

# Un prototipo per lo scheduling di funzioni basato su analisi di costo in piattaforme serverless

Simone Boldrini

Alma Mater Studiorum - Università di Bologna  
Facoltà di Scienze

14 Marzo 2024

**Obiettivo:** Sviluppare un compilatore per funzioni serverless che sfrutti tecniche di analisi di costo per ottimizzare l'esecuzione.

- Definizione di HLCostLan

**Obiettivo:** Sviluppare un compilatore per funzioni serverless che sfrutti tecniche di analisi di costo per ottimizzare l'esecuzione.

- Definizione di HLCostLan
- Analisi del programma

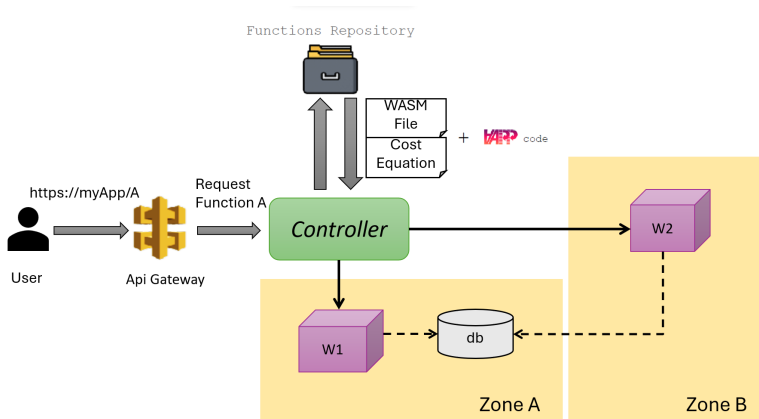
**Obiettivo:** Sviluppare un compilatore per funzioni serverless che sfrutti tecniche di analisi di costo per ottimizzare l'esecuzione.

- Definizione di HLCostLan
- Analisi del programma
- Generazione equazioni di costo

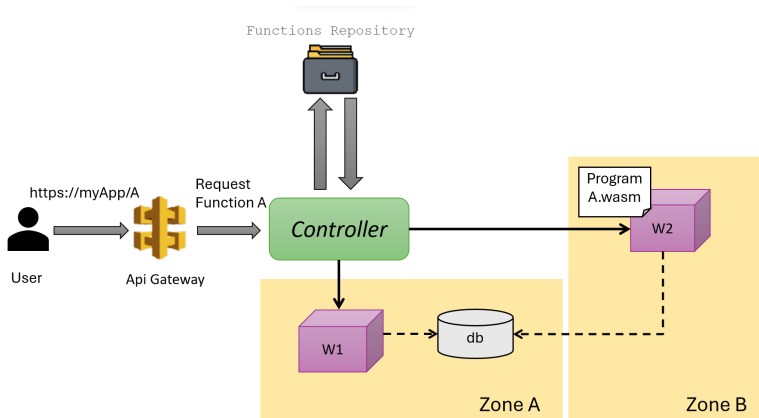
**Obiettivo:** Sviluppare un compilatore per funzioni serverless che sfrutti tecniche di analisi di costo per ottimizzare l'esecuzione.

- Definizione di HLCostLan
- Analisi del programma
- Generazione equazioni di costo
- Generazione del codice WASM

# Esempio cAPP + CostCompiler



# Esempio cAPP + CostCompiler



# Esempio di funzioni in HLCostLan

```
1 struct Params {  
2     address: array[int],  
3     payload: any,  
4     sender: string  
5 }  
6 service PremiumService : (string) -> void;  
7 service BasicService : (any) -> void;  
8 (isPremiumUser: bool, par: any) => {  
9     if ( isPremiumUser ) {  
10         call PremiumService("test");  
11     } else {  
12         call BasicService( par);  
13     }  
14 }  
15
```

Listing 1: Listing8



- Il compilatore ritorna un insieme di equazioni di costo, che sono adeguate per un Cost-Analyzer.
- Il Cost-Analyzer calcola il costo di esecuzione di un programma in base alle equazioni di costo.
- Il Cost Analyzer scelto è PUBS
- I risultati vengono presi dal sistema cAPP permettendo l'allocazione degli scheduling ottimali per le piattaforme serverless.

