Active-Standby for High-Availability in FaaS

Yasmina Bouizem
Univ Tlemcen, LRIT
Univ Rennes, Inria

Djawida Dib
Univ Tlemcen, LRIT

Nikos Parlavantzas
INSA Rennes, IRISA

Christine Morin
Univ Rennes, Inria

https://dl.acm.org/doi/10.1145/3429880.3430097

Paper Review
TCSS 562 Software Engineering For Cloud Computing

By Shishir Reddy & Anindya Dey

OUTLINE

- INTRODUCTION
- RELATED WORK
- FISSION OVERVIEW
- ACTIVE STANDBY APPROACH
- KEY CONTRIBUTIONS
- EXPERIMENTS & RESULTS
- CONCLUSION & FUTURE WORK
- CRITIQUE

INTRODUCTION

- Function-as-a-Service(FaaS) is at the heart of Serverless computing
- High Availability and Fault Tolerance are most essential
- Retry Mechanism (current approaches)
- Alternative Fault Tolerance approach (Active-Standby failover)

RELATED WORK

- AWS Lambda, Google Cloud Functions, Microsoft Azure Functions
- OpenFaaS , Fission
- Fault-Tolerance Shim for Serverless Computing
- Fault-tolerant and transactional stateful serverless workflows

FISSION OVERVIEW

Executor
  Pool Manager : pool of generic warm containers, no auto-scale
  New Deploy   : creates K8s deployments, horizontal auto scaling

Router
  -routes a function call to corresponding pod
  -triggers retries in case of failures

Retry Mechanism

1. Router receives a fn(function) call
2. Checks if the fn service record exists in the cache
    a) No - executor creates a new service for the fn
    b) Yes - sends req to fn pod
3. If req fails, retries for a fixed no. of times & finally removes it from cache & performs step 2a) again.

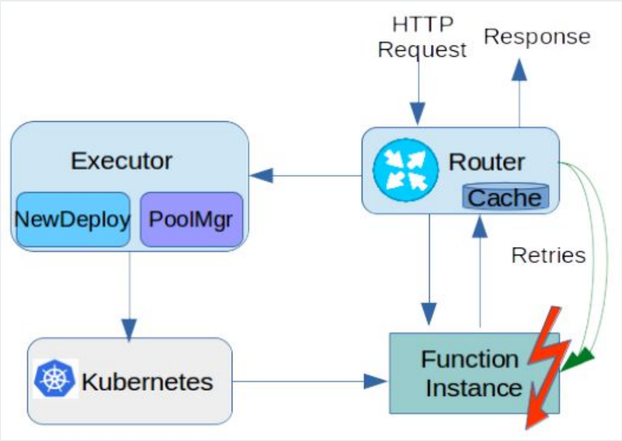


Fig 1 : Overview of retry mechanism in Fission

PROPOSED ACTIVE-STANDBY APPROACH

New Deploy
    creates and maintains two fn instances
K8s Readiness Probe
    specifies state of the pods, configures heartbeat
CoreDNS
    maintains IP address of the pods

Retry Mechanism

1. CoreDNS receives req from user & returns IP of active pod to user.
2. User directly sends req to pod
3. Heartbeat
    a) Every 1 second between active & passive
    b) Active is running, Passive fails readiness test
    c) Active fails, Passive succeeds readiness test and becomes active & a new passive pod is created
    d) Passive fails, a new passive pod is created.

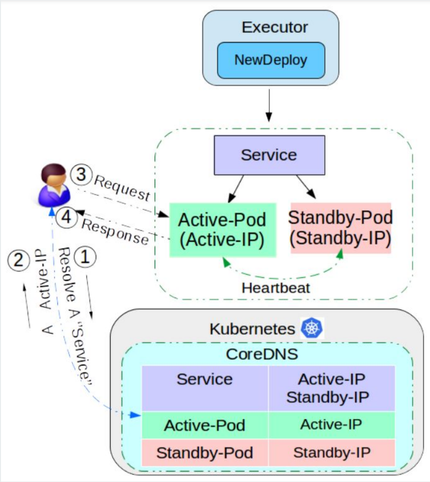


Fig 2 : AS overview in Fission

KEY CONTRIBUTIONS

- High Availability approach for FaaS
  - describes the approach, provides implementation in Fission
- High Availability vs Retry approach comparison
  - experiments and evaluation on Grid' 5000 testbed

