



**HAL**  
open science

## Distributed Function Cache for Heterogeneous Serverless Cloud

Vincent Lannurien, Laurent D'orazio, Olivier Barais, Stephane Paquelet, Jalil Boukhobza

► **To cite this version:**

Vincent Lannurien, Laurent D'orazio, Olivier Barais, Stephane Paquelet, Jalil Boukhobza. Distributed Function Cache for Heterogeneous Serverless Cloud. Per3S - Performance and Scalability of Storage Systems, May 2023, Paris, France. pp.1-1, 2023. hal-04303898

**HAL Id: hal-04303898**

**<https://hal.science/hal-04303898v1>**

Submitted on 23 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# DISTRIBUTED FUNCTION CACHE FOR HETEROGENEOUS SERVERLESS CLOUD

Vincent Lannurien<sup>\*+</sup>, Laurent d’Orazio<sup>+</sup>, Olivier Barais<sup>+</sup>, Stéphane Paquet<sup>+</sup>, Jalil Boukhobza<sup>\*+</sup>

<sup>\*</sup>ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, <sup>+</sup>Rennes 1 University, IRISA UMR6074, <sup>+</sup>b<>com Institute

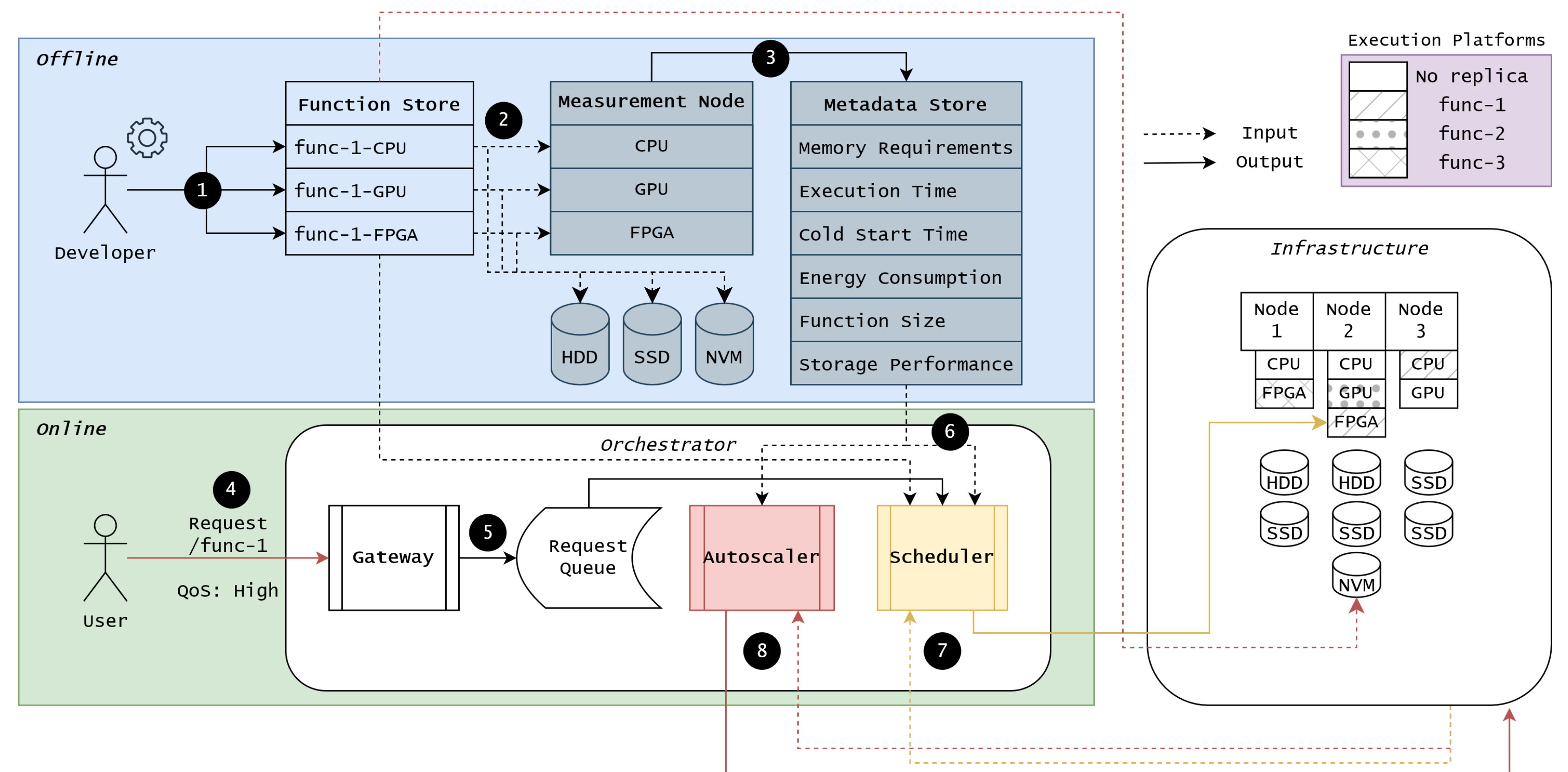
## 1 – Serverless Cloud Challenges [5]

