An Extensible Scheduler for the OpenLambda FaaS Platform

Gustavo Totoy, Edwin F. Boza, Cristina L. Abad

Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador

Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador
- Working on To Leave for its PS4 release

Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador
- Working on To Leave for its PS4 release
- Undergraduate student graduating in September 2018

Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador
- Working on To Leave for its PS4 release
- Undergraduate student graduating in September 2018
  - Applying for US grad school this year!

Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador
- Working on To Leave for its PS4 release
- Undergraduate student graduating in September 2018
  - Applying for US grad school this year!
- Research assistant at the Systems Research Lab lead by Prof. Cristina Abad in ESPOL University

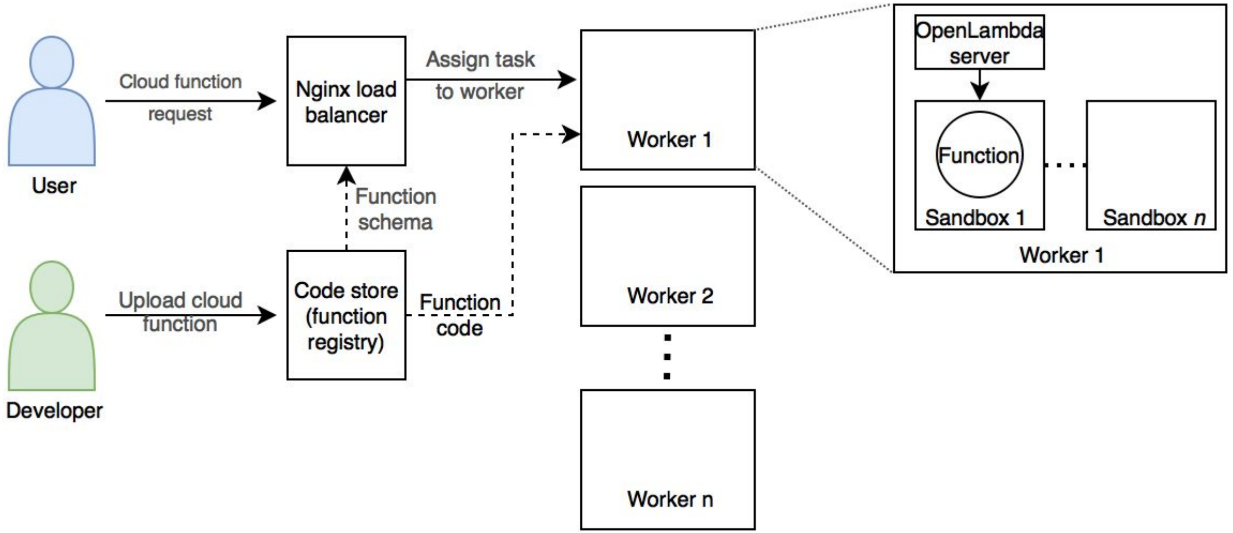
Bio

- Game Developer at http://freakycreations.net/, a studio based in Ecuador
- Working on To Leave for its PS4 release
- Undergraduate student graduating in September 2018
  - Applying for US grad school this year!
- Research assistant at the Systems Research Lab lead by Prof. Cristina Abad in ESPOL University
- Try to continue improving on my hobbies: Cook, CrossFit and BJJ

olscheduler

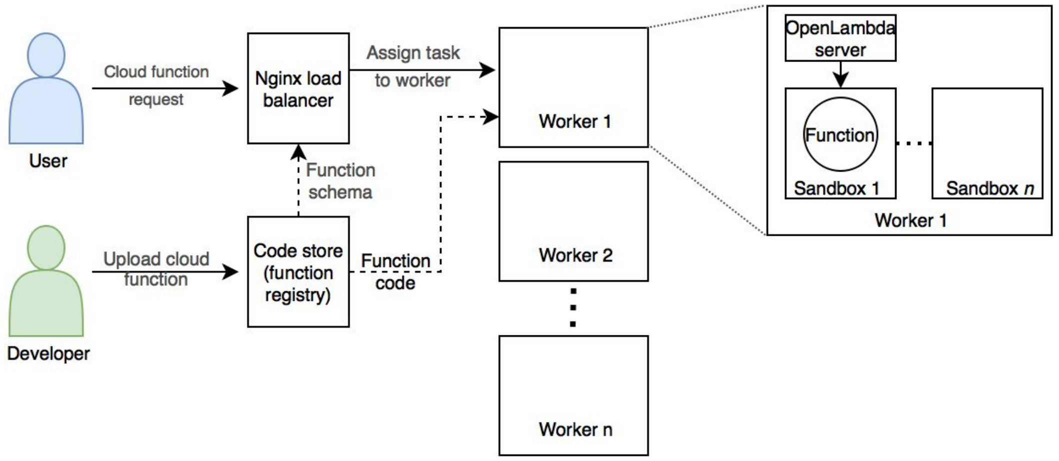
Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform



