D5.5 Evaluation of CLASS Big-Data Analytics Layer

Version 1.0

Document Information

Contract Number	780622
Project Website	https://class-project.eu/
Contractual Deadline	M42, 30th June 2021
Dissemination Level	PU
Nature	R
Author(s)	Erez Hadad (IBM)
Contributor(s)	Pavel Kravchenko (IBM), Rut Palmero-Sanchez (ATOS), Jorge Montero (ATOS)
Reviewer(s)	Roberto Cavicchioli (UNIMORE)
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
0.1	Erez Hadad (IBM)	Initial Draft
0.2	Roberto Cavicchioli (UNIMORE)	Review
0.3	Erez Hadad (IBM)	Final draft for submission
1.0	BSC	Final version ready for EC submission

Table of contents

1 Executive Summary

This document describes deliverable D5.5 “Evaluation of CLASS big data analytics layer”, which focuses on evaluating CLASS analytics, both qualitatively and quantitatively, as required by CLASS DoA [1]. The qualitative evaluation is w.r.t. CLASS requirements and features presented and refined in deliverables D5.1 [2], D5.3 [3], D5.4 [4], and other related CLASS documentation. The quantitative evaluation of CLASS analytics presented in this report is based on CLASS use-case workloads, with the full end-to-end evaluation of the CLASS use-case discussed in deliverable D1.6 [5]. Also, this document reports further optimizations and improvements that have been realized as part of the optimizations and additional impact. Together with the rest of MS4 documents, this document concludes the CLASS project as part of milestone MS4, executed in M31-M42.

Similar to MS3, this milestone has been subject to constraints that have slowed progress behind expectations. We deliver all planned content under some mitigations, as following:

- The EXPRESS prototype is delivered stand-alone, not yet integrated into CLASS applications – see D2.7 [6]. The original estimation of its effort was inadequate, and much additional work was required. Much of this has since happened, but not enough to allow integration. To accommodate and demonstrate value within CLASS project time-frame, we ported several EXPRESS principles directly into Lithops [7] (formerly PyWren), resulting in significant performance improvements and discussed below. The eventual integration of EXPRESS is still planned for additional value, but may happen outside the project time-frame.

The document lays out as follows. Section 2 details technical improvements and optimizations introduced in CLASS components as part of the ongoing integration and evaluation. Section 3 consists of evaluation of CLASS analytics at the end of the project. First, qualitatively, in contrast with CLASS requirements and design, and then quantitatively, based on experimental evaluation. Next, Section 4 provides additional impact details. Last, Section 5 provides a glossary of key technology products involved in CLASS, to assist in overall understanding of the document.

2 Code Updates

The analytics layer of the CLASS software stack has been finalized at MS3, in accordance with the project plan and as reported in D5.4 [4]. Since then, during integration and evaluation, the WP5 work diverged into multiple work streams. One was (as expected) continuous integration. Among the others were evaluations and refinements of analytics back-ends and continued work on EXPRESS. For specific backends, such as COMPSs, DNN in the edge (YOLO) and DNN in the cloud (SLA predictor), the specific evaluations are delegated to the respective evaluation reports of WP2, WP3 and WP4. In this report we therefore report only further changes made to Lithops and to EXPRESS.

2.1 Lithops

As can be seen in Figure 1 below, Lithops plays an important role in the main CLASS use-case application of collision detection. In particular, it is responsible for executing the Trajectory Prediction (TP) and Collision Detection (CD) application components. As such, Lithops has been a focus of attention with several optimizations.

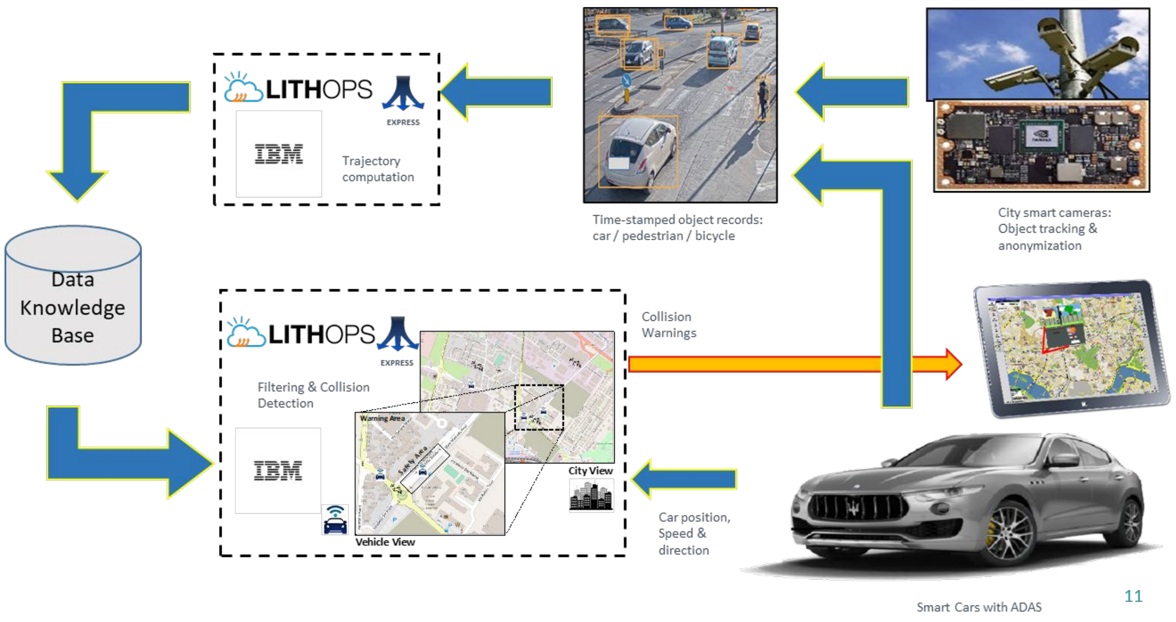

LITHOPS
EXPRESS
IBM
Trajectory computation
Data Knowledge Base
Filtering & Collision Detection
Warning Area
City View
Vehicle View
Time-stamped object records: car / pedestrian / bicycle
City smart cameras: Object tracking & anonymization
Collision Warnings
Car position, Speed & direction
Smart Cars with ADAS
11

Figure 1 : CLASS application for Collision Avoidance Use-Case

2.1.1 A Practical Timing Goal

As discussed in several earlier documents (e.g., D1.4 [8]), the CLASS collision avoidance application aims to handle objects that are in the "Warning Area" of a moving car, i.e., objects that are outside the immediate field of view of the mounted sensors or the driver's senses. Assuming that the warning area objects are at least 20-50 meters away from a car moving at an urban speed limit of 50 Km/h, on a dry road and no slope (matching the Modena testing conditions), we use a Stopping Distance Calculator such as [9] to derive that the response time of the entire system should be 0.5-2.5 seconds. Based on that, we define the practical range of computation time for CLASS prototypes for TP and CD to be no more than 0.5 seconds.

2.1.2 Lithops Operation

Figure 2 below demonstrates how Lithops operates. Lithops is a portable map/reduce engine implemented as a Python library, that is designed for massively-parallel computation. It provides two operations of *map(f, ds)* and *reduce(f, ds)*, of which *map()* is the more important one, as it allows concurrent application of the given function *f* to all elements of the given dataset *ds*, allowing efficient use of parallel infrastructure for a large amount of independent computations. *map()* is used for TP and CD, as described in detail in D5.4 [4].

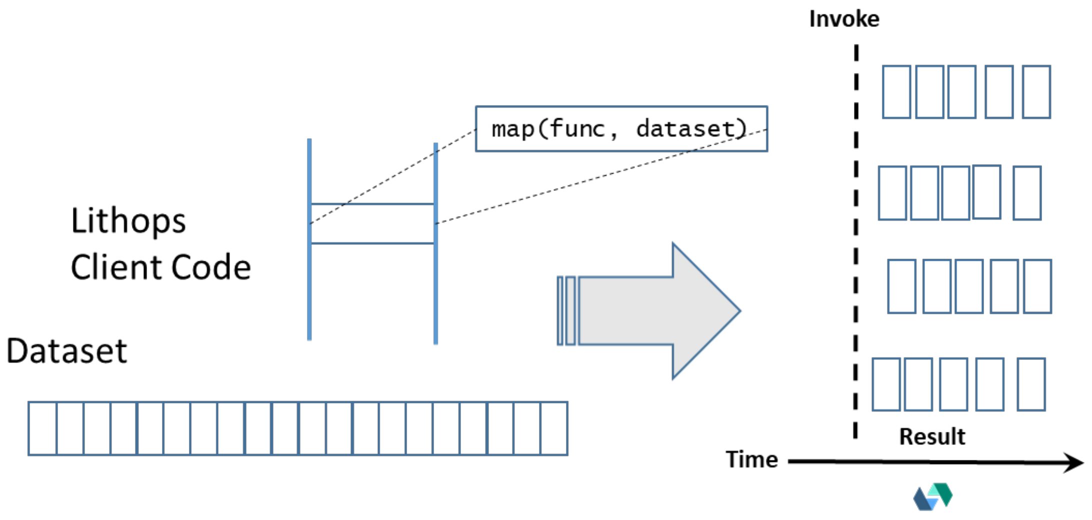


Figure 2: Lithops Map Operation

map(f, ds) works in the original Lithops in the following way: As shown in Figure 2, a Lithops computation begins with a client (using the Lithops library) that invokes the map() operation executed by workers (that implement the concurrent computation). The process is as follows:

1. Lithops registers its worker at the target platform that is used to execute the computation. For example, on a serverless platform (such as Apache OpenWhisk) it registers the worker as a function, based either on an existing Docker image (e.g., from DockerHub) or using a customized Docker image.
2. The function *f* and dataset *ds* are each serialized and stored as a single object in object storage, which is required for Lithops operation.
3. Lithops partitions the dataset *ds* into its elements and builds an invocation for each element. The invocation data contains a reference to the serialized function and data and a specific index range for the specific element's data
4. All invocations are concurrently and asynchronously dispatched from the client, causing workers to be instantiated and started at the target platform. The number of concurrent workers is limited by the target platform, so there might be several rounds of workers being started to cover all elements. In Figure 2, the example maximum concurrency is 4.
5. The client proceeds to wait for result notifications by polling worker-specific objects in object storage.
6. Each worker, once started, uses the invocation data to de-serialize the function *f* and its designated data element *e* from object storage.
7. After de-serialization, each worker invokes *f(e)*, writes the result to its designated object in object storage, updates some finalization statistics and terminates.
8. At the client, Lithops collects results from all the workers' respective objects and returns them as the map() result.

