, time-varying, possibly unpredictable and unbounded streams appears to yield some fundamentally new research problems.

In all of the applications cited above, it is not feasible to simply load the arriving data into a traditional database management system (DBMS) and operate on it there. Traditional DBMS's are not designed for rapid and continuous loading of individual data items, and they do not directly support the *continuous queries* [84] that are typical of data stream applications. Furthermore, it is recognized that both *approximation* [13] and *adaptivity* [8] are key ingredients in executing queries and performing other processing (e.g., data analysis and mining) over rapid data streams, while traditional DBMS's focus largely on the opposite goal of precise answers computed by stable query plans.

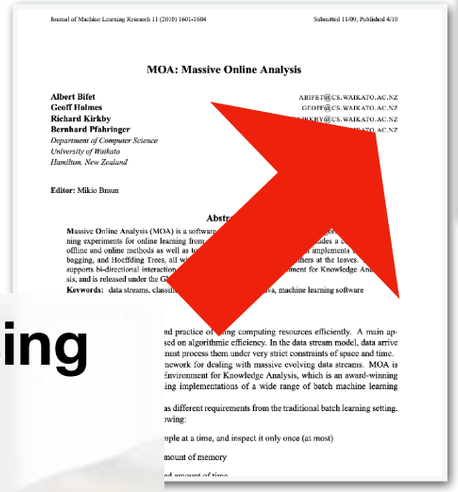
In this paper we consider fundamental models and issues in developing a general-purpose *Data Stream Management System (DSMS)*. We are developing such a system at Stanford [82], and we will touch on some of our own work in this paper. However, we also attempt to provide a general overview of the area, along with its related and current work. (Any glaring omissions are, naturally, our own fault.)

We begin in Section 2 by considering the data stream model and queries over streams. In this section we


PAPER

Shivnath Babu · Jennifer Widom

L continuous query language: semantic foundations
y execution


Journal of Machine Learning Research 11 (2010) 1601-1604 Submitted 11/09; Published 4/10

MOA: Massive Online Analysis

Albert Bifet ABIFET@CS.WAIKATO.AC.NZ
Geoff Holmes GEOFF@CS.WAIKATO.AC.NZ
Richard Kirkby RKIRKBY@CS.WAIKATO.AC.NZ
Bernhard Pfahringer BERNHARD@CS.WAIKATO.AC.NZ
Department of Computer Science
University of Waikato
Hamilton, New Zealand

Editor: Mikio Braun

Abstract

Massive Online Analysis (MOA) is a software
ning experiments for online learning from
offline and online methods as well as to
bagging, and Hoeffding Trees, all w
supports bi-directional interaction
sis, and is released under the G
Keywords: data streams, classifi
nd practice of
sed on algorithmic efficiency. In the data stream model, data arrive
nust process them under very strict constraints of space and time.
nework for dealing with massive evolving data streams. MOA is
nvironment for Knowledge Analysis, which is an award-winning
ng implementations of a wide range of batch machine learning
as different requirements from the traditional batch learning setting.
owing:
mple at a time, and inspect it only once (at most)
mount of memory


Dynamically Scaling
of

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

ABSTRACT

1. INTRODUCTION

Apache Samza


ables: Two Sides of th

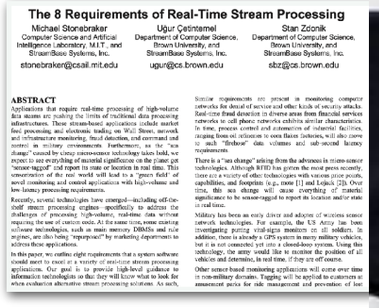

Apache Flink™: Stream and Batch Processing in a Single Engine

Abstract

1 Introduction


Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark


Big Stream Processing Starts


The 8 Requirements of Real-Time Stream Processing

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Stan Zdonik
Department of Computer Science, Brown University, and StreamBase Systems, Inc.
sbz@cs.brown.edu

ABSTRACT

2002 2005 2006 2007 2008 2010 2014 2015 2018

Timeline

Models and Issues in Data Stream Systems *

The CQL continuous query language: semantic foundations and query execution

The 8 Requirements of Real-Time Stream Processing

NiagaraCQ: A Scalable Continuous Query System for Internet Databases.

Aurora: A New Model and Architecture for Data Stream Management

TelegraphCQ: Continuous Dataflow Processing.

It's a Streaming World! Reasoning upon Rapidly Changing Information

MOA: Massive Online Analysis

Apache Samza

Apache Flink™: Stream and Batch Processing in a Single Engine

S. Murthy et al. Pulsar – Real-Time Analytics at Scale. Technical report, eBay, 2015.

Summingbird: A Framework for Integrating Batch and Online MapReduce Computations.

Trill: A High-Performance Incremental Query Processor for Diverse Analytics.

2002 2005 2006 2007 2008 2010 2014 2015 2018

Our Focus

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

a 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, Braz*
ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/32421
3242155

1 INTRODUCTION

Stream processing has emerged as a paradigm to develo
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].

Defining semantics for stream processing, however

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, clouce, 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 [25], 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 [30], to the more recent forays in low-latency processing such as Spark Streaming [34], MillWheel, and Storm [6], 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

---

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

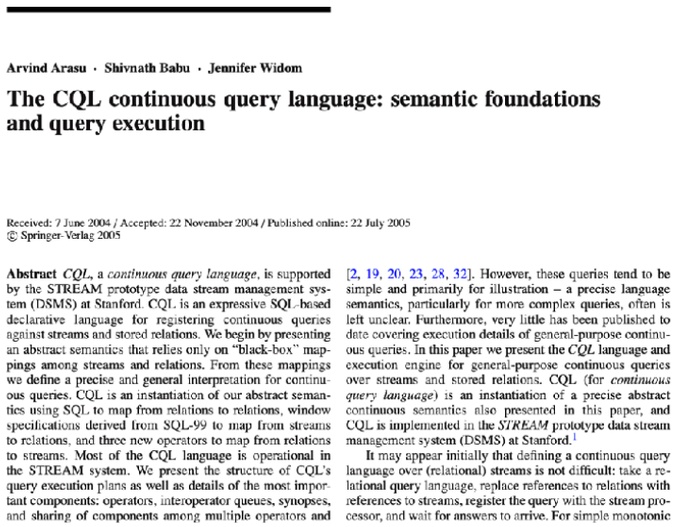
Abstract *CQL*, a *continuous query language*, is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics 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

Is this too ambitious?

https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjaouS-uK3mAhVEqaQKHfdOA4MQjRx6BAgBEAQ&url=https%3A%2F%2Fmedium.com%2Fpersonal-growth-lab%2Fshould-you-set-realistic-or-highly-ambitious-goals-7cf400505444&psig=AOvVaw3xPkWzwvy8SBXD9NS0o_Oe&ust=1576148163494865



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
© Springer-Verlag 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]

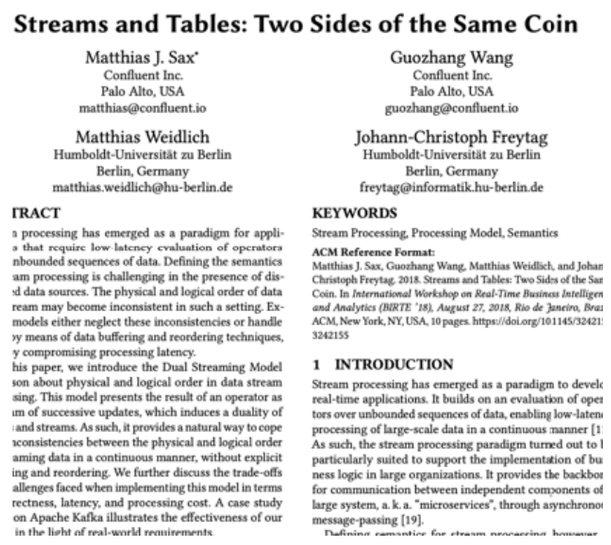
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

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

a processing has emerged as a paradigm for appli-
e 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.

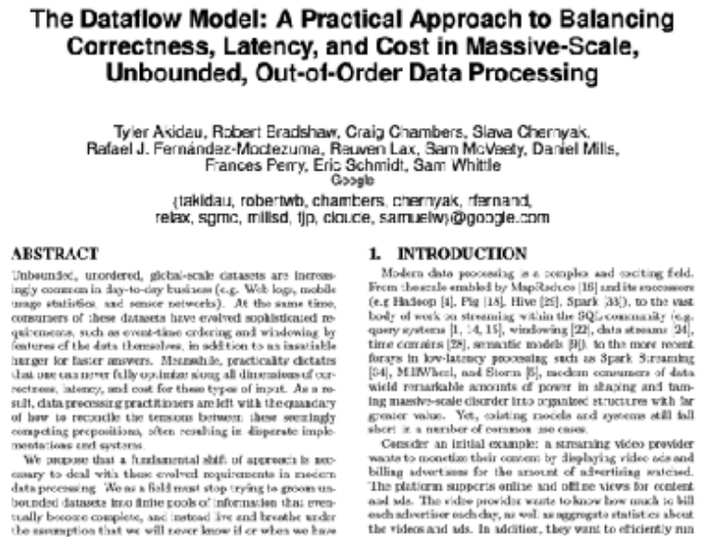
KEYWORDS

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 Sa
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 develo
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
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, clpuice, 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 tools enabled by MapReduce [16] and its successors (e.g. Hadoop [4], Pig [18], Hive [25], Spark [33]), to the vast body of work in streaming within the SQL community (e.g. query systems [1, 14, 15], windowing [22], data streams [24], time semantics [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.

Goals

CQL

- Exploit lesson learnt in RDBMS theory.
- simple, clear, yet powerful language.
- efficiently combine streams and relations.

DSM

- low-latency is paramount.
- take data distribution and ordering into account.
- avoid implicit operator semantics.

DFM

- Correctness first with sessions support.
- Latency requirements vs resource cost should dictate system choice.
- Single processing model for bounded and unbounded data.

Time

CQL

- DSMS Clock Time, i.e., the timestamp assigned by the DSMS.
- Source Time, i.e., the timestamp assigned by the source.
- Source Heartbeat are used to declare no element with lower timestamp will be sent.

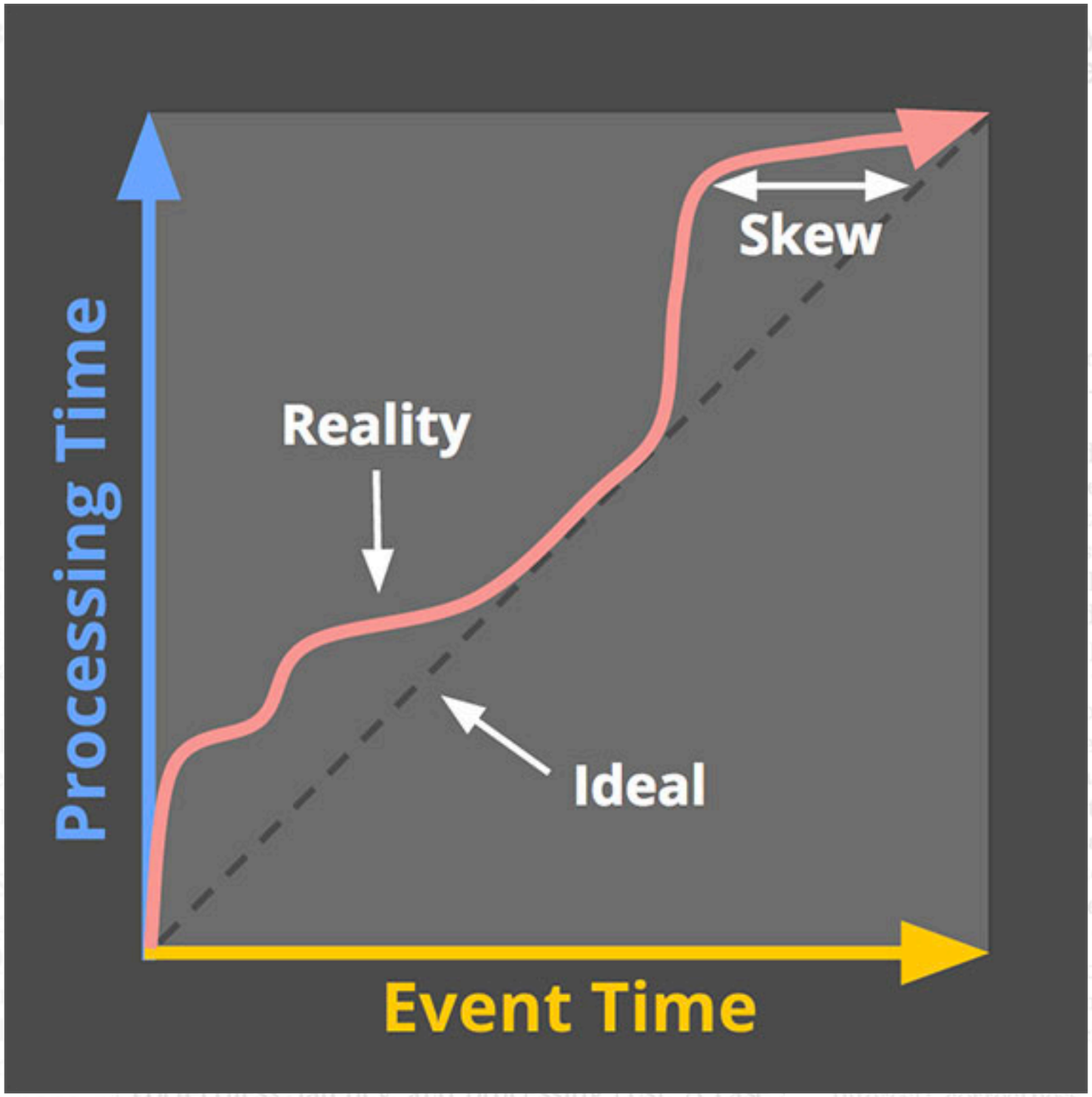
DSM

- Logical Order induced by the timestamp assigned by the data source.
- Physical Order induced by the timestamp assigned by the system at ingestion.