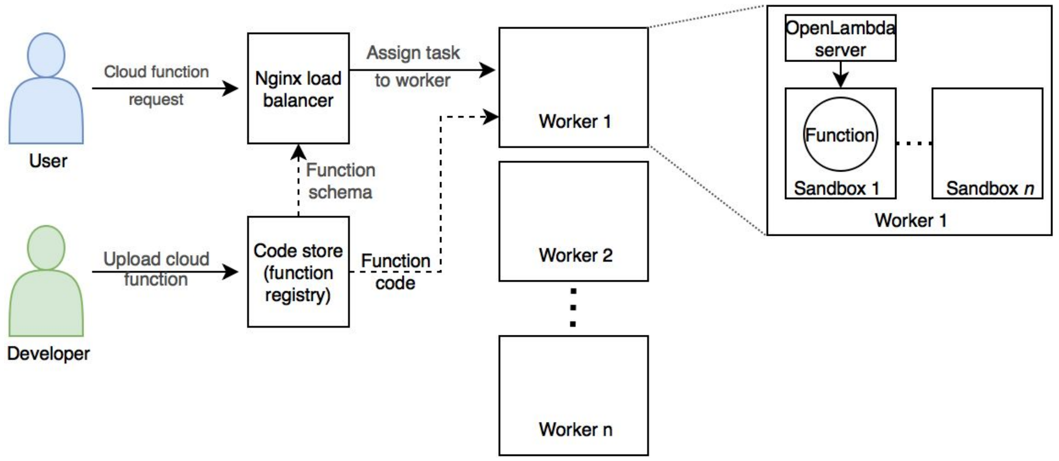
Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform
- Written in Go



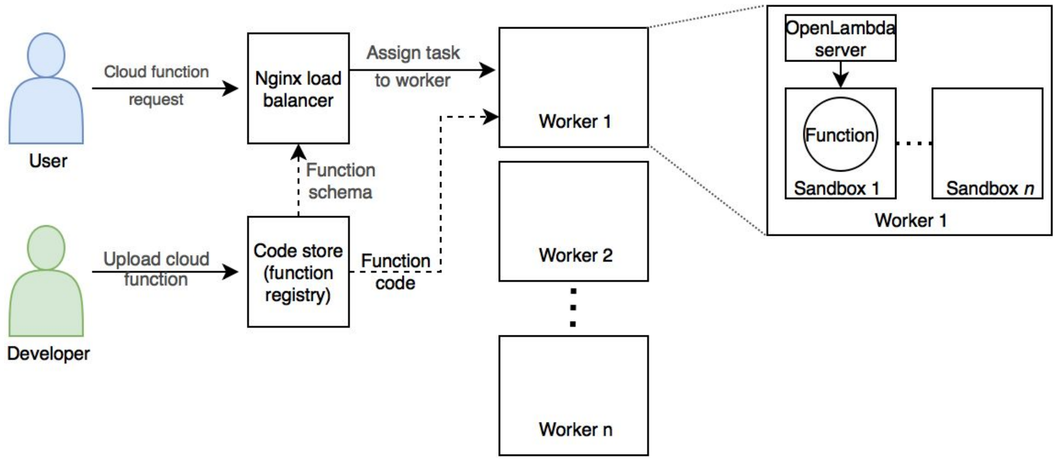
Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform
- Written in Go
- ~400 lines of code



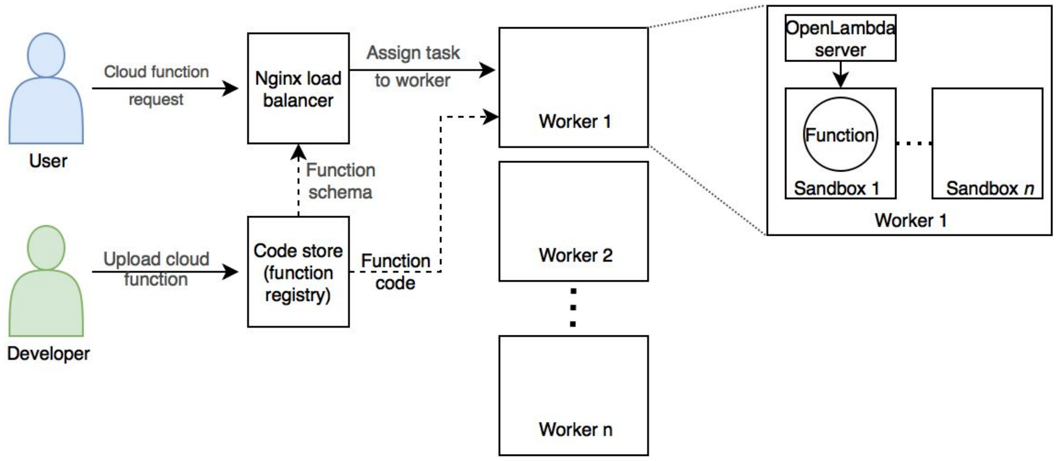
Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform
- Written in Go
- ~400 lines of code
- Uses Go standard library



