Elastic Stream Processing on Serverless Environments

Tsenos Michalis [1], Vana Kalogeraki [1]

[1]Department of Informatics, Athens University of Economics and Business, Greece

1. Introduction

- Serverless Computing
  - One of the newest and most enticing cloud computing models
  - Provides the illusion of always-available resources
  - Maintenance-free, shifts the complexity of allocating and provisioning resources to the cloud provider
- Application Development on Serverless Environments
  - Developers write applications as Functions
  - Each Function and its dependencies are packed and delivered typically as a container
  - Each Function is triggered by specific events such as HTTP calls on specified endpoints
  - Popular platforms: AWS Lambda, IBM Cloud Functions, or open-source systems such as OpenFaas and OpenWhisk

2. Serverless Benefits

- Serverless Computing has important benefits: rather than deploying and managing dedicated virtual machines
  - pay-as-you-use model: users are able to deploy individual functions and pay only for the time that their code is actually running
- Better resource allocation
  - The cloud provider allocates and manages resources depending on current demand. During lengthy periods of inactivity, the number of active instances can be reduced to zero, releasing any resources to other applications.

3. Serverless for real-time stream processing

- Serverless is an ideal model for real-time stream processing for data streams with variable data rates and resource demands due to:
  - high elasticity and auto resource allocation
  - attractive pricing model
- Compared to traditional systems that need:
  - extra maintenance
  - rate limiting techniques in order to handle sudden data bursts

4. Elastic Stream Processing

- In our work we have proposed an architecture for Serverless stream processing and a methodology for supporting real-time elastic operations on data streams.
- Our system employs:
  - Apache Kafka as the data delivery service
  - OpenFaas as our Serverless framework
  - Prometheus as a monitoring system and time series database
  - S3 for persistent storage

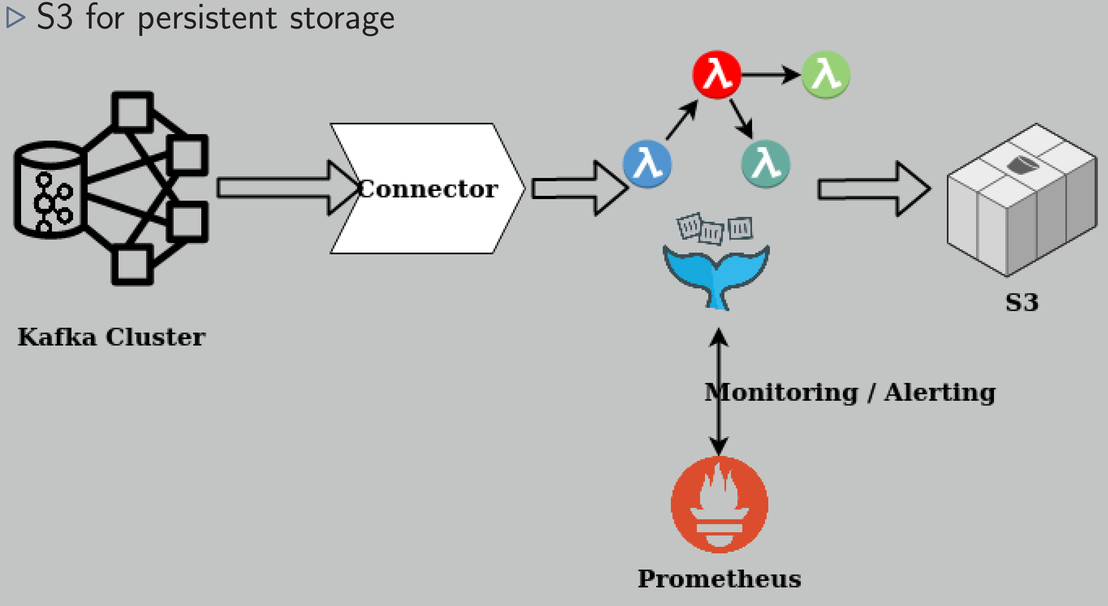

Figure: System architecture

5. System Architecture

- Process Graph
  - describes the function invocation order in a DAG format
  - each node represents a function
  - for each function the Process Graph maintains a pool of workers that invoke the function
  - the execution result is propagated to the next function following the vertices of the DAG
  - allows branching and multiple sinks
  - each function is able to read and write directly to S3 or its open-source version Minio for persistent storage
- Connector
  - custom Kafka connector
  - delivers the function invocation messages from Kafka topics to the Process Graph
  - able to chain multiple Process Graphs in order to create more advanced and dynamic processing flows
- Prometheus
  - monitors the performance of each function in the Process Graph

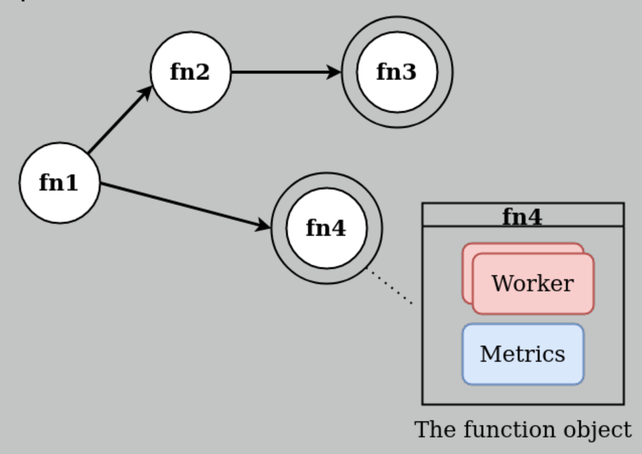

Figure: Process Graph

Methodology

- Auto-scaler component
  - receives the application real-time requirements (typically expressed in the form of end-to-end latency constraints)
  - receives the availability of the computing and storage resources
  - determines the scaling decisions
  - can scale down and can even scale to zero during prolonged periods of inactivity in order to keep the cost low
- Goals
  - perform scaling actions without interrupting the execution of the Process Graph
  - identify the appropriate number of replicas that should be utilized to meet the application requirements while also considering the scaling cost
  - mitigate stragglers by estimating the improvement on the application's execution time and considering how the load is balanced across the nodes

Contact Information

- Michalis Tsenos
  - Email: tsemike@aueb.gr
- Vana Kalogeraki
  - Email: vana@aueb.gr
  - Web: http://www2.cs.aueb.gr/~vana/
- Real-Time Distributed Systems Group
  - Web: http://rtds.aueb.gr