DFM

- Event Time is the time at which the event itself actually *occurred.*
- Processing Time is the time at which an event is *observed* during *processing.*
- Watermark, i.e., global **progress metrics** that tracks the skewness between ET and PT.

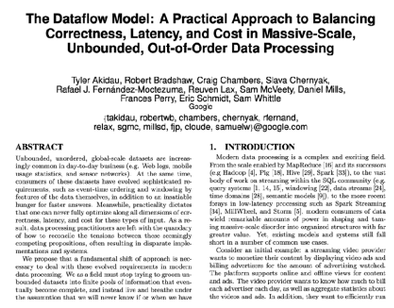
Watermark



Abstractions

CQL	DSM	DFM
• Streams • (Time-Varying) Relations*	• Change-log Streams • Records Streams • Tables	• PCollections (bounded and unbounded)

CQL in 3 Slides



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 [28], 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 [20], semantic models [9]), to the more recent forays in low-latency processing such as Spark Streaming [34], MillWheel, and Storm [3], 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.

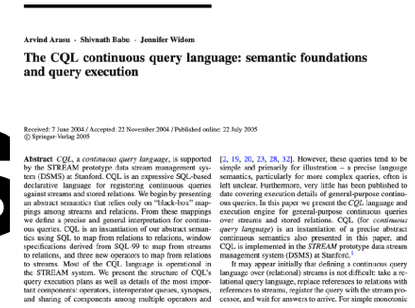
Consider an initial example: a streaming video provider wants to monetize their content by displaying video ads and billing advertisers for the amount of advertising watched. The platform supports online and offline views for content and ads. The video provider wants to know how much to bill each advertiser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

A Stream ***S*** *is a possibly infinite* ***multi-set*** *of elements <s,t> where s is a tuple belonging to the schema of S and t is a timestamp.*

Relation ***R*** is a set of tuples $(d_1, d_2, ..., d_n)$, where each element d_j is a member of D_j, a data domain[1].

[1] *a Data Domain refers to all the values which a data element may contain.*

CQL in ~~3~~ 4 Slides



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]

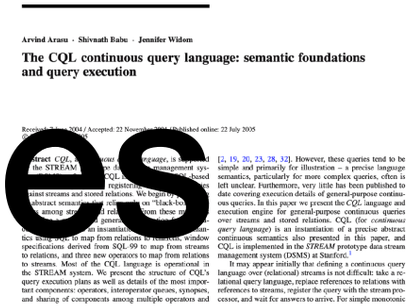
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

A Stream ***S*** *is a possibly infinite* ***multi-set*** *of elements <s,t> where s is a tuple belonging to the schema of S and t is a timestamp.*

~~*Relation* ***R*** is a set of tuples $(d_1, d_2, ..., d_n)$, where each element d_j is a member of D_j, a data domain[1].~~

[1] *a Data Domain refers to all the values which a data element may contain.*

Ok, CQL in ~~4~~ 5 Slides



Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

Received: [illegible] 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

[illegible] language, is [illegible] the STREAM [illegible] management sys- [illegible] CQL [illegible] based [illegible] registering [illegible] queries [illegible] against streams and stored relations. We begin by [illegible] abstract semantics [illegible] relations as "black-box" [illegible] among streams and relations [illegible] from these primitives. [illegible] generality [illegible] an instantiation [illegible] 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]

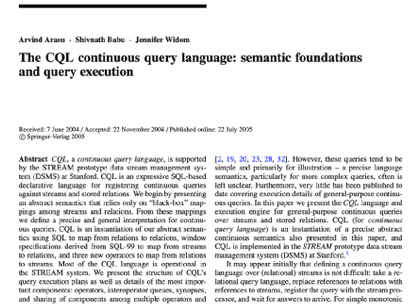
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

A Stream ***S*** *is a possibly infinite* ***multi-set*** *of elements <s,t> where s is a tuple belonging to the schema of S and t is a timestamp.*

Relation ***R*** is a mapping from each time instant in T to a **finite** but **unbounded bag** of tuples belonging to the **schema** of **R**.

[1] *a Data Domain refers to all the values which a data element may contain.*

CQL in 5 Slides



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]

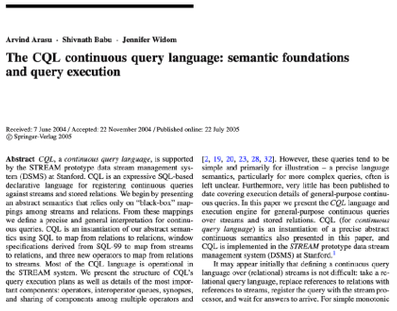
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

A Stream ***S*** *is a possibly infinite* ***multi-set*** *of elements <s,t> where s is a tuple belonging to the schema of S and t is a timestamp.*

Relation ***R*** is a mapping from each time instant in T to a **finite** but **unbounded bag** of tuples belonging to the **schema** of **R**.

[1] *a Data Domain refers to all the values which a data element may contain.*

CQL in 5 Slides



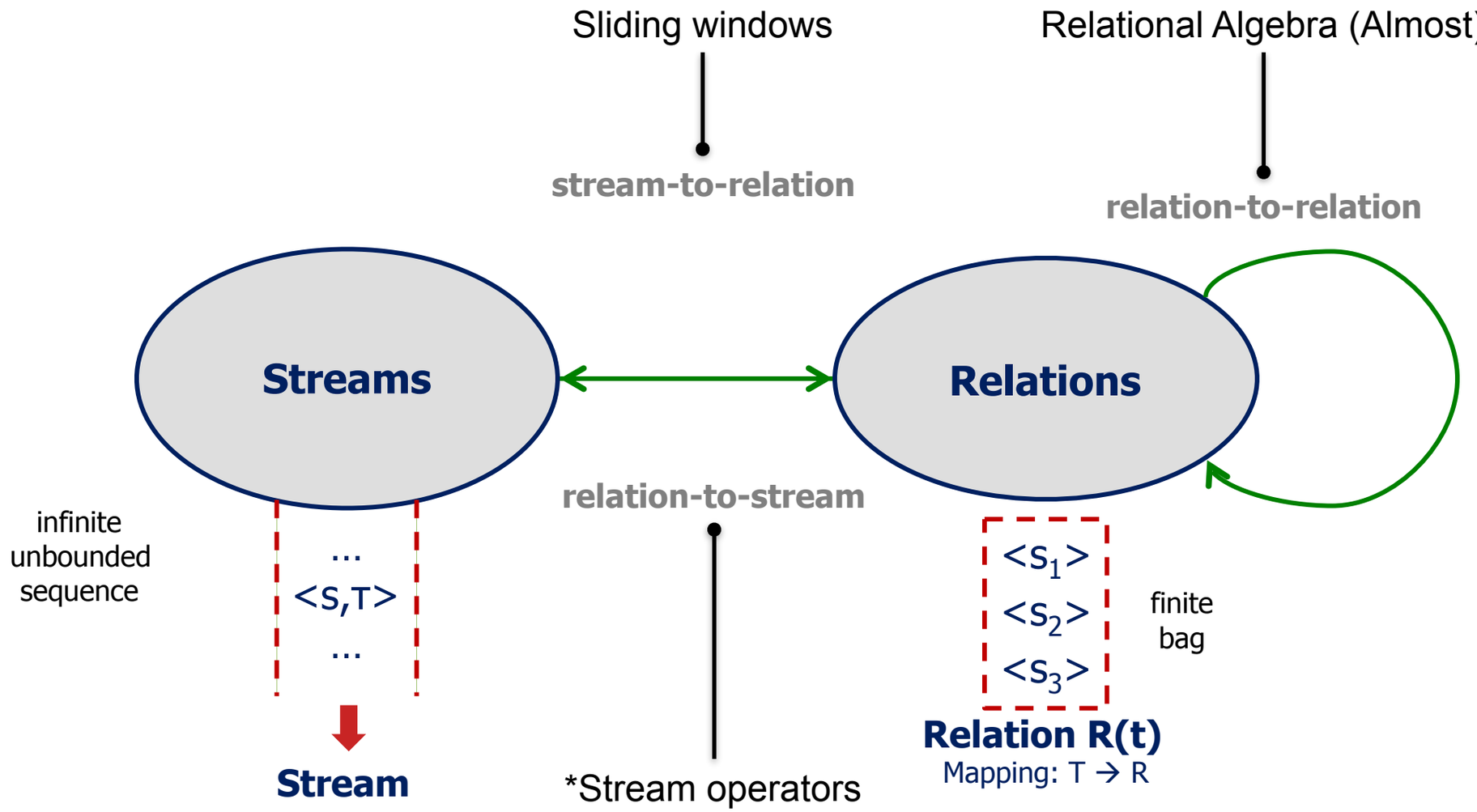
Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

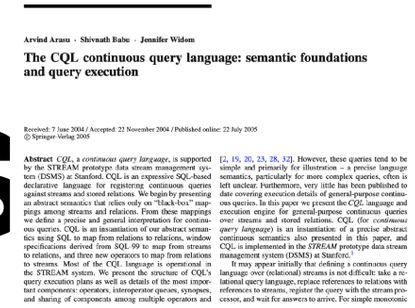
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005
© Springer-Verlag 2005

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



CQL in ~~5~~ 6 Slides



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

Stream-to-Relation Operators:

- **Sliding Window**:
  FROM S [RANGE 5 Minutes]

- **Parametric Sliding Windows:**
  FROM S [RANGE 5 Minutes Slide 1 Min]

- **Partitioned Windows**:
  FROM S [PARTITIONED BY $A_1..A_n$ ROW m]

[1] *a Data Domain refers to all the values which a data element may contain.*

CQL in 6 Slides

Arvind Arasu · Shivnath Babu · Jennifer Widom

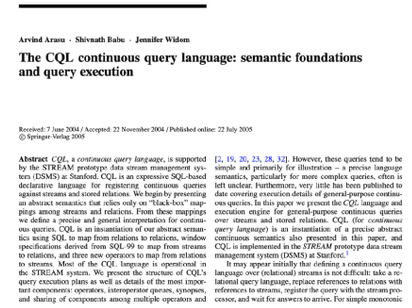
The CQL continuous query language: semantic foundations and query execution

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

CQL in 6 Slides



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

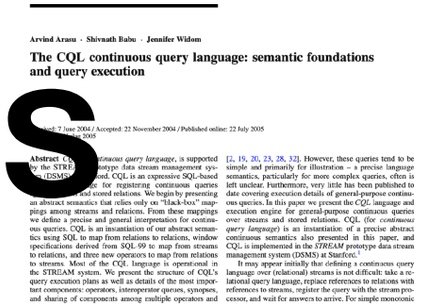
Relation-to-Stream Operators:

- **Rstream**: streams out all data in the last step
- **Istream**: streams out data in the last step that wasn’t on the previous step, i.e. streams out what is new
- **Dstream**: streams out data in the previous step that isn’t in the last step, i.e. streams out what is old

[1] a Data Domain refers to all the values which a data element may contain.