2.1.3 Optimizing Lithops

Early benchmarks showed that the original version of Lithops exhibits high overhead of initialization and finalization, often in the order of multiple seconds. A detailed analysis of the code and execution revealed operation flow details that have been summarized in the previous Section. Discussions about Lithops applicability in many other big-data use-cases in IBM Research and in EU projects (e.g., CloudButton [10]) concluded that for general-purpose batch computation, this kind of performance overhead is actually acceptable:

- The overhead time in many cases is dominated by net computation time when using long-running map functions, e.g., in the order of minutes
- Data is often too large to be passed directly to workers, e.g., due to serverless protocol limitations, so mediating data using a 3rd-party storage such as object storage is required.

However, this is clearly not the case for CLASS applications such as the collision detection application (i.e., TP or CD computations), where big data typically consists of a large number of small elements, per-element computation is short (order of tens of milliseconds) and total computation latency is of extreme importance.

With those differences established, we set out to optimize Lithops for CLASS. The first round of optimizations for Lithops, which has already been reported in D5.4 [4], consisted mostly of tuning and configuration without almost any code changes, and resulted in single object computation (TP) still being above our goal of 0.5 seconds, even with the combined improvements in DataClay, which is an integral part of the computation – in TP, the Lithops code reads and writes to DataClay, and in CD, it reads from DataClay.

In the second round of optimizations, presented in this report, we focused on optimizing the code of Lithops. We profiled the *map()* operation in detail and confirmed our anticipation that a bulk of the overhead time was spent in the cumbersome interaction between the client and the workers using object storage. Not only that, we identified that our on-premise deployment of Lithops in Modena, which is using Minio [11] as a local object storage solution, suffers from increased per-object access time when increasing access concurrency, thus significantly damaging our Lithops solution value as a scalable timely computation. Thus, we decided to make all necessary code changes to completely remove any dependency of Lithops on object storage. The resulting code changes highlights have been as following:

1. Invocation data is now passed directly to workers using the serverless protocol (or that of any other target platform)
2. Results from workers are notified back to the client using a fast message queue (RabbitMQ [12]) instead of polling object storage
3. A new storage backend was added to Lithops: “storageless”, which is a dummy implementation of the Lithops storage interface that throws an exception if asked to perform a meaningful operation such as read or write. This helped us verify that Lithops does not require storage operations anymore.

Additionally, we implemented *chunking*, which allows sending groups of elements (more than one) in each worker invocation. This feature serves two purposes. First, it provides a way for developers to calibrate their Lithops computation by controlling the resulting concurrency to meet a latency requirement (above a certain minimum). Second, it allows to mitigate some performance issues discovered during quantitative evaluation and discussed in Section 3.2. This feature, along with direct communication and message-queue based notifications, are also inspired by similar EXPRESS features. EXPRESS features efficient communication with its workers – the runners, and reusing the runners for multiple executions for amortizing the cost of worker startup and shutdown.

3 Evaluation

In this Section we review and evaluate the fit-for-purpose of the CLASS analytics layer. We begin with a qualitative evaluation w.r.t. CLASS requirements and consequent design and evolution as presented in D5.1 [2], D5.3 [3] and D5.4 [4]. In the second sub-section, we present a quantitative performance evaluation focusing on latency and discuss several related observations that have been collected during evaluation.

3.1 Qualitative Evaluation

To assess the final implementation of the CLASS analytics layer from a quality perspective, we list the main features and discuss how they are addressed.

3.1.1 Inclusive, Event-Driven Programming Model Featuring Multiple Analytics API

As shown in Figure 3 below, the analytics layer of CLASS is based on a serverless platform, using Apache OpenWhisk [13]. This allows CLASS developers to compose their applications from serverless *functions* – pieces of business logic that can be written in different programming languages and employ different libraries and analytics engines, yet be able to invoke one another through the common serverless platform's protocol, thereby forming the complete application. This universal connectivity of the serverless protocol is clearly demonstrated in the main CLASS use-case of collision avoidance application, where the COMPSs workflow invokes the Lithops-based TP and CD logic, which are available as serverless functions. Invocation can be either synchronous (i.e., wait until completion) or asynchronous (invoke and continue).

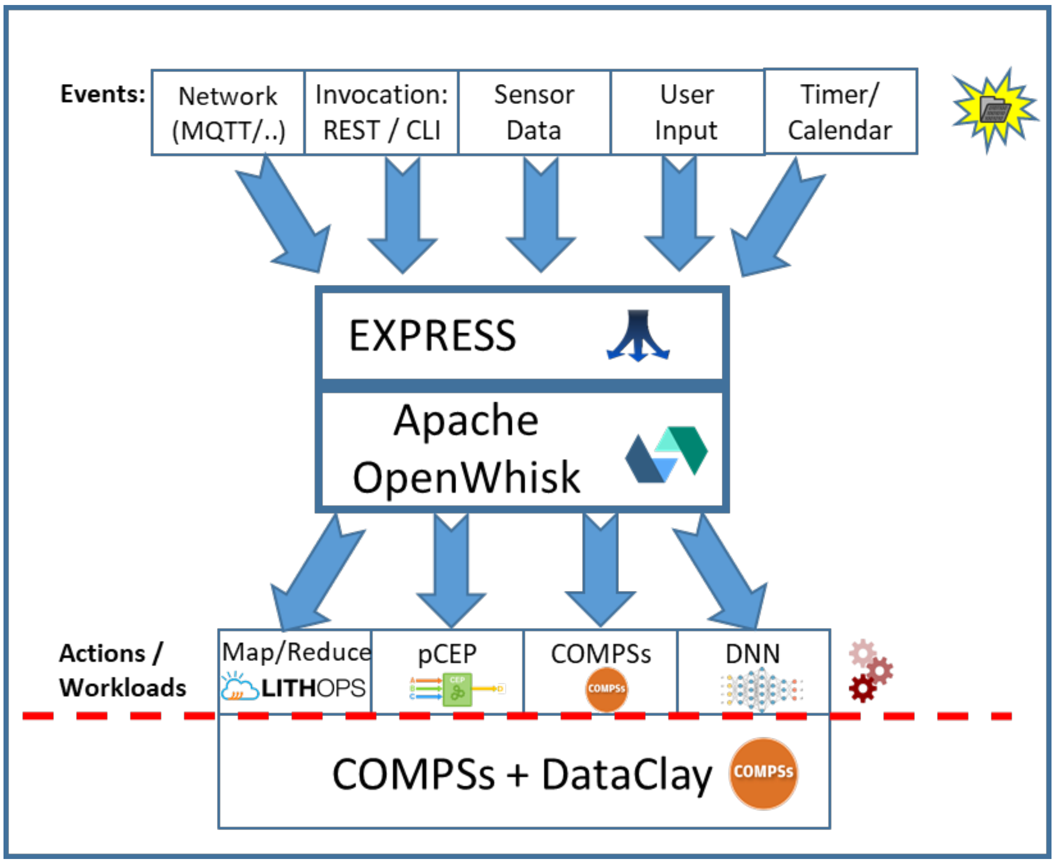


Figure 3: CLASS Analytics Architecture

The same OpenWhisk protocol also allows to bind serverless functions to *events*, which are similar to pub-sub channels. This allows functions to be invoked based on real-world events converted to OpenWhisk events through OpenWhisk *feeds*. Functions can also distribute notifications to other functions using events and cause independent activation of multiple other logic components in response to these notifications.

Another alternative model that is enabled by combining a data back-bone with events is an “enrichment” model in which a function activation is triggered by a timer event, and the function, when triggered, retrieves data from the data back-bone, performs its update and writes the updated data back. An important advantage of the enrichment model is that it allows different stages of the same workflow to operate at different frequencies according to their capabilities, while guaranteeing eventual generation of results by having the data propagated through the different stages using the back-bone.

The integration of EXPRESS on top of OpenWhisk does not lose any properties of the underlying serverless platform. Functions running inside EXPRESS pools are still reachable through the serverless protocol using EXPRESS wrappers, which are deployed as standard (but tiny and efficient) OpenWhisk functions. Similarly, events trigger wrappers which relay the data to EXPRESS functions asynchronously, thereby generating little additional overhead. Also, EXPRESS functions can be written in different languages and use different engines by using matching EXPRESS runners. Last, even the EXPRESS API itself is largely language-independent since it is based on a message queue (RabbitMQ [12]) which has polyglot API and on Protocol Buffers [14] which are polyglot as well. Thus, EXPRESS can be directly invoked from practically all common programming languages.