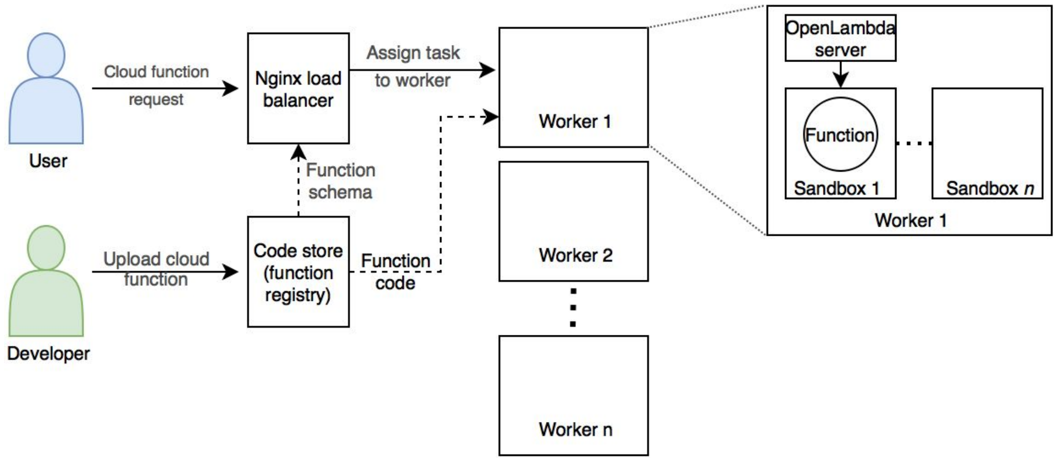
Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform
- Written in Go
- ~400 lines of code
- Uses Go standard library
- Offers four scheduling policies: random, round-robin, least-loaded and pkg-aware



Summary

- Simple and extensible function scheduler for the OpenLambda Function as a Service platform
- Written in Go
- ~400 lines of code
- Uses Go standard library
- Offers four scheduling policies: random, round-robin, least-loaded and pkg-aware
- Open Source: https://github.com/gtotoy/olscheduler



Motivation

Load Balancer -> OpenLambda + PipSqueak

Background

- A Function-as-a-Service (FaaS) cloud platforms enable tenants to deploy and execute functions on the cloud, without having to worry about server provisioning

Background

- A Function-as-a-Service (FaaS) cloud platforms enable tenants to deploy and execute functions on the cloud, without having to worry about server provisioning
- Small cloud functions with big or many packages to be loaded

Background

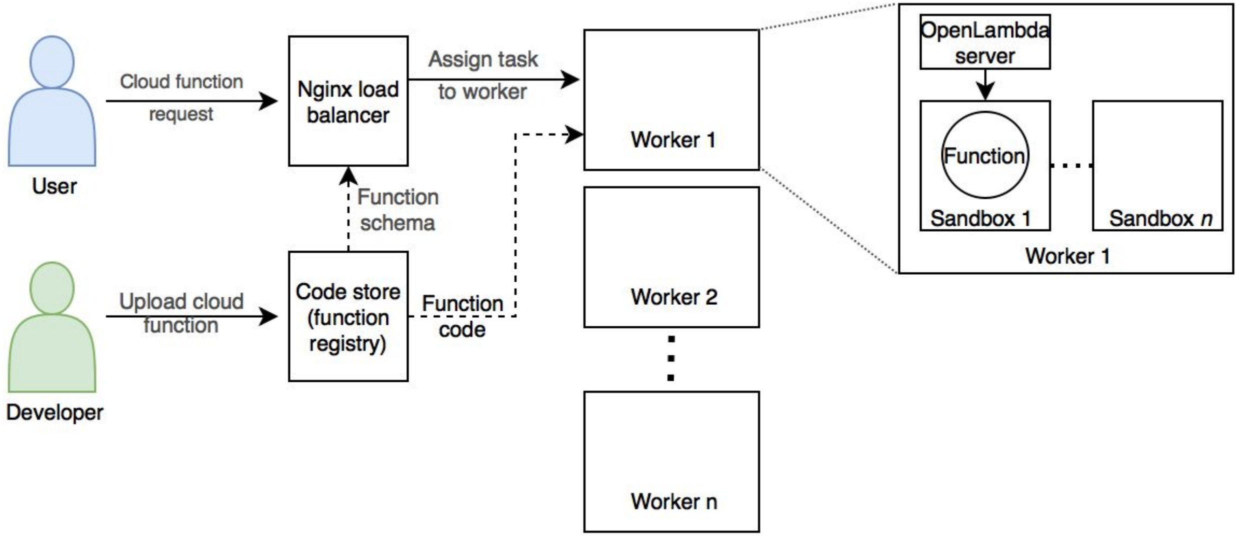
- A Function-as-a-Service (FaaS) cloud platforms enable tenants to deploy and execute functions on the cloud, without having to worry about server provisioning
- Small cloud functions with big or many packages to be loaded
- In a distributed computing platform, co-locating tasks at worker nodes that cache any required files is a time-proven mechanism to reduce task latency

Background

- A Function-as-a-Service (FaaS) cloud platforms enable tenants to deploy and execute functions on the cloud, without having to worry about server provisioning
- Small cloud functions with big or many packages to be loaded
- In a distributed computing platform, co-locating tasks at worker nodes that cache any required files is a time-proven mechanism to reduce task latency
- We seek to increase the hit rate of the package cache and, as a result, reduce the latency of the cloud functions

Load Balancer -> ***OpenLambda*** + PipSqueak

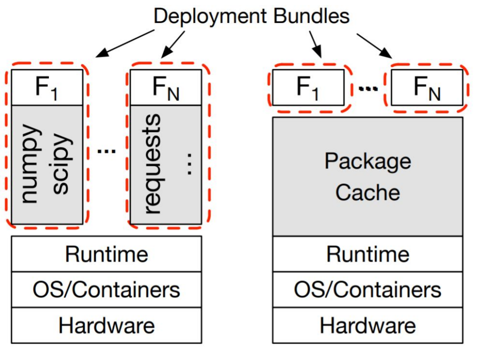
- OpenLambda [1]: is a serverless computing platform that supports the Function as a Service execution model



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

Load Balancer -> OpenLambda + ***PipSqueak***

- Pipsqueak [2]: shared package cache available at each OpenLambda worker
- Stores a set of Python interpreters with pre-imported packages, in a sleeping state



(a) No Support **(b) Package Awareness**

[2] 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).

