Un'euristica distribuita per il controllo dei flussi di richieste in ambiente FaaS

Un approccio basato su simulazione

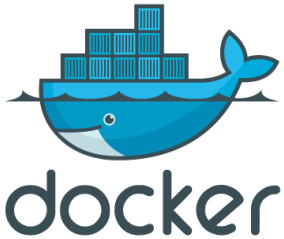
Tesi magistrale di Mattia Vincenzi - 860579

Anno Accademico 2020-2021

Relatore: *Dr. Michele Ciavotta*

Co-relatore: *Dr. Alessandro Tundo*

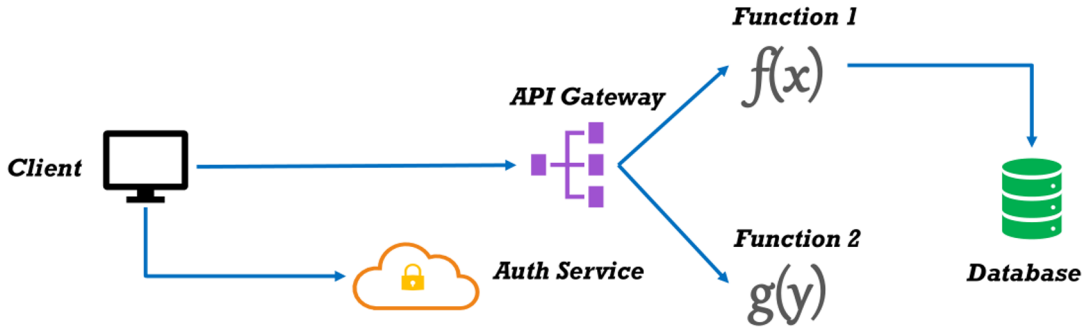
Che cos'è FaaS?



Function-as-a-Service (FaaS) - paradigma di sviluppo di applicazioni la cui logica di backend è scritta per mezzo di funzioni:

- A grana fine e limitate a singole business unit
- Eseguite in funzione del verificarsi di eventi *(event-triggered)*
- Eseguite in **container effimeri** e **stateless**
  - **Container** come tecnologia abilitante

Esempio di applicazione Serverless



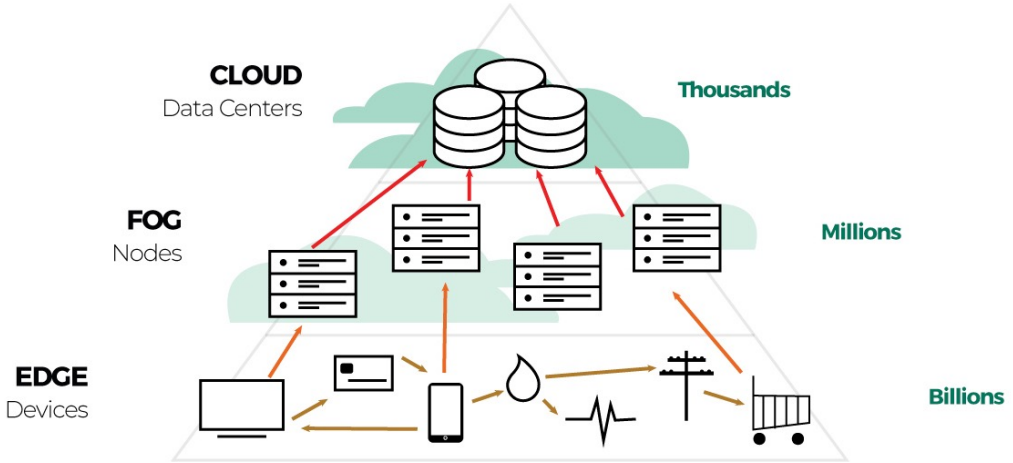
Edge Computing

EDGE COMPUTING

Forma di elaborazione eseguita in **sede** o in **prossimità** di una particolare origine dati

Tra i vantaggi:

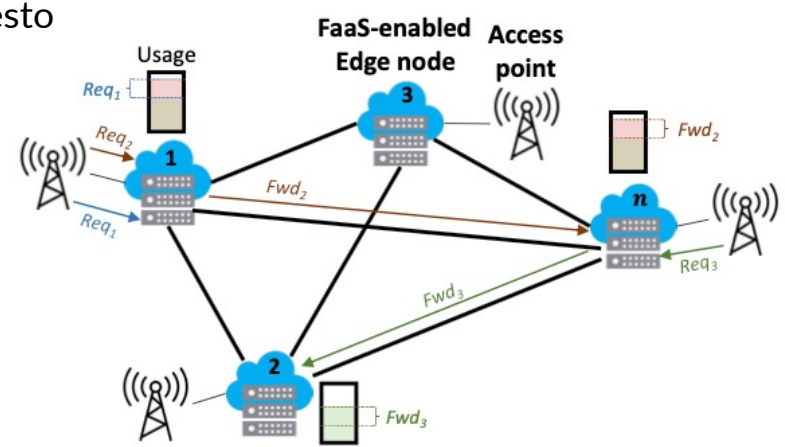
- Riduzione dei tempi di risposta
- Diminuzione del traffico verso il cloud
- Privacy e sicurezza
- Miglior scalabilità delle applicazioni



Problema affrontato

Scenario: erogazione di servizi (realizzati secondo modello **FaaS**) in maniera **geograficamente distribuita** su più piattaforme *micro-cloud* (nodi di Edge)

- Necessità di federare i nodi di Edge
- Necessità di **bilanciare il carico** tra i nodi in un contesto fortemente dinamico ed eterogeneo

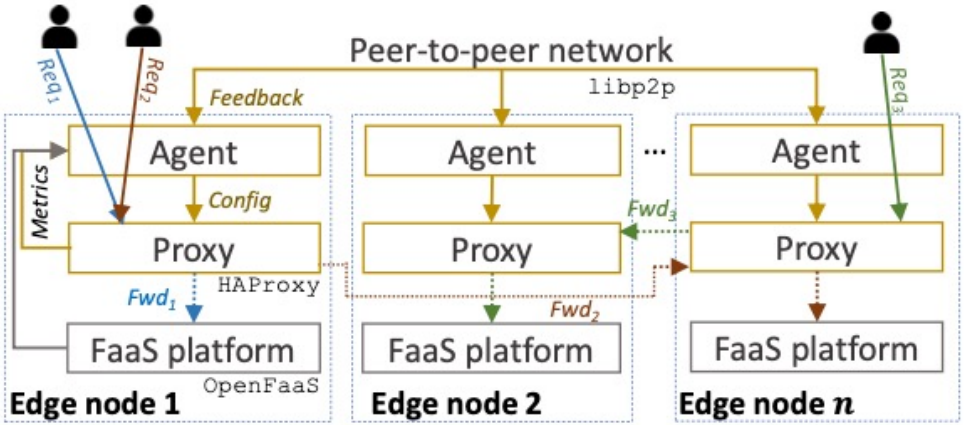


DFaaS

DFaaS [1] (*Decentralized FaaS*) – un'architettura decentralizzata basata su FaaS, progettata per bilanciare autonomamente il carico di traffico tra nodi edge federati

Il prototipo realizzato è stato utilizzato per:

- Monitoring e raccolta dati
- Verificare validità dell'algoritmo



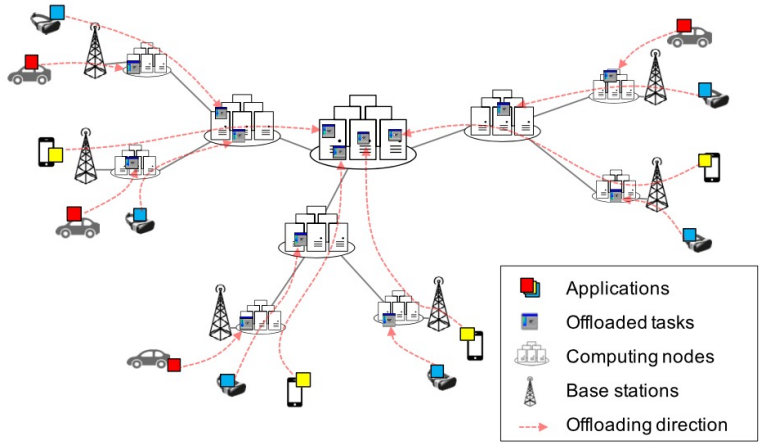
[1] M. Ciavotta, D. Motterlini, M. Savi, and A. Tundo, "Dfaas: Decentralized function-as-a-service for federated edge computing" in 2021 IEEE 10th International Conference on Cloud Networking (CloudNet), pp. 1–4, 2021.

