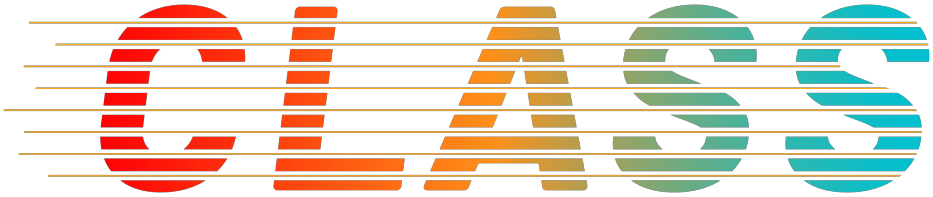

D5.2 Advanced Performance Evaluation Tooling for CLASS Big-Data Services

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M15, March 2019
Dissemination Level	PU
Nature	R + DEM
Author(s)	Erez Hadad (IBM)
Contributor(s)	Sadek Jbara (IBM)
Reviewer(s)	Jorge Montero (ATOS)
Keywords	Analytics, Serverless, Map, Reduce

Notices: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "780622".*

Change Log

Version	Author	Description of Change
V0.1	Erez Hadad (IBM)	Initial Draft
V0.2	Jorge Montero (ATOS)	Review
V0.3	Erez Hadad (IBM)	Final version
V1.0	Eduardo Quiñones (BSC)	Ready for EC submission

Table of contents

1 Executive Summary

This document describes deliverable D5.2 – performance evaluation tooling for CLASS big-data analytics workloads, in accordance with Task T5.2 of the CLASS DoA [1], deliverable D5.1 [2] and other related CLASS documentation. It marks another successful delivery of the CLASS project as part of milestone MS2, executed in M7-M15.

The document is laid out as following. Section 2 consists of overview and feature description of this deliverable, in alignment with other deliverables and project goals. Section 3provides a technical specification of each delivered component. Section 4 provides delivery details: per-component bundling / installation, and demonstration / impact where available. Last, Section 5 provides a glossary of new key technology products involved in CLASS, to assist in overall understanding of the document.

2 Overview

The CLASS analytics workloads, dealing with latency-sensitive computation on one hand and with the need to process large amounts of data on the other hand, needs to be carefully evaluated for throughput and latency, and the trade-off in between. In this deliverable we produce tooling dedicated to this purpose.

As described in D5.1 [2], analytics workloads in CLASS are intended to be delivered, upon integration, wrapped as a set of OpenWhisk actions (functions), which as pieces of logic that can be invoked via REST, CLI or in response to events, as shown in Figure 1 below.