Support for various analytics APIs is enabled in OpenWhisk (with or without EXPRESS) using the standard process of building and deploying OpenWhisk functions as described I D5.1 [2]. Explicit support has been added for COMPSs workflows on OpenWhisk in D5.3 [3]. Lithops-specific actions have also been built, and integrated into the demonstration use-case for Trajectory Prediction and Collision Detection, as explained in detail in D5.4 [4].

3.1.2 Predictable Computation

One major CLASS goal is predictable computation – executing computation in bounded time with high probability. Given that CLASS does not deal with factors below the middleware, such as operating system scheduling and hardware control, it aims only for “soft” or “near” real-time computation.

Enablement of predictable computation in CLASS analytics layer is focused on three aspects. The first is tuning the relevant analytics infrastructure component to generate a bounded[1] *overhead*, as shown in the quantitative analysis below. The second aspect is allowing *scheduling* to be customized in favor of real-time execution. The third aspect is giving developers a way to *calibrate* their computation, by providing different options for executing the same workload yielding different completion times.

The above aspects are apparent in many components of the final CLASS analytics layer, starting with the core. For the underlying Apache OpenWhisk, CLASS leverages *warm containers* where functions are first pre-loaded in memory using a “warm-up” round before engaging in production computation, to make sure function startup overhead, which could be quite significant, is outside the critical invocation path. Furthermore, DataClay session start, which is a CLASS-specific overhead, is included in the function startup and skipped when using warm start, as demonstrated by the TP and CD functions in the collision avoidance application.

Unfortunately, OpenWhisk has its own fixed scheduling algorithm, and calibration is mainly handled by OpenWhisk configuration tweaking, which is a cumbersome process. Given the declining business interest in pursuing OpenWhisk-specific optimizations, as reported in D5.4

[1] Without complete control of the entire software stack, it is not reasonable to guarantee absolute bounds on computation overhead. Therefore, we aim to show “bounded with high probability” as discussed in the quantitative evaluation below.

[4], we initiated EXPRESS around mid-project, which is a significant leap beyond OpenWhisk limitations, and is completely independent of OpenWhisk itself.

EXPRESS, which we presented in detail in D5.4 [4], supports predictable execution at the serverless level much more extensively than OpenWhisk. Recall that the method of operation of EXPRESS is “nested computation” - it deploys its own custom runners (CRs) as serverless functions and then executes functions inside the CRs according to its own *custom* scheduler. By decoupling the serverless function life-cycle from the executed function’s life-cycle it allows for a customary model which separates initialization and finalization from the critical invocation path. EXPRESS support for custom scheduling allows explicit integration of a real-time scheduling algorithm. Last, calibration in EXPRESS is enabled by *dynamic* control of the size of its runner pools. This means the application developer gets to control both the static limits of the concurrency – minimum and maximum number of concurrent execution slots in the runner pool, as well as the policy by which the pool is dynamically resized in response to event load. As explained in D5.4, each EXPRESS slot is used for repeatedly executing requests (without initialization and finalization) directed to it from the EXPRESS controller through a fast message queue-based protocol. As can be understood, this is what inspired the “chunking” feature of Lithops. However, Lithops chunking is only a crude approximation, since a chunk size is fixed, whereas EXPRESS control is fine-grained and dynamic, allowing features such as load-balancing and deadline awareness.

Aside from the infrastructure components, predictable computation is also implemented in analytics backends. COMPSs has a designated deadline-aware scheduler for workflows that is separately discussed in D2.7 [6]. In Lithops, computation overhead was intensively optimized and reduced, as discussed in Section 2.1.3 above and evaluated below. Also, calibration is now available in Lithops via chunk size control, with the resulting maximum concurrency equals the dataset size divided by chunk size. Custom scheduling is not required in Lithops since it has a fixed simple fan-out workflow with all elements of equal priority.

3.1.3 Cloud and Edge Operation

CLASS aims to allow developers to deploy applications easily and efficiently anywhere across the compute continuum – both on cloud and on edge nodes. For edge nodes / clusters, which are separate and less reliably accessible than cloud, CLASS analytics features a *federation* mechanism that allows eventually consistent propagation of software assets deployed at the cloud to designated edge nodes, operating as programmable analytics agents of CLASS. Federation has been implemented and reported in D3.2 [15] and in D3.3 [16]. Furthermore, federation has been put to use in the CLASS use-case as a means to update the edge code for detector/aggregator in CLASS nodes, proving its value as a simplified means of controlling edge nodes at scale, and reported in D3.4.

3.1.4 Data Backend Connectivity

As originally discussed in D5.1 [2], universal computation across the compute continuum requires both compute and data. For universal data access across the continuum, as described in in D2.7 [6], CLASS relies on dataClay. Support for dataClay for analytics applications is demonstrated both in COMPSs (reported separately in in D2.7 [6]) and in Lithops. For Lithops applications, this has been implemented and reported starting in D5.3 [3]. Specifically, we implemented a “data managers” layer that has both a dataClay back-end and a simple CSV file back-end, to allow porting the original TP and CD logic, which was contributed by ATOS as a sequential application that used CSV input, into a Lithops-based serverless function that uses dataClay.

3.2 Quantitative Evaluation

In this section we report and discuss empirical performance evaluation results conducted for the CLASS analytics infrastructure and analytics backends. Following the general approach of this document, we focus on OpenWhisk and on Lithops, as other components are being separately evaluated and reported in respective MS4 documents.

3.2.1 Evaluation Method

Our evaluation method is designed to be closely aligned with the main CLASS collision avoidance use-case, as following. We use the CLASS deployment in Modena, with OpenWhisk deployed in a Kubernetes cluster in the Modena data center. We use the actual TP and CD serverless functions that are used in the collision avoidance use-case as our benchmark applications. The datasets used for computation are also generated from actual Modena input videos recorded during integration sessions.

Our Kubernetes cluster in Modena consists of 1 master node and 4 workers nodes (provisioned as virtual machines), all deployed with Kubernetes v1.19.3. 3 worker nodes have 8GB memory each and 4 cores of Intel Xeon Gold 5120 operating at 2.2GHz. 1 worker node has 60GB of memory and 16 cores of the same mode. So, in total there are 28 cores.

We use official OpenWhisk version tag “ed3f76e” (released 11/20) pulled from Docker Hub. We use a standard deployment scheme of one OpenWhisk invoker and one OpenWhisk controller per node, one global nginx for load-balancing across controllers and one global Kafka for controller-invoker communication. Note that available memory for OpenWhisk is restricted to 7GB per node due to a recent feature that sets a uniform limit on memory per invoker, so we use the common size of usable memory. The translates to a global limitation of about 100 concurrent functions for the entire cluster.

We instrumented the TP and CD functions, both at the client and at the worker components, with code that recorded time-stamps of different stages, and used that information to generate detailed profiling information from each invocation.

Last, the benchmarks consisted of running TP and CD at different workload size and chunk size and collecting profiling information. Recall that a chunk size controls how many concurrent invocations are generated by Lithops and it is designated as the key factor in calibrating Lithops performance (aside from the infrastructure size itself). TP (Trajectory Prediction) is a unary operation performed at each object individually, so workload size for TP is essentially the number for objects in the set. CD (Collision Detection) is a binary operation, taking two objects as input, with at least one of which being a connected car. For isolated performance evaluation and stress generation, all objects are labeled as connected cars. Thus, workload size is determined by number of pairs - all 2-combinations (i.e., unordered unique pairs) of objects from a given object set. Each workload is executed 10 times as warm containers (i.e., after a warm-up round) and profiling data is collected and analyzed, as discussed further below.

To demonstrate the improvements introduced during project life-time and documented in D5.4 and in Section 2.1.3, we compare two versions of Lithops. One is *baseline*, in which the improvements are turned off (by configuration). The other is *current*, which employs all improvements. The version of dataClay that is used with both current and baseline is the latest that was available during the benchmarks. This makes the comparison more accurate and focused only on Lithops and OpenWhisk, as opposed to previous performance enhancement indications in D5.4 [4] that were based on comparing together OpenWhisk + Lithops + dataClay using a few point values.

3.2.2 Evaluation Results

Evaluation results are presented as following. We start with overall end-to-end computation latency for TP and for CD, comparing baseline and current. We then discuss some insights and observations arising from the collected data. Finally, we present and discuss OpenWhisk invocation overhead and its significance in practical use of serverless infrastructure for latency-sensitive computation.

3.2.2.1 Total End-To-End Latency

Figure 4 below presents the total end-to-end latency results for both baseline and current CD functions. The measurements are taken for chunk size of 1, 3, 5 and 10 objects, for workload sizes of 5, 10, 20 and 50 objects. Both graphs (baseline and current) are shown on the same X and Y axis.