Approccio Utilizzato

Problema: bilanciamento del carico in ambiente decentralizzato *(Edge Computing)*

Approccio utilizzato:

1. Implementazione di un **Simulatore**
2. Definizione **Strategia Empirica**
3. Creazione di un sistema di *Monitoring*
   - Raccolta dati su DFaaS
4. Validazione e confronto con altre strategie



Simulatore (1/3)

Problema: Necessità di avere a disposizione un ambiente separato in cui poter validare, testare e confrontare i diversi algoritmi di **bilanciamento del carico**

- Realizzazione di un **simulatore**

Progettato sfruttando i principi di una buona *Architettura del Software*:

- *Modularizzando* le componenti
- Attribuendo ad ognuna di esse una *singola responsabilità*
- Generalizzandolo per facilitarne l'espansione

Vantaggi:

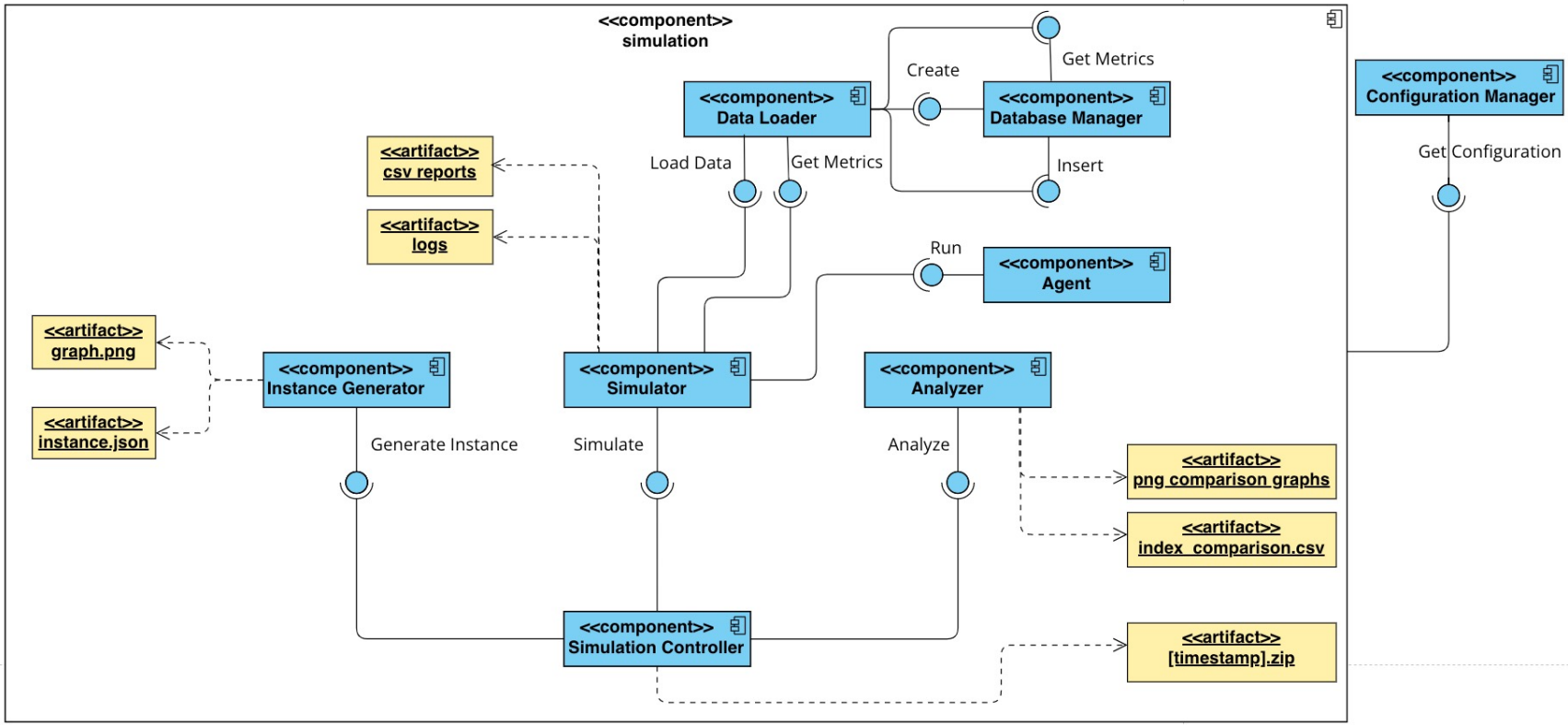
- **Manutenibilità**
- **Riusabilità**
- **Evolvibilità**

