Arcon: Continuous and Deep Data Stream Analytics

Max Meldrum
RISE SICS
Stockholm, Sweden
max.meldrum@ri.se

Klas Segeljakt
KTH Royal Institute of Technology
Stockholm, Sweden
klasseg@kth.se

Lars Kroll
KTH Royal Institute of Technology
Stockholm, Sweden
lkroll@kth.se

Paris Carbone
RISE SICS
Stockholm, Sweden
paris.carbone@ri.se

Christian Schulte
KTH Royal Institute of Technology
Stockholm, Sweden
cschulte@kth.se

Seif Haridi*
KTH Royal Institute of Technology
Stockholm, Sweden
haridi@kth.se

ABSTRACT

Contemporary end-to-end data pipelines need to combine many diverse workloads such as machine learning, relational operations, stream dataflows, tensor transformations, and graphs. For each of these workload types, there exists several frontends (e.g., SQL, Beam, Keras) based on different programming languages as well as different runtimes (e.g., Spark, Flink, Tensorflow) that optimize for a particular frontend and possibly a hardware architecture (e.g., GPUs). The resulting pipelines suffer in terms of complexity and performance due to excessive type conversions, materialization of intermediate results, and lack of cross-framework optimizations. Arcon aims to provide a unified approach to declare and execute tasks across frontend-boundaries as well as enabling their seamless integration with event-driven services at scale. In this demonstration, we present Arcon and through a series of use-case scenarios demonstrate that its execution model is powerful enough to cover existing as well as upcoming real-time computations for analytics and application-specific needs.

CCS CONCEPTS

• **Information systems → Data streams**; • **Software and its engineering → Data flow languages**.

ACM Reference Format:
Max Meldrum, Klas Segeljakt, Lars Kroll, Paris Carbone, Christian Schulte, and Seif Haridi. 2019. Arcon: Continuous and Deep Data Stream Analytics. In *Real-Time Business Intelligence and Analytics (BIRTE 2019), August 26, 2019, Los Angeles, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3350489.3350492

1 INTRODUCTION

A wide variety of scalable data processing frameworks are available today, each tailored to a certain type of computation and manifested in a supported frontend. Examples are Spark focusing on *DataFrame*/Relational operations, Flink [2] adopting the Beam windowing model [1], TensorFlow for programming with Tensors etc. In practice, complete data pipelines typically involve more than a single framework. If we take a typical ML pipeline as an example there is a need to apply feature generation, model training, and serve these models right away within a live event-based application. Combining multiple frameworks in an application can lead to complexities and performance penalties such as mismatch of processing guarantees, hardware under-utilization, and high latency from rematerialization across frameworks. Furthermore, no user-facing programming model subsumes the others while no existing runtime has the capabilities to execute fundamentally different workloads (e.g., dataflow tasks and dynamically scheduled computation).

Arcon is an open-source system that aims to tackle this problem at both of its ends, by providing (1) **Arc**: a common intermediate representation (IR) to declare data-driven computation, and (2) **Arcon Runtime**: a general-purpose distributed runtime to execute programs compiled and optimized through Arc. Through this unique set of common features Arcon allows users to capture and execute a diverse range of computations such as live graph mining, training and serving machine learning models, and declaring stream window aggregates within a single application.

The Arcon runtime combines two distinct execution models into one: the long-running dataflow model seen in stream processing specializing on continuous, uninterrupted processing and the dynamic task scheduling model seen in batch processing systems which further specialize on deeper analytics. This demonstration showcases the seamless integration of both of these execution models within emerging application domains such as online business analytics and dynamic actor-based applications.

Demo Outline. We will cover (1) the internals of Arcon from its overall design and compilation down to its execution model, covering the main components of Arcon – its Arc compiler and the compilation phases that lead to code generation, as well as the Arcon runtime that deploys and executes the generated code fragments at scale – (2) we then provide a step-by-step examination of application use-cases that highlight Arcon's main features, in which we present and execute a set of distinct programs that make use of different existing frontends.

2 SYSTEM OVERVIEW

The main components of the Arcon system are the Arc program compiler and the Arcon runtime, as depicted in Fig.1. Arcon allows

*Also with RISE SICS.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

BIRTE 2019, August 26, 2019, Los Angeles, CA, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7660-0/19/08.
https://doi.org/10.1145/3350489.3350492

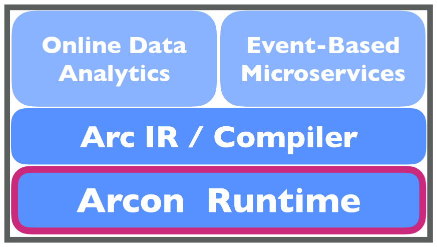


Figure 1: System Overview

existing and future frontends to be supported seamlessly by adding a direct translation of their operations to Arc. In this section, we provide a design overview of Arc and Arcon's runtime, while focusing on the respective characteristics of the demonstrated system.

