
Università degli Studi di Salerno
Dipartimento di Informatica

Tesi di Laurea di I livello in
Informatica

# Implementazione di algoritmi genetici in linguaggio FLY

**Relatore**
Prof. Vittorio Scarano
**Correlatore**
Prof. Carmine Spagnuolo
Dott. Giuseppe D'Ambrosio

**Candidato**
Andrea Di Pierno
**Mat.** 0512105724

Anno Accademico 2020-2021

# Abstract

Il Cloud Computing è considerato fondamentale per la creazione di infrastrutture di calcolo grazie alle risorse messe a disposizione sotto forma di servizi. L'alto numero di Cloud Provider presenti sul mercato ha dato origine al paradigma del multi-cloud, che prevede l'integrazione all'interno della stessa architettura di più ambienti Cloud. I vantaggi che fornisce comprendono la possibilità di ridurre la dipendenza dal singolo provider, avere una maggiore scelta di servizi e tolleranza ai guasti. L'eterogeneità del mercato, tuttavia, implica forti differenze anche negli strumenti di accesso dei vari provider per usufruire dei loro servizi. Ciò rende l'utilizzo del multi-cloud un processo complesso per i meno esperti, creando la necessità di strumenti che facilitino il suo impiego. FLY è un Domain-Specific Language per il calcolo scientifico su multi-cloud che ha come obiettivo la semplificazione dello sviluppo di applicazioni che sfruttino tale paradigma. Il linguaggio FLY mette a disposizione una serie di costrutti che astraggono all'utente l'interazione con il provider, permettendogli di utilizzare servizi di diversi ambienti Cloud senza dover conoscere librerie e strumenti proprietari. Essendo FLY ancora in fase di sviluppo, i programmi di esempio scritti in FLY sono ancora pochi e poco complessi, rendendo necessaria l'introduzione di nuovi che fungano da guida ai nuovi sviluppatori. Questo lavoro di tesi ha come obiettivo l'implementazione di una serie di algoritmi genetici scritti in FLY, presentandone la progettazione e lo sviluppo. Gli algoritmi genetici sono algoritmi ispirati ai processi biochimici descritti nella Teoria dell'Evoluzione di Charles Darwin, per la risoluzione di problemi di ottimizzazione per cui non esistono soluzioni di complessità polinomiale o che sono particolarmente esosi in termini di risorse. Il successo degli algoritmi genetici deriva dalla loro relativa semplicità concettuale, pur essendo non banali da implementare. Per questo motivo, fornire degli esempi di programmi FLY che impiegano algoritmi genetici consente di dimostrare le funzionalità e le potenzialità del linguaggio oltre a risultare un buon punto di partenza per gli utenti che si approcciano a FLY per la prima volta.

Questa tesi è stata sviluppata in 

# Indice

# Capitolo 1

# Introduzione

Negli ultimi anni il Cloud Computing è divenuto uno dei paradigmi più impiegati per la creazione di infrastrutture di calcolo. Il suo successo è dovuto alla possibilità di ottenere risorse computazionali sotto forma di servizi, permettendo all'utenza di eliminare la necessità di avere un proprio centro di calcolo e di evitare i costi collegati all'acquisto, configurazione e manutenzione di quest'ultimo. L'impiego di questo paradigma ha permesso di ottenere numerosi vantaggi che nell'ultimo decennio hanno permesso al Cloud Computing di acquisire sempre più importanza. Un vantaggio paticolarmente rilevante è l'applicazione del modello di costo pay-as-you-go, grazie al quale è possibile pagare soltanto per le risorse effettivamente impiegate.
L'evoluzione del Cloud Computing ha portato alla nascita dei modelli di servizio **Infrastructure as a Service (IaaS)**, **Platform as a Service (PaaS)** e **Software as a Service (SaaS)**, che oggi sono largamente conosciuti ed utilizzati. Nello specifico, il modello IaaS offre la possibilità di ricreare su Cloud gli ambienti di esecuzione utilizzati nelle proprie infrastrutture di calcolo, di fatto eliminando la necessità di queste ultime, oltre che i costi ad esse collegati. Questi modelli di servizio, tuttavia, non consentono di sfruttare pienamente le potenzialità del Cloud Computing, poiché richiedono comunque una forma di gestione delle risorse da parte dell'utente. In mancanza di un paradigma che permettesse di sfruttare a pieno le caratteristiche del Cloud, gli sviluppatori dovevano gestire le risorse allo stesso modo di come avrebbero fatto con un'infrastruttura fisica. La volontà di sfruttare a pieno le potenzialità di questo paradigma ha portato alla nascita del modello di servizio Function as a Service (FaaS). Anche noto come Serverless Computing, il modello FaaS è la naturale evoluzione dell'architettura basata su microservizi, basato sul concetto di funzione, ovvero una porzione di codice molto piccola

eseguita su Cloud in maniera trasparente. Ciò ci consente di ottenere un forte disaccoppiamento ed un'elevata scalabilità, perfettamente in linea con le moderne esigenze computazionali. Le caratteristiche peculiari di questo nuovo modello di servizio si sono rivelate fondamentali per lo sviluppo di branche dell'informatica note per la rapidità di crescita dei numeri con cui lavorano, come il mondo mobile e l'Internet of Things.
La repentina crescita del mercato del Cloud ha portato alla nascita di numerosi Cloud Service Provider, ovvero fornitori di servizi Cloud dotati di una propria infrastruttura e dei propri servizi. I provider maggiormente conosciuti, come ad esempio AWS, Azure e Google Cloud Platform, si contendono il mercato dei servizi Cloud attraverso un'offerta di servizi in costante evoluzione. La mutevolezza dell'offerta dei provider, che si arricchisce costantemente di nuovi e peculiari servizi, ha spinto gli sviluppatori a creare soluzioni in grado di unire le piattaforme Cloud dei vari provider in una singola architettura. Questo approccio, noto con il nome di multi-cloud, permette di sfruttare al meglio i vantaggi di ogni provider e di ottenere non solo maggiore affidabilità, ma anche un miglior rapporto qualità-prezzo ed una riduzione della dipendenza dal singolo provider, aspetto particolarmente critico della computazione Cloud, meglio noto come Vendor Lock-in. L'introduzione del multi-cloud ha però portato alla luce ulteriori criticità del Cloud Computing, come l'eterogeneità delle **API (Application Programming Interface)** fornite dai provider per la fruizione dei proprio servizi.
In questo capitolo verrà descritto il paradigma del Cloud Computing, prestando attenzione ai diversi modelli di servizio disponibili ed in particolare al modello Function as a Service e al paradigma multi-cloud.