DFM in X<42* Slides

Reasoning about time



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
© 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 over 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

- **What** results are being computed
- **Where** in event time are being computed
- **When** in processing time are being materialised
- **How** “earlier results” relate to “later refinements”

*overestimation

DFM in X Slides

(WHAT) Processing Model

A Stream is represented as a possibly unbounded collection of key-value pairs, i.e., a PCollection<K,V>.

PTransforms are PCollections-to-PCollections operations.

Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

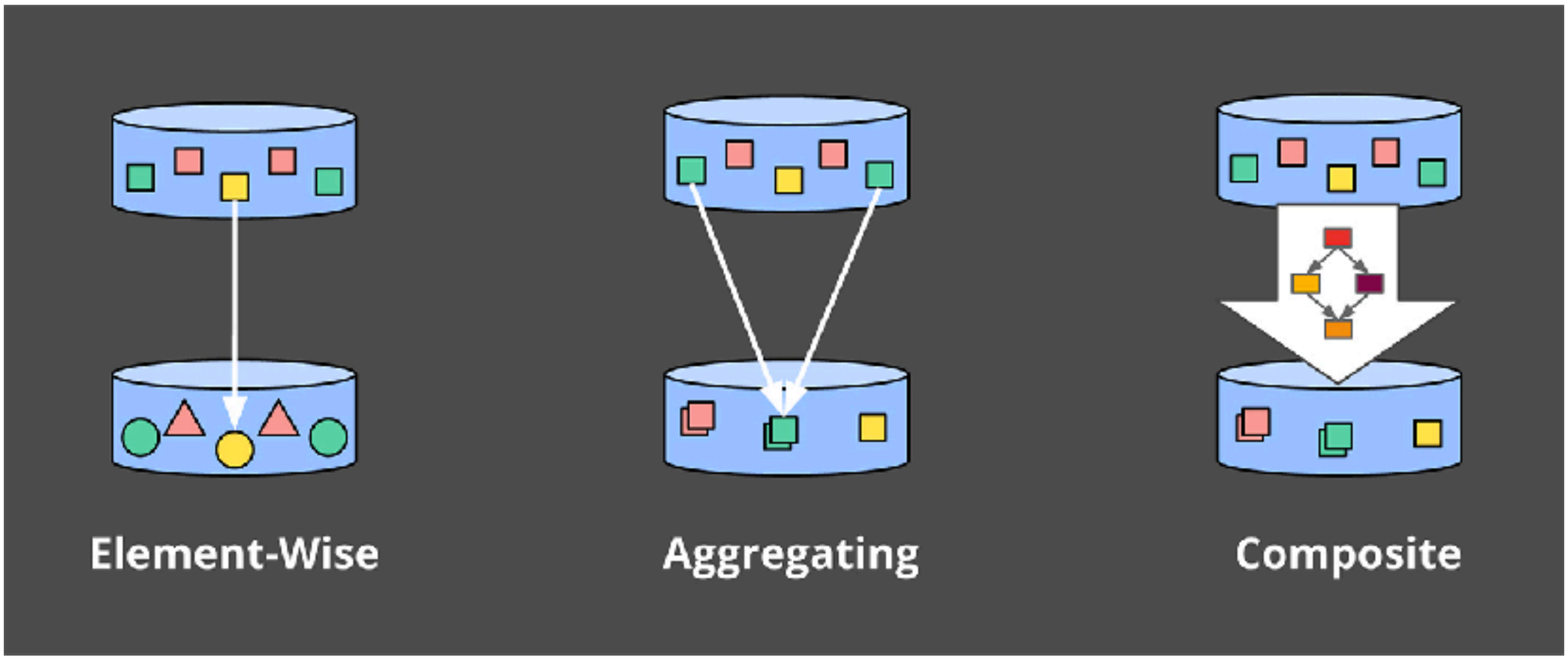
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

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

DFM in X Slides

(WHAT) Processing Model



Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

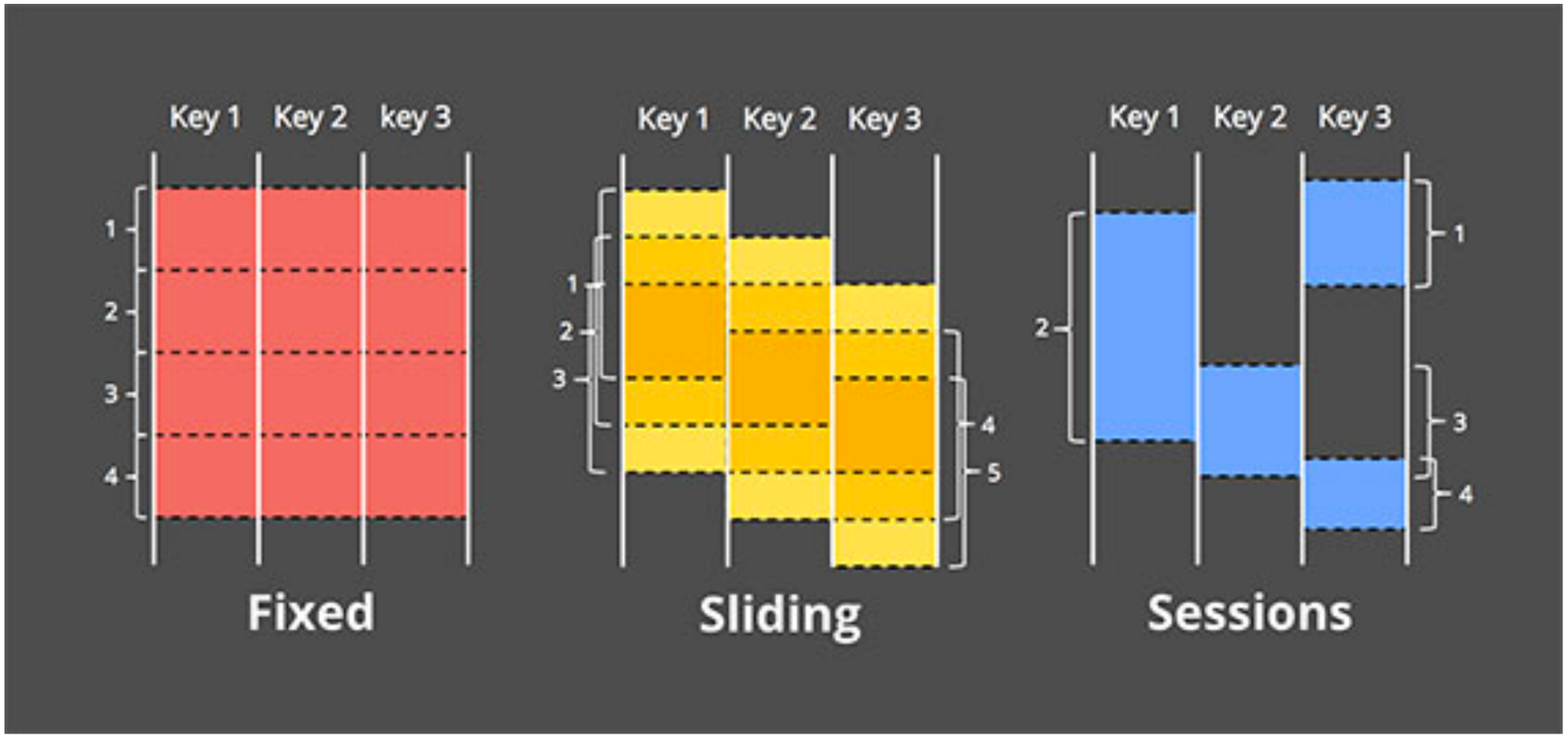
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

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

DFM in X Slides

(WHERE) Windowing Model



Arvind Arasu · Shivnath Babu · Jennifer Widom

The CQL continuous query language: semantic foundations and query execution

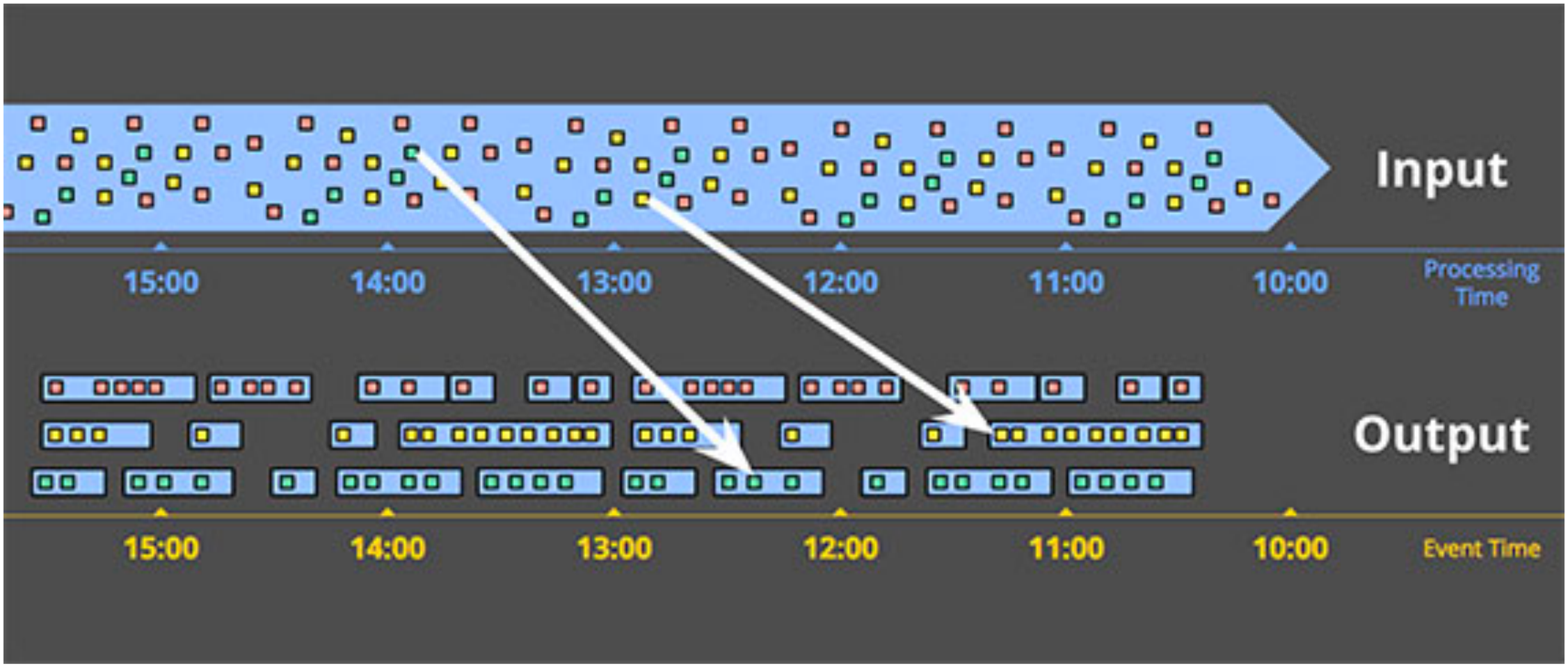
Received: 7 June 2004 / Accepted: 22 November 2004 / Published online: 22 July 2005

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

DFM in X Slides

(WHERE) Windowing Model



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]

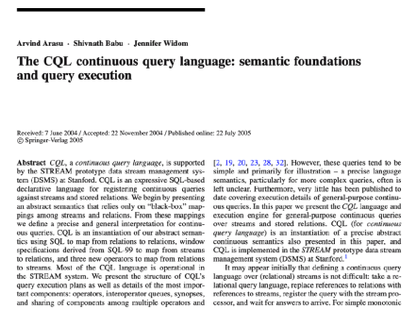
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

Wall of Text

Disclaimer

DFM in X Slides

(WHERE) Windowing Model



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
© Springer-Verlag 2005

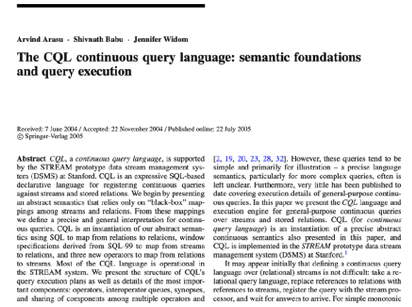
Abstract *CQL, a continuous query language,* is supported by the STREAM prototype data stream management system (DSMS) at Stanford. CQL is an expressive SQL-based declarative language for registering continuous queries against streams and stored relations. We begin by presenting an abstract semantics 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

- The DFM windowing model requires extended primitives than CQL's one.
- Window Assignment, i.e., each element is assigned to a corresponding window.
- Window Merge:
  - **Drop Timestamp: only the window interval is relevant here on**
  - GroupBy Key: to enable parallel execution
  - Window Merge (the merge logic depends on the window strategy)
  - GroupByWindow: to ensure elements are processed in sequence window-wise.
- ExpandToElements: assigned a valid timestamp to the elements.