Characteristic	Serverful (IaaS, PaaS)	Serverless (FaaS)
Provisioning	Customer responsibility	Fully <b>managed</b> (i.e. by the provider)
Scaling	Customer responsibility	<b>Auto-scaling</b> built in
Availability	Depends on <b>provisioned</b> resources	Code runs in multiple high availability zones
Fault tolerance	Depends on deployment strategy	Retries of <b>stateless</b> functions
Concurrency	Depends on <b>provisioned</b> resources	Virtually <b>infinite</b>

Tab. 1: Comparison of key characteristics in serverless and serverful service models

- Serverless resources are *not reserved* [6]
  - Increased provider’s responsibility
    - \* Dynamic **allocation** (following load variations) ⑧
    - \* Dynamic **placement** (mapping requests to resources) ⑦
- Functions are *not network-addressable* [1]
  - Communications between functions use **slow storage**
  - Initialization times can **dominate** total function execution time
- Cloud resources are *heterogeneous* [3]
  - Various levels of **performance** and **cost**
- Load is *unpredictable* [7]
  - Stochastic barrier: need for an online solution
- Users have various *QoS requirements* [2]
  - Throughput-centric (batch jobs) vs. lower latency (interactive jobs)

## 2 – Platform Overview and Lifecycle of a Request [4]



**PROBLEM STATEMENT** – How to account for storage costs when deploying short-lived serverless functions in a private cloud, leveraging heterogeneous hardware to optimize for quality of service?

## 3 – Problem Justification

- Function instantiation requires **pulling the function image** (2 to 4 GB) from a remote image repository (Fig. 2)
- Functions on the same node **benefit from local storage** to achieve communications (Fig. 3)
- Short execution times (10 to 100 ms) can be offset by longer or more frequent **cold start delays** (1 to 15 s) (Fig. 2)
- There is a **contention on local storage** between function images and function data (Fig. 3)
- Risk of saturating node function cache with function images (Tab. 2)
- Local image cache starts thrashing, resulting in SLA violations (Tab. 2)

## 4 – Thinking in Terms of Cost

We extend the cost model for resources allocation and function scheduling we published in [4]. We identify the implications of storage at each step:

	Impact	Cost
Resources allocation	Function response time	I/O bandwidth (Gbps)
	Resource contention	I/O capacity (GB)
Function scheduling	SLA penalties	I/O latency (ms)
	Tasks consolidation	I/O capacity (GB)
Application execution	Inter-function communications	I/O latency (ms)
	Output data storage	I/O capacity (MB)