Load Balancer -> OpenLambda + PipSqueak

- Nginx (www.nginx.com), which comes with load balancing methods: round robin, least connected and ip hash
- gobetween (gobetween.io)
- But, What if we want to make smarter scheduling policies?

olscheduler

Our scheduler exposes functions and data structures that can be used to get the following information:

1. **Workers:** Number and references to the worker lambda nodes
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, exposed for each function call

olscheduler

Currently supports the following scheduling algorithms:

- **Round-robin**

olscheduler

Currently supports the following scheduling algorithms:

- **Round-robin**
- **Least-loaded**

olscheduler

Currently supports the following scheduling algorithms:

- **Round-robin**
- **Least-loaded**
- **Random**

olscheduler

Currently supports the following scheduling algorithms:

- **Round-robin**
- **Least-loaded**
- **Random**
- **Pkg-aware:** Seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

olscheduler

- Small code base ~400 LOCs

olscheduler

- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer

olscheduler

- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer
- Written in a modern programming language

olscheduler

- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer
- Written in a modern programming language
- It can be adapted to OpenLambda specific requirements

olscheduler

- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer
- Written in a modern programming language
- It can be adapted to OpenLambda specific requirements
- Offers pkg-aware, a novel scheduler policy

olscheduler

- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer
- Written in a modern programming language
- It can be adapted to OpenLambda specific requirements
- Offers pkg-aware, a novel scheduler policy
- Allows development of additional scheduling policies

olscheduler

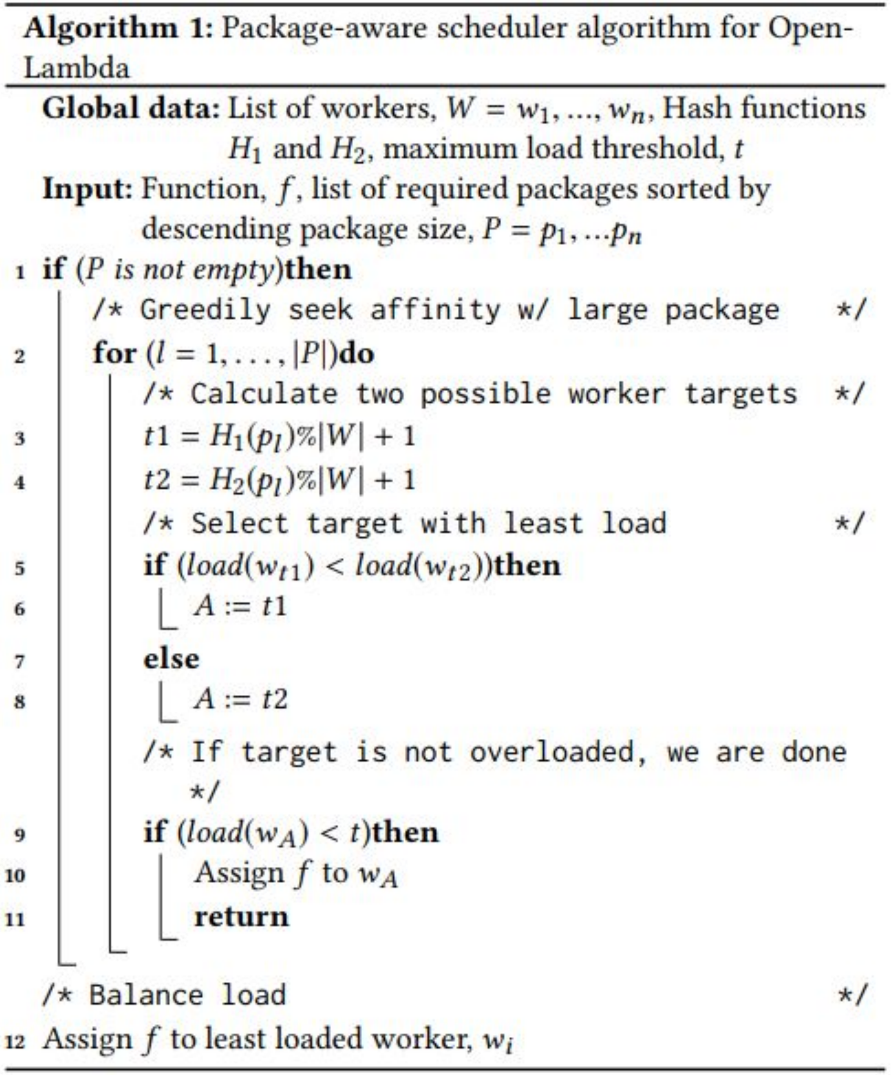
- Small code base ~400 LOCs
- Simpler to modify and extend than the nginx load balancer
- Written in a modern programming language
- It can be adapted to OpenLambda specific requirements
- Offers pkg-aware, a novel scheduler policy
- Allows development of additional scheduling policies
- Facilitates experiment reproducibility

Pkg-Aware Algorithm

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)