DFM in X Slides

(WHEN) Triggering Model



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
© Springer-Verlag 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]

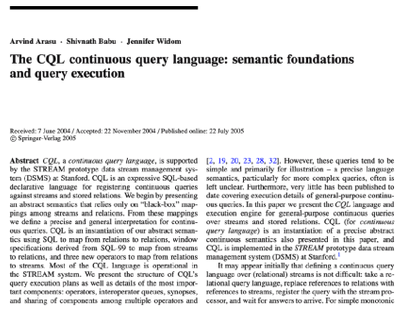
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

- In order to build unaligned (apply across subset of the data) event-time windows DFM is forced to decouple the reporting of the window content.

- To do so, DFM introduces an orthogonal model that allows to signal when a window is ready to be processed.

DFM in X Slides

(WHEN) Triggering Model



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
© Springer-Verlag 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

- In order to build unaligned (apply across subset of the data) event-time windows DFM is forced to decouple the reporting of the window content.

- To do so, DFM introduces an orthogonal model that allows to signal when a window is ready to be processed.

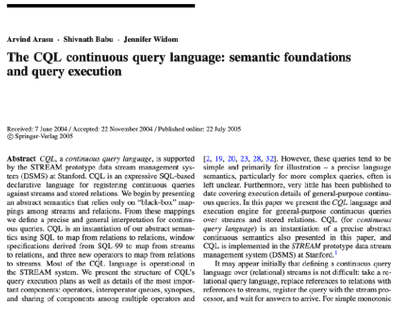
- Triggers solve this issue allowing DFM users to write an arbitrary logic to signal the completion of a window.

Wall of Text

Disclaimer

DFM in X Slides

(HOW) Triggering Model (part 2)



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
© Springer-Verlag 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]

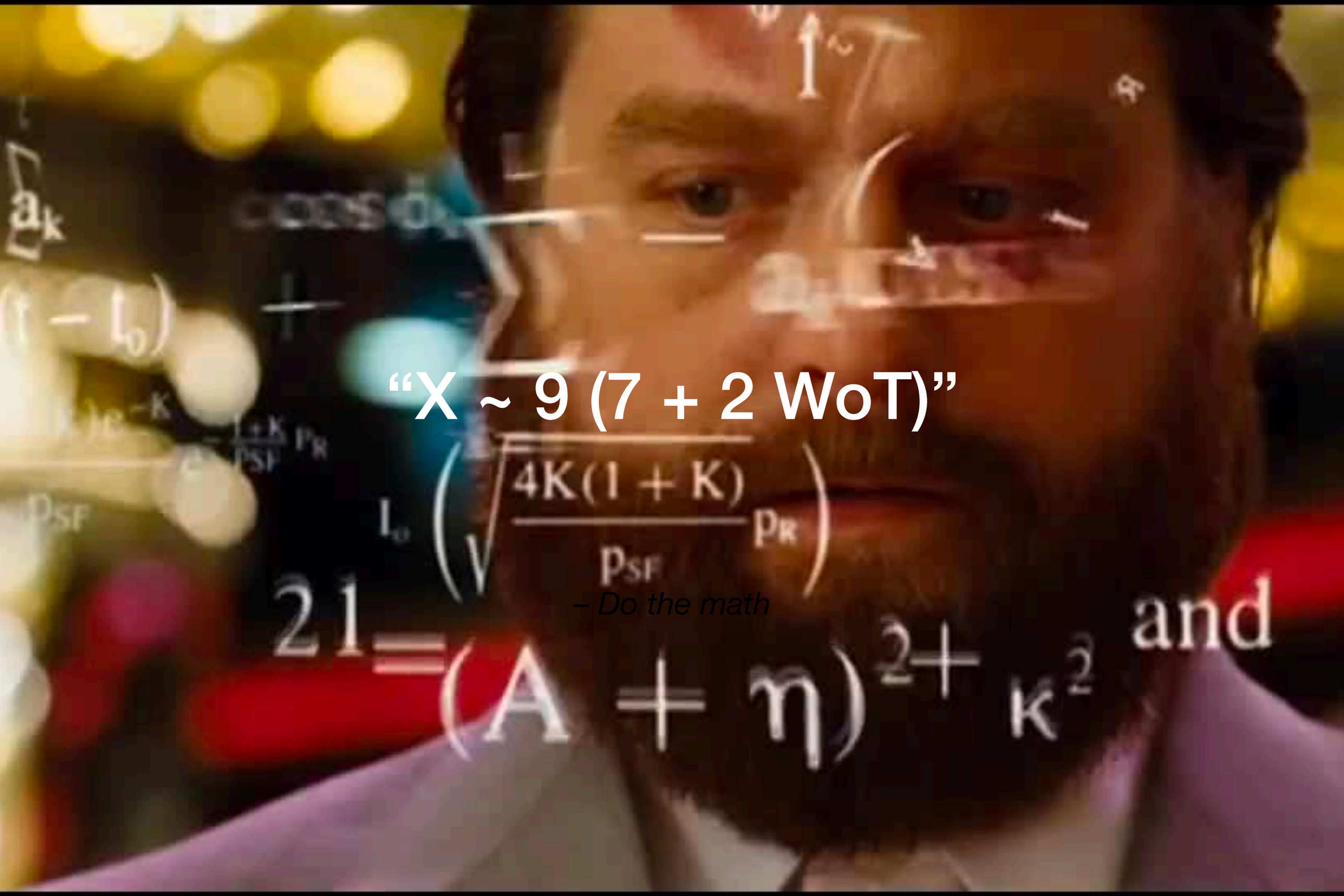
It may appear initially that defining a continuous query language over (relational) streams is not difficult: take a relational query language, replace references to relations with references to streams, register the query with the stream processor, and wait for answers to arrive. For simple monotonic

In addition to the triggering semantics, DFM introduces different refinements models to deal with late arrivals

- Discarding, i.e., upon triggering, window contents are discarded and later results bear no relation to previous results.

- Accumulating, i.e., upon triggering, window contents are left intact in a persistent state and later results will become a refinement to previous results

- Accumulating & Retracting, i.e, it extends the accumulating semantics with retraction of the previous value.


"X ~ 9 (7 + 2 WoT)"
– Do the math
$$21 = \left(\sqrt{\frac{4K(1+K)}{P_{SF}} P_R}\right) \text{ and } (A+\eta)^2 + \kappa^2$$

Was this too ambitious?

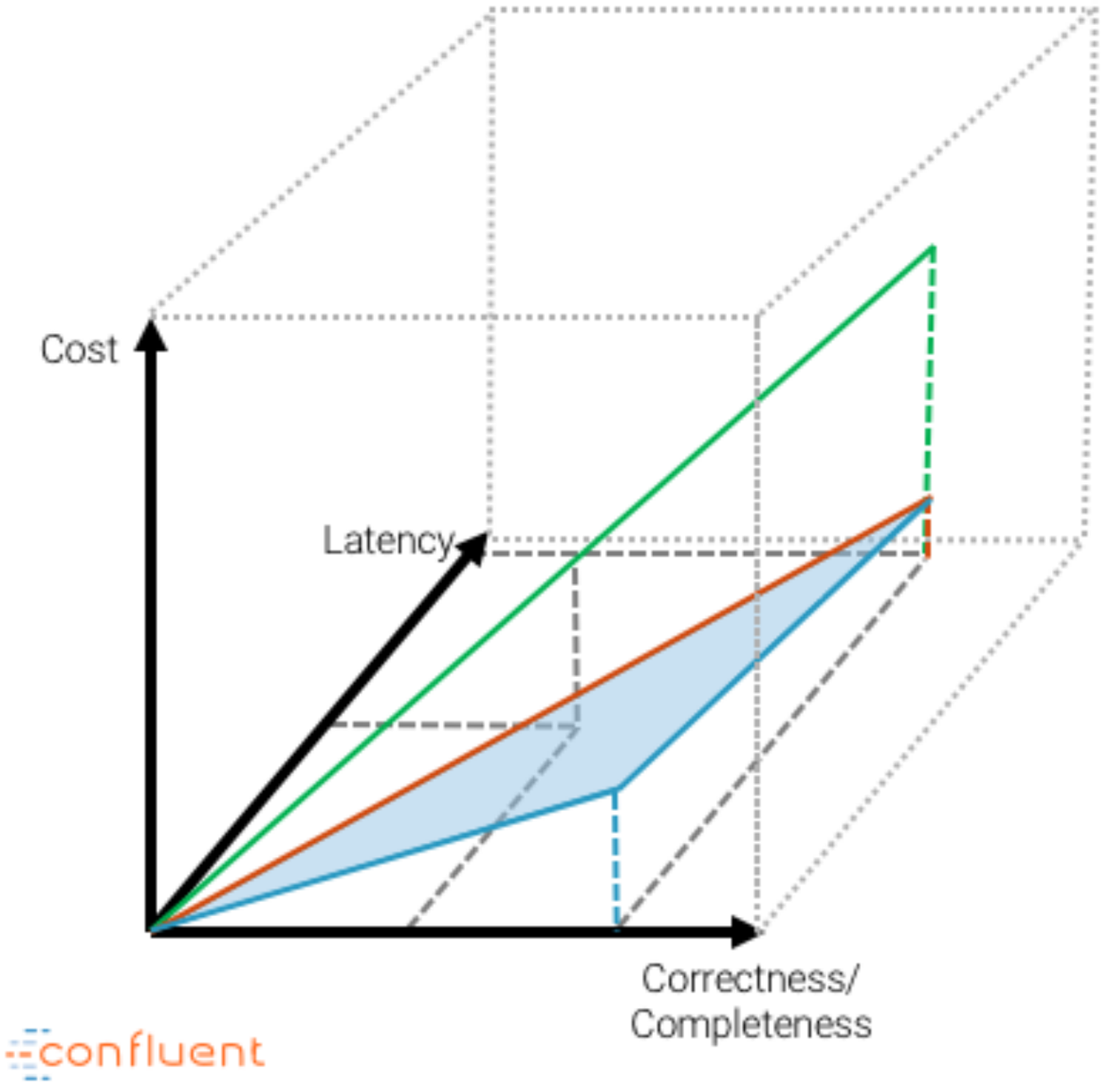
https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwjaouS-uK3mAhVEqaQKHfdOA4MQjRx6BAgBEAQ&url=https%3A%2F%2Fmedium.com%2Fpersonal-growth-lab%2Fshould-you-set-realistic-or-highly-ambitious-goals-7cf400505444&psig=AOvVaw3xPkWzwvy8SBXD9NS0o_Oe&ust=1576148163494865

ZZZZ


I'M NOT DONE

Dual Stream Model

The design space



Buffering and Reordering
- Ref: CQL[1], Trill[2]

Punctuations/Watermarks
- Ref: Li et al.[3], Krishnamurthy et al.[4]

Dual Streaming Model
- Continuous updates/changelogs
- Decouple latency from correctness
- Trade-off latency and cost
- Trade-off cost and completeness (retention time)

confluent

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
ma ias.weidlich@hu-berlin.de

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

TRACT
as eme radigm appli-
that n low- cy eva n of ators
nbounde uenc data. Defi the antics
eam proce is d ese of dis-
d data so . Thi sical and logical or f data
eam m ome sistent in a s g. Ex-
neglect encies 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

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 Sa
Coin. In *International Workshop on Real-Time Business Intelliger
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