## 1.1 Cloud Computing

Esistono molte definizioni diverse di Cloud Computing, fornite sia da Cloud Service Provider che da sviluppatori. Una definizione data da Amazon Web Services, uno dei leader tra i Cloud Service Provider, è la seguente:
*"Il cloud computing consiste nella distribuzione on-demand delle risorse IT tramite Internet, con una tariffazione basata sul consumo. Piuttosto che acquistare, possedere e mantenere i data center e i server fisici, è possibile accedere a servizi tecnologici, quali capacità di calcolo, storage e database, sulla base delle proprie necessità affidandosi a un fornitore cloud."* [50].
Il Cloud permette quindi di ottenere risorse computazionali in modo semplice e rapido, consentendo una gestione semplificata dei carichi di lavoro. Il Cloud supporta modelli di programmazione ridondanti, capaci di gestire gli errori

in autonomia e altamente scalabili. Inoltre, il Cloud consente di avere pieno controllo sulle risorse che si utilizzano, permettendo di bilanciarne le quantità assegnate attraverso un monitoraggio in tempo reale dei carichi di lavoro. Grazie al Cloud Computing, le aziende e gli sviluppatori possono noleggiare risorse computazionali in maniera semplice e veloce da grandi aziende che possiedono centri di calcolo di elevata potenza e che mettono a disposizione parte di queste ultime sotto forma di servizi. Ciò permette di evitare i grandi investimenti necessari per creare una propria infrastruttura ed offre agli utenti la possibilità di eseguire applicazioni da qualsiasi posto nel mondo con costi molto bassi.

Le applicazioni di calcolo scientifico, ad esempio, possono trarre numerosi benefici dall'impiego del Cloud Computing. Tali applicazioni sono tipicamente sviluppate utilizzando linguaggi general-purpose oppure framework o linguaggi paralleli come il C, Java, Python e così via, inoltre, richiedono codice scritto appositamente che implementi algoritmi per risolvere problemi specifici. In aggiunta, i problemi di Calcolo Scientifico sono di solito computing-intensive e richiedono elevata potenza computazionale ottenibile da sistemi distribuiti come cluster o supercomputer. L'infrastruttura Cloud, invece, fornisce servizi general-purpose accessibili attraverso endpoint o API. Tuttavia, nonostante molti Cloud provider abbiano inglobato nei loro servizi *Apache Hadoop* e *MapReduce*, molti problemi computing-intensive risultano inadatti ad essere eseguiti su Cloud. Inoltre, nonostante il Cloud offra soluzioni con elevata scalabilità, molto spesso migrare applicazioni di Calcolo Scientifico su modelli IaaS o PaaS risulta essere complesso e tedioso. Ciò va a precludere agli sviluppatori di ambito scientifico di sfruttare a pieno la scalabilità e l'efficienza del Cloud Computing e genera costi elevati [17].

Alla base del Cloud Computing troviamo la virtualizzazione, tecnologia che consente di gestire al meglio le capacità computazionali dei sistemi. Questa tecnologia consente infatti di astrarre le componenti hardware e di fornirle ai software sotto forma di risorse virtuali. Grazie alla virtualizzazione, le aziende che possiedono grandi centri di calcolo possono suddividere le risorse fisiche non impiegate dalle macchine in risorse virtuali più piccole, che possono essere quindi assegnate facilmente agli utenti che ne fanno richiesta. L'affitto di queste risorse virtuali costituisce la base delle tecnologie Cloud, portando numerosi vantaggi sia ai provider che agli sviluppatori. Il consumatore ha la possibilità di sfruttare le risorse di cui ha bisogno per il tempo necessario, senza dover creare una propria infrastruttura, mentre il provider può ricavare profitti fornendo le risorse non impiegate agli utenti. Grazie alla

virtualizzazione delle risorse è inoltre possibile gestire in maniera elastica le risorse assegnate, in modo da richiederne una quantità maggiore durante i picchi di carico ed una quantità minore per i periodi di lavoro meno intenso. Questo meccanismo consente di ottenere un notevole risparmio all'utente ed un'ottimizzazione nell'impiego delle risorse al provider. L'utilizzo del Cloud Computing risulta più vantaggioso rispetto a quello dei sistemi on-premise, ovvero le macchine di proprietà dell'utente o dell'azienda, poiché permette di eliminare tutti i costi collegati all'acquisto, alla manutenzione e all'aggiornamento delle infrastrutture, oltre che ad un miglioramento dell'efficienza energetica e ad una vasta disponibilità dei servizi, anche in caso di guasti.
Le architetture di Cloud Computing possono essere suddivise in tre modelli.

**Public Cloud** Appartenente a grandi aziende come Amazon, Microsoft e Google, che offrono servizi accessibili tramite Internet e utilizzabili mediante apposite interfacce pubbliche. L'utilizzo è consentito a qualsiasi utente abbia pagato per i servizi che intende utilizzare. Grazie all'alta scalabilità ed al modello di costo pay-as-you-go, permette di pagare solo per le risorse effettivamente utilizzate, disponibili in quantità virtualmente illimitata. I Cloud pubblici permettono di evitare gli investimenti per gestire le infrastrutture, offrono alta flessibilità e promuovono la standardizzazione, tuttavia dati e applicazioni sono in un sistema in possesso di un'azienda esterna. Ciò richiede un elevato livello di fiducia del cliente verso il provider.

**Private Cloud** Un Cloud privato è controllato e gestito completamente dall'azienda proprietaria che ne fornisce l'accesso solo alla propria rete interna. Fornisce un'alta personalizzazione e flessibilità poiché l'infrastruttura viene progettata dall'azienda che lo utilizza, permettendo di ottenere alta efficienza e prestazioni. Un Cloud privato rappresenta un ottimo modello per ciò che concerne la privacy e la sicurezza, in quanto non progettato per la vendita di servizi all'esterno. I servizi disponibili sono utilizzabili attraverso interfacce private.

**Hybrid Cloud** Integra i servizi di un Cloud privato con quelli forniti da un Cloud pubblico. L'accesso avviene attraverso delle interfacce private, poiché non è progettato per la vendita. La combinazione dei due modelli permette di ottenere una maggiore flessibilità e sicurezza. Si ha infatti la possibilità di sfruttare il Cloud privato per memorizzare dati sensibili e applicazioni critiche e di impiegare le risorse del Cloud pubblico. Risulta essere il miglior compromesso tra elevato controllo e alta scalabilità.

### 1.1.1 Modelli di servizio

Esistono quattro principali modelli di servizi offerti dai Cloud Provider, ciascuno dei quali offre un diverso livello di astrazione in base alle necessità degli utenti. Di seguito verranno descritti i primi tre modelli di servizio, mentre il quarto verrà descritto nel paragrafo 1.1.2.

**Infrastructure as a Service (IaaS)** Il modello IaaS permette di ottenere macchine virtuali di cui è possibile specificare potenza computazionale, capacità di storage e caratteristiche del network. Tale modello infatti, fornisce all'utente le infrastrutture con cui lavorare. Attraverso questo modello è possibile configurare una macchina con elevate prestazioni o un cluster di grandi dimensioni per mandare in esecuzione una particolare applicazione. L'utente, tuttavia, non ha il controllo dell'infrastruttura sottostante, quella che permette alla macchina di funzionare, ma ha la responsabilità di gestirla in ogni suo aspetto, dal sistema operativo alle applicazioni che esegue.