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

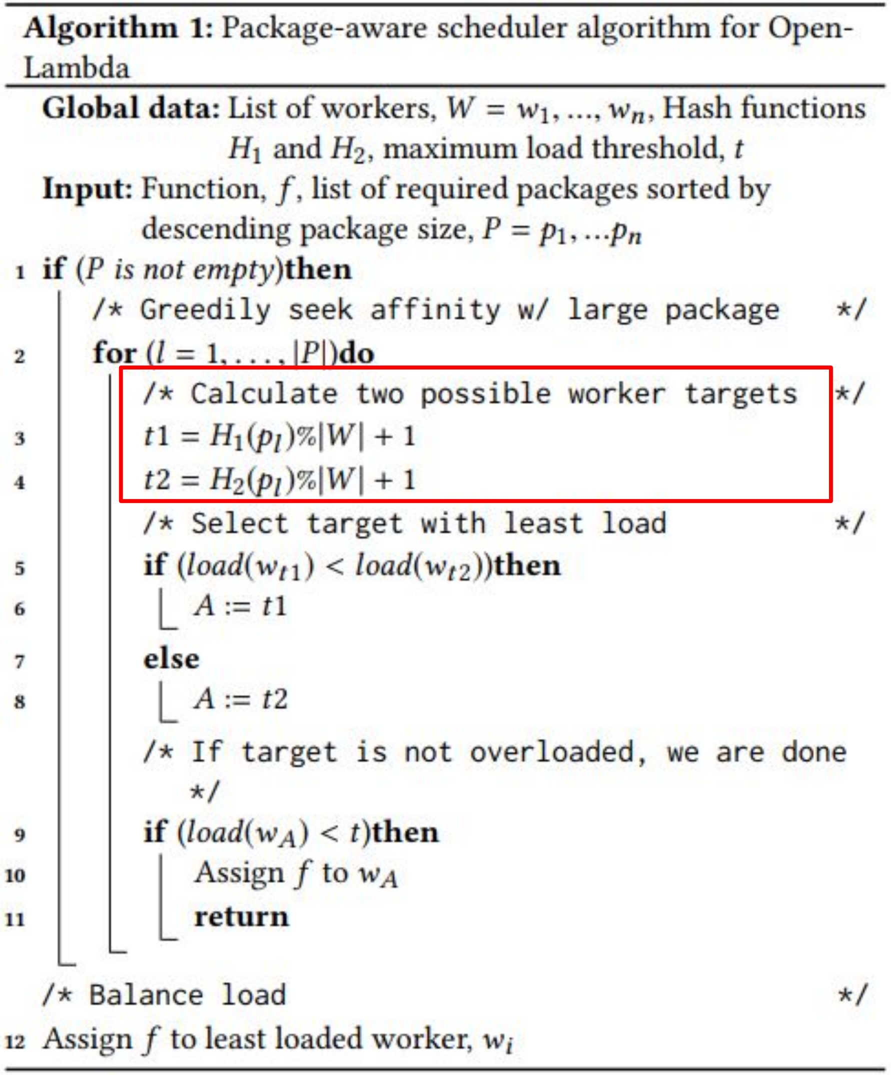


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package */
    for ($l = 1, \ldots, |P|$) do
        /* Calculate two possible worker targets */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load */
Assign $f$ to least loaded worker, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