Il processo di simulazione deve aderire a tre *macro-fasi*:

- **Generazione dell'istanza da simulare** (numero nodi, topologia di rete, repliche e carico per ogni funzione) per ogni *step* di simulazione
- **Simulazione**. Reperimento delle *metriche* per ogni configurazione da simulare (da *database*) ed effettiva simulazione del bilanciamento del carico
- **Analisi** dei risultati prodotti dalla simulazione e calcolo degli *indici di confronto*

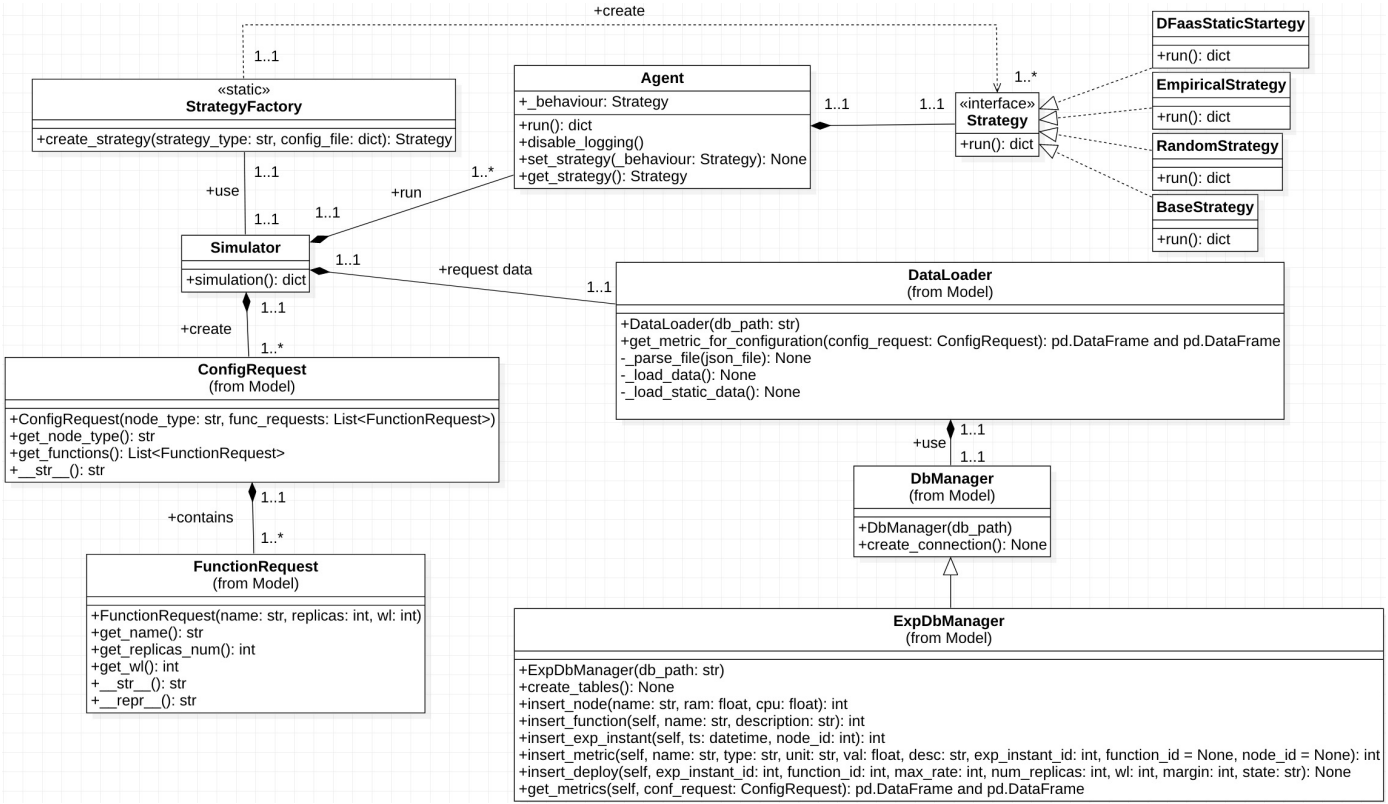
Simulatore (2/3)