**Platform as a Service (PaaS)** Il modello PaaS fornisce piattaforme virtualizzate utilizzabili per lo sviluppo di applicazioni, nonché per la loro esecuzione. La piattaforma include tutto ciò di cui si ha bisogno per lo sviluppo software, come ambienti di runtime, database, IDE (Integrated Development Environment) e middleware. L'utente può gestire la piattaforma in ogni suo aspetto attraverso l'utilizzo di API e strumenti appositi, mentre hardware e software sono completamente gestiti dal Cloud Provider. Il modello PaaS è utilizzato soprattutto per lo sviluppo ed il supporto del software durante tutto il suo ciclo di vita.

**Software as a Service (SaaS)** Il modello SaaS consente di utilizzare software attraverso il browser, sotto forma di servizio Web. Grazie a questo modello, l'utente è esentato dall'acquisto e dalla gestione dell'hardware, nonché dalla gestione dei sofware e delle licenze necessarie per utilizzarli. In questo modello rientrano servizi come applicazioni industriali, servizi di posta elettronica o i più recenti servizi di collaborazione per lo smart working.

### 1.1.2 Serverless Computing

L'evoluzione del Cloud Computing ha portato alla nascita dei modelli di servizio **Infrastructure as a Service (IaaS)**, **Platform as a Service (PaaS)** e **Software as a Service (SaaS)**, che oggi sono largamente conosciuti ed utilizzati. Questi modelli hanno velocizzato l'espansione del Cloud

Computing grazie alla possibilità per gli utenti di ricerare su Cloud gli ambienti utilizzati nelle proprie infrastrutture di calcolo. Tuttavia, ciò ha anche rallentato lo sviluppo di nuove metodologie che permettano di implementare applicazioni sviluppate espressamente per l'esecuzione in ambiente Cloud. In mancanza di un paradigma che permettesse di sfruttare a pieno le caratteristiche del Cloud, gli sviluppatori dovevano gestire le risorse allo stesso modo di come avrebbero fatto con un'infrastruttura fisica. La necessità di questo paradigma ha portato alla nascita del modello di servizio Function as a Service (FaaS), meglio noto con l'appellativo di Serverless Computing. Benché il nome lasci pensare alla totale mancanza di un server, in realtà il termine "Serverless" indica la totale gestione di quest'ultimo da parte del Cloud Provider, dall'infrastruttura necessaria alla sua configurazione e manutenzione.[29].
FaaS è la naturale evoluzione dell'architettura basata su microservizi, basato sul concetto di funzione serverless, ovvero una porzione di codice molto piccola eseguita su Cloud in maniera trasparente. Lo sviluppatore ha quindi l'unico onere di occuparsi della funzione e degli eventi da essa generati. Di fatto, una funzione gestisce un singolo evento per volta e questo non solo ne migliora la scalabilità, ma anche la facilità di utilizzo, andando ad ottimizzare carico di lavoro e tempi di sviluppo. Poiché basato sugli eventi, questo paradigma viene definito un modello di programmazione event-driven, dato che ogni funzione viene eseguita in risposta ad un evento specifico. Tale approccio consente di migliorare la produttività ed ottenere un notevole risparmio, grazie al forte disaccoppiamento derivante dal fatto che le funzioni vengono eseguite solo in risposta all'evento ad esse associato [45].
Per lo sviluppo di applicazioni Serverless è necessario prendere in considerazione tutti gli aspetti compresi in questo paradigma, i quali rendono necessario un approccio diverso all'implementazione. Un'applicazione Serverless ha una diversa funzione per ognuna delle sue funzionalità e deve, inoltre, prevedere un modello di comunicazione ad eventi, che sarà onere dello sviluppatore generare e gestire. Il ruolo del Cloud provider è quello di allocare le risorse necessarie al funzionamento dell'applicazione, tenendo conto del carico di lavoro. Alla ricezione dell'evento ad essa associato, infatti, la funzione viene mandata in esecuzione su un container indipendente, la cui gestione è completamente delegata al Cloud provider, che viene lanciato e, quando necessario, spento. Attraverso l'impiego dei container e quindi della virtualizzazione, già largamente impiegata dagli altri modelli di servizio del Cloud Computing, si ottiene un'elevata scalabilità, poichè all'aumentare del carico di lavoro e delle funzioni da eseguire viene lanciato un nuovo container in modo automatico. A differenza delle macchine virtuali, un container risulta decisamente più

leggero, grazie al peso di pochi megabyte ed alla quantità di risorse richieste nettamente inferiore. Ciò permette ad un container di completare l'avvio in pochi secondi.
Da ciò possiamo comprendere quanto sia vantaggioso l'utilizzo del Serverless Computing, che consente di impiegare in maniera efficiente le risorse, minimizzare il consumo di memoria e ridurre i costi. Il Serverless Computing ha inoltre favorito lo sviluppo dell'Internet of Things e dei servizi mobile, noti per lavorare con numeri in rapida crescita. Grazie al Cloud Computing eliminiamo la necessità di possedere e gestire l'infrastruttura hardware, mentre con il Serverless Computing eliminiamo le necessità relative alla parte software. Ciò consente agli sviluppatori di concentrarsi unicamente sulla logica dell'applicazione, migliorando così la produttività e la qualità del software implementato. Il livello di astrazione è tale da non rendere necessaria alcuna azione dell'utente in quanto hardware e software sono completamente gestiti dal Cloud Provider in ogni loro aspetto [26].

### 1.1.3 Multi-cloud

Con multi-cloud si intende l'uso dei servizi di diversi Cloud provider combinati tra loro in una singola architettura eterogenea in cui carichi di lavoro, dati, applicazioni vengono distribuiti tra diversi ambienti Cloud. Questo approccio permette di sfruttare al meglio i vantaggi di ogni provider e di ottenere non solo maggiore affidabilità, ma anche un miglior rapporto qualità-prezzo, una riduzione della dipendenza dal singolo provider ed un aumento della flessibilità di scelta. Inoltre, tra i motivi per cui nasce la necessità di un'architettura multi-cloud figurano l'aderenza alle politiche locali che richiedono dati fisicamente presenti all'interno della propria regione e distribuzione geografica delle richieste di elaborazioni dal Cloud fisicamente più vicino al fine di ridurre la latenza. La volontà di aumentare la flessibilità nella scelta dei servizi e la consapevolezza che non esista nessun Cloud provider che possa fornire la giusta combinazione di servizi per ogni esigenza hanno portato alla nascita dell'approccio al multi-cloud. Il ricorso al multi-cloud, inoltre, consente alle aziende di replicare parzialmente o totalmente la propria architettura su ambienti differenti. Tali ambienti possono essere sempre attivi e suddividersi il carico di lavoro oppure soltanto uno di essi può essere attivo mentre gli altri fungono da ambienti di backup. Queste due tipologie di funzionamento prendono rispettivamente il nome di active-active ed active-passive. Al fine di ottenere elevata disponibilità e tolleranza ai guasti è inoltre possibile replicare in modo parziale o totale servizi e applicazioni su diverse piattaforme.
Una tra le criticità del Cloud Computing che il multi-cloud mira a superare

è il problema del vendor lock-in, ovvero la dipendenza dal singolo provider. La possibilità di scegliere tra un più ampio catalogo di servizi permette di scegliere la soluzione più adatta alle proprie esigenze, sfruttando i servizi esclusivi di determinati provider [37]. La scelta di un singolo Cloud provider limita le possibilità implementative per via della possibile mancanza di disponibilità di alcuni servizi in un determinato ambiente Cloud. Di fatto, i Cloud provider talvolta offrono servizi ad-hoc per alcune tecnologie o piattaforme, le quali possono tuttavia rivelarsi inadatte a specifiche necessità di sviluppo [20].
I vantaggi più significativi offerti dal multi-cloud sono:

- *localizzazione* - controllo sulla posizione dei dati attraverso la selezione dell'area geografica in cui si desidera che questi vengano memorizzati. Ciò consente di scegliere le aree geografiche con una legislazione il più possibile conforme alle politiche interne;

- *scalabilità* - grazie alla possibilità di accedere alle risorse messe a disposizione da più provider, queste ultime risultano virtualmente illimitate. Ciò consente di gestire facilmente applicazioni complesse e picchi di carico;

- *disponibilità* - la possibilità di replicare l'intera infrastruttura su un provider di backup garantisce una maggiore tolleranza ai guasti e consente di avere un sistema scalabile e con le stesse prestazioni anche in caso di guasti seri;

- *flessibilità* - la maggiore possibilità di scelta implica la possibilità di soddisfare qualsiasi tipo di esigenza. Se un provider non dovesse offrire un servizio adatto ad una specifica esigenza, si potrà decidere di utilizzare uno dei servizi messi a disposizione dagli altri provider;

- *risparmio* - possibilità di ottenere un ambiente efficiente dal punto di vista dei costi, attraverso la scelta della combinazione di servizi più conveniente.

- *prestazioni* - possibilità di spostare i carichi di lavoro verso server fisicamente più vicini all'utente finale nel caso dovessero essere eseguite applicazioni per cui la latenza è un aspetto particolarmente critico.

# Capitolo 2

# Stato dell'Arte

Visto il crescente interesse per l'impiego del multi-cloud sono nate nuove tecnologie in grado di permetterne l'utilizzo, ciascuna con metodi specifici per la fruizione dei servizi. Lo scopo di queste tecnologie è semplificare l'uso del multi-cloud tramite l'unificazione degli strumenti di gestione e di controllo delle risorse messe a disposizione dai provider. Tra i prodotti maggiormente conosciuti troviamo OpenStack [36], che consente di creare e gestire ambienti cloud pubblici e privati, sfruttando il modello IaaS. Criticità di questo prodotto è la mancanza di supporto al Serverless Computing e ai restanti modelli.
Il supporto al Serverless Computing è invece offerto da TOSCA [48], un linguaggio per lo sviluppo di applicazioni su Cloud, attraverso TOSCA-Serverless [51]. Le applicazioni sviluppate con TOSCA godono di ottime interoperabilità e portabilità. Criticità di questo linguaggio è la mancanza di costrutti per la descrizione dei requisiti in termini di risorse. Una tipologia di approccio simile è offerta da CAMEL [14], acronimo di Cloud Application Modelling and Execution Language. Si tratta di un linguaggio che consente di gestire il ciclo di vita delle applicazioni attraverso la modellazione di componenti e servizi. Similmente a quanto fatto per TOSCA, CAMEL viene esteso da ServerlessCamel [28]. Basato sulla tecnologia Xtext di Eclipse, CAMEL unisce le funzionalità di più DSL per gestire ogni aspetto del ciclo di vita di un'applicazione e, pertanto, viene definito come multi-DSL.
Approccio differente è quello offerto dal framework Serverless [44], che permette di sviluppare applicazioni basate sui servizi serverless di diversi provider, tra cui AWS, Microsoft Azure, IBM e Google Cloud. Serverless non utilizza una GUI (Graphical User Interface), bensì sfrutta un'interfaccia CLI (Command Line Interface) attraverso la quale è possibile sviluppare le proprie

applicazioni. Altri punti di forza sono la disponibilità di esempi e strutture pronte all'uso che agevolano il lavoro dello sviluppatore e la possibilità di effettuare testing in locale e, di conseguenza, evitare costi di esecuzione. Essendo un servizio specifico per il Serverless Computing, Serverless consente unicamente lo sviluppo di applicazioni serverless. Un altro servizio particolarmente interessante è Pulumi [39], uno strumento open source che sfrutta Infrastructure as Code per creare e gestire infrastrutture Cloud. Punti di forza di Pulumi sono il supporto ai linguaggi più comuni, il supporto ad infrastrutture tradizionali (macchine virtuali, network, ecc.), ma anche a quelle più moderne, come i cluster Kubernetes, le funzioni serverless e i container. Altra funzionalità di spicco è la possibilità di definire attraverso il codice anche i metodi e gli ambienti di deploy delle applicazioni.
Un'altro tipo di approccio è quello offrerto da DistrubutedFaas [49] che opera sfruttando un'architettura basata su container. DistrubutedFaas consente di progettare e costruire un sistema distribuito tra diversi Cloud provider. Attraverso questo metodo, vengono di fatto costruiti sistemi di HPC (High Performance Computing) su Cloud, sfruttando il modello FaaS. Il risultato è dato da meccanismi di auto-scaling compatibili sia con container che con macchine virtuali e da un sistema di bilanciamento del carico di lavoro molto performante.

# Capitolo 3

# FLY Language

Quando si parla di implementazione di un'architettura multi-cloud l'obiettivo è quello di gestire con efficacia la collaborazione tra i vari Cloud attraverso un ambiente coeso, prestando particolare attenzione alla complessità di gestione, che non deve essere eccessiva. L'implementazione non consiste quindi nella mera aggregazione di servizi di provider diversi e, di fatto, richiede un approccio ponderato per rimanere in linea con i suddetti obiettivi. La necessità di una simile architettura nasce dalla mancanza di standardizzazione tra i vari provider, che utilizzano una serie di API e protocolli specifici per la fruizione dei propri servizi. Un'ulteriore criticità è collegata agli aggiornamenti dei servizi offerti, con cui spesso vengono modificate o rimpiazzate le API. In questo scenario è semplice comprendere che nonostante si possano ottenere diversi vantaggi dal multi-cloud, le difficoltà da affrontare sono numerose e possono scoraggiare le aziende e gli sviluppatori che intendono approcciarsi a questo mondo.
Una soluzione efficace è quella di aggiungere dei livelli di astrazione, ovvero utilizzare strumenti che consentano all'utente di sfruttare il multi-cloud senza però necessariamente conoscere i dettagli implementativi che consentono di gestire l'ambiente. Ciò permette all'utente di concentrarsi sullo sviluppo della propria applicazione e di ottenere tutti i vantaggi che derivano dall'utilizzo del multi-cloud [46]. Per far fronte a questa esigenza è nato **FLY**, un **Domain-Specific Language** per il Calcolo Scientifico sul multi-cloud che ha come obiettivo la semplificazione dello sviluppo di applicazioni che sfruttino il multi-cloud, mediante il paradigma FaaS, che consente di ottenere applicazioni con scalabilità e prestazioni elevate. FLY permette di astrarre completamente la gestione dell'interazione con il Cloud, semplificando l'esperienza dello sviluppatore [17].

