# An Extensible Scheduler for the OpenLambda FaaS Platform

Gustavo Totoy
Escuela Superior Politécnica del Litoral, ESPOL
Guayaquil, Ecuador
gtotoy@fiec.espol.edu.ec

Edwin F. Boza
Escuela Superior Politécnica del Litoral, ESPOL
Guayaquil, Ecuador
eboza@fiec.espol.edu.ec

Cristina L. Abad
Escuela Superior Politécnica del Litoral, ESPOL
Guayaquil, Ecuador
cabad@fiec.espol.edu.ec

## ABSTRACT

In a distributed computing platform, co-locating tasks at worker nodes that store or cache any required files is a time-proven mechanism to reduce task latency. While this problem has been extensively studied in the Web and Big Data processing domains, it is only recently gaining attention in the serverless computing domain. One proposed optimization for Function-as-a-Service (FaaS) clouds is to cache required packages at the worker nodes instead of bundling them with the cloud functions, thus significantly reducing the function launch time. However, existing FaaS schedulers are vanilla load balancers that do not attempt to minimize the movement of packages or files across the network. As researchers start tackling the problem of *package-aware scheduling* and other near-data scheduling optimizations for FaaS platforms, having a common framework on top of which to implement and evaluate their ideas would be beneficial, as this would encourage fair comparisons between different solutions and facilitate experiment reproducibility. To address this problem, we present a simple and extensible function scheduler for the OpenLambda FaaS platform. Our scheduler is implemented in Go, and is simpler to modify and extend than the nginx load balancer used by the original OpenLambda. To illustrate the usefulness of our scheduler, we added a package-aware scheduling algorithm to it. We have released our code so that others can easily implement new scheduling algorithms for OpenLambda.

**ACM Reference Format:**
Gustavo Totoy, Edwin F. Boza, and Cristina L. Abad. 2018. An Extensible Scheduler for the OpenLambda FaaS Platform. In *Proceedings of Workshop on Hardware/Software Techniques for Minimizing Data Movement (Min-Move'18).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

*Functions-as-a-Service* (FaaS) cloud platforms enable tenants to deploy and execute functions on the cloud, without having to worry about server provisioning. In a FaaS platform, *cloud functions* are (typically) small, stateless tasks, with a single functional responsibility, and are triggered by events. The FaaS cloud provider manages the infrastructure and other operational concerns, enabling developers to easily deploy, monitor, and invoke cloud functions [16]. These functions can be executed on any of a pool of servers managed by the provider and potentially shared between the tenants.

One proposed optimization for FaaS platforms is to cache required packages or libraries at the worker nodes instead of bundling them with the functions, thus making the functions more lean and as a result, significantly reducing their launch time [2, 11]. However, existing FaaS schedulers are vanilla load balancers that do not attempt to maximize package-locality when assigning functions to workers. For this reason, the development of new scheduling algorithms for FaaS platforms is considered an important challenge in the serverless computing domain [2, 6, 15, 16].

As researchers start tackling the problem of *package-aware scheduling* and other near-data scheduling optimizations for FaaS platforms, having a common platform on top of which to implement and evaluate their ideas would be beneficial, as this would encourage fair comparisons between different solutions, facilitate experiment reproducibility and reduce development time.

Towards this goal, we have implemented `olscheduler`, a simple and extensible function scheduler for the OpenLambda FaaS platform. Our scheduler has 331 lines of Go code, and is simpler to modify and extend than the ngninx load balancer used by the original OpenLambda. `olscheduler` comes with three scheduling policies (random, round-robin and least-loaded), and exposes useful information about the platform to the system developer, so that additional scheduling policies can be easily implemented.

To illustrate the usefulness of `olscheduler`, we added a simple package-aware scheduling algorithm that seeks to improve package-affinity while avoiding unmanageable worker overload. We were able to implement this algorithm with only 46 additional lines of code (LOCs). Preliminary simulation results show that this simple approach can cut the function latency by more than 65%. In the future, we will validate our results in real cloud experiments.

We have released our code so that others can easily implement new scheduling algorithms for OpenLambda[1].