2.1 The Arc Intermediate Language

The Arc IR [3] is an extension of the Weld IR [4], bringing the ability to express continuous and scalable computations on streams and windows. Weld is a language for describing transformations on bounded size datasets, and was designed to improve the performance of data analysis applications which depend on numerous libraries. By nature, libraries are optimized internally, but typically not with respect to each other. In consequence, expensive data movement costs may arise when functions of different libraries are pipelined, as intermediate results may need to be eagerly materialized. Weld addresses this issue by acting as a common translation layer for different computations. Weld's programming model builds on monadic comprehensions, enabling data parallelism, while abstracting over hardware. Data types in Weld are categorized as either value- or builder-types. The former are read-only data types, which can either be scalars or collections, and the latter are write-only data types, equivalent to additive monads.

Whereas Weld's builders are used to create values, the builders that Arc adds are used to construct stream processing pipelines of operators, with sources and sinks, connected by channels. Arc introduces streams and stream builders as new data types, and additionally supports windowed streams through higher-order builders referred to as windowers. As Arc retains the full expressiveness of Weld, developers will be able to harness the strengths of both stream and batch processing in the same application, enabling both continuous and deep analytics.

2.2 The Arcon Runtime

Arcon's distributed runtime is designed to reflect a clear separation of concerns between high performance and operability. At one end, it is meant to exploit special hardware as well as low-level system and network optimizations to execute the application critical path efficiently and reliably. Whereas, at the same time it has to integrate well with the existing cloud computing ecosystem (e.g., resource managers, schedulers, data stores etc.) for its non-performance-critical operational needs. For these reasons Arcon is divided into *Execution* and *Operational* planes, to achieve a suitable degree of synergy and independence between these two requirements. In Fig.2, we depict a physical deployment model of the runtime, the internals of which we summarize in the rest of this section.

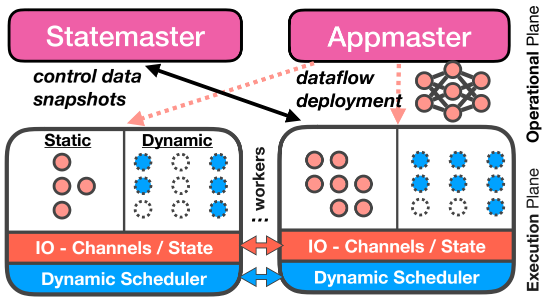


Figure 2: Runtime Overview

2.2.1 The Operational Plane. The operational plane is responsible for coordinating the distributed execution of an application including task deployment, monitoring, and state management. This part of the system is built in Scala Akka to acquire interoperability with existing tools and data processing libraries (e.g., YARN, HDFS) and a mature actor programming model to implement crucial operational services such as state management and job monitoring.

2.2.2 The Execution Plane. The Execution plane of Arcon provides a high-performance computation environment that is available locally to each operator. This includes support for all critical auxiliary mechanisms (e.g., multiplexed network IO, local state, and dynamic task scheduling). It is, in essence, a set of system libraries which ensure that all time-critical operations are performed without unnecessary delays. For that reason, we have implemented these libraries using a specialized middleware to execute program fragments written entirely in Rust for high performance, memory safety, and no garbage collection. As shown in Fig.2 each worker supports the execution of both long-running streaming tasks and short-running dynamic tasks. Short-running tasks are supported via dynamic scheduling and essentially offer the capability to support applications that involve intermittent batch computation (e.g., Reinforcement Learning) or "preempt" computation within stream dataflow graphs which are otherwise static. This can be further used to offload the critical path of a continuous application from time-consuming work such as performing external IO or computationally intensive computation (i.e. iterative computation, simulations).

3 SYSTEM DEMONSTRATION

Our demonstration focuses primarily on how problems from different application domains can be abstracted over and handled by a single runtime system. Onlookers should be able to follow the path from a domain specific problem description to how it is executed on a general purpose distributed processing infrastructure.

3.1 Demonstration Flow

During the system demonstration we will show a number of use-case scenarios that highlight Arcon's capabilities. Through Fig. 3, 4, and 5, we showcase the general demonstration flow that we will follow throughout each of these scenarios. Every scenario will begin by describing a high-level domain specific description of the problem in a common data science language like Python and Scala. Based on this, we will describe the Arc IR that implements this code in a domain independent manner that is understood by Arcon.

```python
import arc_beam as beam
import arc_beam.transforms.window as window
import weld_pandas as pd

def normalize(data):
    s = pd.Series(data)
    return s / s.sum()

p = beam.Pipeline(...)
(p | 'source' >> beam.io.ReadFromPubSub(Kafka(..))
                    .with_output_types(int)
   | 'filter' >> beam.Filter(lambda x: x > 0)
   | 'window' >> beam.WindowInto(window.FixedWindows(60))
   | 'map'    >> beam.Map(normalize)
   | 'sink'   >> beam.io.WriteToSink(Kafka(..)))
p.run()
```

Figure 3: Example Application using Beam & Pandas

Going further, we will demonstrate how this logical representation is converted into a deployment-ready dataflow graph, that we then execute on our runtime. While the application is running, we will describe the system's internals using the dataflow graph and elaborate on how different parts of the system interact. Finally, we will show the results of the execution, in some manner that is easy to follow for the onlookers. Fig. 3 highlights an application which combines Beam and Pandas. Beam constructs a pipeline of operators, reading input as a stream of integers from a Kafka source, and writing output to a Kafka sink. The intermediate processing steps involve filtering out negative integers, and creating a tumbling, one-minute long, window. Each window outputs a vector of its elements in a normalized form. The generated Arc IR and runtime deployment graph are illustrated in Fig. 4, and Fig. 5 respectively.