Diagramma UML delle **componenti**

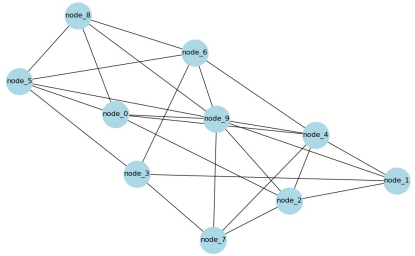


Simulatore (3/3)

Diagramma UML delle **classi** del *simulator*



Strategia Empirica (1/2)

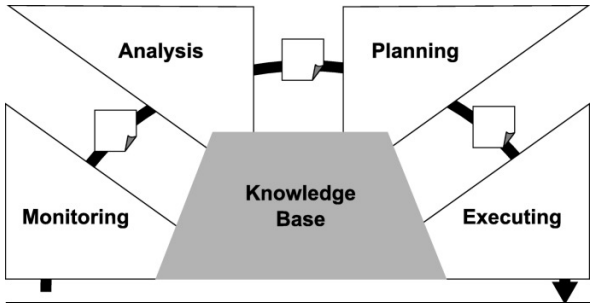


Output. Pesi di inoltro [1] delle richieste verso i nodi del vicinato, per ogni funzione **sovraccaricata**

Euristica distribuita, *asincrona*, che basa il calcolo dei pesi di inoltro su un insieme di **metriche**, ovvero evidenze raccolte sul campo, che sintetizzano lo stato di salute dei nodi vicini e delle funzioni su di essi in esecuzione

Control Loop dell'agente eseguito ciclicamente, progettato ispirandosi al **MAPE-K** *feedback-loop*

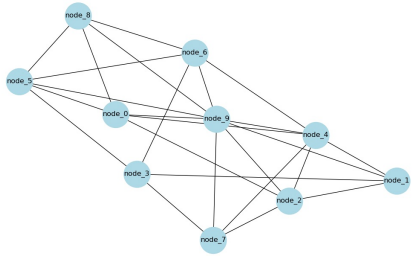
$$w_{func\_name \rightarrow node_{id}} = \sum_{h_k, w_k \; \forall \; k \in \{metric\_set\}} h_k w_k$$



Output: {«funca»: node1: **10%**, node2: **90%**}

[1] Pesi di inoltro: Rappresentano la percentuale di richieste inviate da ogni agente verso gli altri agenti che può raggiungere. Vengono calcolati per ogni funzione nello stato di sovraccarico. Rappresentano, per ogni funzione, una distribuzione di probabilità.

Strategia Empirica (2/2)



Control Loop dell'agente eseguito ciclicamente:

- **Monitor**. Fase di raccolta delle *metriche* relative allo stato del nodo e delle funzioni
  - AFET, utilizzo di RAM e CPU, numero di repliche, rateo di invocazione, max_rate, ecc.
  - Determinazione dello stato di ogni funzione: **overload** o **normal**
- **Exchange**. Invio delle *metriche* raccolte ai nodi del vicinato
  - La comunicazione non deve avvenire in maniera *sincrona*
- **Analyze**. Per ogni funzione **overloaded** è necessario bilanciare il carico verso i nodi del vicinato, con la stessa funzione nello stato di **normal**. Algoritmo diviso in due *macro-fasi*:
  - Calcolo dei pesi relativi alle singole metriche: viene attribuito un peso ad ogni *metrica* per ogni nodo del vicinato (es: $w_{AFET(node_1)}$)
  - Aggregazione dei pesi per ogni nodo del vicinato: aggregazione di tutti i pesi delle *metriche* verso uno specifico nodo (es: $w_{funca \rightarrow node_1}$)
- **Plan**. Aggiunta di rumore probabilistico ai pesi calcolati nella fase precedente
  - Con una probabilità del 50% viene aggiunto del rumore ai pesi → per evitare *convergenza* del traffico
- **Execute**. Applicazione dei pesi sul file di configurazione del *proxy*