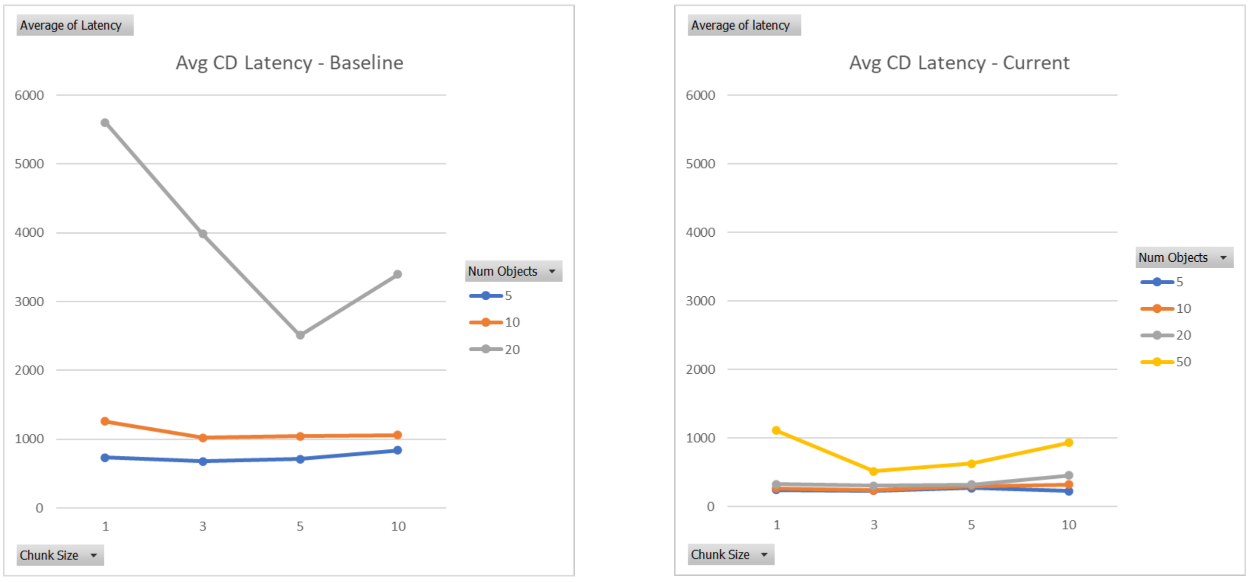


Figure 4: Average end-to-end latency of Collision Detection

A few observations:

1. The current Lithops improves in total latency up to 7X over baseline, e.g., when comparing the 20 objects workload latency
2. The current Lithops delivers CD of 50 objects (>1200 object pairs) in ~500 msec, which is the upper practical limit, as described in Section 2.1.1. Baseline CD for 50 objects is too slow for any practical reasons (tens of seconds) so it is not included.
3. A typical and expected graph shape of "smiling curve" is formed for all workload sizes. This demonstrates two conflicting apparent behaviors:
    a. Too-high chunk sizes imply reduced concurrency, where the existing infrastructure is under-utilized and few Lithops workers have to process lots of object pairs sequentially, yielding high total latency.
    b. Too-low chunk sizes for large workloads yield a number of invocations that is higher than the number of maximum concurrent workers allowed by OpenWhisk, thus causing workers to be re-invoked with new sets of pairs but with additional overhead of worker reset (finalize and restart). For small workloads, too-small chunks cause straggler sensitivity – i.e., that workload is thinly spread over many workers, and that late-starting workers (stragglers) delay total completion time, yielding again increased total latency.

4. Calibration is indeed valuable, as the shortest total latency for a given workload size is attained at different chunk sizes – sometimes 1, sometimes 3, sometimes 5. A future logic (e.g., DNN) can help in automatically determining the right configuration.

Figure 5 below presents total end-to-end latency for TP, baseline and current. In a TP computation, processing is done per-object, as opposed to per-object-pair in CD. Because of that, the same size of object set yields a much smaller workload, compared to the square-size workload of CD. Thus, we benchmark TP with bigger object set sizes to examine its behavior realistically compared to CD. As expected, the results exhibit behavior similar to CD with similar observations. Note that the improvement demonstrated through project optimizations in TP is as much as 3X, demonstrated for 50 object workload. Another minor observation that is more apparent with TP is that small workloads with large chunk size yield similar latencies, because all computation is handled essentially in a single Lithops worker (a trivial case).

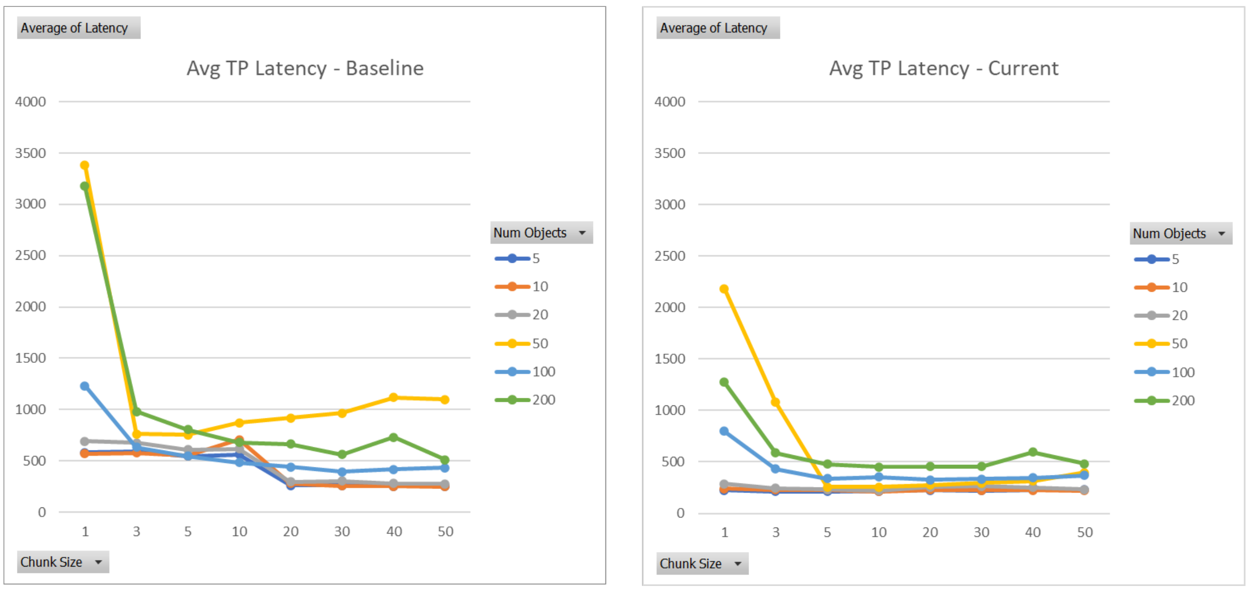


Figure 5: Average end-to-end latency of Trajectory Prediction

3.2.2.2 OpenWhisk Overhead

In this Section we evaluate the invocation overhead imposed by OpenWhisk. Figure 6 below presents the overhead measured for TP and CD invocations. As we are dealing with platform overhead, we consider results collected from both baseline and current versions of both applications.

Each of the 4 graphs in Figure 6 presents the 90th percentile of OpenWhisk invocation overhead, for TP and CD, baseline and current. The calculation of per-invocation overhead is done by subtracting the net execution time of the respective TP or CD application from the total (gross) execution time, which was presented and discussed in the previous Section.

The most important observation from this data, which can be easily seen, is **the critical importance of warm start**. Note that the difference between measurements is in order of 10s of msec for all chunk sizes and workload types, except for a few outliers that stand out, e.g., CD baseline, chunk size 10 and workload of 20 objects. Analyzing the logs, we traced those outliers to two cases of cold start:

- Lithops worker cold start – this can happen e.g., when increasing the number of workers between two computations. The resulting delay can be in the order of 100s of msec.
- Lithops client cold start – this can happen e.g., when changing workloads (TP to CD or baseline to current and vice-versa). The resulting delay is very large (1000s of msec) since it also involves re-connecting with dataClay.

In both cases, delaying too long after a warmup round (e.g., resetting dataClay with test content) or improperly-tuned warmup, may cause the cold start to happen. This is because warm-up in serverless platforms is an *opportunistic* feature. This clearly indicates one key motivation of EXPRESS – *guaranteed* warm-container behavior with warm concurrency completely at the control of the application developer. This is also the reason why various market competitors developed their own custom-made solutions to address this exact issue, such as AWS Lambda Provisioned Concurrency [17]. However, as already explained in D5.4 [4], EXPRESS will provide this capability on top of any serverless platform, and as an open-source project. In that sense, this evaluation provides empirical evidence to the necessity of EXPRESS.