Funzionalità centrale di FLY sono le **funzioni FLY**, ovvero blocchi di codice indipendente che possono essere eseguiti in modo concorrente. Ciò permette di eseguire diverse istanze della stessa funzione in parallelo su Cloud o in locale, in linea con il modello FaaS. In particolare, l'esecuzione delle funzioni FLY su ambiente Cloud prevede che a ciascuna funzione FLY venga associata una funzione Serverless.
I tre aspetti principali che caratterizzano FLY sono:

- *Potenza*, poiché consente di sfruttare i servizi di diversi provider in una singola applicazione, permettendo di utilizzare le soluzioni più efficienti;
- *Efficacia*, poiché consente al programmatore di evitare la gestione degli ambienti di esecuzione, pur essendo semplice da comprendere e da utilizzare;
- *Efficienza*, poiché permette di ridurre i tempi di sviluppo grazie all'astrazione e di scegliere i servizi da utilizzare in base al costo ed alle funzionalità disponibili.

In questo capitolo sarà descritto dettagliatamente il linguaggio FLY, prestando particolare attenzione agli obiettivi per cui è stato sviluppato, alla sua architettura ed al suo funzionamento, nonché ai concetti ed agli strumenti che sono alla base di questo linguaggio.

## 3.1 Obiettivi

Lo scopo per cui è nato FLY è quello di rendere compatibili il mondo del Cloud Computing e quello del calcolo scientifico. FLY è stato concepito come uno strumento semplice, potente ed efficace per consentire lo sviluppo di applicazioni Serverless che sfruttino i vantaggi del multi-cloud, ottenendo così elevata scalabilità.
Gli obiettivi di FLY sono:

- garantire *espressività*, ovvero permettere di scrivere algoritmi facilmente leggibili con un basso numero di istruzioni, consentendo quindi lo sviluppo di applicazioni di Calcolo Scientifico in modo semplice ed intuitivo anche per gli sviluppatori meno esperti.
- garantire *alta usabilità* attraverso la completa astrazione dell'interazione con il Cloud e, conseguentemente, l'eliminazione della necessità di

conoscere protocolli ed API dei vari provider. Un esperto del dominio deve poter sviluppare applicazioni in modo semplice.

- garantire *scalabilità*, ovvero permettere sia alle architetture Cloud che a quelle Symmetric Multiprocessing (SMP) di scalare facilmente, senza aggiungere eccessiva complessità al linguaggio.

FLY nasce per consentire agli sviluppatori di dominio, ovvero coloro che hanno particolare esperienza in un determinato campo, ma conoscenza limitate sui sistemi paralleli e distribuiti, di sviluppare le proprie applicazioni sfruttando le architetture Serverless per poter eseguire del codice in parallelo. Per tale motivo, la sintassi di FLY ha preso ispirazione da quella dei linguaggi maggiormente utilizzati per il Calcolo Scientifico (Java, JavaScript, Python, R, ecc.). Grazie ai molti costrutti specifici per i vari domini di interesse, FLY risulta essere un linguaggio semplice da utilizzare per interagire con il Cloud, ma anche ricco.
FLY supporta anche il paradigma di calcolo distribuito, la gestione della memoria e fornisce un valido sistema di comunicazione tra i processi, attraverso specifici canali di comunicazione. Per tale ragione, un programma FLY è eseguibile sia su un'architettura multiprocessore, sia in ambiente Cloud, senza che l'utente debba configurare tutte le risorse necessarie per l'esecuzione [17].

## 3.2 Domain-Specific Language

Un **Domain-Specific Language (DSL)** è un linguaggio di programmazione appositamente progettato per l'utilizzo in un determinato contesto relativo ad un dominio e che fornisce un'apposita notazione sviluppata sulla base delle caratteristiche e dei concetti principali di quest'ultimo. Al contrario dei linguaggi General Purpose, che vengono comunemente utilizzati per un vasto insieme di problemi, il campo di applicazione dei DSL risulta essere piuttosto ristretto. Grazie ai DSL si aggiunge un livello di astrazione tale da rendere semplice l'implementazione di applicazioni anche ad esperti del dominio che non hanno competenze tecniche avanzate, permettendo quindi di affidare lo sviluppo di un progetto non solo agli informatici, ma anche ad esperti del settore.
Alcuni celebri esempi di DSL sono: SQL per la scrittura di query su database relazionali, HTML per la costruzione di siti web, R per le applicazioni in ambito statistico e LaTex per la preparazione di testi. Tutti questi linguaggi non sono di certo adatti per la creazione di applicazioni di tutti i tipo, ma soltanto per le applicazioni che fanno parte del proprio dominio.

L'obiettivo di un DSL consiste nell'essere semplice da utilizzare e da imparare, facendo leva proprio sulla specificità di utilizzo che consente a questi linguaggi di essere piccoli ed espressivi e di avere poca ridondanza, conciliando le necessità degli esperti di dominio e quelle degli sviluppatori.
Per realizzare un DSL occorre sviluppare un compilatore che possa leggere il programma scritto nel linguaggio che si vuole creare, effettuane l'analisi, processarlo e interpretarlo. Ciò consente al compilatore di generare codice eseguibile. Il processo di compilazione è suddiviso in diverse fasi:

- *analisi lessicale* - che consiste nella suddivisione del programma originale in unità più piccole dette *token*, ciascuna corrispondente ad un singolo elemento del linguaggio;
- *analisi sintattica* - che consiste nell'analisi dei *token* identificati per verificare che formino uno statement valido per il linguaggio. In questa fase che viene generato l'Abstract Syntax Tree (AST), cioè una rappresentazione della struttura, dal punto di vista sintattico, del programma;
- *analisi semantica* - che consiste nell'unione dei processi di type checking, ovvero il controllo degli assegnamenti di valore per verificarne la compatibilità con il relativo tipo di dati, e di tracciamento degli identificatori, del loro tipo e delle espressioni, processo che consente di verificare che la dichiarazione degli identificatori preceda il loro uso nelle espressioni;
- *generazione del codice* - che consiste nella generazione del codice eseguibile o del codice in un altro linguaggio, servendosi dei risultati ottenuti dalle fasi precedenti.

### Xtext

**Xtext** [52] è un framework open-source per lo sviluppo di DSL e linguaggi di programmazione in grado di generare non solo un parser ma anche un class model per l'ABS, fornendo anche un IDE personalizzabile basato su Eclipse. Alla base dello sviluppo di un linguaggio con Xtext troviamo la scrittura di una grammatica che definisce ogni parte del linguaggio e che permette di ottenere un'infrastruttura composta da parser, linker, typechecker, compilatore e supporto all'editing per Eclipse. L'utilizzo di Xtext risulta particolarmente vantaggioso per gli sviluppatori di DSL. Per tale motivo, Xtext è stato scelto come framework per lo sviluppo del compilatore di FLY.