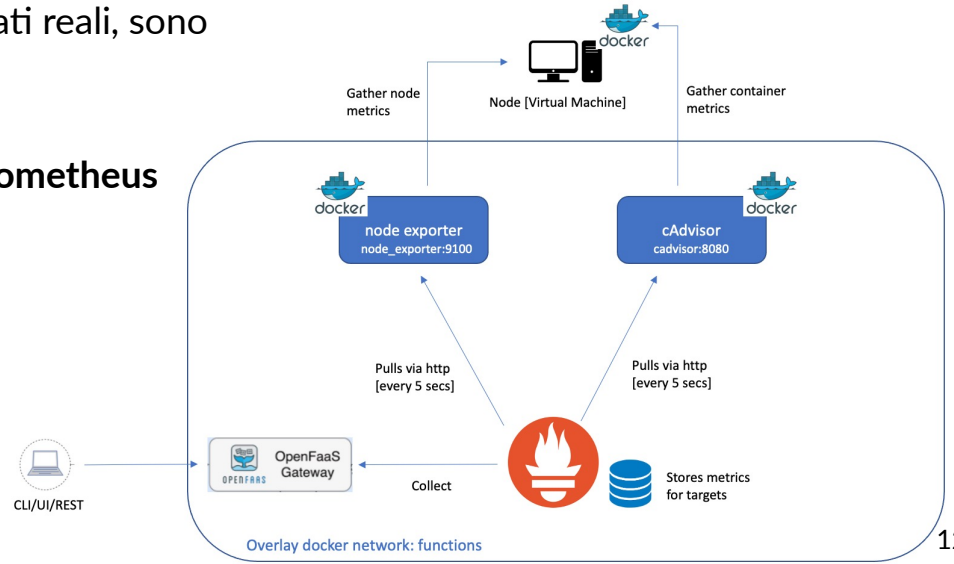
Architettura Monitoring

La **strategia empirica** basa il calcolo dei pesi su un insieme di *metriche*

Al fine di realizzare una simulazione basata su dati reali, sono stati raccolti sul prototipo di DFaaS

Ampliato l'insieme delle metriche raccolte da **Prometheus** tramite l'utilizzo di due **exporters**:

- Node exporter
- cAdvisor



Raccolta dati

Utilizzando l'architettura di *monitoring* è stato predisposto uno scenario di raccolta delle *metriche*, sottoponendo un *singolo* nodo del prototipo di DFaaS a diverse situazioni di deployment e carico

Nota: la fase di generazione degli scenari di simulazione è stata *automatizzata*

Numero massimo di repliche

	funca	qrcode	ocr
node of type 1	5	2	1
node of type 2	5	2	1
node of type 3	5	2	1

Numero massimo di invoc. x replica

	funca	qrcode	ocr
node of type 1	15	2	1
node of type 2	20	5	3
node of type 3	30	10	5

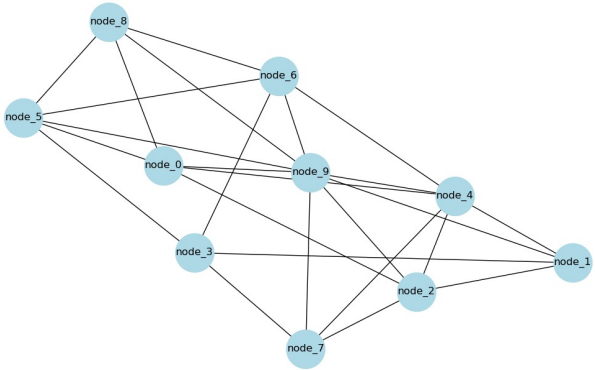
	RAM (Gb)	# of (virtual) CPUs
node of type 1	2	1
node of type 2	4	2
node of type 3	6	4