EXPERIMENTAL SETUP

Test Scenarios
Pod failure & Node failure

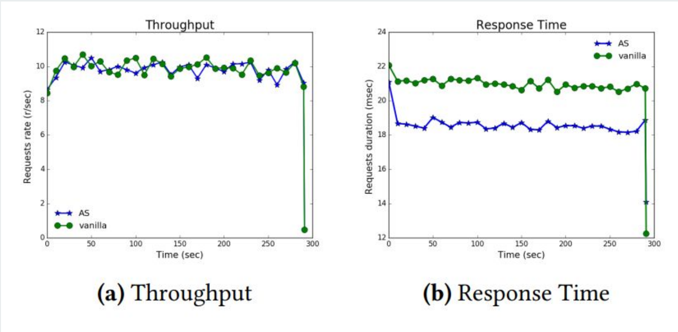
Metrics
Performance (Throughput & Response Time)
Availability (and HTTP status code)
Resource Consumption

Test Environment
Grid'5000 testbed
5 nodes on Lyon site to deploy K8s (1.11), Fission AS, Fission Vanilla(1.5.0)
1 node to invoke functions
1 node for fault injection
Each node - 2 CPUs Intel Xeon E5-2620 v4, 8 cores/CPU, 64 GB memory

Applications
Fibonacci & Guestbook

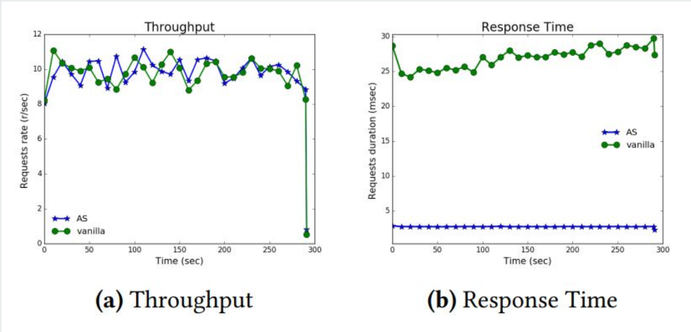
Workload
3000 requests in 5 minutes(Tsung)

RESULTS - NO FAILURE



(a) Throughput (b) Response Time

Fig 3 : Fibonacci without failures



(a) Throughput (b) Response Time

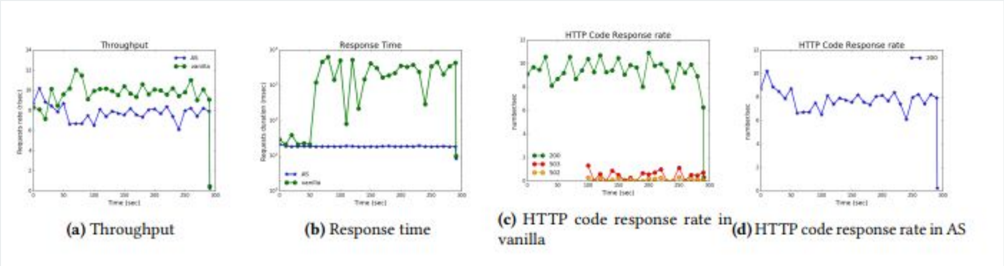
Fig 4 : Guestbook without failures

Throughput : Same in both (11 req/sec)

Response Time : Fission : 16 ms, Fission AS : 2 ms
(Router component (vanilla) vs Core DNS (in AS))

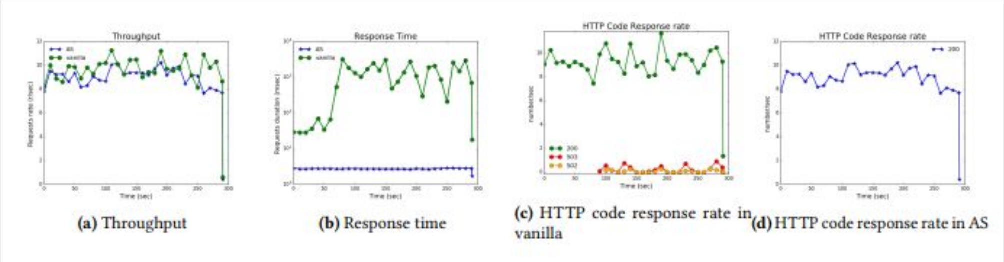
RESULTS - POD FAILURE

- Active-Standby and vanilla react to the pod failure differently
- Vanilla retries the function execution many times
- Active-Standby immediately forwards traffic to the standby instance



(a) Throughput (b) Response time (c) HTTP code response rate in vanilla (d) HTTP code response rate in AS