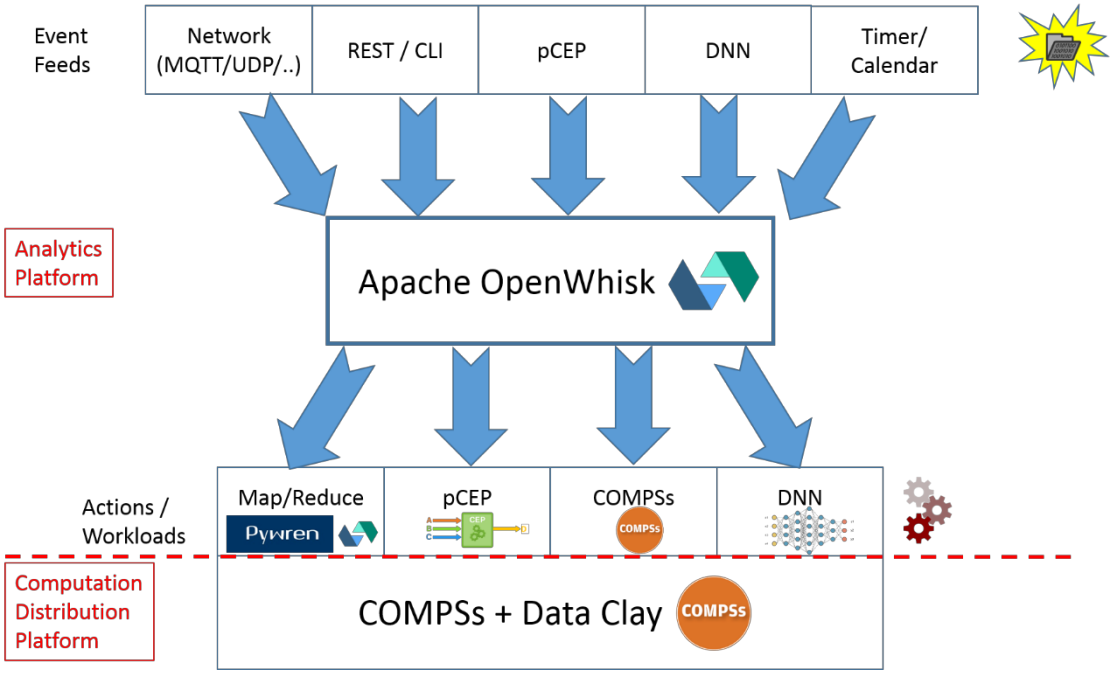

Figure 1. CLASS Analytics Layer Architecture

Putting OpenWhisk as the connecting hub of the analytics architecture naturally leads to a stress-testing technique that relies on the OpenWhisk API, for the purpose of characterizing a CLASS deployment from performance perspective – latency and throughput at various stress levels, maximum throughput (service rate) of each workload, etc.

A second important tooling capability described in CLASS DoA [1] is to *correlate* the workload execution with the underlying infrastructure behaviour – CPU utilization, memory utilization etc. Such correlation is key to identifying and understanding phenomena that affect performance, such as bottlenecks and "noisy neighbour" interference [3]. The infrastructure used by CLASS is Kubernetes in the cloud (D2.1 [4], D4.1 [5]) and native Linux and/or Docker in the edge (D5.1 [2], D3.1 [6]).

Following the above, the performance tooling release contains the following features:

1. **Benchmark tool:** *owperf* – a tool for stress-testing an OpenWhisk-based deployment and recording the measured latency and throughput. owperf can measure latency and throughput both of actions (from invocation to response) invoked synchronously or asynchronously, and of rules, invoked through event generation. It further provides a breakdown of the measured latency to separate the overhead of the OpenWhisk infrastructure from the core workload processing itself.
2. **Monitoring facility:** this facility consists of a set of services that is shared in part with the CLASS cloud infrastructure, managed by WP4 [7]. For this deliverable, this facility is intended for *correlation* - identifying the time-frame of workload execution stages of interest, and for *drill-down* - inspecting resource behaviour during said time-frames to identify performance issues.

3 Technical Specification

This Section provides a more detailed description of the specific components that are included in this release. Each component Section should include the technical details of the component along with a usage description and example, if such is available and relevant.

3.1 owperf – Benchmark Tool For OpenWhisk

3.1.1 Technical Description

owperf is a performance benchmark tool for OpenWhisk-based deployments. It has been developed by our team in IBM as open-source. A highly-detailed technical description of owperf is available in the git repository of owperf in public github at [8]. Therefore, in this section, only a brief description is provided.

owperf is a CLI tool with two main functions:

1. Generating a load of invocations for specific actions or rules with specific behavior parameters, such as arrival rate and maximal burst size.
2. Measuring latency and throughput (service rate) of the OpenWhisk service interaction, reported in a CSV format that can be consumed in a spreadsheet post-factum for further analysis.

The operation of an owperf benchmark consists of the following steps:

1. **Setup:** the tool creates OpenWhisk assets to be tested
2. **Test:** the tool fires up a specified number of concurrent clients - a master and workers.
    - Each client wakes up once every *delta* msec (iteration) and invokes the specified activity: either the event trigger or multiple concurrent actions forming a burst. Action invocations can be blocking or not.
    - After each client has completed a number of initial iterations (*warmup*), measurement begins, controlled by the master client, for either a specified number of iterations or for specified time.

- At the end of the measurement, each client retrieves the OpenWhisk activation records of its triggers and/or actions, and generates summary data that is sent to the master, which generates and prints the final results.

3. **Teardown:** clean up the OpenWhisk assets created during setup

Both setup and teardown can be disabled in CLI so owperf can be integrated as a building block for complex multi-workload test scenarios (see Usage below).