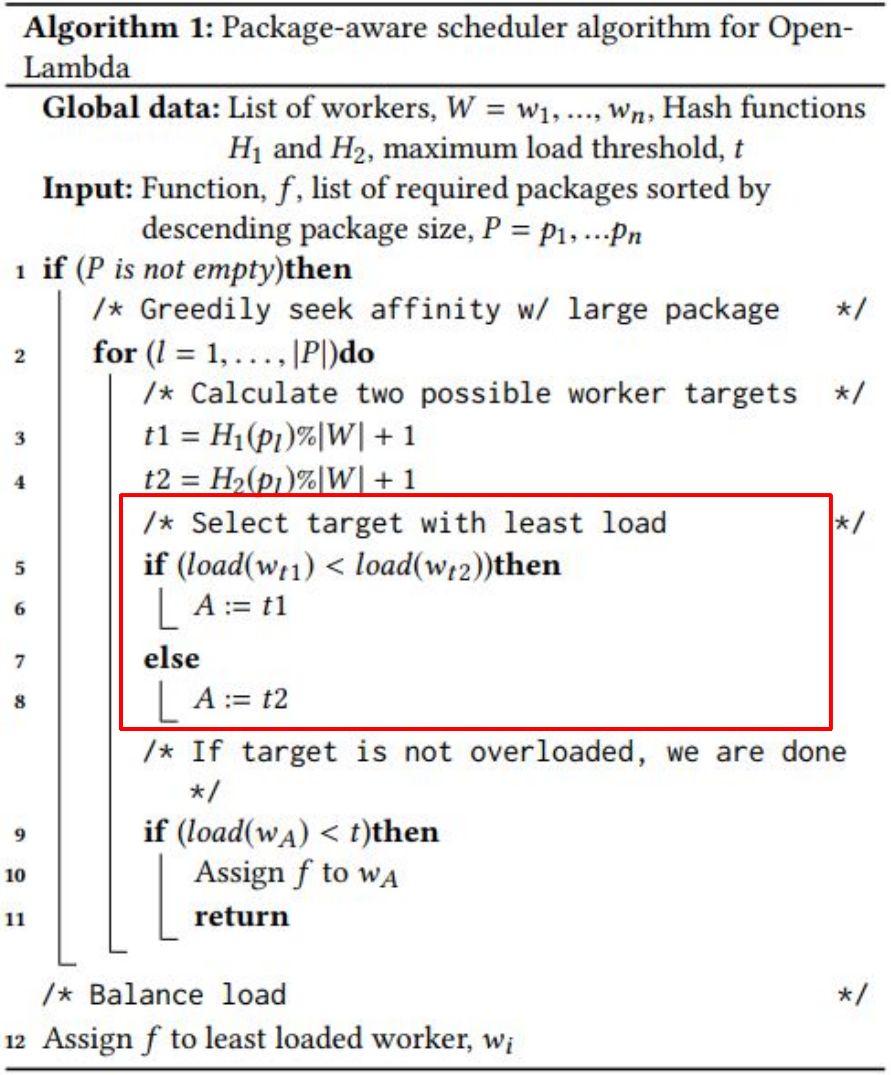


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package */
    for ($l = 1, \dots, |P|$) do
        /* Calculate two possible worker targets */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load */
Assign $f$ to least loaded worker, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

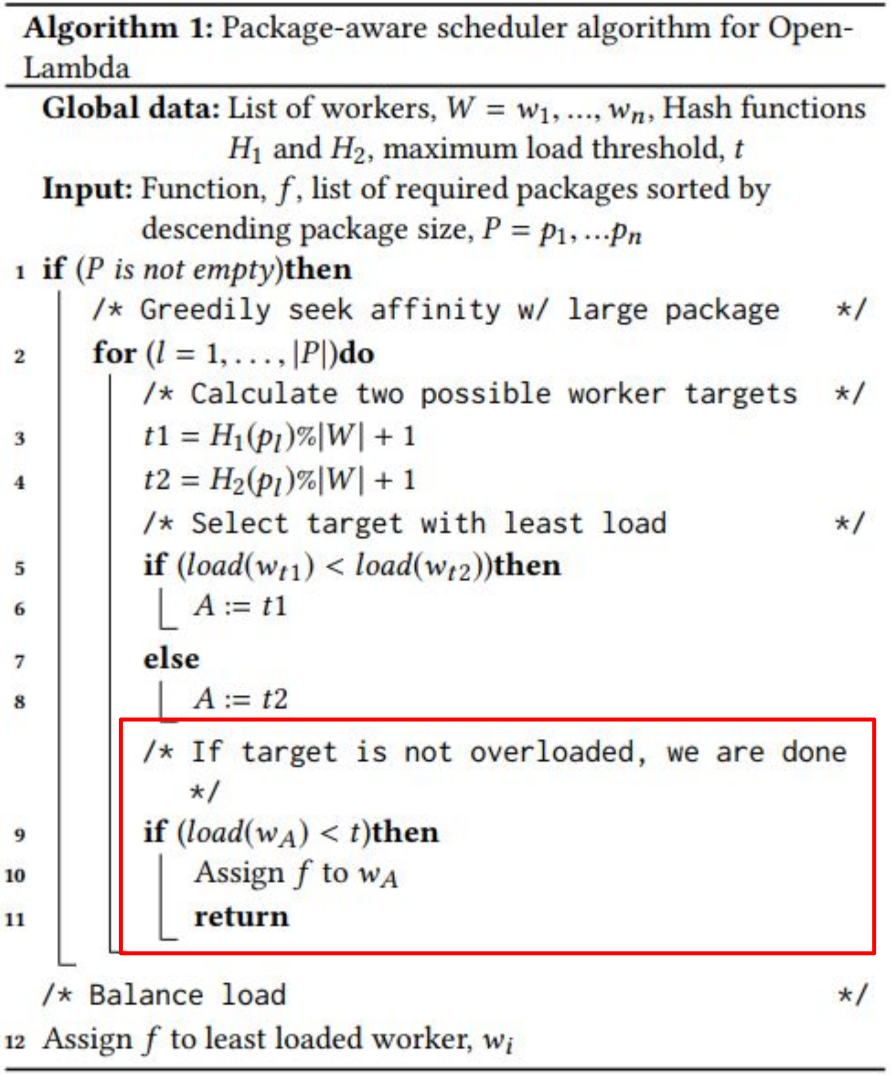


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package */
    for ($l = 1, . . . , |P|$) do
        /* Calculate two possible worker targets */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load */
Assign $f$ to least loaded worker, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