## 2 BACKGROUND

A Function-as-a-Service (FaaS) cloud platform supports the creation of distributed applications composed by a number of small, single-task, cloud functions. These functions run in lightweight sandbox environments, which run on top of virtual machines. The sandboxes, runtime environments, and virtual machines are managed by the cloud platform. Thus, a developer can create elastic cloud applications without having to worry about server provisioning and elasticity managers. Examples of FaaS platforms include OpenLambda[2], Fission[3], OpenWhisk[4], AWS Lambda[5] and Azure Functions[6].

[1]https://github.com/gtotoy/olscheduler
[2]open-lambda.org
[3]fission.io
[4]openwhisk.apache.org
[5]aws.amazon.com/lambda
[6]azure.microsoft.com/en-us/services/functions

*Min-Move'18, March 2018, Williamsburg, VA*
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

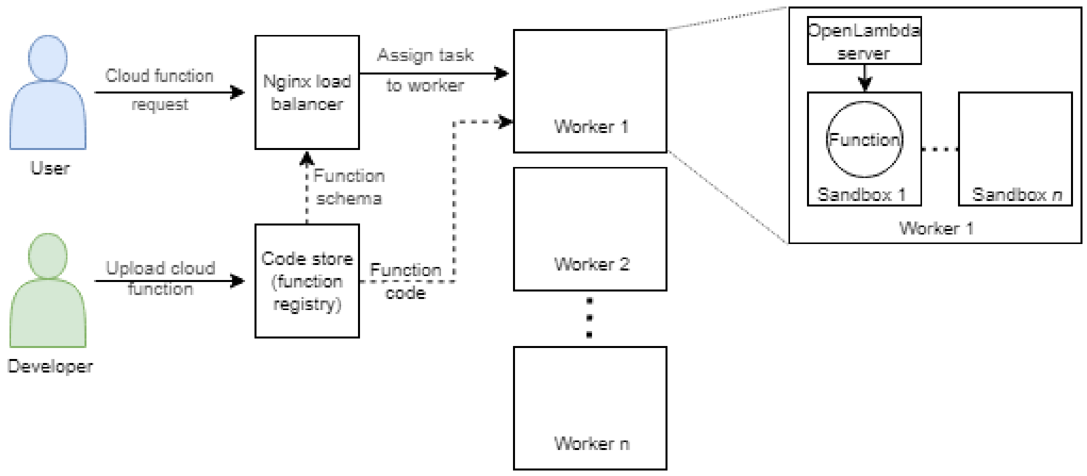

**Figure 1: The OpenLambda architecture [6].**

## 2.1 OpenLambda

OpenLambda is serverless computing platform that supports the FaaS execution model [6, 7]. Figure 1 shows the OpenLambda architecture. In OpenLambda, a developer must upload the code of her cloud functions to the *code store* or registry. When a user triggers the execution of cloud functions, a request is sent to the *load balancer*, which selects a *worker* based on the configured load balancing mechanism. When a worker receives a request, it runs the cloud function in a *sandbox*, currently implemented with Docker containers. The first time a function runs on a worker, the worker has to contact the code store to get the code of the function; the code is cached so that this step is not needed in future invocations.

## 2.2 Function scheduling in OpenLambda

In OpenLambda, the function scheduling (or task of assigning cloud functions to workers) is performed by nginx, with its role of software load balancer. The load balancing methods currently supported by ngninx are [1]:

- Round-robin: requests are assigned to servers in a round-robin fashion.
- Least-connected: an incoming request is assigned to the server with the least number of active connections.
- IP-hash: uses a hash-function map all requests coming from the same IP address to the same server.

These load balancing methods distribute the load between the workers, but lack functionality to make intelligent decisions that seek to, for example, minimize data transfers between workers or between a worker and an external repository (e.g., a distributed file system or a repository of packages required by the cloud functions).

## 2.3 Package caching with Pipsqueak

Oakes et al. [11] recently proposed Pipsqueak, a shared package cache available at each OpenLambda worker. Pipsqueak seeks to reduce the start-up time of cloud functions via supporting lean functions whose required packages are cached at the worker nodes (see Figure 2). The cache maintains a set of Python interpreters with packages pre-imported, in a sleeping state. When a cloud function is assigned to a worker node, it checks if the required packages are cached. To use a cached entry, Pipsqueak: (1) Wakes up and forks the corresponding sleeping Python interpreter from the cache, (2) relocates its child process into the handler container, and (3) handles the request. If a cloud function requires two packages that are cached in different sleeping interpreters, then only one can be used and the missing package must be loaded into the child of that container (created by step 2 above). To deal with cloud functions with multiple package dependencies, Pipsqueak supports a tree cache in which one entry can cache package A, another entry can cache package B, and a child of either of these entries can cache both A *and* B.