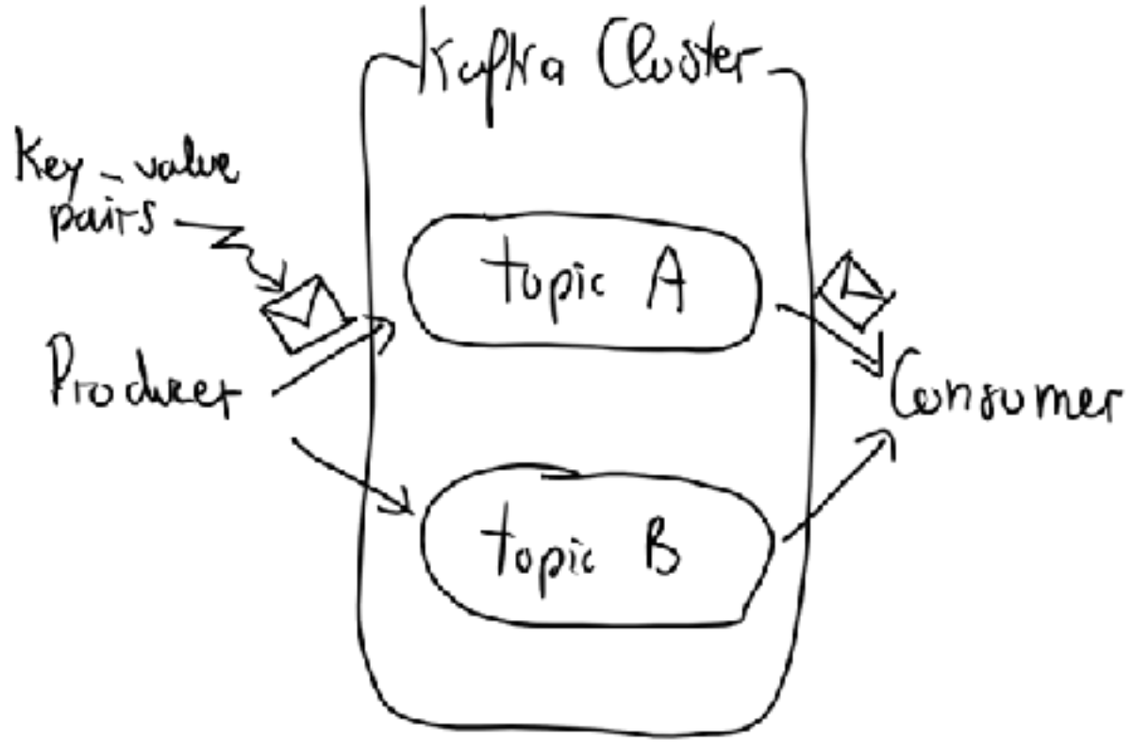
1 INTRODUCTION

Stream processing has emerged as a paradigm to devel
real-time applications. It builds on an evaluation of ope
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
particularly suited to support the implementation of bu
ness logic in large organizations. It provides the backbo
for communication between independent components o
asynchrono

Should I Take a Step back?

A Conceptual View of Kafka

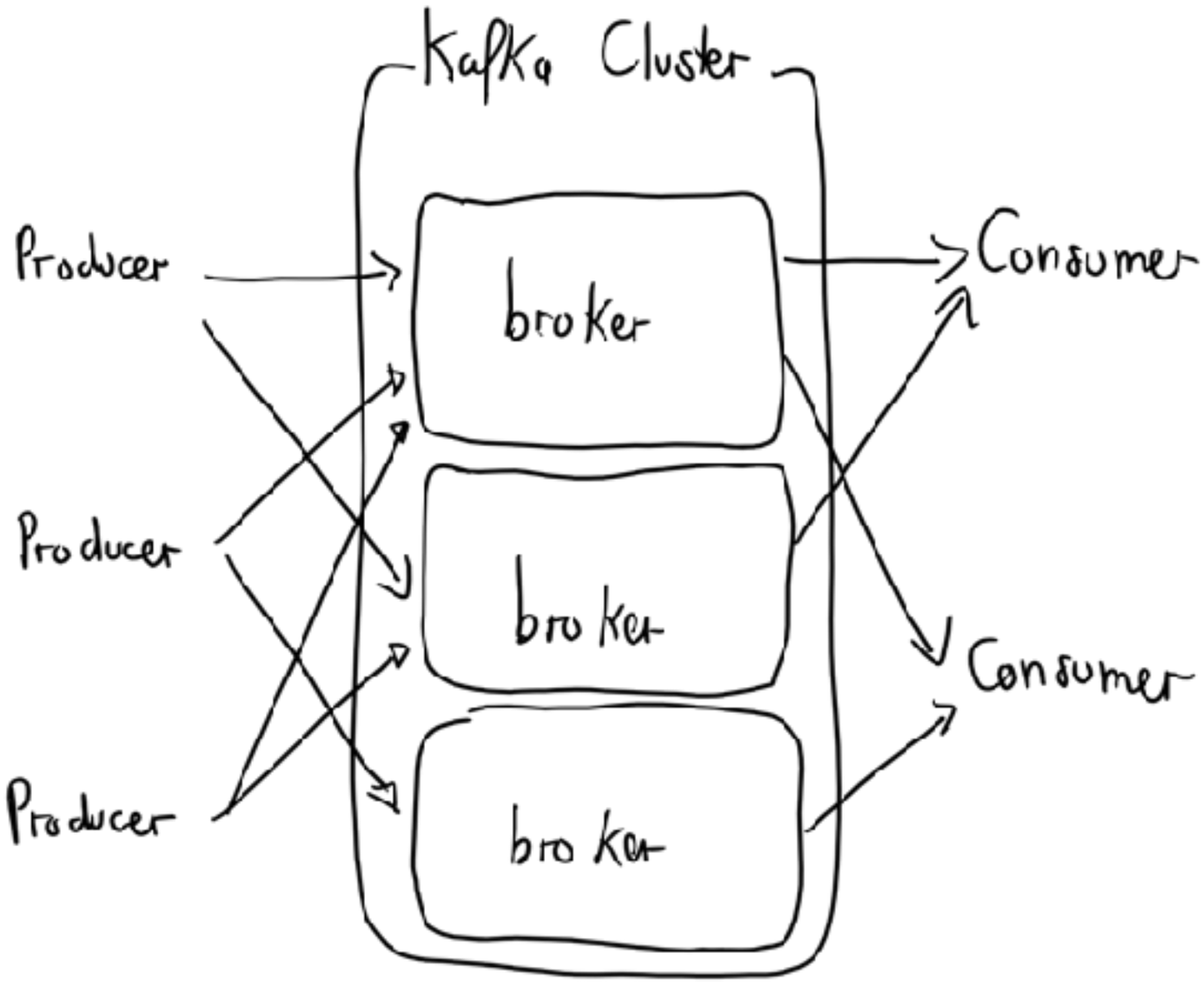
- **Producers** send messages on topics
- **Consumers** read messages from topics
- **Messages** are key-value pairs
- **Topics** are streams of messages
- Kafka cluster manages topics



curtesy of Emanuele Della Valle - http://emanueledellavalle.org

A Logical View of Kafka

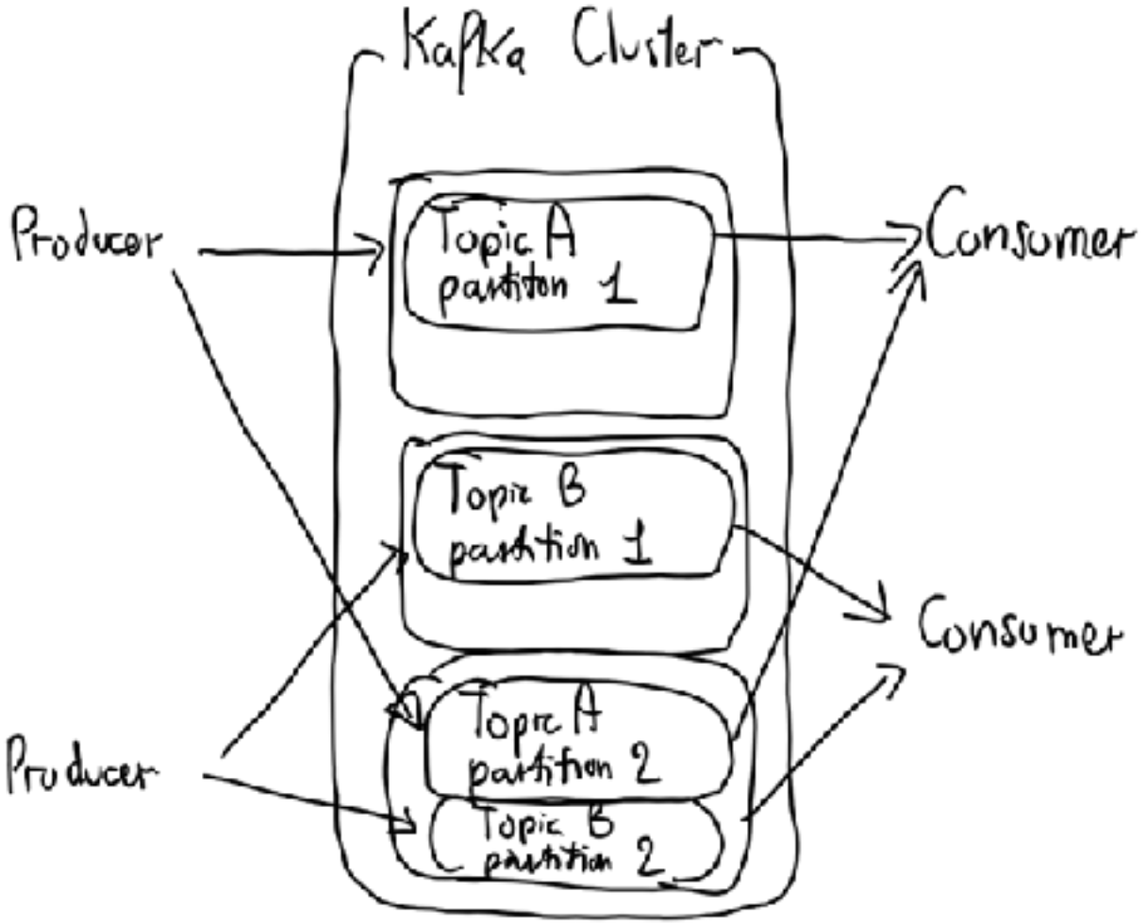
- **Brokers** are the main storage and messaging components of the Kafka cluster



curtesy of Emanuele Della Valle - http://emanueledellavalle.org

Reconciling the two views of Kafka

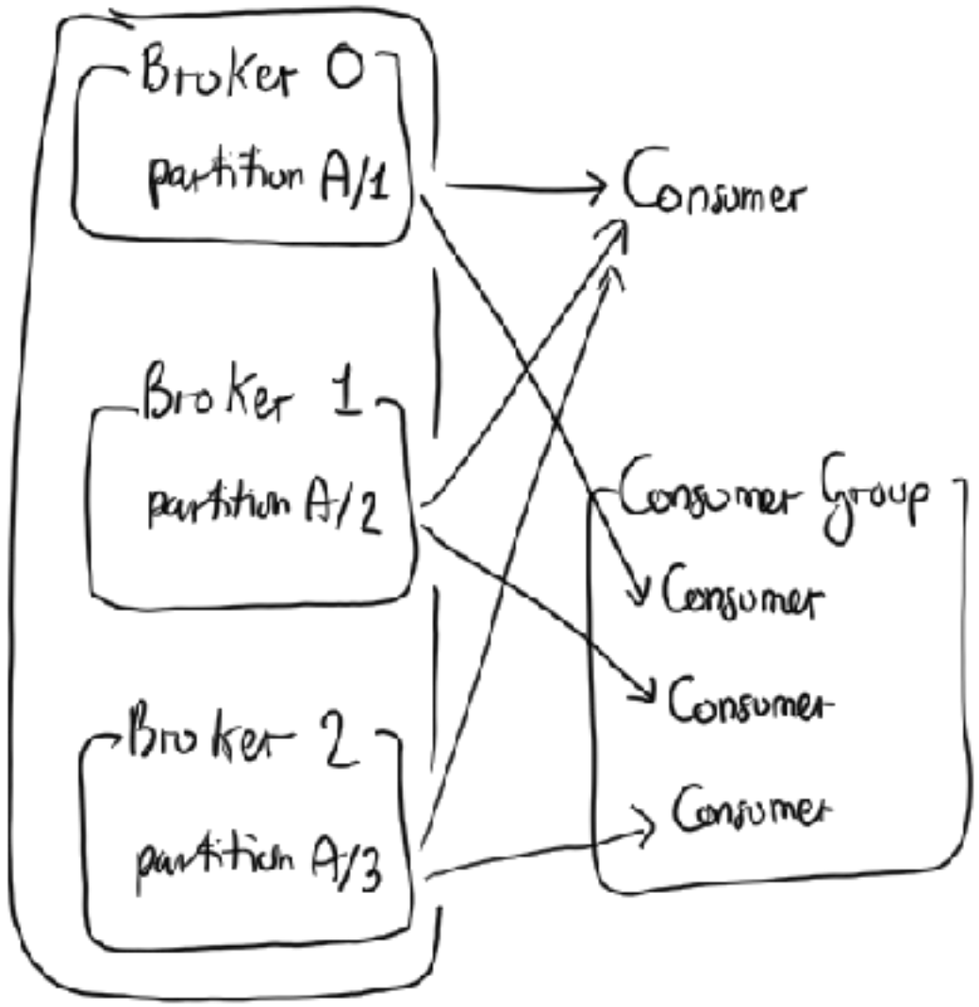
- Topics are partitioned across brokers
- Producers shard messages over the partitions of a certain topic
- Typically, the message key determines which Partition a message is assigned to