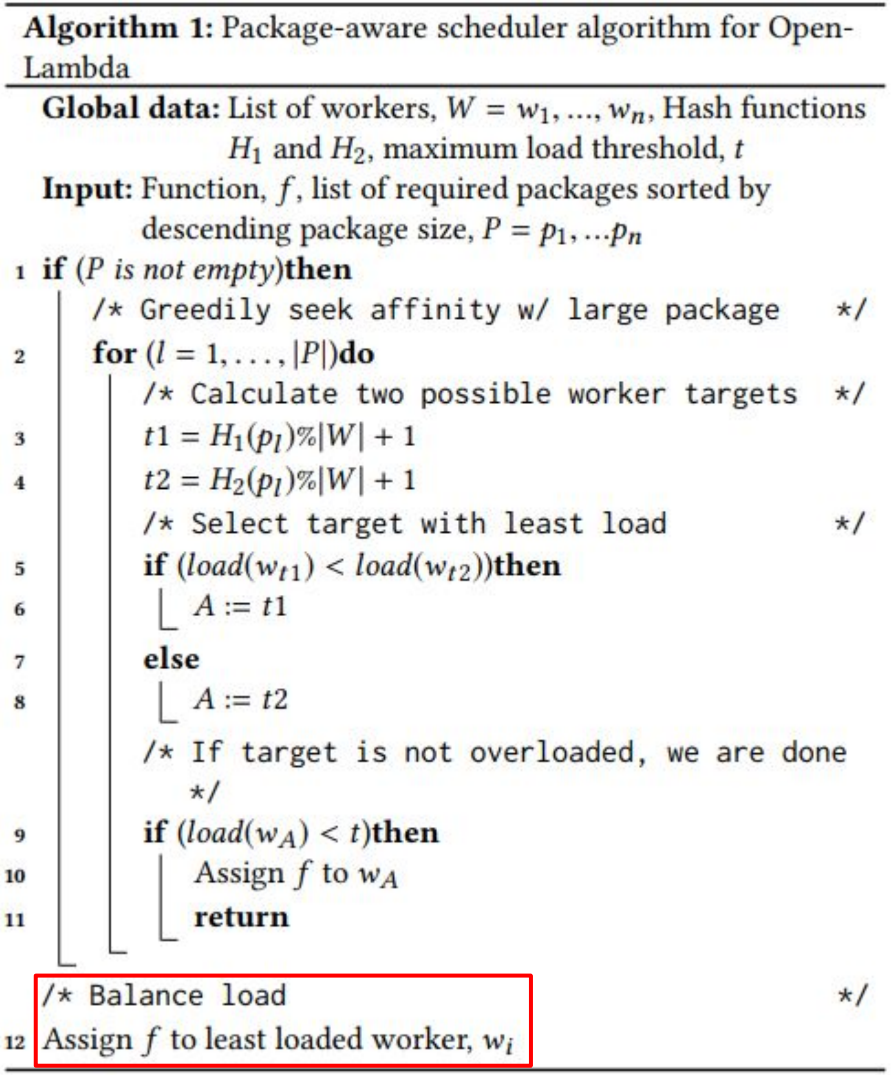


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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 ($l = 1, . . . , |P|$) 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, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

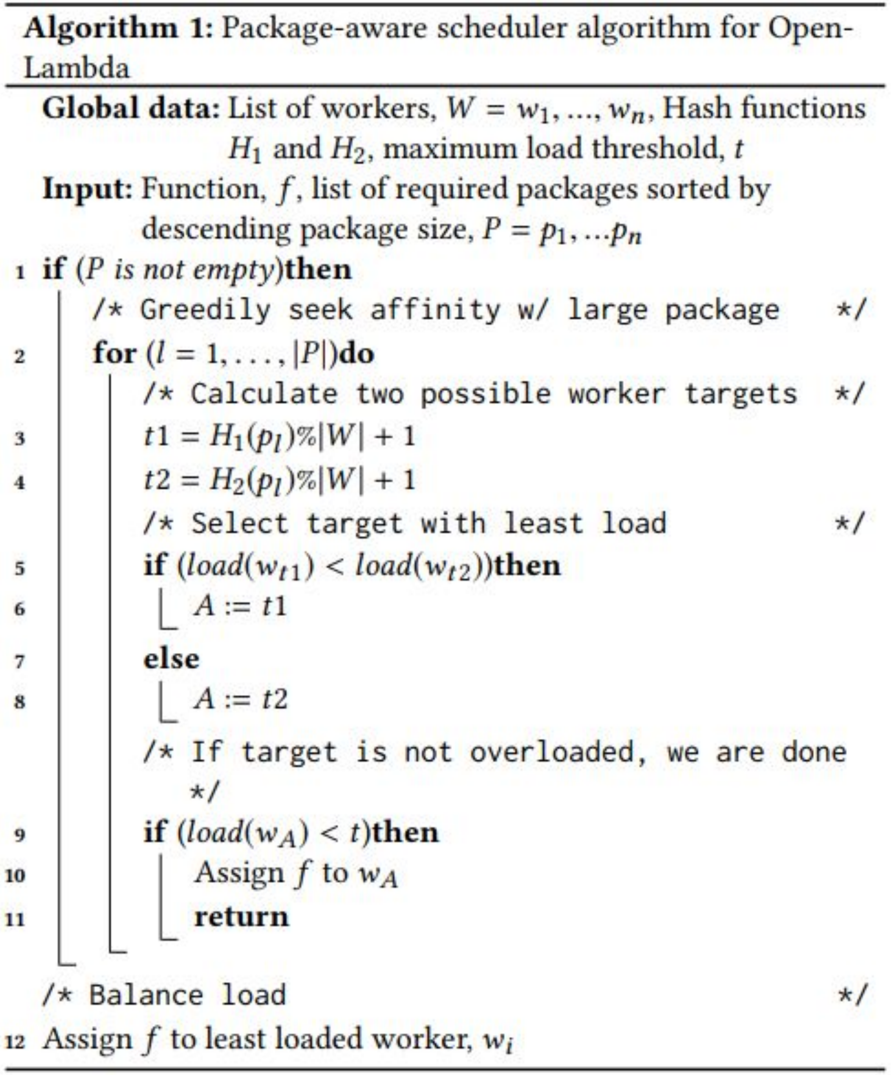


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package */
    for ($l = 1, \ldots, |P|$) do
        /* Calculate two possible worker targets */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load */
Assign $f$ to least loaded worker, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold

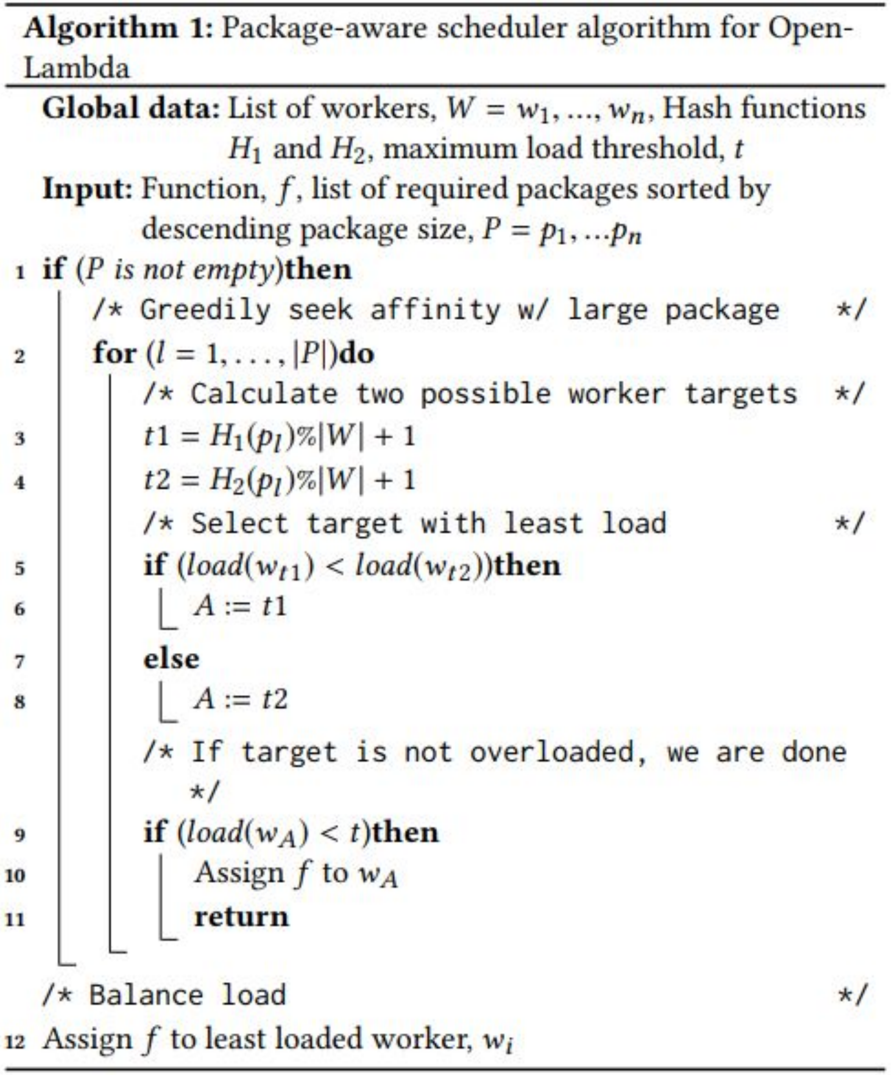


Algorithm 1: Package-aware scheduler algorithm for Open-Lambda

```
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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package   */
    for ($l = 1, \dots, |P|$) do
        /* Calculate two possible worker targets  */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load           */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done
           */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load                                   */
Assign $f$ to least loaded worker, $w_i$
```

Algorithm

- The algorithm seeks to maximize cache affinity (with respect to the packages in the package cache), while avoiding overloading workers beyond a configurable threshold
- With the information exposed by olscheduler, implementing this policy was straightforward, and took only 46 LOCs



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, $P = p_1, ...p_n$
if ($P$ is not empty) then
    /* Greedily seek affinity w/ large package */
    for ($l = 1, \dots, |P|$) do
        /* Calculate two possible worker targets */
        $t1 = H_1(p_l)\%|W| + 1$
        $t2 = H_2(p_l)\%|W| + 1$
        /* Select target with least load */
        if ($load(w_{t1}) < load(w_{t2})$) then
            $A := t1$
        else
            $A := t2$
        /* If target is not overloaded, we are done */
        if ($load(w_A) < t$) then
            Assign $f$ to $w_A$
            return
/* Balance load */
Assign $f$ to least loaded worker, $w_i$
```

65%

Preliminary simulation results show that this simple approach can cut the function latency by more than 65%

Results

- Improved median hit rate from 51.15% (least-loaded) to 63.52% (pkg-aware)

Results

- Improved median hit rate from 51.15% (least-loaded) to 63.52% (pkg-aware)
- Median latency improves by 65.8% (pkg-aware vs. least-loaded), and tail latency improves by 41.9% (90th percentile)

Results

- Improved median hit rate from 51.15% (least-loaded) to 63.52% (pkg-aware)
- Median latency improves by 65.8% (pkg-aware vs. least-loaded), and tail latency improves by 41.9% (90th 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

Future Work

Real Cloud Experiments + More scheduling algorithms

Planning

- PipSqueak has recently released a benchmarking suite for OpenLambda to evaluate its cache performance improvements
- We are going to use this suite to measure latency when pkg-aware scheduling policy is selected and compare against other available policies
- We plan to implement more refined version of pkg-aware based on the work of Package-aware scheduling of FaaS functions [3]

[3] 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)

Thanks!

Contact:

Systems Research Lab
ESPOL University
Guayaquil, Ecuador

gtotoy@fiec.espol.edu.ec
cabad@fiec.espol.edu.ec