Having pre-initialized packages in sleeping containers speeds up function start-up time because this eliminates the following steps present in an unoptimized implementation: (1) downloading the package, (2) installing the package, and (3) importing the package. The last step also includes the time to initialize the module and its dependencies. Especially for cloud functions with large libraries,

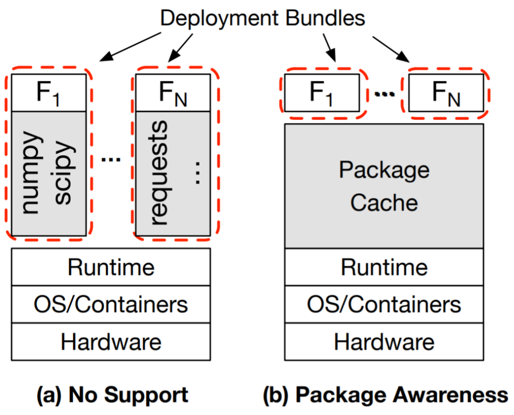

**Figure 2: The Pipsqueak package cache in OpenLambda. Figure reproduced from [11].**

this process can be extremely time consuming, as it can take $4.007s$ on average and as much as $12.8s$ for a large library like Pandas [12].

## 3 DESIGN AND IMPLEMENTATION

We decided to implement `olscheduler` using Go, as its primitives lets the developer easily build high-performant distributed systems.

To facilitate the implementation of future schedulers, `olscheduler` exposes functions and data structures that can be used to get the following information:

(1) Workers: Number and references to the workers.
(2) Per-worker load: Measured as the number of active requests that each worker is currently handling.
(3) Required packages: The list of required packages, sorted by size, is exposed for each function call.
(4) Function schemas: Obtained by querying the code repository and cached in memory.

To get the number of required packages (item 3 above), we extended the HTTP Post request (cloud function request in Figure 1) so that it supports receiving the list as an annotation on the function call. An alternative design would have been to query the cloud store to get this information; we rejected this idea to avoid an extra step on the critical path of the function requests.

`olscheduler` currently supports the following scheduling algorithms:

- Round-robin: Distributes the requests uniformly between the workers, in round robin fashion.
- Least-loaded: An incoming request is assigned to the worker that currently has the least number of active requests.
- Random: Distributes requests randomly between the workers, according to user defined worker weights.

The functionality described above was implemented in 331 lines of code (LOCs). In addition, we have implemented an additional, package-aware scheduling policy, as described next.

### 3.1 Package-aware scheduling in olscheduler

To illustrate that `olscheduler` can be easily extended to implement a package-aware algorithm, we implemented a variant of an algorithm we proposed in prior work [2], adapting it so that it is suitable for the OpenLambda scheduler model: a push-based, centralized, scheduler with a processor-sharing service discipline (workers).

The algorithm we implemented seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold. The scheduler uses hashing to try to assign all tasks that require the same package to the same worker, to encourage cache affinity. To avoid overloading a worker to which one or more popular packages map, a configurable maximum load threshold is used. If the scheduler cannot achieve affinity without assigning a task to an overloaded node (defined as one for which its number of active requests has exceeded a threshold, $t$), then the scheduler chooses the worker with the least load. To improve cache affinity while improving load balance, we apply the power-of-2 choices technique [10], by using two hashing functions to map a task to a worker; each hash function maps the task to a different worker, and the task is assigned to the least loaded one. Algorithm 1 shows our package-aware scheduling algorithm for OpenLambda.