3.1.2 Usage

The envisioned use of owperf in CLASS consists of the following scenarios:

1. Comprehensive benchmarking of a single component (action / rule) – by directly using owperf to invoke it. For example, have a PyWren workload invoked in a specific frequency and measure the service rate (i.e., rate of responses) to see if it meets a designated rate goal.
2. Construct a complex multi-workload benchmark for testing. This involves running multiple instances of owperf concurrently (via a script), one for each workload, with a specific invocation load, for a reasonably long period of time, and measure the resulting latency and throughput for each workload. Such a test can help identify interference and bottlenecks rising from the joint operation of workloads.

To adjust owperf to invoke a particular workload, the default test action that comes with owperf needs to be replaced with the action of the workload. It is required that an action invoked by owperf returns the unique activation ID and the duration it took to execute – both simple operations that can be completed in any programming language. There is also a detailed reference to how to implement them in the default test action.

The launcher of owperf is called `owperf.sh` . Details of how to invoke owperf by CLI can be obtained by running

```
./owperf.sh -h
```

The following example stresses the test action using 3 concurrent clients firing in each iteration a burst of 4 invocations, 5 times per second (once every 200 msec, which is the default delta). So the total arrival rate is 60 invocations per second (3x4x5) with cumulative bursts between 4 and 12 (3x4) invocations each. The benchmark lasts for 100 iterations past warmup, which is roughly 20 seconds (100:5).

```
./owperf.sh -a action -w 3 -i 100 -r 4
```

The output is twofold. On stderr (and can be silenced) is a log of the operation, as well as a header of the output CSV record. On stdout is the CSV output record itself.

3.2 Monitoring Facility

3.2.1 Technical Description

The monitoring facility is a set of services that is deployed on top of the infrastructure, in conjunction with D4.2 [7], which uses a subset of these services for general monitoring and SLA management. Its main goals are, with respect to this particular deliverable:

- **Correlation**: provide a means that given an execution stage of interest of a workload, allows extracting the *time frame* at which the stage happened
- **Drill-down**: Provide a means for visualizing and closely inspecting the system behavior in a given time frame in order to understand a particular issue, such as bottleneck or cross-workload interference.

The monitoring facility consists of the following services, all open-source and well-known in the IT industry:

1. *collectd* [9] – a metric collector. It is agent-based (i.e., has an agent deployed in every CLASS node) and is responsible for collecting metrics from the infrastructure using various plugins – for Linux, for Docker, for Kubernetes, etc
2. *Prometheus* [10] – a database for storing and querying metrics. It interfaces with collectd to retrieve collected metrics.
3. *Grafana* [11] – an engine for visualizing metrics (time-series data) on a web UI. Grafana can retrieve data from metric databases, such as Prometheus and Graphite [12], and has innate zooming and shifting capabilities for focusing on specific time ranges.
4. *Elasticsearch* [13] – a text search engine. It is capable of operating on index data built from raw text and performing queries.
5. *Logstash* [14] – a log processing service. It is capable of periodically retrieving logs of applications based on configuration, directing the logs to a common storage location / service, and then processing the logs to generate index data that can be used by e.g., Elasticsearch.
6. *Kibana* [15] – a web UI for Elasticsearch. It accepts queries, directs them to Elasticsearch and presents the results, typically visualized via a bar chart mapping entry time to result count, but it also supports far more complex visualizations.

The services of 1-3 are used for general collection, storage and retrieval of infrastructure metrics. In the cloud, they are shared with WP4 (see D4.1 [5]). The services 4-6 are commonly known together as *ELK stack* (ELK is an acronym of the service names), and are intended to retrieve, index and analyze the workload operation, by processing the log entries of EXPRESS / OpenWhisk (see D5.3 [16]), COMPSs, etc. Their combined operation delivers the capabilities of correlation and drill-down, as explained in the next sub-section.

Connecting workloads and frameworks to the ELK stack is done typically via *beats* / *forwarders* [17], which are components that actually monitor the logs of a specific source (e.g., specific workload), optionally do some pre-processing, and push the data to Logstash.