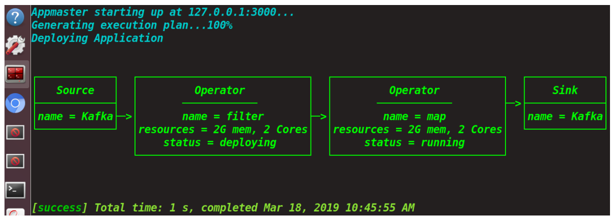
3.2 Application Domains

We will use the same demonstration flow to showcase a number of pipelines that currently suffer from the absence of either a cross-framework SDK integration or a common runtime. For that purpose we choose the application domains of "Online Analytics" and "Actor-Based Services".

Online Business Analytics: Business analytics are becoming increasingly important for day-to-day decision making rather than yearly or monthly retrospective reports. While the "online" aspect is inherently important for decision making, several stages within business analytics pipelines today are complex and require days of data reconciliation due to the combination of technologies involved. For example, if the business analytics models make use of a graph representation, underlying tasks can either perform deep analysis on graph snapshots (e.g., identifying the graph backbone using a system like Giraph) or incremental/approximate analysis on dynamic graphs (e.g., vertex counts, triangle counts, etc.). Nevertheless, since different runtimes are involved (e.g., a batch and a stream processing framework respectively) no data or computation sharing is possible in the overall process. Thus, a backbone identification algorithm could not make use of the "running" set of triangles in a dynamically updating graph. Machine Learning tasks are also known for the diversity of frameworks used across different stages of a pipeline (Pytorch, Keras, Pandas, MLlib, Ray, xarray etc.) and thus, the possibilities of data and computation sharing are

```
type ts = u64, val = i64, elem = { ts, val };
|source: stream[elem], sink: streamappender[elem]|
 let filtered = filter(source, |e: elem| e.$1 > 0);
 let windowed = for(filtered,
  windower[unit, appender[val]](
   |e:elem,_,_| {[e.$0/60L], ()},
   |wm:ts,open:vec[ts],_| {filter(open,|t|t<wm),()},
   |t:ts,agg:appender[val]| {t, normalize(result(agg))}),
   |w:windower,e:elem| merge(w,e));
 drain(windowed, sink)
```

Figure 4: Produced Arc IR



```
Appmaster starting up at 127.0.0.1:3000...
Generating execution plan...100%
Deploying Application

Source            Operator                           Operator                          Sink
name = Kafka -->  name = filter                -->   name = map                   -->  name = Kafka
                  resources = 2G mem, 2 Cores        resources = 2G mem, 2 Cores
                  status = deploying                 status = running

[success] Total time: 1 s, completed Mar 18, 2019 10:45:55 AM
```

Figure 5: Dataflow deployment

relatively low. Throughout the demo we will show how Arcon aims to solve these problems through its IR enabling common expression, optimization, and execution of fundamentally different tasks and execute them by the same runtime.

Stateful Actor-based Services: The use of actor-based frameworks is the current standard of writing general event-based services and applications. Typically, application logic is divided into logical units that are being implemented and executed in an actor framework such as Akka or Orleans. A shortcoming of structuring services in this way (often termed a "microservice") is the lack of providing integrated state management and fault tolerance across microservices or actors. Arcon aims to support such an application while providing uniform state management and seamless integration of long-running event-based task logic and dynamic calls, e.g., to fetch external resources for data enrichment.

4 CONCLUSIONS AND FUTURE WORK

We provide a detailed, interactive demonstration of Arcon, a general-purpose distributed computing framework that aims to unify the declaration and execution of data-driven applications. Throughout a series of use-cases we show how programs that make use of multiple data processing frontends can be compiled, optimized and get executed in a unified way. One of the main future directions of Arcon is to add support for most existing data computing frameworks via a direct translation of their primitives to Arc.

REFERENCES

[1] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *VLDB*, 2015.

[2] P. Carbone, S. Ewen, G. Fóra, S. Haridi, S. Richter, and K. Tzoumas. State Management in Apache Flink: Consistent Stateful Distributed Stream Processing. *Proc. VLDB Endow.*, 10(12):1718–1729, Aug. 2017.

[3] L. Kroll, K. Segeljakt, P. Carbone, C. Schulte, and S. Haridi. Arc: An IR for Batch and Stream Programming. *Proceedings of the 17th ACM SIGPLAN International Symposium on Database Programming Languages*, 2019.

[4] S. Palkar, J. J. Thomas, A. Shanbhag, D. Narayanan, H. Pirk, M. Schwarzkopf, S. Amarasinghe, M. Zaharia, and S. InfoLab. Weld: A common runtime for high performance data analytics. In *Conference on Innovative Data Systems Research (CIDR)*, 2017.