**Algorithm 1:** Package-aware scheduler algorithm for OpenLambda

```
Global data: List of workers, $W = w_1, ..., w_n$, Hash functions
             $H_1$ and $H_2$, maximum load threshold, $t$
Input: Function, $f$, list of required packages sorted by
       descending package size, $P = p_1, ...p_n$
1  if ($P$ is not empty) then
       /* Greedily seek affinity w/ large package */
2      for ($l = 1, \ldots, |P|$) do
           /* Calculate two possible worker targets */
3          $t1 = H_1(p_l)\%|W| + 1$
4          $t2 = H_2(p_l)\%|W| + 1$
           /* Select target with least load */
5          if ($load(w_{t1}) < load(w_{t2})$) then
6              $A := t1$
7          else
8              $A := t2$
           /* If target is not overloaded, we are done */
9          if ($load(w_A) < t$) then
10             Assign $f$ to $w_A$
11             return
   /* Balance load */
12 Assign $f$ to least loaded worker, $w_i$
```

With the information exposed by `olscheduler`, implementing this policy was straightforward, and took only 46 LOCs.

### 3.2 Expected Results

We have performed validation tests to make sure our scheduler works according to our design, and are working on doing a thorough evaluation in a public cloud. In order to do so, we first need a suitable benchmark or workload; as no such benchmark exists, we are in the process of compiling different cloud functions to be used in our evaluation.

In the meantime, we implemented both the least-loaded and the proposed package-aware algorithm in a simulator, with the following configuration parameters:

- Mean inter-arrival time = $0.1ms$ (exponentially distributed).
- 1 000 worker nodes; each can run up to 100 functions simultaneously ($st = 100$).
- Distribution of packages popularity: Zipfian ($s = 1.1$).
- Time to start the packages is exponentially distributed (average time to start = $4.007s$).
- Number of packages required by a function is exponentially distributed (mean required packages = 3).
- Each worker has a LRU package cache (capacity = $500MB$).
- The sizes of the (cacheable) packages is modeled after the sizes of the packages in the PyPi repository.
- Time to launch a function that requires no packages: $1s$.

- The running time of a task (after loading required packages) is exponentially distributed with mean = $100ms$.
- Experiment duration: 30 minutes.
- Overload threshold: $t = 200$.

While the configuration described above represents an artificial scenario, the configuration values were chosen to closely model real observed behavior, as reported by related work [6, 8, 11].

Our preliminary **results** show we can improve the median **hit rate** from 51.15% (least-loaded) to 63.52% (Proposed). This has a direct impact on latency, as shown in Table 1: median latency improves by 65.8% (Proposed vs. least-loaded), and tail latency improves by 41.9% (90$th$ percentile). If we compare against a least-loaded load balancer in an unoptimized platform that does not cache function packages, our algorithm improves median latency by 189.9 times.

**Table 1: Task latency percentiles (in seconds). Our algorithm improves latency due to improved cache hit rate.**

| Algorithm | 50$th$ | 90$th$ | 95$th$ | 99$th$ |
|---|---|---|---|---|
| Least-loaded, no pkg cache | 256.42 | 455.76 | 480.71 | 503.72 |
| Least-loaded | 3.95 | 18.53 | 25.29 | 40.86 |
| Proposed | 1.35 | 10.76 | 15.90 | 28.98 |

## 4 RELATED WORK

We build upon prior work in load balancing for server clusters [5, 9, 10]. Most of this work is specific to the Web server farms, though these balancing algorithms are easy to extend and apply to FaaS architectures. In the FaaS domain, industry schedulers—like the generic nginx and the custom function schedulers for OpenWhish, Fission–are vanilla load balancers that do not seek to minimize data movement in the system.

Another line of research is work in task scheduling while trying to improve data-locality, for example, for the Web [3, 4, 13] and Big Data processing [17–20] domains.

The near-data scheduling problem arises in frameworks like Hadoop, where each type of task has different processing rates on different subsets of servers [18]; tasks that process data that is stored locally execute the fastest, followed by tasks whose input data is stored in the same rack, followed by tasks whose input data is stored remotely (in a different rack). Several near-data schedulers have been proposed for Hadoop [17, 19, 20]. One thing that has helped the community propose new schedulers for Hadoop—some of which have later made it to the Hadoop codebase, like [20]—is the fact that Hadoop's design makes it easy to replace the scheduler with a new algorithm.

Finally, our work joins recent efforts by other researchers in seeking to advance the state-of-the-art in the management of resources in serverless computing clouds [11, 14, 15].