## 3.3 Architettura

L'utilizzo di FLY per lo sviluppo delle proprie applicazioni permette di evitare la configurazione delle risorse necessarie per l'esecuzione, in quanto queste ultime vengono stabilite automaticamente sulla base dei requisiti necessari per la computazione. FLY offre supporto sia a modelli di esecuzione sincroni che asincroni, inoltre, permette di scegliere tra due ambienti di run-time, ovvero la macchina locale e l'infrastruttura Cloud, che viene considerata come un'architettura di calcolo parallelo che è possibile utilizzare per l'esecuzione di codice concorrente, semplificando l'esperienza dell'utente.
FLY possiede una tipizzazione statica e forte, che sfrutta diversi tipi di inferenza per la determinazione dei tipi di variabili e costanti. I programmi scritti in FLY vengono automaticamente tradotti dal compilatore in programmi Java. Per rendere possibile ciò è stato necessario determinare un corrispettivo in Java per ogni tipo di dato definito in FLY, oltre che definire i costrutti Java da utilizzare per l'implementazione delle funzionalità specifiche del dominio.
Elemento centrale del linguaggio FLY è il concetto di **funzione FLY**, ovvero una porzione di codice indipendente, eseguibile in maniera concorrente, in maniera del tutto simile a quanto avviene con le funzioni Serverless. Le funzioni FLY possono essere eseguite sia in sequenziale che in parallelo, sia in locale che su Cloud. Quest'ultima tipologia di esecuzione è resa possibile da costrutti che permettono la definizione. l'esecuzione, la sincronizzazione e la comunicazione delle funzioni FLY. In particolare, la comunicazione è possibile grazie all'utilizzo di canali di comunicazione virtuali che permettono lo scambio di dati tra diversi ambienti e tra diverse funzioni FLY.


*Figura 3.1: Flusso di compilazione del codice scritto in FLY.*

Nella Figura 3.1 viene mostrato il **flusso di compilazione** di un'applicazione scritta in FLY. All'inizio della compilazione, il programma FLY viene dato in input al compilatore, il quale genera un *Abstract Syntax Tree*, cioè

un albero che rappresenta la struttura sintattica del programma, che non rappresenta però tutti i dettagli del codice sorgente. L'AST viene successivamente trasformato in codice Java, dal quale vengono estratte le funzioni FLY, che a loro volta verranno trasformate, una ad una, in codice eseguibile. Il prodotto finale della compilazione è il codice compilato delle funzioni FLY pronto per essere eseguito sui vari ambienti dichiarati dallo sviluppatore.

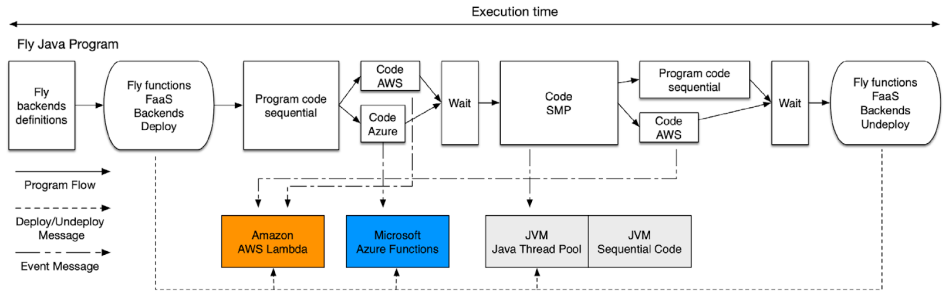


*Figura 3.2: Flusso di esecuzione di un programma FLY.*

La Figura 3.2 mostra dettagliatamente il flusso di esecuzione di un programma FLY. Inizialmente, si inizializzano gli ambienti di back-end dichiarati nel codice, effettuando il login sul Cloud provider e istanziando i servizi necessari. Il codice della funzione FLY viene poi caricato sul rispettivo ambiente Cloud e compilato, in moda da evitare la compilazione a tempo di esecuzione. Completata l'inizializzazione, il programma principale viene eseguito e, alla sua terminazione, vengono effettuate le procedure di undeploy sull'ambiente di back-end, in modo da eliminare tutte le istanze dei servizi Cloud create in precedenza.

## 3.4 Definizione del linguaggio

FLY fornisce due tipologie di tipi, tipi *basici* e tipi di *dominio*. I tipi basici sono ereditati direttamente da Java, includendo *booleani*, *interi*, *reali* e *stringhe*, utilizzabili anche per dichiarare array monodimensionali, bidimensionali e tridimensionali. I tipi di dominio, invece, sono in numero maggiore e sono utilizzabili per interagire e comunicare con l'ambiente di esecuzione.
Il Listato 3.1 mostra un semplice esempio di programma FLY per il calcolo del Pi Greco con il metodo Monte Carlo su ambiente AWS. Tale esempio sfrutta AWS Lambda [6]. Di seguito verrà brevemente descritto il codice, mentre gli elementi che lo compongono verranno approfoditi successivamente. La prima istruzione consiste nella dichiarazione dell'ambiente di esecuzione

AWS. Su tale ambiente è dichiarato subito dopo un `channel`, ovvero un canale di comunicazione virtuale che permette al programma principale di comunicare con la funzione FLY `pi`, che genera un punto casuale e calcola se appartiene o meno ad un cerchio con origine al punto 1.0. Il risultato della funzione viene inviato sul canale `ch`. La funzione `estimation` legge l'output della funzione `pi` e scrive a schermo la stima di Pi ottenuta. L'ultima riga mostra il lancio delle funzioni FLY, che prevede l'utilizzo della parola chiave `FLY` per eseguire 10 funzioni `pi` sull'ambiente AWS. Grazie alla parola chiave `thenall`, quando tutte le funzioni `pi` hanno terminato la propria esecuzione, viene eseguita la funzione `estimation`.

```
1  var local = [type="smp", nthread=4]
2
3  var cloud = [type="aws", user="user_name", access_id_key = "ACCESS_ID"
       , secret_access_key="ACCESS_KEY", region="eu-west-2", language="
       nodejs12.x", thread=10, memory=256, time=300]
4
5  var ch = [type="channel"] on cloud
6
7  func pi(){
8      var r = [type="random"]
9      var x = r.nextDouble()
10     var y = r.nextDouble()
11     var msg = 0
12
13     if((x * x)+(y * y) < 1.0){msg = 1}
14     ch!msg on cloud
15 }
16
17
18 func estimation(){
19     var sum = 0
20     var crt = 0
21     for i in [0:2] {
22         sum += ch? as Integer
23         crt += 1
24     }
25     println "pi estimation: " + (sum*4.0) \ crt
26 }
27
28 FLY pi in [0:10] on cloud thenall estimation
```

*Listing 3.1: Stima del PI Greco con il metodo Monte Carlo su ambiente AWS.*

### Object

Il tipo di dato di dominio principale è il tipo `object`, un insieme di elementi eterogenei. Un FLY `object` consiste in una struttura dati in grado di

memorizzare elementi nella forma di coppie chiave-valore. Il valore di un elemento può essere ottenuto utilizzando la sua posizione, similmente ai metodi usati per gli array, oppure la sua chiave, come per le mappe.
Ogni tipo di dato di dominio è un'istanza di `object`. Per tale motivo, tutti i tipi di dominio utilizzano una sintassi simile sfruttando il campo `type` per specificarne il tipo.