# Analisi di costo

Un'analisi di costo è fortemente dipendente dal modello di costo preso in considerazione:

- **Costo di esecuzione**: il costo di esecuzione di una funzione
- **Costo di allocazione**: il costo di allocazione di una variabile nell'heap
- **Costo di latenza**: il costo di latenza nell'invocazione di una funzione

I vantaggi delle equazioni di costo:

- Sono **indipendenti** dal linguaggio di programmazione
- Possono rappresentare diverse classi di **complessità**
- Possono catturare una varietà di nozioni non banali di risorse.

# Esempio Cost Equation

```
1 eq(main(P,ISPREMIUMUSER0,B),0,[if9(ISPREMIUMUSER0,P,B)],[]).  
2 eq(if9(ISPREMIUMUSER0,P,B),nat(P),[],[ISPREMIUMUSER0=1]).  
3 eq(if9(ISPREMIUMUSER0,P,B),nat(B),[],[ISPREMIUMUSER0=0]).
```

Listing 2: Equazioni di costo per Listing 8

Eseguendo PUBS infine otteniamo:

*pubs\_aux\_entry*(A,B,C) – THE MAIN ENTRY

\* Non Asymptotic Upper Bound:  $\max([\text{nat}(C), \text{nat}(A)])$

# Generazione del codice WASM

Una volta ottenute le equazioni di costo, abbiamo sviluppato un interprete per la generazione del codice WASM.

**WebAssembly** rappresenta una tecnologia versatile e potente che offre un'alternativa efficace alle tradizionali soluzioni di esecuzione di codice.

Le istruzioni **Wasm** si distinguono dalle istruzioni di un processore reale in quanto sono progettate per l'esecuzione in un ambiente virtuale.

# Esempio HLCostLan

```
1  fn svc(i: int, n:int) -> any{  
2      return n * i  
3  }  
4  (len : int) => {  
5      return svc(len,4)  
6  }  
7
```

Listing 3: Esempio di funzione HLCostLan, Listing15

# Equivalente WAT

```
1  (func $svc (param $i i32) (param $n i32) (result i32)
2  (local $res i32)
3  (if(i32.eq
4  (local.get $i)
5  (local.get $n))
6  (then
7  (i32.const 1)
8  (local.set $res))
9  (else
10 (i32.const 0)
11 (local.set $res))
12 )(local.get $res))
13
14 (func $main (export "main") (param $a i32) (param $b i32) (result
15 i32)
16 (local.get $a)
17 (local.get $b)
18 (call $svc)
19 )
```

# Vantaggi e Svantaggi di WASM

## Vantaggi:

- **Velocità:** progettato per essere eseguito in modo efficiente su hardware moderno.
- **Sicurezza:** progettato per essere eseguito in un ambiente sandbox.
- **Portabilità:** progettato per essere eseguito su qualsiasi piattaforma.

## Svantaggi:

- **Complessità:** linguaggio a basso livello, quindi è più difficile da scrivere e leggere rispetto ad altri linguaggi.
- **Mancanza di supporto:** è relativamente nuovo, quindi non tutti i browser e le piattaforme lo supportano.



# Conclusioni

- È stato sviluppato un **interprete prototipale** attraverso il quale è possibile eseguire programmi scritti in HLCostLan, un linguaggio di programmazione appositamente progettato per l'analisi e l'ottimizzazione dei costi computazionali.
- L'interprete offre anche la possibilità di derivare **l'equazione di costo** associata a tali programmi, che viene poi calcolata attraverso il tool PUBS, per descrivere in modo formale il carico computazionale generato dal programma.
- CostCompiler, infine, offre inoltre la capacità di generare automaticamente il corrispondente codice **WebAssembly**.

Alcuni possibili sviluppi futuri che potrebbero arricchire ulteriormente il nostro lavoro includono:

- Integrazione con Kubernetes
- Estendere il linguaggio HLCostLan
- Condurre studi sperimentali e valutazioni empiriche per testare l'efficacia del sistema in scenari realistici di utilizzo