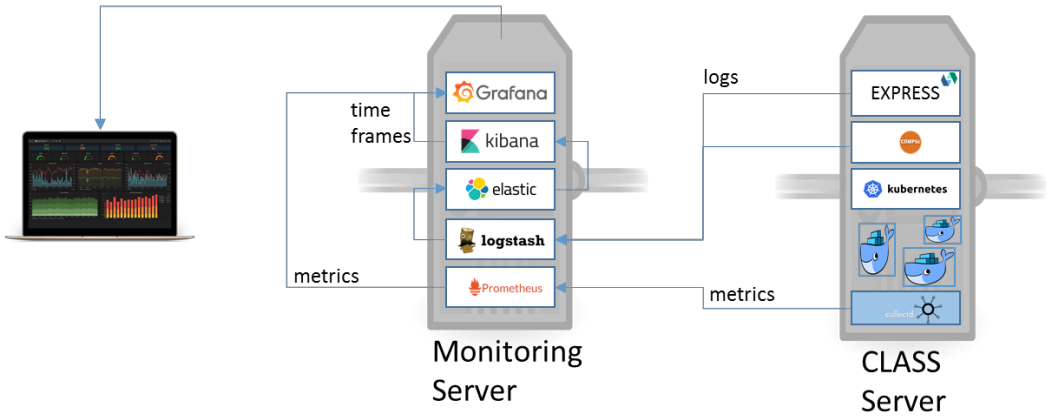

Figure 2. Monitoring Facility Deployment in CLASS

For the CLASS prototype, the core logic of the above services (save collectd) is intended to be deployed in a monitoring server the cloud, and provide monitoring for both cloud nodes and fixed edge nodes (street nodes). See Figure 2, showing the facility collecting metrics via collectd and logs from EXPRESS, COMPSs, Kubernetes and any other framework / workload

that needs to be connected to it. The mobile edge node (car node) monitoring configuration is still being discussed among CLASS partners, planned to resolve towards MS3.

3.2.2 Usage

The monitoring facility is intended to be used in the following manner:

1. During normal operation, both collectd and Logstash periodically collect their respective data: metrics from the infrastructure and logs entries (via forwarders or filebeats) from monitored workloads and framworks.
2. Collected metrics are stored in Prometheus. Collected log events are stored and indexed for Elasticsearch.
3. When an issue is detected with performance of a certain workload component, a time-frame of the issue needs to be defined:
    a. If the issue is raised by the SLA manager (see D4.2 [7]), it is time-stamped and refers to specific Kubernetes pods, which are also monitored. The identities of the pods can be looked up in the Kubernetes logs using Kibana to identify the specific time frame. The basic process of using Kibana to look up matching log entries is simple and demonstrated in [18].
    b. If the issue is raised for a specific OpenWhisk activation, then the time-frame of the activation can be looked up directly using `wsk activation logs <activation_id>` and then extracting the time-stamp from the log entries, or by using Kibana.
    c. If the issue is raised in other service or framework that has time-stamped logs, then the principle is the same: use an identifier or keyword specific to stage in question, to look up the stage log in Kibana. Use the time-stamps of the log entries to establish a time-frame that encapsulates the issue.
4. The time-frame can be used in Grafana to zoom in on the infrastructure behavior, the metric values during the issue can be inspected to identify the root cause and possibly tune CLASS workloads.

4 Delivery

This Section contains all the specifics of the delivery, including: bundling, installation instructions, and related demonstrations and impact, in accordance with the DoA and evolution of work. As before, there is a sub-section per-component.

4.1 Bundling and Installation

4.1.1 owperf – A Benchmark Tool For OpenWhisk

owperf is a node.js application, hosted in a public git repository at [8].

Its dependencies are:

1. A Linux system – Tested on Ubuntu 16.04.
2. *git* [19] – available as a package for most Linux distributions.
3. *node.js* [20](including *npm*) – install version 8: https://github.com/nodesource/distributions/blob/master/README.md#debinstall
4. An OpenWhisk system – CLASS analytics is based on EXPRESS (see D5.3 [16]). Installing it also delivers the *wsk* CLI client with settings located in `~/.wskprops`

After having all the dependencies installed, run `npm install` in the owperf folder to install the npm dependencies. Now you can start using the tool.

4.1.2 Monitoring Facility

Of the set of services comprising the monitoring facility, as listed in Section 3.2.1, services 1-3 are the responsibility of WP4. Therefore, please refer to D4.2 [7] to learn how to install them. In this section we further discuss only the ELK stack (services 4-6).

ELK Stack

The ELK stack is available online in a public github repository from its maker (Elastic). It can be easily installed on Kubernetes (CLASS cloud) using the *helm* tool [21], which is a tool for executing and managing complex deployments on top of Kubernetes. The blueprint for deployment via helm is called a *helm chart*.