curtesy of Emanuele Della Valle - http://emanueledellavalle.org

Topic partitioning invites distributed consumption

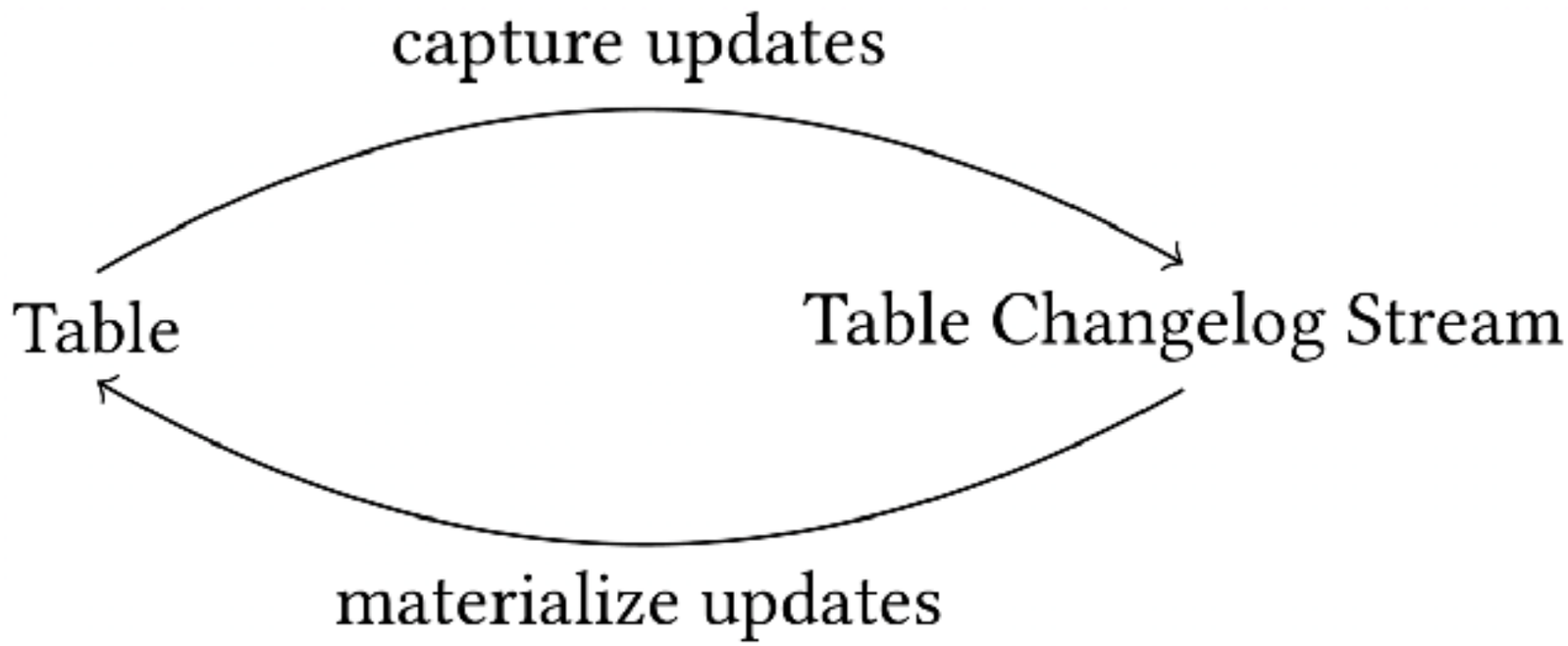
- Different Consumers can read data from the same Topic
  - By default, each Consumer will receive all the messages in the Topic
- Multiple Consumers can be combined into a Consumer Group
  - Consumer Groups provide scaling capabilities
  - Each Consumer is assigned a subset of Partitions for consumption



curtesy of Emanuele Della Valle - http://emanueledellavalle.org

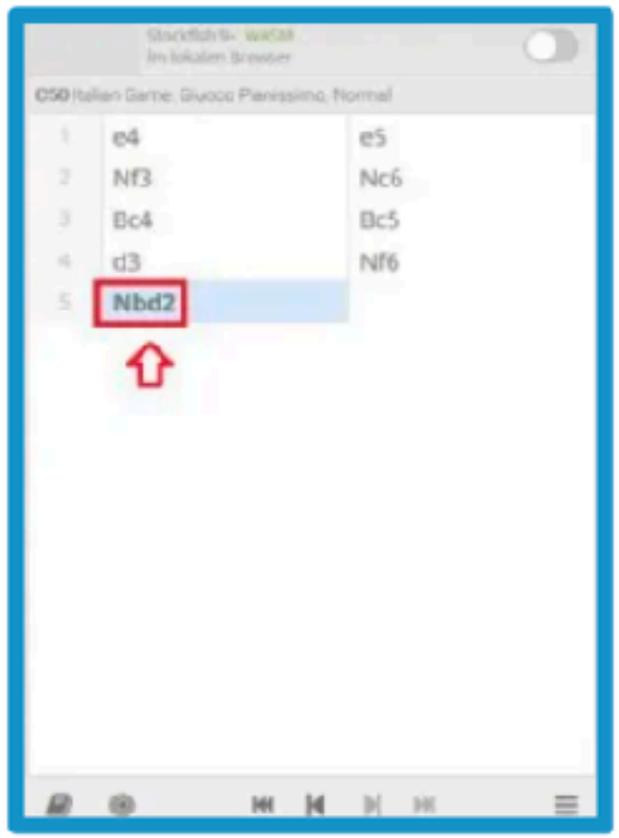
Dual Stream Model

The intuition



Dual Stream Model

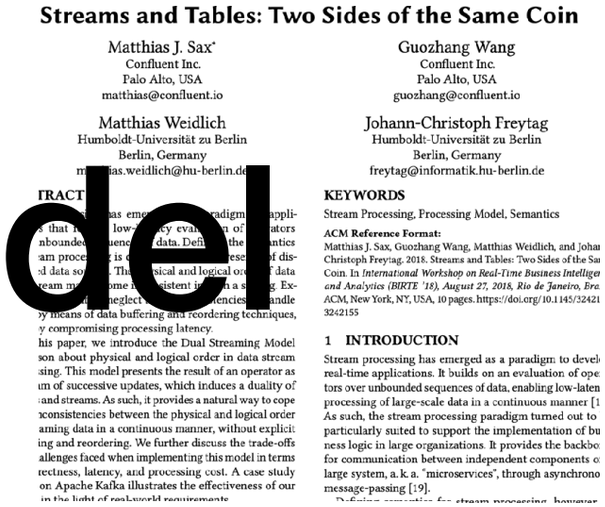
The intuition



Streams record History
"The sequence of moves."

Tables represent State
"The state of the board at last move."

https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/



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

has emerged as a paradigm for appli- that require low-latency evaluation of operators unbounded sequences of data. Defining the semantics eam processing is difficult because of dis- d data sources. The physical and logical order of data eam may become inconsistent in such a setting. Ex- mechanisms neglect inconsistencies or handle by 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 Sa Coin. In *International Workshop on Real-Time Business Intelliger 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

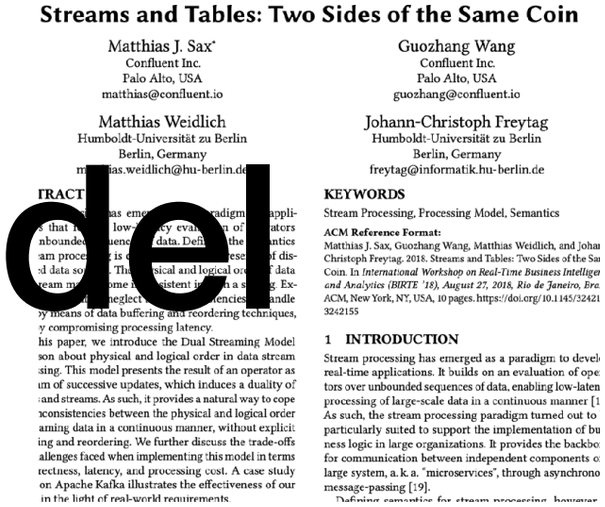
Stream processing has emerged as a paradigm to devel real-time applications. It builds on an evaluation of ope tors over unbounded sequences of data, enabling low-laten processing of large-scale data in a continuous manner [1 As such, the stream processing paradigm turned out to particularly suited to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

Dual Stream Model

The intuition

Table

https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/



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

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 Sa
Coin. In *International Workshop on Real-Time Business Intelliger
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 develo
real-time applications. It builds on an evaluation of ope
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
particularly suited to support the implementation of bu
ness logic in large organizations. It provides the backbo
for communication between independent components o
large system, a. k. a. "microservices", through asynchrono
message-passing [19].

Dual Stream Model

The Truth about Streams

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
has emer radigm appli-
s that low- y evalu n of ators
nbounded uenc data. Defi the antics
eam proce is d ese of dis-
d data so . Thi sical and logical or f data
ream m ome nsistent in h a s g. Ex-
neglect encies 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 Sa
Coin. In *International Workshop on Real-Time Business Intelliger
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 develo
real-time applications. It builds on an evaluation of ope
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
particularly suited to support the implementation of bu
ness logic in large organizations. It provides the backbo
for communication between independent components o
large system, a. k. a. "microservices", through asynchrono
message-passing [19].

Unbounded and ordered sequence of key-value pairs

Stream

Change-log Stream

A streams whose records are updates to a table

records are identified by a primary key

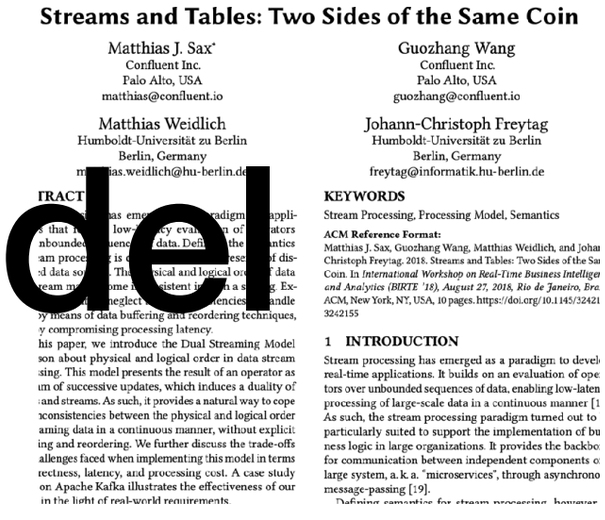
Record Stream

A streams whose records are facts

records are not identified by a primary key

Dual Stream Model

The Truth about Tables


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

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

KEYWORDS

Stream Processing, Processing Model, Semantics

ACM Reference Format:

1 INTRODUCTION

A table is a collection of table versions; one version for each point in time using the timestamp as a version number.