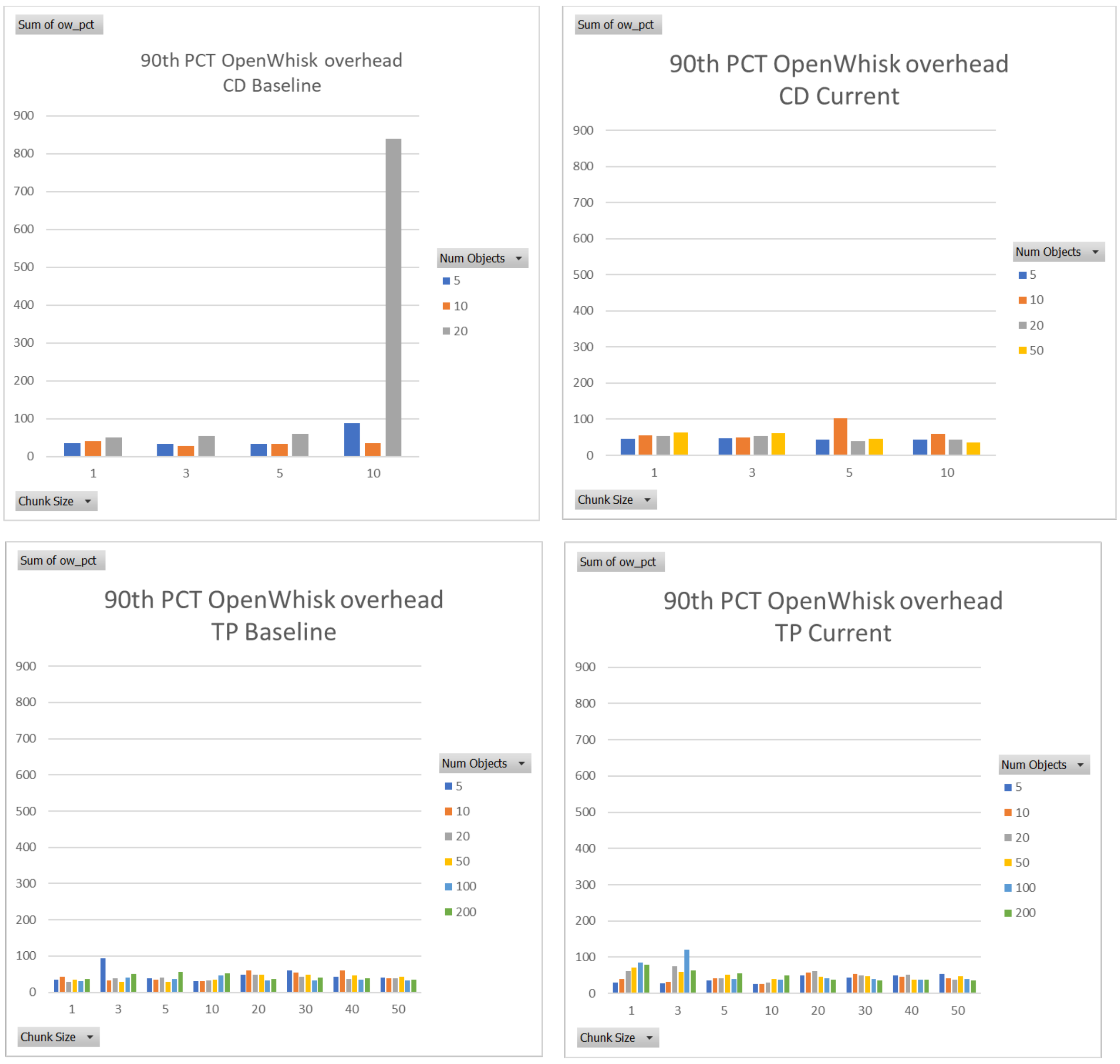


Figure 6: 90th percentile of OpenWhisk Invocation Overhead

To further emphasize the practical value of serverless when overcoming warm start, we present a second set of graphs in Figure 7, where we remove from the result dataset all the cold-start results – whether client or worker cold-start. As can be seen now, we get a practical solution for soft-real-time systems with 90th percentile of invocation overhead in all cases of invocation overhead is less than 85 msec[2]. We expect that with EXPRESS, this threshold on the same hardware resources may be significantly lower, because of a much-improved interaction of event-to-client and between client and workers (in addition to guaranteed warm-up behavior).

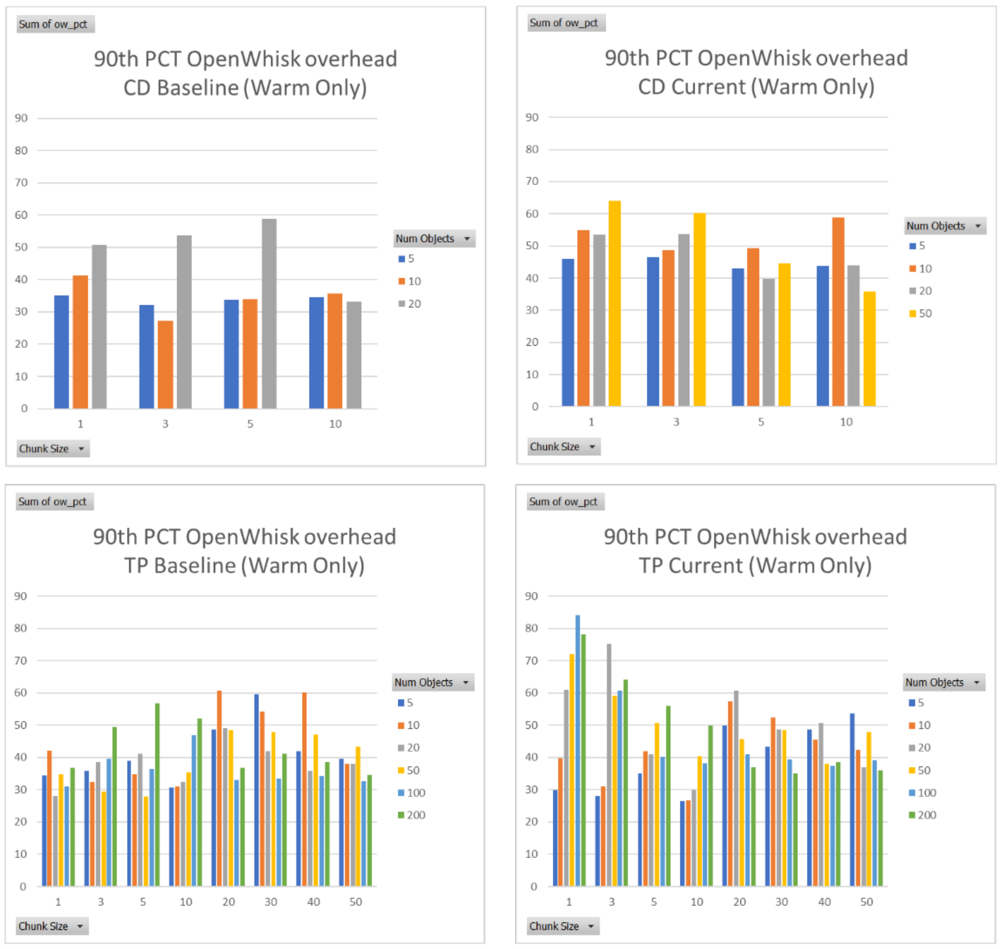


Figure 7: 90th percentile of OpenWhisk Invocation Overhead - Warm Only

4 Impact, Collaboration and Demonstrations

In this Section we describe additional impact gathered by CLASS analytics since MS3, collaboration efforts with other EU projects and bundled demonstrations.

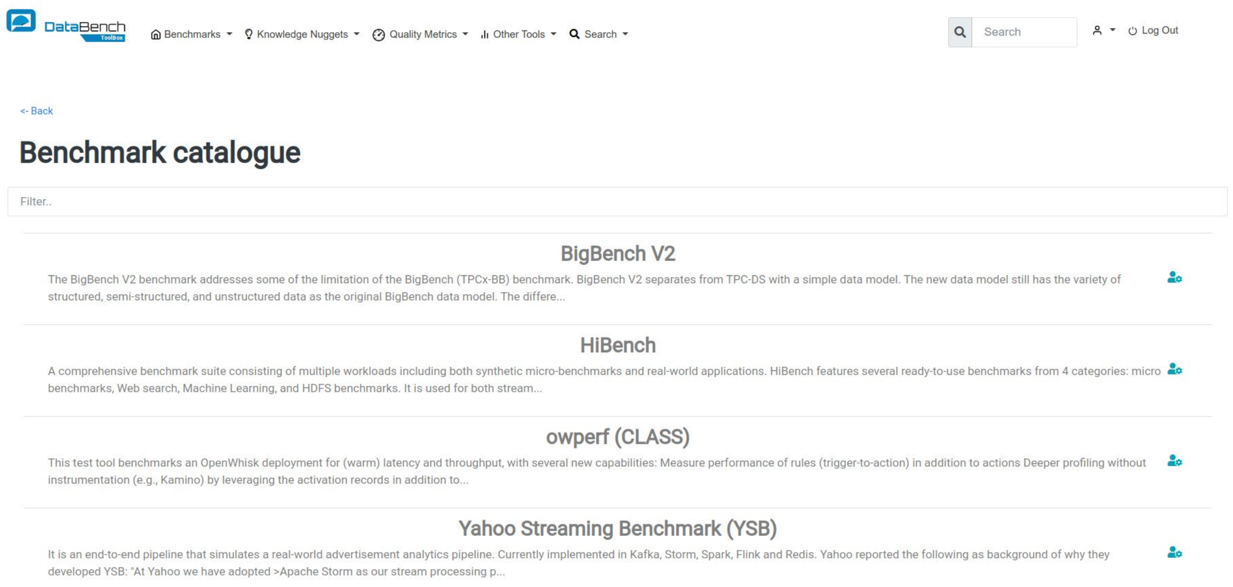
4.1 DataBench Support

The idea is to integrate a benchmark into the DataBench project [18] big data and AI catalogue based on the trajectory prediction analytic developed and using the potential of OpenWhisk [13] and owperf [19]. The DataBench catalogue lists many of the most used benchmarks

[2] As explained before, CLASS operates at the middleware level and thus does not control the entire software stack. As such, it cannot guarantee absolute bounds. Rather, it aims for practical solution for soft- or near- real-time solutions.

related to big data. The blog https://class-project.eu/news/benchmarking-real-time-serverless-applications-owperf describes deeply how to benchmark serverless functions and measure the response times. The collaboration with DataBench consisted in the automation of the deployment and the execution of the CLASS benchmark within the DataBench Toolbox [20].

Being listed in the DataBench entails the possibility of reaching out to a bigger audience, as well as bringing to the end users of the benchmark an easier and more automated way to deploy and run the benchmark. Figure 8 shows the DataBench Toolbox catalogue where the CLASS benchmark is placed.



Benchmark catalogue

BigBench V2

The BigBench V2 benchmark addresses some of the limitation of the BigBench (TPCx-BB) benchmark. BigBench V2 separates from TPC-DS with a simple data model. The new data model still has the variety of structured, semi-structured, and unstructured data as the original BigBench data model. The differe...