Generazione di scenari di carico

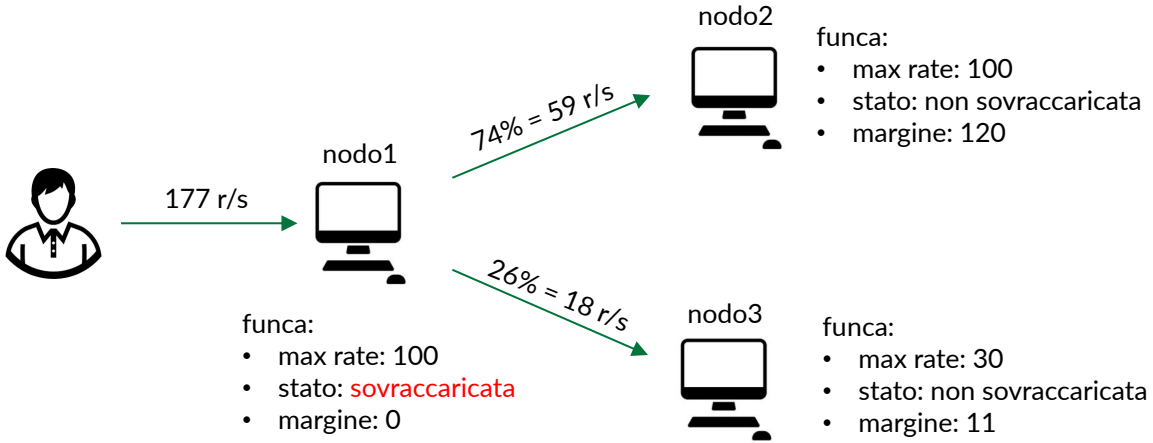
Report Json contenente le metriche raccolte durante l'esperimento condotto

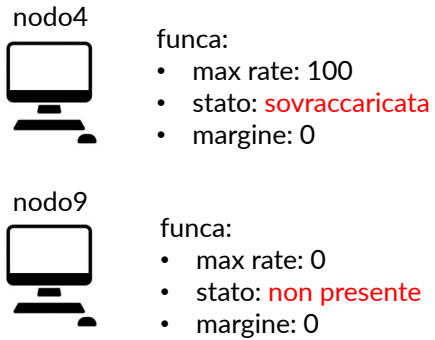
Validazione

Di seguito viene riportata un esempio di istanza simulata che dimostra la correttezza della **strategia empirica**



Vicinato nodo1: 2, 3, 4, 9





Comparazione risultati (1/3)

Strategie di distribuzione del carico utilizzate per comparare i risultati:

- **Base Strategy**. Nessuna logica di distribuzione del carico
- **Random Strategy**. Inoltro casuale delle richieste
- **DFaaS Static Strategy**. Strategia implementata sul prototipo di DFaaS

Comparazione eseguita in termini dei seguenti **indici**:

- **Success Rate**
- **Success Rate** (durante periodo di stress)
- **Numero totale di rigetti**