Tab. 2: Breakdown of storage impacts on cost throughout the request lifecycle

$$scaleCost_{f_{N,P}} = k_{TT} \cdot TT_{f_{N,P}} + k_{EC} \cdot EC_{f_{N,P}} + k_{HP} \cdot HP_{f_{N,P}}$$

$$schedCost_{f_{N,P}} = k_{QP} \cdot QP_{f_{N,P}} + k_{EC} \cdot EC_{f_{N,P}} + k_{TC} \cdot TC_{f_{N,P}}$$

## 3a – Function Cache

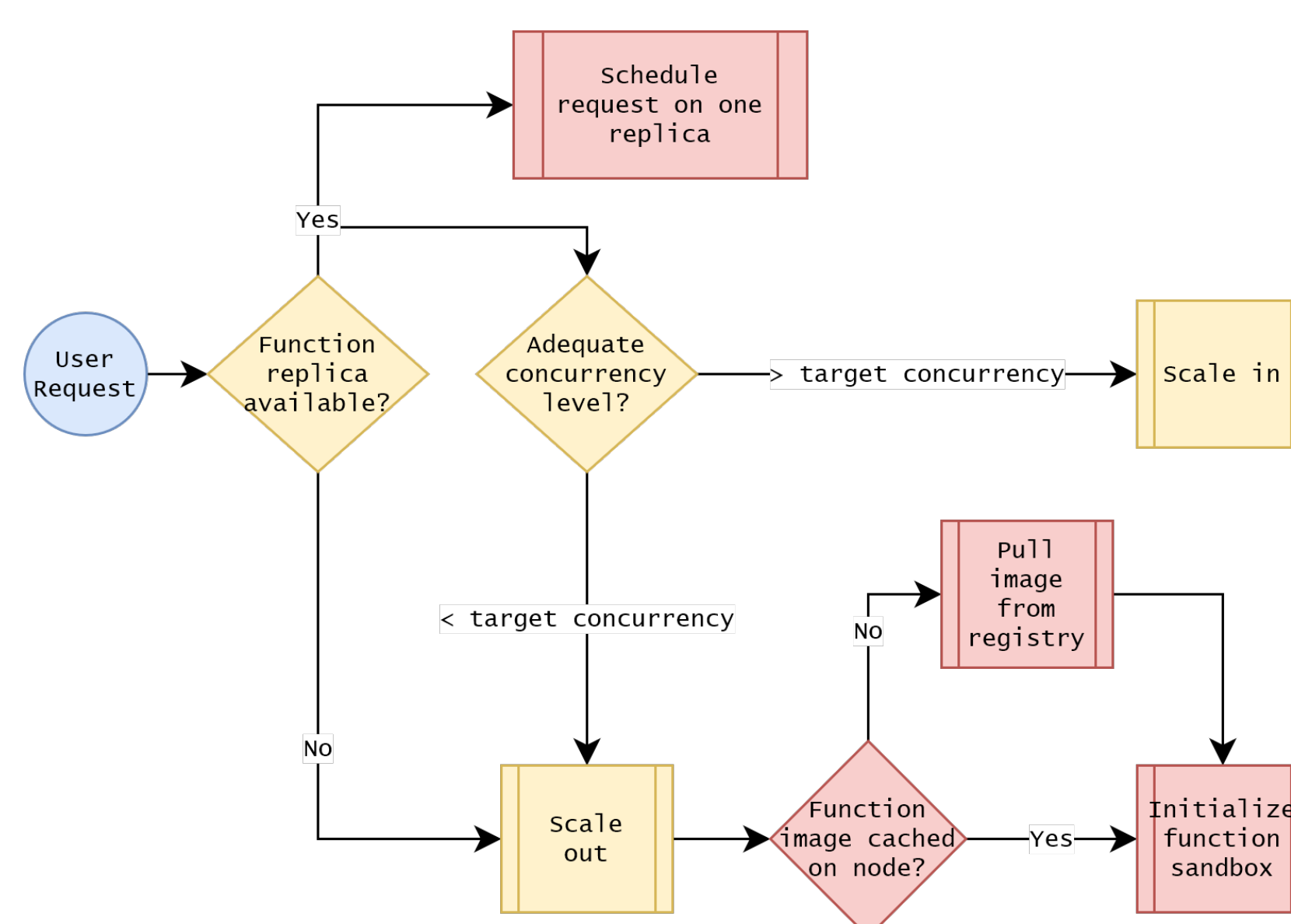


Fig. 2: Flowchart: allocation (yellow) and scheduling (red) decisions

## 3b – Function Communications

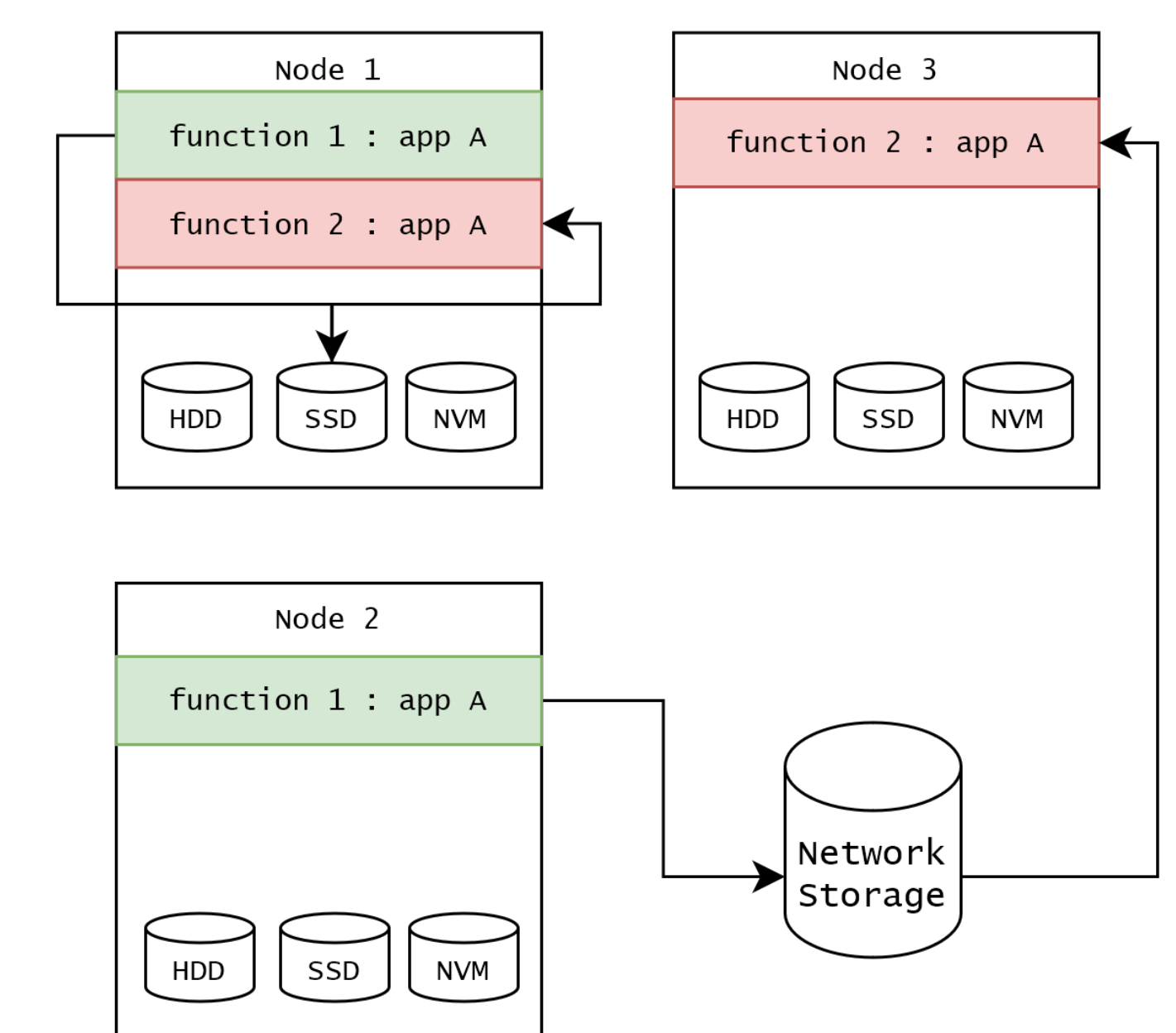


Fig. 3: Functions can achieve communications through either local or remote storage

## 5 – Leveraging Node Storage

We propose to extend the platform in Fig. 1 with a **distributed, cost-aware function cache** that opportunistically leverages the heterogeneity of available disk space and memory on worker nodes to:

- accelerate function initialization by minimizing cold start delays;
- lower application response time by minimizing inter-function communications through remote storage.

$TT_{f_{N,P}}$	function response time (total execution time)
$WT_{f_{N,P}}$	duration of the scheduling decision, including queuing time
$CST_{f_{N,P}}$	initialization time for the function, including cold start