T(1)={T5 ={⟨A,7.2⟩}}

T(2) = {T5 = {⟨A,7.2⟩},T6 = {⟨B,14.7⟩}}

T(3) = {T5 = {⟨A,7.2⟩}, T6 = {⟨A,8.9⟩,⟨B,14.7⟩}}

T(4) = {T3 = {⟨B,12.1⟩},T5 = {⟨A,7.2⟩,⟨B,12.1⟩}, T6 = {⟨A, 8.9⟩, ⟨B, 14.7⟩}}

T(5) = {T3 = {⟨B,12.1⟩},T5 = {⟨A,7.2⟩,⟨B,12.1⟩}, T6 = {⟨A, 8.9⟩, ⟨B, 14.7⟩},T8 = {⟨A, 8.9⟩, ⟨B, 16.7⟩}}

Dual Stream Mod

The Truth about Operations

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

t 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
eam 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

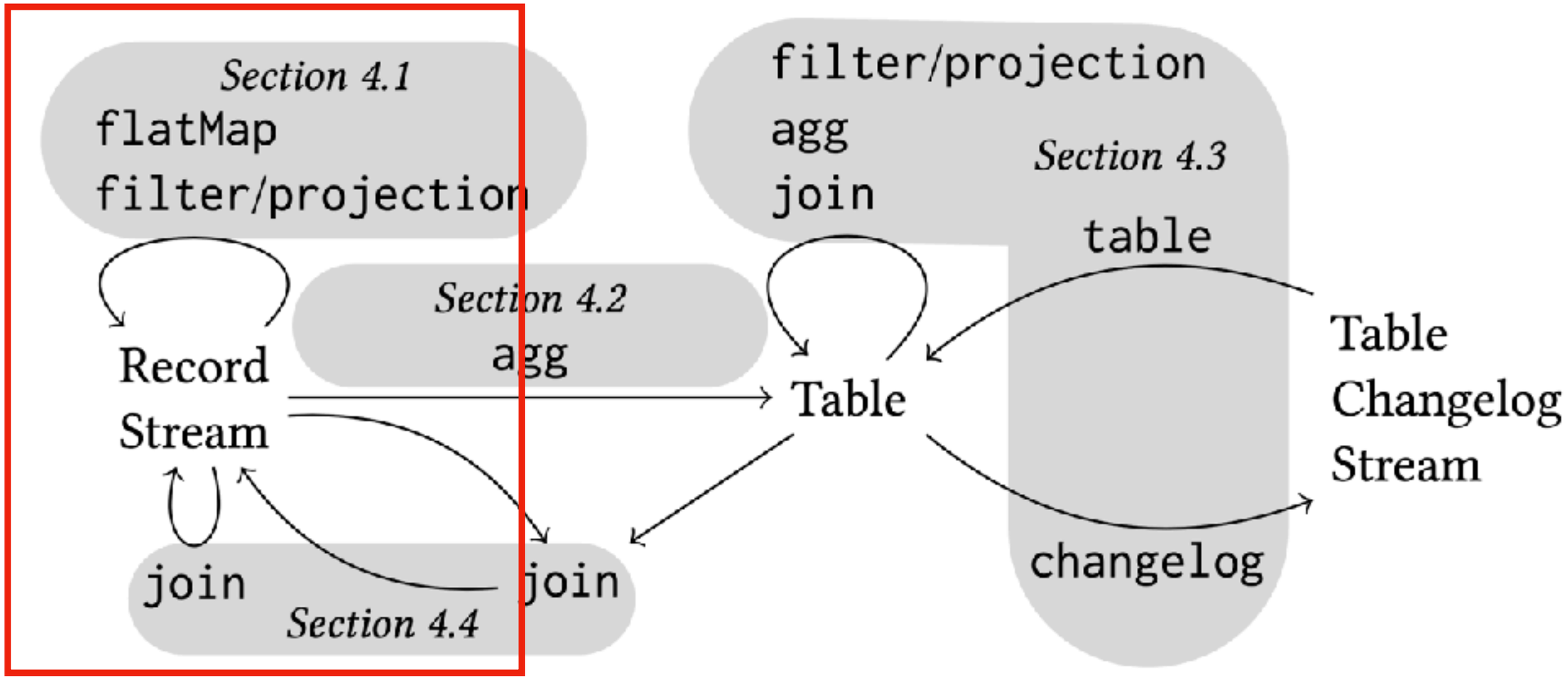
KEYWORDS

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 Sa
Coin. In *International Workshop on Real-Time Business Intelliger
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

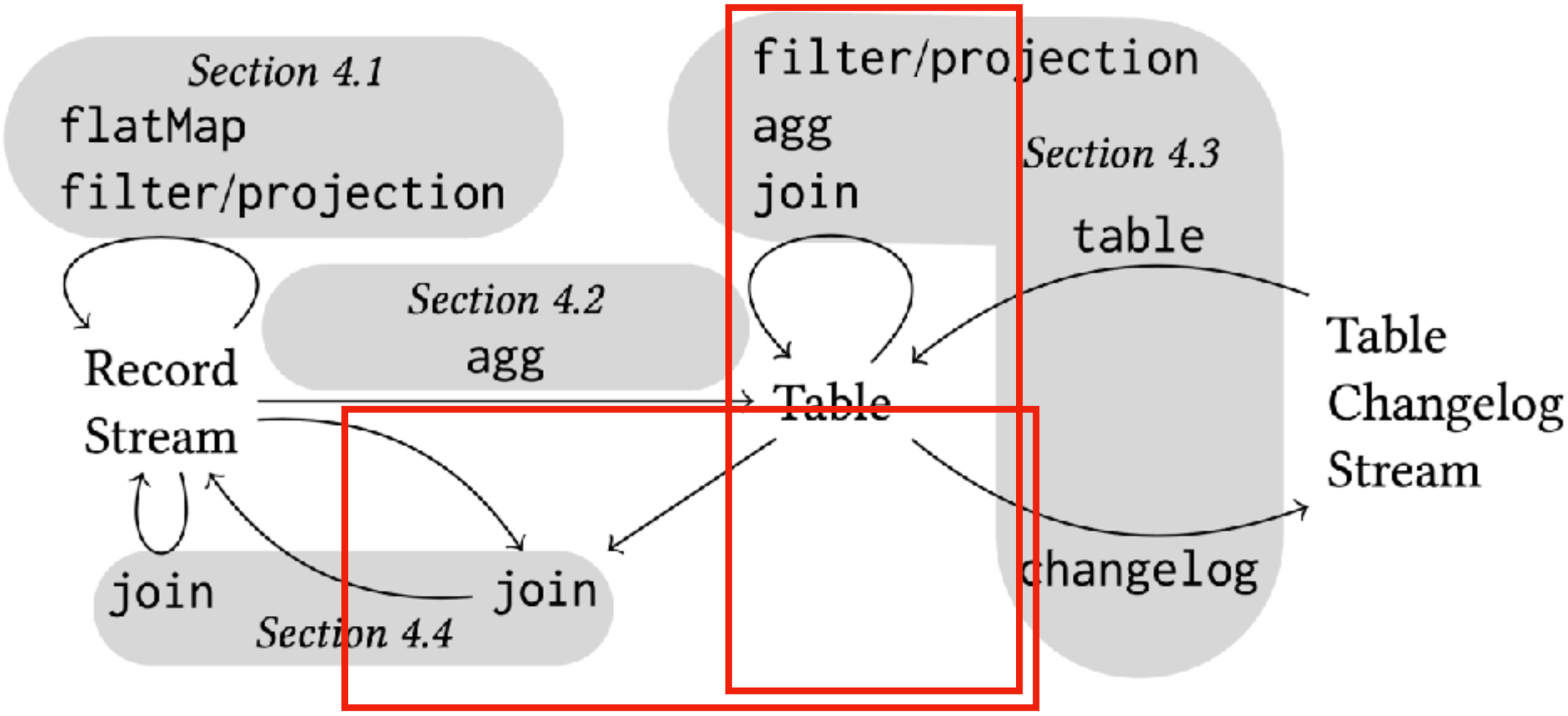
Stream processing has emerged as a paradigm to devel
real-time applications. It builds on an evaluation of ope
tors over unbounded sequences of data, enabling low-laten
processing of large-scale data in a continuous manner [1



Stateless Operations

Dual Stream Mod

The Truth about Operations



Stateful Operations

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

t 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
eam 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

KEYWORDS

Stream Processing, Processing Model, Semantics

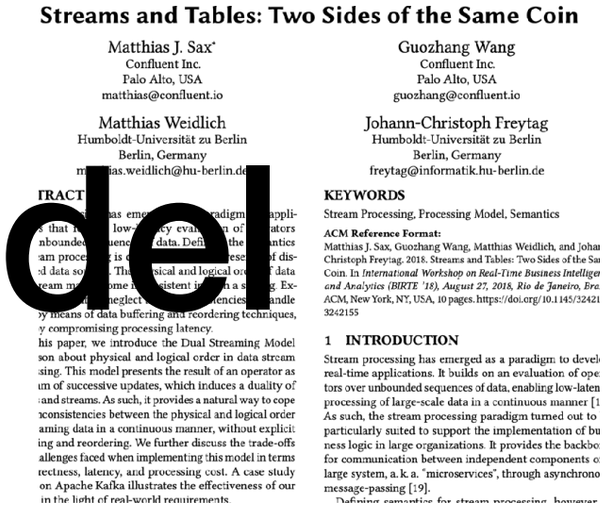
ACM Reference Format:
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johan
Christoph Freytag. 2018. Streams and Tables: Two Sides of the Sa
Coin. In *International Workshop on Real-Time Business Intelliger
and Analytics (BIRTE '18), August 27, 2018, Rio de Janeiro, Bra.*
ACM, New York, NY, USA, 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 ope
tors over unbounded sequences of data, enabling low-laten
processing of large-scale data in a continuous manner [1

Dual Stream Model

The Complete Picture



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

has emer radigm appli- that low- y evalu n of ators nbounded uenc data. Defi the antics am proce g is d ese of dis- d data so . Thi sical and logical or f data eam m ome i sistent in h a s g. Ex- neglect encies andle 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

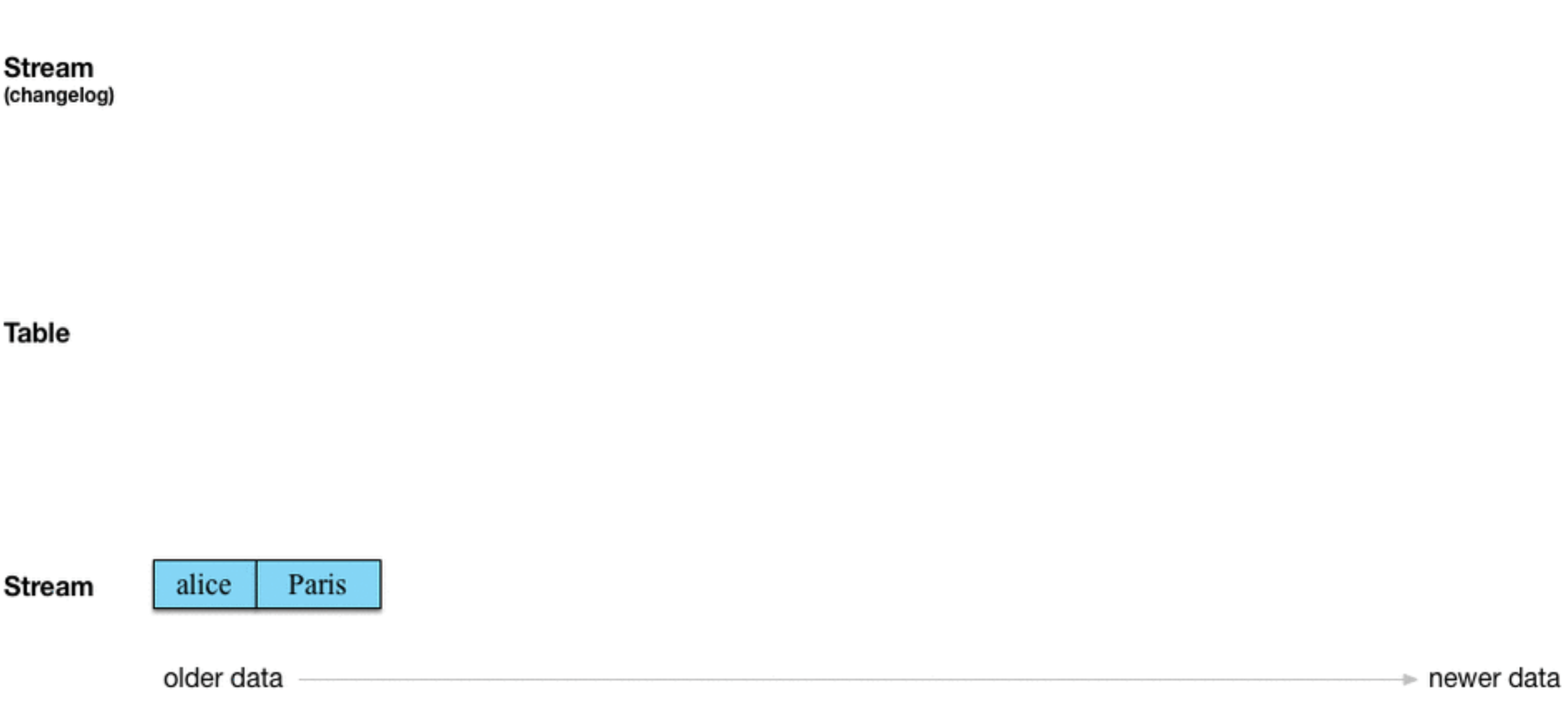
Stream Processing, Processing Model, Semantics

ACM Reference Format:
Matthias J. Sax, Guozhang Wang, Matthias Weidlich, and Johan Christoph Freytag. 2018. Streams and Tables: Two Sides of the Sa Coin. In *International Workshop on Real-Time Business Intelliger 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 ope 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 particularly suited to support the implementation of bu ness logic in large organizations. It provides the backbo for communication between independent components o large system, a. k. a. "microservices", through asynchrono message-passing [19].