HiBench

A comprehensive benchmark suite consisting of multiple workloads including both synthetic micro-benchmarks and real-world applications. HiBench features several ready-to-use benchmarks from 4 categories: micro benchmarks, Web search, Machine Learning, and HDFS benchmarks. It is used for both stream...

owperf (CLASS)

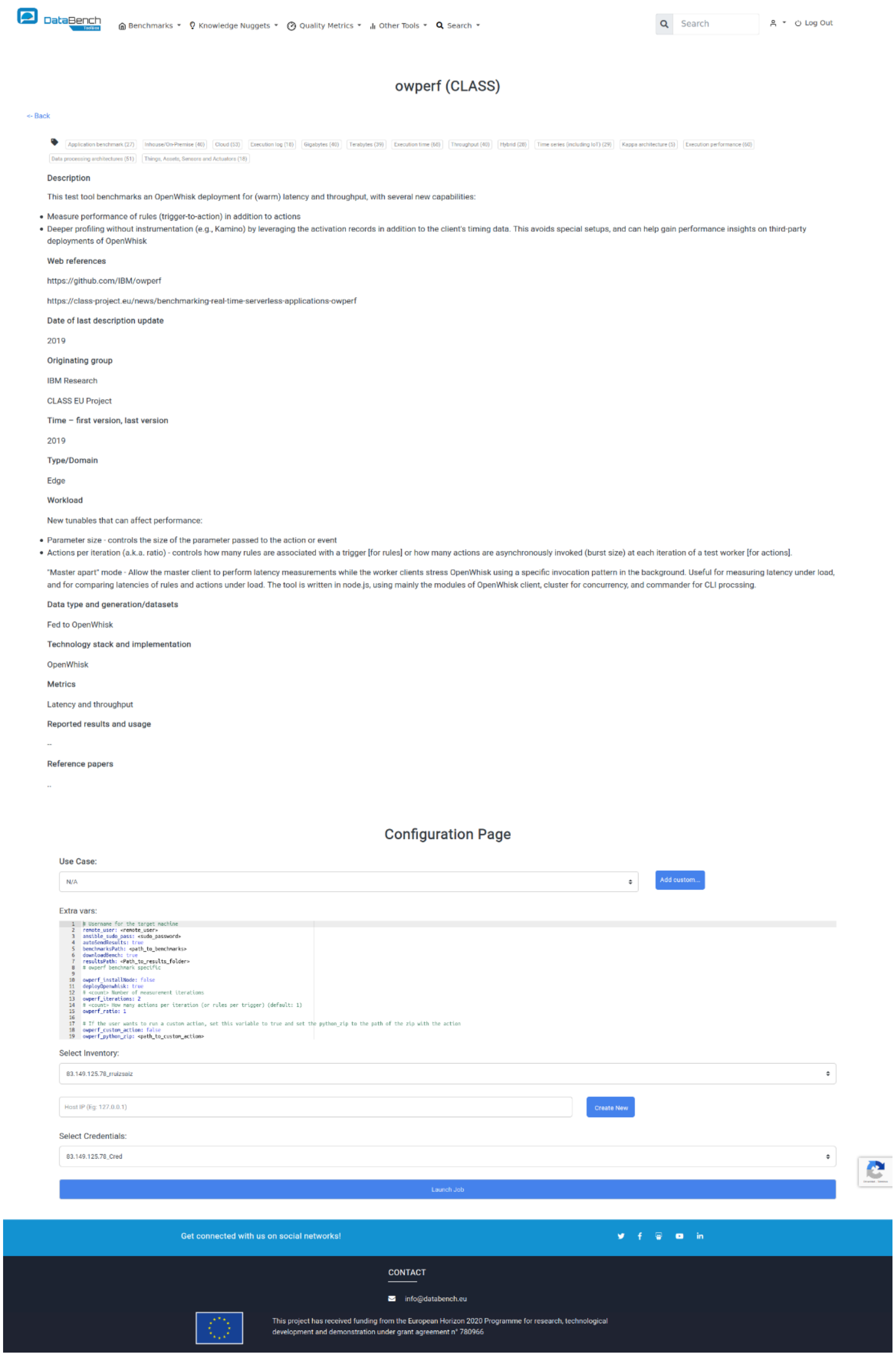
This test tool benchmarks an OpenWhisk deployment for (warm) latency and throughput, with several new capabilities: Measure performance of rules (trigger-to-action) in addition to actions Deeper profiling without instrumentation (e.g., Kamino) by leveraging the activation records in addition to...

Yahoo Streaming Benchmark (YSB)

It is an end-to-end pipeline that simulates a real-world advertisement analytics pipeline. Currently implemented in Kafka, Storm, Spark, Flink and Redis. Yahoo reported the following as background of why they developed YSB: "At Yahoo we have adopted >Apache Storm as our stream processing p...

Figure 8. DataBench benchmark catalogue

Figure 9 shows the description of the CLASS benchmark in the DataBench Toolbox, where users are able to set the configuration parameters to deploy and run the benchmark, specify where to store the results, and configure the parameters of owperf about the number of workers or number of iterations, among others.



owperf (CLASS)

<- Back

Application benchmark (27) | Inhouse/On-Premise (40) | Cloud (53) | Execution log (18) | Gigabytes (40) | Terabytes (39) | Execution time (68) | Throughput (40) | Hybrid (28) | Time series (including IoT) (29) | Kappa architecture (5) | Execution performance (60) | Data processing architectures (51) | Things, Assets, Sensors and Actuators (18)

Description

This test tool benchmarks an OpenWhisk deployment for (warm) latency and throughput, with several new capabilities:

- Measure performance of rules (trigger-to-action) in addition to actions
- Deeper profiling without instrumentation (e.g., Kamino) by leveraging the activation records in addition to the client's timing data. This avoids special setups, and can help gain performance insights on third-party deployments of OpenWhisk

Web references

https://github.com/IBM/owperf

https://class-project.eu/news/benchmarking-real-time-serverless-applications-owperf

Date of last description update

2019

Originating group

IBM Research

CLASS EU Project

Time – first version, last version

2019

Type/Domain

Edge

Workload

New tunables that can affect performance:

- Parameter size - controls the size of the parameter passed to the action or event
- Actions per iteration (a.k.a. ratio) - controls how many rules are associated with a trigger [for rules] or how many actions are asynchronously invoked (burst size) at each iteration of a test worker [for actions].

"Master apart" mode - Allow the master client to perform latency measurements while the worker clients stress OpenWhisk using a specific invocation pattern in the background. Useful for measuring latency under load, and for comparing latencies of rules and actions under load. The tool is written in node.js, using mainly the modules of OpenWhisk client, cluster for concurrency, and commander for CLI procssing.

Data type and generation/datasets

Fed to OpenWhisk

Technology stack and implementation

OpenWhisk

Metrics

Latency and throughput

Reported results and usage

--

Reference papers

--

Configuration Page

Use Case:

N/A

Add custom...

Extra vars:

```
# Username for the target machine
remote_user: <remote_user>
ansible_sudo_pass: <sudo_password>
autoSendResults: true
benchmarksPath: <path_to_benchmarks>
downloadBench: true
resultsPath: <Path_to_results_folder>
# owperf benchmark specific

owperf_installNode: false
deployOpenwhisk: true
# <count> Number of measurement iterations
owperf_iterations: 2
# <count> How many actions per iteration (or rules per trigger) (default: 1)
owperf_ratio: 1

# If the user wants to run a custom action, set this variable to true and set the python_zip to the path of the zip with the action
owperf_custom_action: false
owperf_python_zip: <path_to_custom_action>
```

Select Inventory:

83.149.125.78_rruizsaiz

Host IP (Eg: 127.0.0.1)

Create New

Select Credentials:

83.149.125.78_Cred

Launch Job

Get connected with us on social networks!

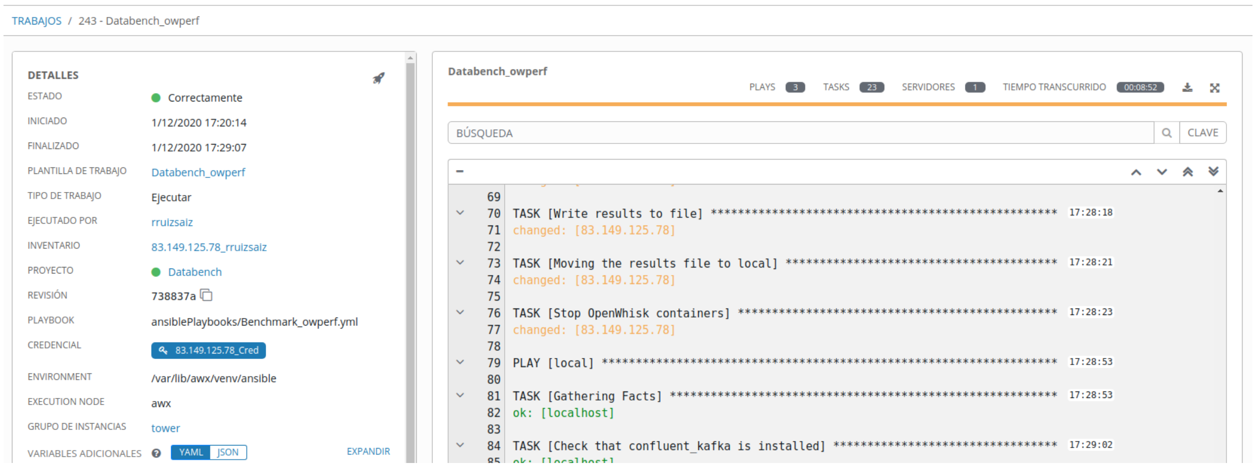
CONTACT

info@databench.eu

This project has received funding from the European Horizon 2020 Programme for research, technological development and demonstration under grant agreement n° 780966

Figure 9. CLASS benchmark description

Figure 10 and Figure 11 represent the execution phase of the owperf tool in the DataBench infrastructure, and the results related to the response times. The description of each metric is well described in the previous blog entry https://class-project.eu/news/benchmarking-real-time-serverless-applications-owperf