## 5 CONCLUSIONS AND FUTURE WORK

Function-as-a-Service platforms—like OpenWhisk, OpenLambda and Fission—could benefit significantly from intelligent schedulers that seek to minimize data transfers, as this can be an expensive step (e.g., reading input data from a distributed file system, or downloading and installing packages from a package repository). In this work, we presented a simple yet extensible function scheduler for OpenLambda, which can be used as a basis for future research in smart scheduling for FaaS platforms. In the future we will use our scheduler to evaluate different package-aware scheduling algorithms.

## REFERENCES

[1] Using nginx as HTTP load balancer. Available at: http://nginx.org/en/docs/http/load_balancing.html.

[2] Abad, C. L., Boza, E. F., and van Eyk, E. Package-aware scheduling of FaaS functions. In *HotCloudPerf workshop, co-located with ACM/SPEC Intl. Conf. Perf. Eng. (ICPE)* (2018).

[3] Cardellini, V., Casalicchio, E., Colajanni, M., and Yu, P. The state of the art in locally distributed Web-server systems. *ACM Comput. Surv. 34*, 2 (2002).

[4] Cherkasova, L., and Ponnekanti, S. Optimizing a content-aware load balancing strategy for shared web hosting service. In *Intl. Symp. Model., Anal. and Sim. of Comp. and Telecomm. Sys. (MASCOTS)* (2000).

[5] Gupta, V., Harchol Balter, M., Sigman, K., and Whitt, W. Analysis of Join-the-Shortest-Queue Routing for Web server farms. *Perform. Eval. 64* (2007).

[6] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. Serverless computation with OpenLambda. In *USENIX Work. Hot Topics in Cloud Comp. (HotCloud)* (2016).

[7] Hendrickson, S., Sturdevant, S., Oakes, E., Harter, T., Venkataramani, V., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. Serverless computation with OpenLambda. *Usenix ;login: 41*.

[8] Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., and Pallickara, S. Serverless computing: An investigation of factors influencing microservice performance. In *IEEE Intl. Conf. Cloud Eng. (ICPE), to appear* (2018).

[9] Lu, Y., Xie, Q., Kliot, G., Geller, A., Larus, J., and Greenberg, A. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable Web services. *Perform. Eval. 68*, 11 (2011).

[10] Mitzenmacher, M. The power of two choices in randomized load balancing. *IEEE Trans. Par. Distrib. Sys. 12*, 10 (2001).

[11] Oakes, E., Yang, L., Houck, K., Harter, T., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. Pipsqueak: Lean Lambdas with large libraries. In *IEEE Intl. Conf. Distrib. Comp. Sys. Workshops (ICDCSW)* (2017).

[12] Oakes, E., Yang, L., Houck, K., Harter, T., Arpaci-Dusseau, A., and Arpaci-Dusseau, R. Pipsqueak: Lean Lambdas with large libraries, 2017. (Presentation given at the) Workshop on Serverless Computing (WoSC).

[13] Pai, V., Aron, M., Banga, G., Svendsen, M., Druschel, P., Zwaenepoel, W., and Nahum, E. Locality-aware request distribution in cluster-based network servers. *SIGOPS Oper. Syst. Rev. 32*, 5 (1998).

[14] Sampé, J., Sánchez-Artigas, M., García-López, P., and París, G. Data-driven serverless functions for object storage. In *ACM/IFIP/USENIX Middleware* (2017).

[15] van Eyk, E., Iosup, A., Abad, C. L., Grohmann, J., and Eismann, S. A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures. In *ACM/SPEC Intl. Conf. Perf. Eng. (ICPE)* (2018).

[16] van Eyk, E., Iosup, A., Seif, S., and Thömmes, M. The SPEC cloud group's research vision on FaaS and serverless architectures. In *Intl. Workshop on Serverless Comp. (WoSC)* (2017).

[17] Wang, W., Zhu, K., Ying, L., Tan, J., and Zhang, L. MapTask scheduling in MapReduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Trans. Netw. 24*, 1 (2016).

[18] Xie, Q., and Lu, Y. Priority algorithm for near-data scheduling: Throughput and heavy-traffic optimality. In *IEEE Conf. Comp. Comm. (INFOCOM)* (2015).

[19] Xie, Q., Pundir, M., Lu, Y., Abad, and Campbell. Pandas: Robust locality-aware scheduling with stochastic delay optimality. *IEEE/ACM Trans. Netw. 25*, 2 (2017).

[20] Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In *European Conf. Comp. Sys. (EuroSys)* (2010).