1. Instructions for installing and using helm: https://helm.sh/docs/using_helm/#quickstart-guide
2. The helm chart for deploying ELK stack: https://github.com/helm/charts/tree/master/stable/elastic-stack

To add OpenWhisk logs (in EXPRESS) to be processed in the ELK stack, please also install the OpenWhisk Logstash Forwarder [22], using instructions found here:

https://github.com/jthomas/openwhisk-logstash-forwarder

Logs of other frameworks or workloads (COMPSs, DNN, etc), may be added to ELK in the next integration phase, towards MS3.

4.2 Demonstration and Impact

This deliverable contains a demonstration of owperf, in the form of a video showing the tool being presented to the Apache OpenWhisk community in one of their bi-weekly video conferences, on February 20, 2019: https://www.youtube.com/watch?v=gJg8gOeYUX8

The tool was initially named “overhead” and later renamed to “owperf” by community response. The presentation starts approximately at 23:50 minutes from video start.

As the video shows, the community response was quite positive, with explicit support from the OpenWhisk maintainer, Carlos Santana. Following this presentation, a process started of contributing owperf to the core Apache OpenWhisk project via PR #4320 located at:

https://github.com/apache/incubator-openwhisk/pull/4320

5 Product Glossary

This auxiliary Section provides a brief overview of the products involved in CLASS analytics, to put the above discussion in context.

5.1 Apache OpenWhisk

Apache OpenWhisk [23] (OW for short) is a serverless, open source cloud platform, which was initiated, and is still maintained, by IBM. OpenWhisk executes functions (called *actions*) in response to events, at scale. Both actions and events are high-level abstractions that can be implemented in various ways. Actions, as code, can be written in virtually any programming language (although there are 7+ languages that have official support), and using many platforms and SDKs. Similarly, events can represent any concrete event or signal, such as message arrival, command invocation, device signals, or mark the occurrence of a higher logic result, such as complex events or other decision logic. Once defined, events can be bound to actions using *rules* to create event-driven applications, with simple facilities for relaying event

data to invoked actions. Such applications are cloud-native, in the sense that events can arrive and be processed by actions anywhere in the cloud, and actions are elastically auto-scaled to match the event load.

The resulting programming model of OpenWhisk offers several attractive advantages to developers, in addition to polyglot programming and auto-scaling. Developers do not need to manage the location of their code (hence the term "serverless"), its life-cycle or its resource allocation – OpenWhisk uses a default (but customizable) resource allocation for each action. Actions are time-limited to keep consistent with the original serverless model from AWS, but time-limit is configurable.

The architecture of OpenWhisk consists of the following components, as shown in Figure 3 below.

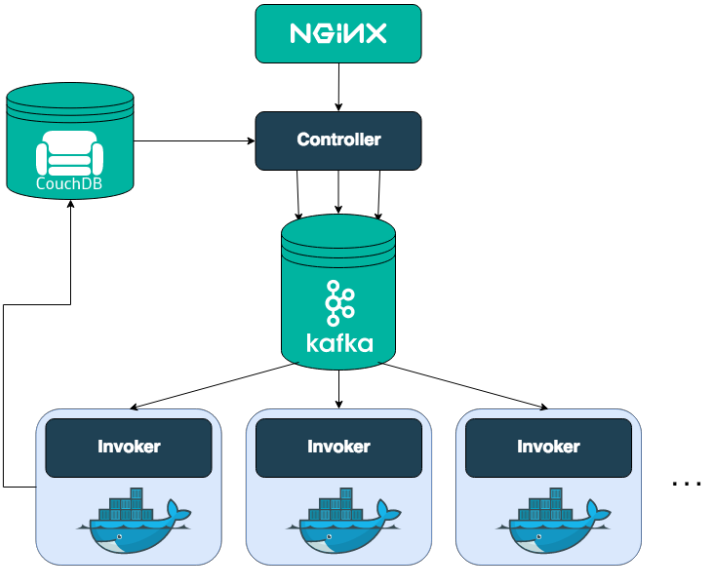

Figure 3: OpenWhisk Architecture

The OW architecture consists of the following components:

- **NGINX** is an optional reverse proxy, used for load-balancing controllers and for SSL termination.
- **CouchDB** is a database used for storing the assets created by users – actions, triggers, rules, packages and records of action activations.
- **Controller** is a management logic of OW. It implements the OW REST API, and dispatches actions for execution at invokers in response to events.
- **Kafka** is a message bus used to distribute messages from controllers to invokers in a cloud setting.
- **Invoker** is a "worker" of OW. It executes actions using IaaS or cluster facilities. By default, an invoker uses Docker containers for running actions, but there are variations that use Kubernetes and other facilities.

