Big Data Platforms

Stragglers and serverless computing

Oren Ben Eliyahu

Yuval Barkan

February 16th, 2022

# Table of Contents

[1. Abstract: 3](#_Toc94638114)

[1.1. Sub-title 3](#_Toc94638115)

[1.1.1. Sub-sub title 3](#_Toc94638116)

[2. Motivation and background: 3](#_Toc94638117)

[3. Stragglers in serverless computing: 3](#_Toc94638118)

[4. Stragglers 3](#_Toc94638119)

# Abstract:

# Motivation and background:

**רקע כללי על סטרגלר, קשיים שעלו בדרך (בדגש על משימות סטרגלר) דוגמאות, והשוני בין ההתמודדות של מנועי ביג דאטא עם משימות סטרגלר.**

# Stragglers in serverless computing:

להסביר מה זה חישוב ללא שרת, הבעיות של סטרגלר בשילוב עם סרברלס, להציג נקודות דמיון ושוני בנוגע להבדל בין סטרגלר בסרברלס למנועי ביג דאטא.

# Stragglers in jobs submitted by Lithops

* Explore and explain how Lithops submit it’s job and knows which tasks are finished and which tasks yet running.

Lithops is a Python multi-cloud agnostic serverless computing framework that can run Local Python code can be run at a massive scale on serverless computing platforms without modification. Using Lithops, the user does not have to worry about managing boilerplate code or deploying and running it. Lithops provides a simple and intuitive interface for handling data volumes by abstracting the underlying APIs for cloud storage[[1]](#footnote-1).

Map and reduce computations are each executed as separate jobs in Lithops.

There is an orchestrator component responsible for orchestrating all the computations in Lithops. It is instantiated prior to any use of Lithops. Its initialization includes these essential steps:

1. Set up the workers[[2]](#footnote-2) by constructing docker images and defining serverless functions.
2. Define a bucket in object storage (pre-configured) in which each job will store the job and input data prior to computation and results when computation is complete.
3. Create an Invoker object and execute each job as independent worker tasks

In map job, a partitioner object first partitions the data.

Each worker should be able to process each partition independently.

In reduce jobs, Lithops provide two methods of exectution:

1. Reduce per object, where each object is processed by a reduce function, and the data is partitioned as one partition per storage object.
2. Global reduce, where a single reduce function processes all data. Additionally, the reduce function is forced to wait for data before invocation since the output of the map jobs needs to be finished before the reduce job can run.

Both functions also construct jobs, described as follows:

1. A job description is defined for the job
2. The partition map and the data processing function are serialized.
3. The serialized partition map is stored in the object storage bucket.
4. The serialized processing function and its module dependencies are stored in the same bucket

Lithops proceeds to execute the job by the following steps:

1. The job is executed as independent and concurrent invocations from the beginning.
2. An internal invoker creates a dictionary with all the data the invocation needs. It includes a copy of some of the job descriptions as well as some essential metadata regarding its job (e.g., call\_id, data length, log location output)
3. Invocation proceeds to the computation part with a retry mechanism for the current call. that depends on the configured backend computation API
4. When computation part completes, each invocation commits the result to object storage and returns a response of the computed result, which is used to wait for job completion and retrieve all results.

Explain how Lithops know when entire job is finished and how it waits for the running tasks to complete.

Eventually, Lithops detects job completion in one of two following configurable techniques:

1. RabbitMQ[[3]](#footnote-3): A unique RabbitMQ topic is defined for each job, combining the executor id and job id. Once each worker completes its invocation, it posts a notification message on that topic, the orchestrator consumes a number of messages on that topic and compare it with the expected total calls that determine completion of the job.
2. Object Storage Polling: each invocation persists its computation results in a specific object. The orchestrator is repeatedly, once per fixed period, polls the executor’s bucket for status objects of a subset of invocations that have still not completed. This allows control of resource usage and eventual detection of all calls.

* Provide two different approaches to extend Lithops with mechanism to prevent jobs to be affected by straggler tasks.
* Address different aspects of complexity of implementation, costs, performs.
* Explain positive and negative effects of the approaches you propose.

# Prototype

* Provide pseudo-code for the approaches you propose. Explain how it works.

# Next steps

* Suggest next steps to the solutions you proposed

# Conclusion

* Short conclusion of the work you did

# Bibliography

1. <https://lithops-cloud.github.io/docs/index.html> [↑](#footnote-ref-1)
2. A worker performs one unit of computation (e.g., processing one dataset chunk or one object) within a more extensive job of Lithops [↑](#footnote-ref-2)
3. Open-source message-broker software initially implemented the Advanced Message Queuing Protocol and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol, MQ Telemetry Transport, and other protocols. [↑](#footnote-ref-3)