$ET_{f_{N,P}}$	execution time of the function on the platform
$TC_{f_{N,P}}$	sum of total times of all tasks in platform’s queue
$QP_{f_{N,P}}$	QoS penalty for a given scheduling
$HP_{f_{N,P}}$	hardware price for a given allocation or scheduling
$HP_{f_{N,P}}$	energy consumption for a given scheduling
$IFC_{f_{N,P}}$	inter-function communications

$CST_{f_{N,P}}$  is impacted by either pulling the function image from a repository or deploying from the local cache and increases function response time and thus potential  $QP_{f_{N,P}}$  penalties due to missed deadlines:

$$TT_{f_{N,P}} = WT_{f_{N,P}} + CST_{f_{N,P}} + ET_{f_{N,P}}$$

$TC_{f_{N,P}}$  has to be maximized for functions of a same application so as to allow node-local communications between functions of an application and minimize the use of remote storage:

$$ET_{app_{N,P}} = \sum ET_{f_{app_{N,P}}} + \sum IFC_{app}$$

$$IFC_{app} = \sum IFC_{local_{f_{N,P}}} + \sum IFC_{remote_{f_{N,P}}}$$

## 6 – Perspectives

- We are extending the simulator developed for [4] to integrate our updated cost model to the resources allocation and task scheduling policy;