TRABAJOS / 243 - Databench_owperf

DETALLES

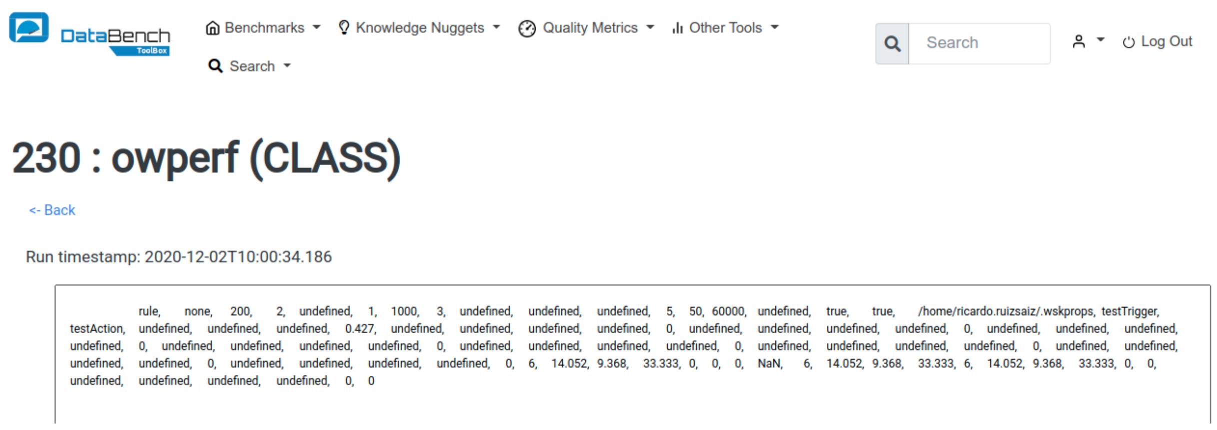
ESTADO	● Correctamente
INICIADO	1/12/2020 17:20:14
FINALIZADO	1/12/2020 17:29:07
PLANTILLA DE TRABAJO	Databench_owperf
TIPO DE TRABAJO	Ejecutar
EJECUTADO POR	rruizsaiz
INVENTARIO	83.149.125.78_rruizsaiz
PROYECTO	● Databench
REVISIÓN	738837a
PLAYBOOK	ansiblePlaybooks/Benchmark_owperf.yml
CREDENCIAL	83.149.125.78_Cred
ENVIRONMENT	/var/lib/awx/venv/ansible
EXECUTION NODE	awx
GRUPO DE INSTANCIAS	tower
VARIABLES ADICIONALES	YAML JSON

Databench_owperf

PLAYS 3 TASKS 23 SERVIDORES 1 TIEMPO TRANSCURRIDO 00:08:52

```

TASK [Write results to file] ***************************************************** 17:28:18
changed: [83.149.125.78]

TASK [Moving the results file to local] ****************************************** 17:28:21
changed: [83.149.125.78]

TASK [Stop OpenWhisk containers] ************************************************* 17:28:23
changed: [83.149.125.78]

PLAY [local] ********************************************************************* 17:28:53

TASK [Gathering Facts] *********************************************************** 17:28:53
ok: [localhost]

TASK [Check that confluent_kafka is installed] *********************************** 17:29:02
```

Figure 10. CLASS benchmark execution



230 : owperf (CLASS)

<- Back

Run timestamp: 2020-12-02T10:00:34.186

```
          rule,    none,    200,    2,  undefined,   1,   1000,   3,  undefined,   undefined,   undefined,   5,   50,  60000,  undefined,   true,    true,   /home/ricardo.ruizsaiz/.wskprops,  testTrigger,
testAction,  undefined,   undefined,   undefined,   0.427,   undefined,   undefined,   undefined,   undefined,   0,  undefined,   undefined,   undefined,   undefined,   0,  undefined,   undefined,   undefined,
undefined,   0,  undefined,   undefined,   undefined,   undefined,   0,  undefined,   undefined,   undefined,   undefined,   0,  undefined,   undefined,   undefined,   undefined,   0,  undefined,   undefined,
undefined,   undefined,   0,  undefined,   undefined,   undefined,   undefined,   0,   6,   14.052,  9.368,   33.333,  0,   0,   0,   NaN,    6,   14.052,  9.368,   33.333,  6,   14.052,  9.368,   33.333,  0,   0,
undefined,   undefined,   undefined,   undefined,   0,   0
```

Figure 11. CLASS benchmark results

4.2 EXPRESS

EXPRESS is maturing and gaining attention, so far only inside IBM, because of a prolonged patent disclosure process. However, it did accumulate additional impact:

- The patent disclosure has been filed by IBM at the USPTO, with application number 17/315422.
- At the time of writing this report, EXPRESS is included in future proposals of CONGENIAL+ and MetOS for the future DATA-01-05 call in Cluster 4 of Horizon Europe. In this call we aim to demonstrate EXPRESS in different Non-Functional Requirements for serverless applications including predictability but also exploring locality, heterogeneity and composability.
- With management agreement, EXPRESS is going to pilot integration with an existing IBM Cloud offering. If the prototype convinces the IBM Cloud division, it might become an official part of the offering.
- EXPRESS is planned to become an open-source project on its own.

Now that the EXPRESS IP is filed, we expect to also start publishing about it in papers and blogs.

4.3 Lithops

In CLASS, we have taken Lithops, a map/reduce on serverless engine designed for massive parallelism, and transformed it from a high-overhead execution designated for long per-element computation into another big-data use-case that consists of a large amount of short computations and focuses on small overhead and low overall latency. All the changes documented in D5.4 [4] and in Section 2.1.3 are planned to undergo a review in IBM and if approved be contributed back to the Lithops open-source project [7]. Some of the changes have already passed the process and got contributed, such as persisting the map function in the container image.

We conclude the work on Lithops with a demonstration of the TP (Trajectory Prediction) application invoked as a Lithops-based OpenWhisk function, with the new chunk size feature enabled and specified. The demonstration video is available at the CLASS intranet:

https://class-project.eu/user/login

A dedicated user has been created for demonstration purposes, with limited access to deliverables and related videos. The credentials to access this service are the following:

Username: **EC_user**

Password: **@Hz.52qXXF#K23**

After logging in, click on “Intranet”, the demonstration videos and files of this deliverable are located in “PU_D5-5Report” directory.

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 [13] (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 components shown in Figure 12.

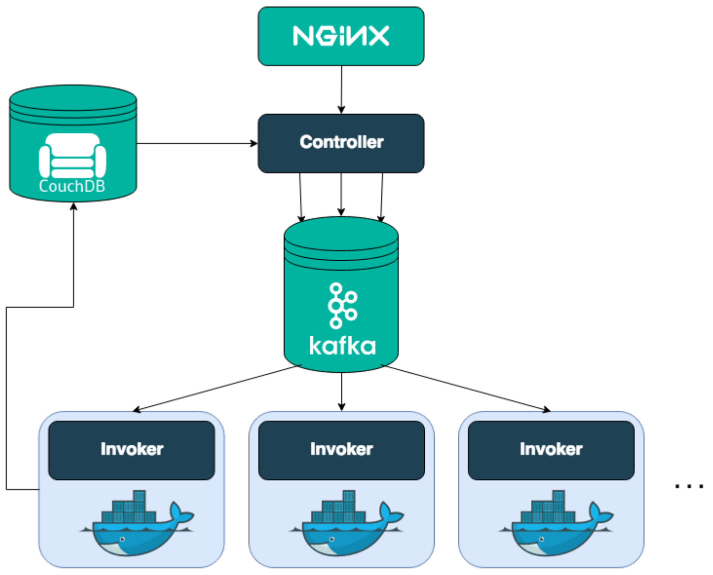


Figure 12: OpenWhisk Architecture.

The components of the OW architecture are introduced as follows:

- **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 [21].

5.2 Lithops

Lithops [7] is a rebranding of the former project of PyWren [22], which 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 [23] to use IBM's Cloud Functions, based on Apache OpenWhisk.

Lithops'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 are 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 Lithops 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 Lithops is as following [23]:

- `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 results from one computation to the next, ending with a single final result. Figure 13 demonstrates the operation of a `map` computation in PyWren/Lithops, involving the client, IBM Cloud Functions (OpenWhisk) and object storage, in an IBM Cloud setup.

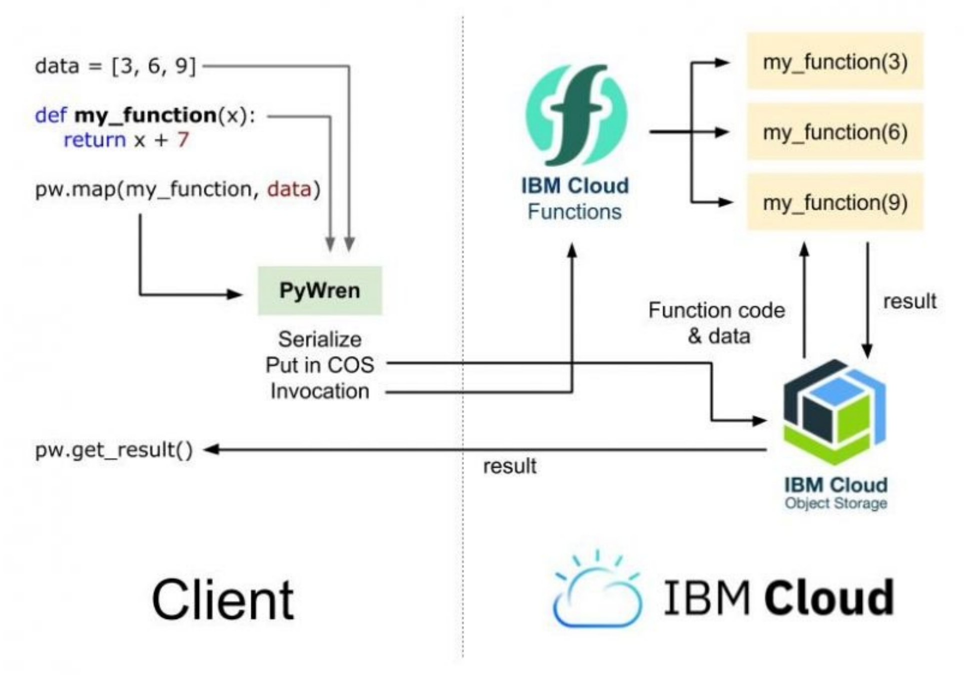


Figure 13: Execution of a PyWren map operation.