OW has a simple interface consisting of a REST API and a CLI (`wsk` command) which wraps the REST API. It allows creating actions and invoking them, creating event triggers from event feeds of actual events, binding event triggers to actions via rules, and several secondary operations. OW programming model is documented in detail in [24].

5.2 PyWren

PyWren [25] is a system that was built at UC Berkeley's RISELAB to enable highly scalable execution of existing Python functions on the cloud using the serverless platform. It started on AWS Lambda, the serverless platform of AWS Cloud, and later it was converted [26] to use IBM's Cloud Functions, based on Apache OpenWhisk.

PyWren's programming interface is based on Map/Reduce. The developer writes a *client* program that during runtime, creates an *executor*, which issues highly parallel computations as `map` and `reduce` operations in its API. These operations get executed using a set of concurrent serverless functions / actions. Each action's environment is prepared to include the dependencies needed to execute the map or reduce function, including dependencies of the function code and of PyWren itself. The functions used in `map` and `reduce`, as well as the input and output datasets, are shared between the client and the actions via object storage.

The current basic API for *map* and *reduce* in PyWren is as following [26]:

- `executor.map(func, dataset)`
- `executor.map_reduce(map_func, dataset, reduce_func)`

Both operations perform map first, which applies `func` (`map_func` in `reduce`) to all elements of the dataset. For `reduce`, it later applies a reduce function to the dataset resulting from map. The `reduce` function has an internal accumulator carrying result from one computation to the next, ending with a single final result. Figure *4* below demonstrates the operation of a `map` computation in PyWren, involving the client, IBM Cloud Functions (OpenWhisk) and object storage, in an IBM Cloud setup.

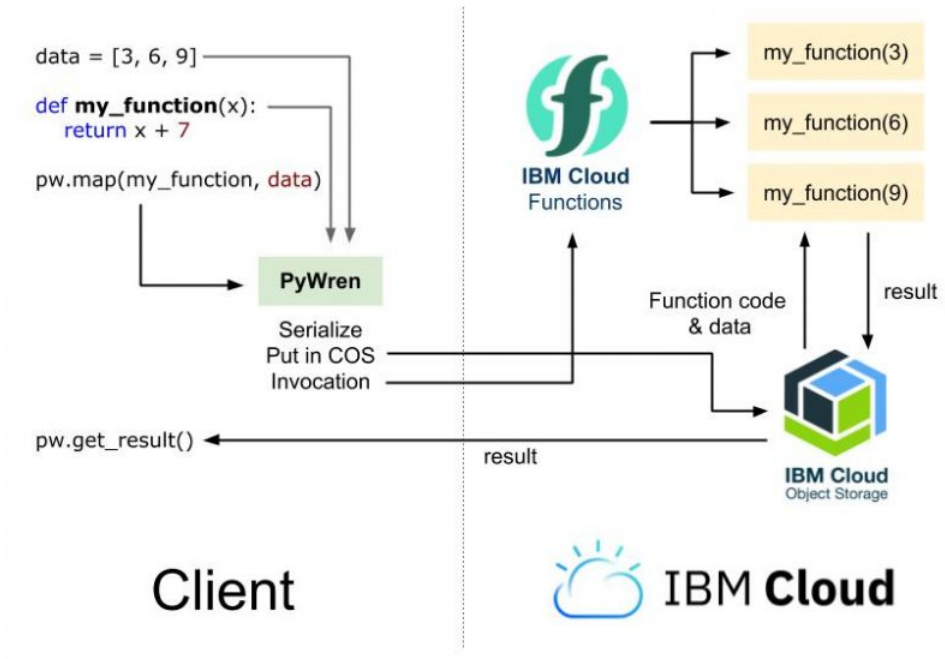

Figure 4: Execution of a PyWren map operation

5.3 COMPSs

Deliverable D2.1 [4] provides a detailed description of COMPSs.