Fig 5 : Fibonacci with pod failures

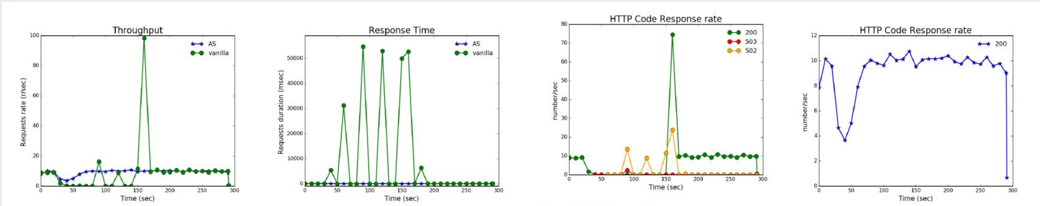


(a) Throughput (b) Response time (c) HTTP code response rate in vanilla (d) HTTP code response rate in AS

Fig 6 :Guestbook application with pod failures

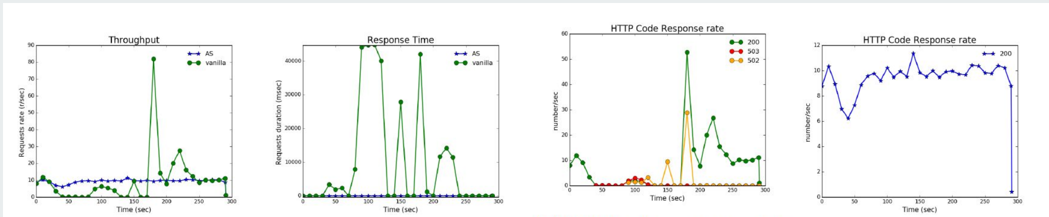
RESULTS - NODE FAILURE

- Figures 7(a) and 8(a) show peaks in throughput for vanilla
- After a node crash, requests are queued for vanilla, resulting in increased waiting and response times
- Vanilla tolerates short failures better



(a) Throughput **(b)** Response time **(c)** HTTP code response rate in vanilla **(d)** HTTP code response rate in AS

Fig 7 : Fibonacci with node failures



(a) Throughput **(b)** Response time **(c)** HTTP code response rate in vanilla **(d)** HTTP code response rate in AS

Fig 8 : Guestbook application with node failures

CONCLUSION

Problem

Increase availability of serverless functions in FaaS platforms

Method

Active-Standby failover approach for FaaS platforms.

Results

Active Standby outperforms vanilla in terms of response time and availability while incurring an overhead in resource consumption

FUTURE WORK

Additional Fault-Tolerance Techniques	○ Explore additional fault-tolerance techniques within a FaaS context like check-point restart, logging, replication. ○ Passive node can operate more as a load-balancer with smart management.
Serverless Application Testing	○ Use applications that give a better standard of performance. ○ Use applications that have more real-world significance
Goals	○ Design a smart fault tolerant system for FaaS which can use these techniques to automatically make the right trade off between availability, performance, energy consumption

REFERENCES

1. Active-Standby for High-Availability in FaaS (https://doi.org/10.1145/3429880.343009)

2. A Fault-Tolerance Shim for Serverless Computing (https://dl.acm.org/doi/pdf/10.1145/3342195.3387535)

3. Fault-tolerant and transactional stateful serverless workflows(https://www.usenix.org/system/files/osdi20-zhang_haoran.pdf)

CRITIQUE: STRENGTHS

Table 1. Recovery Time with AS and vanilla in pod failures

	Fission Vanilla	Fission AS
Finonacci Function	2.840s	1.814s
Guestbook application	3.614s	1.528s

Table 2. Recovery Time with AS and vanilla in node failures

	Fission Vanilla	Fission AS
Finonacci Function	3min7s	6.384s
Guestbook application	2min39s	6.194s

Observations:

- The vanilla fault-tolerance system of Fission reacts much harsher to node failures over pod failures

Performance Increase:

- With node failures, recovery times are significantly better for AS

CRITIQUE: WEAKNESSES

Trade-Offs:

- 15% CPU and 12% in-memory for a pod failure recovery time gain of 55% and 140%
- This time might be significantly less if retry counts are reduced in Vanilla

Scalability:

- CPU and Memory overhead do not scale well across networks of larger functions
- If every function requires a copy, might become cost-prohibitive

Assumptions:

- Assumed that functions are idempotent in both approaches (may not be the case in real world scenarios)