5.3 COMPSs

Deliverable D2.1 [24] 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,* 2018.

[3] E. Hadad, “D5.3 Initial Release of CLASS Big-Data Analytics Layer,” CLASS, 2019.

[4] E. Hadad, “D5.4 Final Release of CLASS Big-Data Analytics Layer,” CLASS, 2020.

[5] CLASS Consortium, “D1.6 CLASS Use Case Evaluation,” 2021.

[6] E. Kartsakli, “D2.7 Final Release of the CLASS software architecture,” CLASS Consoirtium, 2021.

[7] Lithops Community, “Lithops - Lightweight Optimized Processing,” [Online]. Available: https://github.com/lithops-cloud/lithops. [Accessed June 2020].

[8] D. Amendola, “D1.4 - Final Release Of The Smart City Use Case,” CLASS, 2020.

[9] B. Szyk, “Stopping Distance Calculator,” [Online]. Available: https://www.omnicalculator.com/physics/stopping-distance. [Accessed May 2021].

[10] CloudButton Consortium, “CloudButton - Serverless Data Analytics Platform,” [Online]. Available: https://cloudbutton.eu/. [Accessed May 2021].

[11] MinIO, Inc., “MinIO - Object Storage for AI,” [Online]. Available: https://min.io/. [Accessed 3 2019].

[12] VMware, inc, “RabbitMQ - Messaging That Just Works,” [Online]. Available: https://www.rabbitmq.com/. [Accessed May 2021].

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

[14] Google inc., “Protocol Buffers | Google Developers,” [Online]. Available: https://developers.google.com/protocol-buffers. [Accessed May 2021].

[15] E. Hadad, “D3.2 - First Release of Edge Analytics Platform Agent,” CLASS, 2019.

[16] E. Hadad, “D3.3 - Final Release Of CLASS Analytics Platform Agent,” CLASS, 2020.

[17] J. Beswick, “New for AWS Lambda – Predictable start-up times with Provisioned Concurrency,” 4 December 2019. [Online]. Available: https://aws.amazon.com/blogs/compute/new-for-aws-lambda-predictable-start-up-times-with-provisioned-concurrency/. [Accessed 2 2020].

[18] DataBench consortium, “DataBench - Big Data Benchmarking,” [Online]. Available: https://www.databench.eu/. [Accessed May 2021].

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

[20] DataBench Consortium, "DataBench Toolbox," [Online]. Available: https://databench.ijs.si/. [Accessed March 2021].

[21] 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].

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

[23] 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].

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

[25] "NVIDIA TensorRT | NVIDIA Developer," [Online]. Available: https://developer.nvidia.com/tensorrt.

[26] M. Corredoira, *D1.2 First release of the smart city use cases,* 2019.

[27] J. Redmon, "YOLO: Real-Time Object Detection," [Online]. Available: https://pjreddie.com/darknet/yolo/.

[28] Barcelona Supercomputing Center, "Application development guide. COMPSs user manual," 9 November 2018. [Online]. Available: http://compss.bsc.es/releases/compss/latest/docs/COMPSs_User_Manual_App_Development.pdf.

[29] Barcelona Supercomputing Center, "Application execution guide. COMPSs User Manual," 9 November 2018. [Online]. Available: http://compss.bsc.es/releases/compss/latest/docs/COMPSs_User_Manual_App_Exec.pdf.

[30] Linux Foundation, "Production-Grade Container Orchestration - Kubernetes," [Online]. Available: https://kubernetes.io. [Accessed 3 2019].

[31] Docker Inc, "Enterprise Application Container Platform - Docker," [Online]. Available: https://www.docker.com/. [Accessed 3 2019].

[32] Barcelona Supercomputing Center, "COMP Superscalar - BSC-CNS," [Online]. Available: https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar/. [Accessed 3 2019].

[33] Barcelona Supercomputing Center, "dataClay - BSC-CNS," [Online]. Available: https://www.bsc.es/research-and-development/software-and-apps/software-list/dataclay. [Accessed 3 2019].

[34] E. Quinones, “D2.4 First Release Of The CLASS Software Architecture,” 2019.

[35] M. Corredoira, “D1.2 Final Release Of The Smart City Use Cases,” 2019.

[36] D. Breitgand, “Lean OpenWhisk: Open Source FaaS for Edge Computing,” 7 2018. [Online]. Available: https://medium.com/openwhisk/lean-openwhisk-open-source-faas-for-edge-computing-fb823c6bbb9b. [Accessed 3 2019].

[37] IBM, “OpenWhisk REST API,” [Online]. Available: https://openwhisk.ng.bluemix.net/api/v1/docs/index.html?url=/api/v1/api-docs#/. [Accessed 3 2019].

[38] OpenWhisk Community, “[OpenWhisk] Documenation,” [Online]. Available: https://openwhisk.apache.org/documentation.html. [Accessed 3 2019].

[39] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes and J. Labarta, “PyCOMPSs: Parallel computational workflows in Python,” *IJHPCA,* vol. 31, no. 1, pp. 66-82, 2017.

[40] OpenWhisk Community, “Creating triggers and rules,” [Online]. Available: https://github.com/apache/incubator-openwhisk/blob/master/docs/triggers_rules.md. [Accessed 3 2019].

[41] Wikipedia, “Network Time Protocol,” [Online]. Available: https://en.wikipedia.org/wiki/Network_Time_Protocol. [Accessed 3 2019].

[42] R. Cavicchioli, “D3.4 First release of the real-time analysis methods and tools on the edge,” 2019.

[43] OpenWhisk Community, “Setting up OpenWhisk on Ubuntu server(s),” [Online]. Available: https://github.com/apache/incubator-openwhisk/blob/master/tools/ubuntu-setup/README.md. [Accessed 3 2019].

[44] MASERATI, BSC, “D1.2 - Final release of the smart city use cases”.

[45] ATOS, “D4.1. - Cloud requirement specification and definition,” 2018.

[46] OpenWhisk Community, “Apache OpenWhisk package that can be used to create periodic, time-based alarms,” [Online]. Available: https://github.com/apache/incubator-openwhisk-package-alarms. [Accessed 3 2019].

[47] E. Hadad, “erezh16/express: A serverless platform with predictability enhancements - based on OpenWhisk,” IBM Research, [Online]. Available: https://github.com/erezh16/express. [Accessed 3 2019].

[48] IBM Corp., “IBM Cloud Object Storage - Overview,” [Online]. Available: https://www.ibm.com/cloud/object-storage. [Accessed 3 2019].

[49] N. Mammadli and J. Álvarez , “D2.5 – Final release of Spark and COMPSs integrated in CLASS architecture,” CLASS, 2020.

[50] M. Shilkov, “Cold Starts in AWS Lambda,” 26 September 2019. [Online]. Available: https://mikhail.io/serverless/coldstarts/aws/.

[51] H2 Database, "H2 Performance Comparison," [Online]. Available: http://www.h2database.com/html/performance.html. [Accessed 2 2020].

[52] Knative Community, "Knative," [Online]. Available: https://knative.dev/. [Accessed March 2020].

[53] Amazon Web Services, "AWS Lambda Execution Context," [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html. [Accessed 2 2020].

[54] M. Groves, "Lazy Initializing Within Azure Functions," 15 September 2017. [Online]. Available: https://blog.couchbase.com/azure-functions-lazy-initialization-couchbase-server/.

[55] J. Daly, "Lambda Warmer: Optimize AWS Lambda Function Cold Starts," 14 July 2018. [Online]. Available: https://www.jeremydaly.com/lambda-warmer-optimize-aws-lambda-function-cold-starts/.

[56] iguazio, inc, "Nuclio: Serverless Platform for Automated Data Science," [Online]. Available: https://nuclio.io/. [Accessed 2 2020].

[57] A. Singhvi, K. Houck, A. Balasubramanian, M. D. Shaikh, S. Venkataraman and A. Akella, "Archipelago: A Scalable Low-Latency Serverless Platform," 22 November 2019. [Online]. Available: https://arxiv.org/pdf/1911.09849.pdf.

[58] C. W. X. C. L. J. S.-S. J. M. F. J. E. G. J. M. H. A. T. Vikram Sreekanti, "Cloudburst: Stateful Functions-as-a-Service," 27 January 2020. [Online]. Available: https://arxiv.org/pdf/2001.04592.pdf.

[59] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya and V. Hilt, "SAND: Towards High-Performance Serverless Computin," in *USENIX Annual Technical Conference*, Boston, MA, 2018.

[60] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak and V. Sukhomlinov, "Agile Cold Starts for Scalable Serverless," in *HotCloud'19*, Renton, WA, 2019.