### Ambiente di esecuzione

Un programma FLY, nelle sue prime istruzioni, prevede la dichiarazione degli **ambienti di esecuzione** per consentire al generatore di configurare le risorse necessarie al funzionamento dell'applicazione. Per esempio, nel caso di un ambiente locale multiprocessore è necessaria una `Java Thread Pool`, mentre per gli ambienti su Cloud è necessaria la creazione delle istanze dei servizi necessari.
Per dichiarare un ambiente di esecuzione è necessario fornire alcuni parametri aggiuntivi, oltre ai quali, nel caso della dichiarazione di un ambiente Cloud, è necessario che l'utente specifichi il linguaggio di programmazione in cui tradurre le funzioni FLY. In tal caso, le possibili scelte sono JavaScript [25] o Python [40].
Attualmente gli ambienti supportati sono quattro:

- **smp** - ambiente locale che sfrutta i thread Java per realizzare un parallelismo basato su un sistema multiprocessore simmetrico;
- **aws** - ambiente Cloud collegato ad Amazon Web Services (AWS) [3] che permette di sfruttare AWS Lambda [6] per l'esecuzione delle funzioni FLY;
- **aws-debug** - ambiente locale che simula l'esecuzione delle funzioni FLY in ambiente AWS, appositamente sviluppato per consentire il testing dei programmi senza dover sfruttare il Cloud;
- **azure** - ambiente Cloud collegato a Microsoft Azure [34] che sfrutta Azure Function [12] per l'esecuzione delle funzioni FLY.

### Channel

Per la comunicazione tra le funzioni FLY su un determinato ambiente ed il programma principale occorre la dichiarazione di un `"channel"`, che necessita, mediante la parola chiave `on`, la specificazione dell'ambiente su cui operare. I channel si basano sulle code di messaggi bloccanti, ovvero quando si tenta

di ricevere un messaggio l'esecuzione si ferma fino a che un nuovo messaggio non viene ricevuto. L'invio di messaggi è possibile attraverso il carattere `"!"`, mentre il carattere `"?"` ne permette la ricezione. Per consentire di scambiare messaggi con l'ambiente su Cloud, FLY implementa un meccanismo di serializzazione.

### Funzione FLY

Le **funzioni FLY**, di cui è già stata fornita una parziale descrizione, sono frammenti di codice indipendente dal programma principale e dalle altre funzioni, eseguibili in maniera concorrente.
La dichiarazione di una funzione FLY avviene mediante l'utilizzo della parola chiave `func` seguita dal nome della funzione e dagli eventuali parametri di input, che vengono passati per copia. Le funzioni FLY restituiscono un valore con parola chiave `return` ed hanno scoping privato, per cui solo i parametri della funzione e le variabili locali sono visibili all'interno della funzione. Per oltrepassare questo limite, si possono utilizzare i `channel` oppure le costanti, la cui dichiarazione ed il successivo accesso devono avvenire nello stesso ambiente di esecuzione. Ciò consente ad una funzione in esecuzione su un determinato ambiente di poter utilizzare canali e oggetti disponibili su tale ambiente, indipendentemente dall'origine della loro dichiarazione. Le funzioni FLY vengono eseguite mediante la parola chiave `FLY`. Tuttavia, tale parola chiave non può essere utilizzata all'interno del corpo di una funzione, poiché non è ammessa la ricorsione di funzioni FLY. Il suo utilizzo permette la generazione di un evento sull'ambiente che si vuole utilizzare. Ciò consente di rimanere in linea con il modello di programmazione event-driven, che caratterizza il Serverless Computing. FLY sfrutta il parallelismo attraverso la sintassi che permette l'esecuzione di una funzione FLY. A seguito dell'utilizzo della parola chiave `FLY`, viene dichiarato il nome della funzione da eseguire e, se necessario, numero di istanze da eseguire e l'ambiente di esecuzione da utilizzare, quest'ultimo a seguito della parola chiave `on`.

### Funzione di Callback

Le **funzioni di Callback** sono funzioni FLY da eseguire al termine dell'esecuzione di una precedente funzione. Vanno dichiarate dopo la specifica dell'ambiente di esecuzione di una funzione FLY. FLY prevede due tipi di funzioni di Callback, uno riguarda le funzioni specificate in seguito alla parola chiave `then`, che vengono eseguite al termine di ogni istanza della relativa funzione FLY, mentre l'altro sfrutta la parola chiave `thenall` per indicare

che l'esecuzione della funzione di Callback deve avvenire solo a seguito della terminazione di tutte le istanze della relativa funzione FLY.

#### Esecuzioni asincrone

FLY consente le **esecuzioni asincrone** attraverso l'uso della parola chiave `async` ed il tipo di dato di dominio chiamato `async-object` che permette l'interazione ed il controllo da parte dell'utente. L'ivocazione di una funzione utilizzando `async` restituisce immediatamente il controllo al programma principale in modo da continuare l'esecuzione. A seguito di ciò, l'utente può controllare lo stato delle funzioni asincrone con il metodo `status()` dell'`async-object`, mentre può mettere in pausa le funzioni che non hanno terminato l'esecuzione con il metodo `wait()`.

#### Codice nativo

FLY permette di includere **codice nativo** all'interno di applicazioni FLY grazie alla parola chiave `native`. Ciò consente di inserire in una funzione FLY codice scritto in Python o in Javascript che non verrà tradotto dal compilatore FLY, ma semplicemente copiato. La parola chiave `require`, inoltre, consente di installare librerie esterne addizionali nell'ambiente di esecuzione.

### 3.4.1 Struttura di un progetto FLY

FLY utilizza *Maven* [32] come gestore di pacchetti per la costruzione dei progetti. Maven consente di gestire progetti software e di automatizzare il processo di build, basandosi sul concetto di **Project Object Model (POM)**, ovvero un file XML che contiene tutte le dipendenze necessarie.
La creazione di un **progetto FLY** genera un progetto Java Maven con all'interno due cartelle principali ed un file POM, tutte le dipendenze necessarie ed il codice delle funzioni FLY. Il progetto viene costruito sulla base del programma FLY da parte del compilatore. Con il comando `mvn package` di Maven, è possibile utilizzare il progetto per generare un file *JAR* eseguibile [17].
La struttura di un progetto FLY è la seguente:

- **cartella *src*** - in cui vanno inseriti i file con estensione *.FLY* contenenti il codice dei programmi scritto in linguaggio FLY;
- **cartella *src-gen*** - in cui vengono inseriti i file generati dal compilatore, ovvero i file Java e gli file di script di deploy e undeploy con estensione

*.sh*. Il file Java orchestra l'intera esecuzione del programma, incluso il lancio dei file di script.

- **file XML *pom.xml*** - file necessario a Maven per la gestire il progetto. Si tratta di un file XML contenente informazioni e i dettagli di configurazione necessari per effettuare la build, specificamente le librerie necessarie al programma [33].