6 References

[1] CLASS Consortium, "CLASS: Edge&CLoud Computation: A Highly Distributed Software Architecture for Big Data AnalyticS," Baercelona, 2017.

[2] E. Hadad, "D5.1 Analytics Requirements," CLASS, 2018.

[3] D. Hill, "Eliminating Noisy Neighbors in the Public Cloud," 2013. [Online]. Available: https://www.networkcomputing.com/data-centers/eliminating-noisy-neighbors-public-cloud. [Accessed 3 2019].

[4] E. Quinones, "D2.1 – CLASS software architecture requirements and integration plan," CLASS, 2018.

[5] O. Avila, "D4.1 – Cloud requirements specification and definition," CLASS, 2018.

[6] R. Cavicchioli, "D3.1 – Real-time analysis of the edge computing platform," CLASS, 2018.

[7] I. Chulani, R. Sucassas and O. Avila, "D4.2 First release of the Cloud Data Analytics Service Management components," 2019.

[8] E. Hadad, "owperf - A performance evaluation tool for Apache OpenWhisk," IBM Research, 2019. [Online]. Available: https://github.com/IBM/owperf. [Accessed 3 2019].

[9] "collectd – The system statistics collection daemon," [Online]. Available: https://collectd.org/. [Accessed 3 2019].

[10] "Prometheus - Moniotoring system & time series database," Linux Foundation, [Online]. Available: https://prometheus.io/. [Accessed 3 2019].

[11] G. Labs, "Grafana - The open platform for analytics and monitoring," Grafana Labs, [Online]. Available: https://grafana.com/. [Accessed 3 2019].

[12] C. Davis, "Graphite," Orbitz, [Online]. Available: https://graphiteapp.org/. [Accessed 3 2019].

[13] Elasticsearch B.V., "Elasticsearch: RESTful, Distributed Search & Analytics," Elasticsearch B.V., [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed 3 2019].

[14] Elasticsearch B.V., "Logstash: Collect, Parse, Tranform Logs," Elasticsearch B.V., [Online]. Available: https://www.elastic.co/products/logstash. [Accessed 3 2019].

[15] Elasticsearch B.V., "KibanaL Explore, Visualize, Discover Data," Elasticsearch B.V., [Online]. Available: https://www.elastic.co/products/kibana. [Accessed 3 2019].

[16] E. Hadad, "D5.3 Initial Release of CLASS Big-Data Analytics Layer," IBM Research, 2019.

[17] Elasticsearch B.V., "Beats: Data Shippers for Elasticsearch," Elasticsearch B.V., [Online]. Available: https://www.elastic.co/products/beats. [Accessed 3 2019].

[18] T. Roes, "Kibana 5 Introduction," [Online]. Available: https://www.youtube.com/watch?time_continue=179&v=mMhnGjp8oOI. [Accessed 3 2019].

[19] "Git," [Online]. Available: https://git-scm.com/.

[20] "Node.js," Node.js Foundation, [Online]. Available: https://nodejs.org/en/.

[21] ". N. C. Foundation", "helm/helm: The Kubernetes Package Manager," Cloud Native Computing Foundation, [Online]. Available: https://helm.sh/. [Accessed 3 2019].

[22] J. Thomas, "OpenWhisk Logstash Forwarder," [Online]. Available: http://jamesthom.as/blog/2017/11/21/openwhisk-logstash-forwarder/. [Accessed 3 2019].

[23] Apache OpenWhisk, "Apache OpenWhisk is a serverless, open source cloud platform," Apache Foundation, [Online]. Available: http://openwhisk.apache.org/. [Accessed 21 June 2018].

[24] Apache OpenWhisk, "incubator-openwhisk/docs at master · apache/incubator-openwhisk · GitHub," [Online]. Available: https://github.com/apache/incubator-openwhisk/tree/master/docs#readme. [Accessed 21 June 2018].

[25] "pywren -- run your python code on thousands of cores - pywren," RiseLab, UC Berkeley, [Online]. Available: http://pywren.io. [Accessed 21 June 2018].

[26] G. Vernik and J. Sampe, "Process large data sets at massive scale with PyWren over IBM Cloud Functions - IBM Cloud Blog," IBM, [Online]. Available: https://www.ibm.com/blogs/bluemix/2018/04/process-large-data-sets-massive-scale-pywren-ibm-cloud-functions/. [Accessed 21 June 2018].