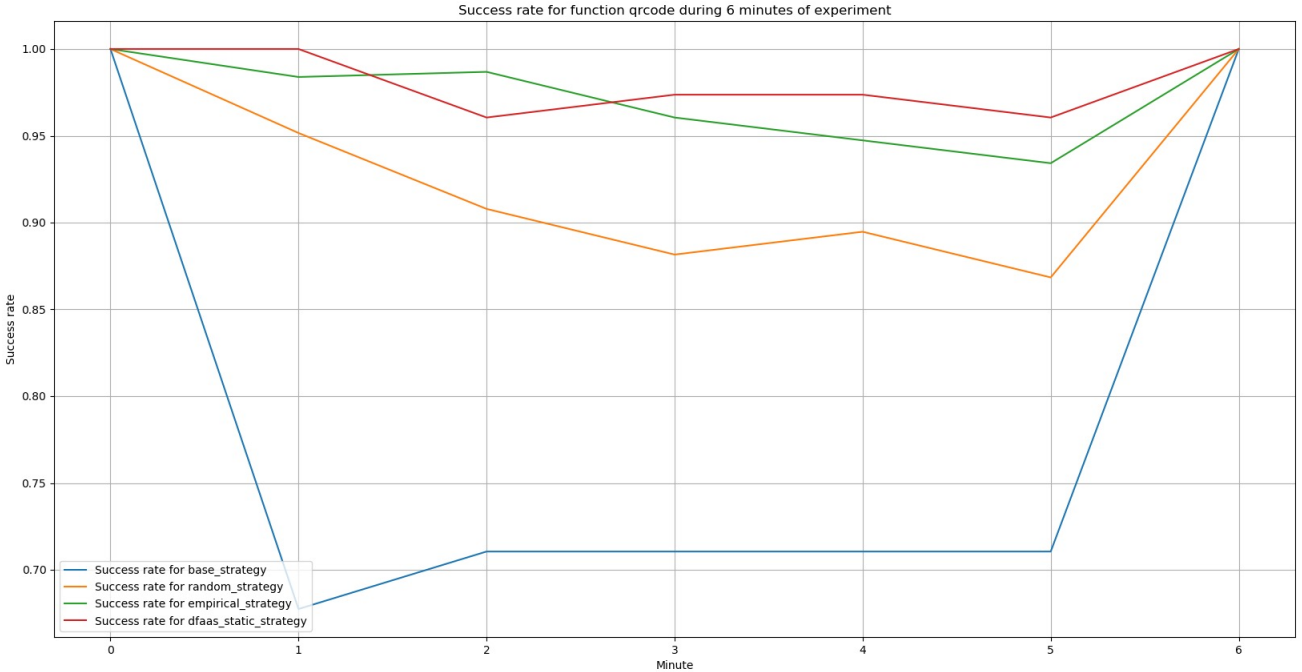
Di tali indici ne viene calcolata: **Media**, **Varianza**, **Mediana** e **Percentile al 90%**

Confronto delle strategie di bilanciamento al termine di 5 simulazioni

Strategies	Base	Random	Empirical	DFaaS static
Mean SR	68.18%	82.22%	92.63%	94.10%
SR variance	632.78	258.93	72.67	51.90
Mean SR (stress period)	55.42%	75.11%	89.69%	91.74%
SR variance (stress period)	318.02	184.97	70.99	52.96
Tot reject number	75540	46956	16560	11844

Comparazione risultati (2/3)

Nel seguente grafico viene riportato l'andamento del **success rate** per quanto concerne la **qrcode**



Comparazione risultati (3/3)

Indici calcolati e *mediati* su tutte le funzioni per il confronto

- **Strategia Empirica** e **DFaaS static strategy** si equivalgono in termini di risultati
- Si distaccano dalle strategie di **baseline**, soprattutto durante il periodo di **stress**

Strategies	Base	Random	Empirical	DFaaS static
Mean SR	68.18%	82.22%	92.63%	94.10%
SR variance	632.78	258.93	72.67	51.80
Mean SR (stress period)	55.42%	75.11%	89.69%	91.74%
SR variance (stress period)	318.02	184.97	70.99	52.96
Tot reject number	75540	46956	16560	11844
Reject num 90% percentile	13800	8880	3060	2340

Tanti esperimenti eseguiti, in situazioni di congestione, in cui si è fatto variare il numero di agenti e la topologia di rete → differenza massima in termini di **success rate** dell'1% o 2%

- Vantaggi della **strategia empirica**: **flessibilità** e **adattabilità** a scenari di bilanciamento differenti, basa il calcolo dei pesi di inoltro su diverse *metriche*

Sviluppi futuri

- Implementare altri algoritmi di distribuzione del carico e confronto con quelli attualmente implementati
  - Il **simulatore** è estendibile, rende semplice implementare altri algoritmi
- Studiare la combinazione migliore dei pesi, attribuiti alle *metriche*, per ottimizzare la distribuzione del carico a seconda dell'obiettivo che si vuole perseguire (es. utilizzo CPU, RAM, ecc.)
  - I pesi attribuiti alle *metriche* potrebbero essere calcolati dinamicamente, ad esempio utilizzando un algoritmo di **Machine Learning**

Grazie per l'attenzione