- We will evaluate the relevance of the full policy using traces derived from Azure datasets [7];
- We consider exploring the use of machine learning algorithms to introduce workload prediction and/or to characterize workloads during execution instead of during an offline measurement phase.

[1] Istemi Ekin Akkus et al. “SAND: Towards High-Performance Serverless Computing”. In: USENIX ATC ’18. 2018.

[2] Rajkumar Buyya et al. “SLA-oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions”. In: *CSC ’11*. IEEE, 2011.

[3] Edson Horta et al. “Xar-Trek: Run-Time Execution Migration among FPGAs and Heterogeneous-ISA CPUs”. In: *Middleware ’22*. ACM, 2021.

[4] Vincent Lannurien et al. “HeROfake: Heterogeneous Resources Orchestration in a Serverless Cloud – An Application to Deepfake Detection”. In: *CCGRID ’23* (2023).

[5] Vincent Lannurien et al. “Serverless Cloud Computing: State of the Art and Challenges”. In: *Serverless Computing: Principles and Paradigms*. 2023.

[6] Johann Schleier-Smith et al. “What Serverless Computing is and Should Become: The next Phase of Cloud Computing”. In: *Commun. ACM* (2021).

[7] Mohammad Shahrads et al. “Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider”. In: USENIX ATC’20. 2020.