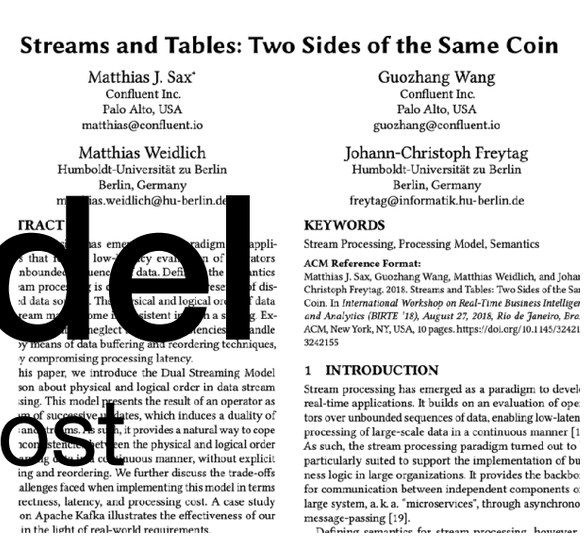
Defining semantics for stream processing, however



https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Dual Stream Model

Result Correctness vs Runtime Cost



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

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

KEYWORDS

Stream Processing, Processing Model, Semantics

ACM Reference Format:

1 INTRODUCTION

The DSM simplifies the **reasoning** about the **transformations**, but does not solve the **unboundedness** problem.

We still need infinite memory for processing an infinite stream.

DSM introduces the retention time to make the trade-off explicit.

Mapping to Kafka

Minimal

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 pro g has paradigm for appli-
s require l ency e ation of operators
nded sequences of data ning the semantics
cessing is he presence of dis-
d s rces, hysical a gical order of data
ream beco consist such a setting. Ex-
models ne encies or handle
by 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.

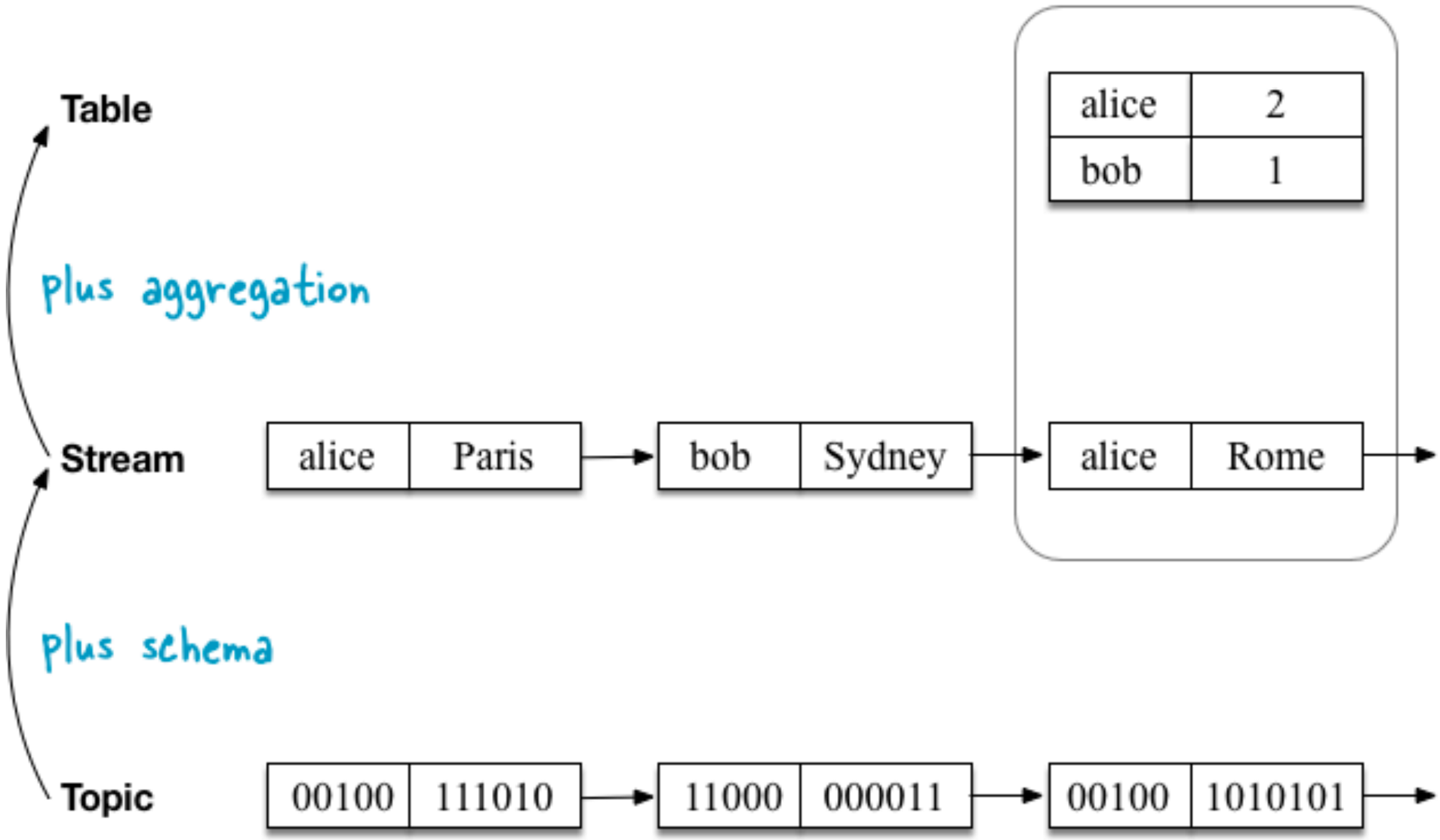
KEYWORDS

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 Sa
Coin. In *International Workshop on Real-Time Business Intelliger
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

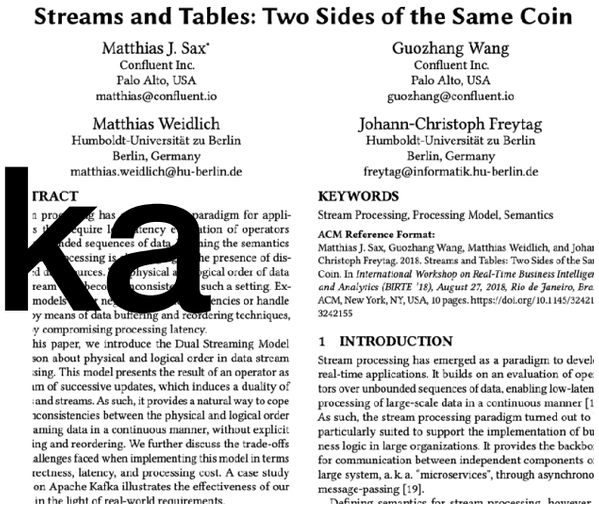
Stream processing has emerged as a paradigm to devel
real-time applications. It builds on an evaluation of ope
tors over unbounded sequences of data, enabling low-laten
processing of large-scale data in a continuous manner [1
As such, the stream processing paradigm turned out to
particularly suited to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



https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Mapping to Kafka

Minimal



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

KEYWORDS

Stream Processing, Processing Model, Semantics

ACM Reference Format:

1 INTRODUCTION

Concept	Partitioned	Unbounded	Ordering	Mutable	Unique key	Schema
Topic	Yes	Yes	Yes	No	No	No (raw
Stream	Yes	Yes	Yes	No	No	Yes
Table	Yes	Yes	No	Yes	Yes	Yes

Concept	Kafka	KSQL	Java	Scala	Python
Topic	-	-	`List/Stream`	`List/Stream[(Array[Byte],`	`[]`
Stream	`KStream`	`STREAM`	`List/Stream`	`List/Stream[(K, V)]`	`[]`
Table	`KTable`	`TABLE`	`HashMap`	`mutable.Map[K, V]`	`{}`

https://www.michael-noll.com/blog/2018/04/05/of-stream-and-tables-in-kafka-and-stream-processing-part1/

Streaming SQL

- New trend is hiding the complexity of the processing behind SQL
- Like CQL, but even simpler.
- Visit streaminglands.io

One SQL to Rule Them All:
An Efficient and Syntactically Idiomatic Approach to Management of Streams and Tables

An Industrial Paper

Edmon Begoli
Oak Ridge National Laboratory / Apache Calcite
Oak Ridge, Tennessee, USA
begoli@apache.org

Tyler Akidau
Google Inc. / Apache Beam
Seattle, WA, USA
takidau@apache.org

Fabian Hueske
Ververica / Apache Flink
Berlin, Germany
fhueske@apache.org

Julian Hyde
Looker Inc. / Apache Calcite
San Francisco, California, USA
jhyde@apache.org

Kathryn Knight
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
knightke@ornl.gov

Kenneth Knowles
Google Inc. / Apache Beam
Seattle, WA, USA
kenn@apache.org

ABSTRACT

Real-time data analysis and management are increasingly critical for today's businesses. SQL is the de facto lingua franca for these endeavors, yet support for robust streaming analysis and management with SQL remains limited. Many approaches restrict semantics to a reduced subset of features and/or require a suite of non-standard constructs. Additionally, use of event timestamps to provide native support for analyzing events according to when they actually occurred is not pervasive, and often comes with important limitations.

We present a three-part proposal for integrating robust streaming into the SQL standard, namely: (1) time-varying relations as a foundation for classical tables as well as streaming data, (2) event time semantics, (3) a limited set of optional keyword extensions to control the materialization of time-varying query results. Motivated and illustrated using examples and lessons learned from implementations in Apache Calcite, Apache Flink, and Apache Beam, we show how with these minimal additions it is possible to utilize the complete suite of standard SQL semantics to perform robust stream processing.

CCS CONCEPTS

• **Information systems** → **Stream management**; **Query languages**;

KEYWORDS

stream processing, data management, query processing

ACM Reference Format:
Edmon Begoli, Tyler Akidau, Fabian Hueske, Julian Hyde, Kathry Knight, and Kenneth Knowles. 2019. One SQL to Rule Them A An Efficient and Syntactically Idiomatic Approach to Managem of Streams and Tables: An Industrial Paper. In *2019 Internation Conference on Management of Data (SIGMOD '19), June 30-July 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 16 pag https://doi.org/10.1145/3299869.3314040

1 INTRODUCTION

The thesis of this paper, supported by experience developin large open-source frameworks supporting real-world stream ing use cases, is that the SQL language and relational mod ns is and with minor non-intrusive extensions, can be ver effective for manipulation of streaming data.

Our motivation is two-fold. First, we want to share our

Thanks!

Questions?