## 3.5 Generazione del codice e configurazione delle risorse

Il principale vantaggio offerto da FLY consiste nel livello di astrazione che consente all'utente di non dover conoscere le API di ogni Cloud provider per scrivere un programma. Tuttavia, per l'interazione con i servizi le API restano comunque essenziali. La responsabilità dell'utilizzo delle API è delegata quindi al generatore FLY. Come anticipato nei precedenti paragrafi, il compito del generatore è quello di tradurre ogni componente nel linguaggio di destinazione. Infatti, il punto focale di FLY sta proprio nella generazione del codice, il cui risultato varia in base all'ambiente di esecuzione dichiarato dall'utente. Per gli ambienti multiprocessore, il parallelismo viene implementato attraverso una `Java Thread Pool` che consente di utilizzare dei thread per l'esecuzione delle istanze delle funzioni e, inoltre, consente di riutilizzare i thread precedentemente creati, riducendo ulteriormente il tempo di esecuzione. Il programma FLY viene eseguito su una *Java Virtual Machine (JVM)* poiché tradotto in codice Java, sulla quale viene eseguito anche l'ambiente SMP, rendendo quindi necessari almeno due core fisici, uno per ciascuno dei suddetti ambienti di esecuzione. L'esecuzione su Cloud prevede l'utilizzo delle API del provider di riferimento, le quali permettono l'interazione con i servizi disponibili. In base a questi ultimi vengono tradotti i vari costrutti e funzionalità di FLY.
Come precedentemente specificato, il codice generato per le funzioni FLY varia a seconda dell'ambiente di esecuzione scelto. Specificamente, il prodotto della compilazione può essere codice in Java, JavaScript o Python, a seconda delle scelte dell'utente. L' ambiente SMP locale prevede che le funzioni siano tradotte in Java, in quanto eseguite su JVM, mentre i servizi FaaS su Cloud permettono solitamente l'esecuzione di codice in JavaScript o Python. L'ambiente di destinazione, inoltre, determina anche il modo in cui viene generato lo script di deploy che costruisce un pacchetto di esecuzione al cui interno troviamo codice sorgente e librerie.

Durante l' esecuzione l'interazione con il Cloud avviene attraverso strumenti di **Command Line Interface (CLI)** forniti dai provider. Alcuni esempi sono la *AWS CLI* [4] o la *Azure CLI* [10]. Infine, le funzioni vengono invocate attraverso le chiamate *HTTP POST* asincrone, che assicurano bassa latenza e permettono al programma di continuare la sua esecuzione in attesa di risposta dal servizio.
I Cloud provider supportati da FLY sono **Amazon Web Services (AWS)** e **Microsoft Azure**. AWS è un Cloud provider di proprietà del gruppo Amazon, celebre per essere stato il primo a supportare il Serverless Computing attraverso il servizio **Lambda** e che attualmente offre più di 150 servizi differenti. Microsoft Azure, invece, è un Cloud provider di proprietà di Microsoft, celebre per il supporto ai database NoSQL grazie al servizio **CosmosDB**.

### Amazon Web Services

Amazon Web Services fornisce diverse SDK che permettono di interagire con i suoi servizi attraverso i principali linguaggi di programmazione [7]. Il programma principale, essendo costituito da codice Java, fa uso delle *AWS SDK for Java* [42] che consentono l'accesso a tutti i servizi forniti da AWS mediante le API dedicate. Gli script di deploy e undeploy, invece, utilizzano la AWS CLI [4] per la creazione delle istanze dei servizi necessari e per il caricamento delle funzioni FLY, insieme ai relativi pacchetti su AWS Lambda. In particolare, le funzioni FLY sfruttano la libreria *boto3* per Python [13] e la libreria *aws-sdk* per JavaScript [43] per interagire con i servizi di AWS. Attualmente, FLY utilizza la versione 1.0 delle SDK, ma è previsto il passaggio alle SDK 2.0.

### Microsoft Azure

Le SDK di Microsoft Azure sono complicate da usare al di fuori dell'IDE VisualStudio Code e mancano di funzionalità per la maggior parte dei servizi disponibili. A causa di queste problematiche, il supporto ad Azure è basato su un servizio REST API che prescinde dal linguaggio di programmazione, in quanto basato su operazioni HTML. Le *Representational State Transfer (REST) API* permettono di avere endpoint specifici per ciascun servizio che supporti le operazioni HTTP, le quali devono permettere l'accesso alle principali funzionalità di questi ultimi come la creazione, l'aggiornamento, la cancellazione o l'ottenimento di informazioni. Per utilizzare le REST API è necessario avere un token di autorizzazione ottenibile tramite il servizio Azure

Active Directory, che attraverso i dati di accesso (ID, password e TenantID) ne permette l'acquisizione. FLY utilizza una libreria specificamente creata per aggiungere il supporto alle REST API.

Nelle seguenti sezioni verrà analizzata la traduzione del codice FLY dal compilatore nei vari linguaggi di destinazione per generare i file eseguibili. I seguenti listati si riferiscono al programma per la stima del Pi Greco, già presentato nel Listato 3.1.

### 3.5.1 Ambiente di esecuzione

#### Ambiente locale - SMP

La dichiarazione di un **ambiente locale** necessita di due parametri, il primo definisce il tipo di ambiente che sarà di `type="smp"`, il secondo specifica il numero di thread che dovranno essere utilizzati.

```
1 var local = [type = "smp", nthread = 10]
```

*Listing 3.2: Dichiarazione di un ambiente SMP locale.*

Il codice generato risulta in un oggetto Java di tipo `ExecutorService`, che semplifica l'esecuzione asincrona permettendo l'esecuzione dei task in modo concorrente.

```
1 static ExecutorService __thread_pool_local = Executors.
      newFixedThreadPool(10);
```

*Listing 3.3: Codice generato per l'ambiente SMP.*

#### Ambiente Cloud - Amazon Web Services

Per utilizzare il **Cloud AWS** come ambiente di esecuzione è necessario dichiarare una variabile di `type="aws"` che necessita di una serie di parametri aggiuntivi, i quali corrispondono alle caratteristiche necessarie al lancio delle funzioni Serverless su AWS Lambda [16]:

- **dati dell'account AWS**: sono necessari `user_name`, `access_id_key` e `secret_access_key`;
- **regione**: id della regione su cui si vogliono istanziare i servizi;

- **linguaggio**: linguaggio di programmazione in cui tradurre le funzioni Lambda. Può essere `nodejs12.x` per JavaScript o `python` per Python;
- **thread**: numero di istanze concorrenti delle funzioni;
- **memoria**: quantità di memoria disponibile per l'esecuzione di una funzione;
- **time**: tempo limite per l'esecuzione di ogni funzione.

```
1 var cloud = [type="aws", user="user_name", access_id_key = "
      access_id_key", secret_access_key="secret_access_key", region="eu-
      west-2", language="nodejs12.x", thread=10, memory=256, time_=300]
```

*Listing 3.4: Dichiarazione di un ambiente AWS.*

Nel Listato 3.5 vediamo come vengono istanziati i servizi necessari all'esecuzione di un programma FLY utilizzando le informazioni fornite nella dichiarazione dell'ambiente. Tra i servizi troviamo AWS SQS [1] per le code di messaggi, AWS IAM [5] per l'autenticazione, AWS S3 [2] per la memorizzazione di dati e AWS Lambda [6] per l'esecuzione di funzioni Serverless.

```
1  static BasicAWSCredentials cloud = new BasicAWSCredentials("
       access_id_key", "secret_access_key");
2
3  static AmazonSQS __sqs_cloud  = AmazonSQSClient.builder()
4      .withCredentials(new AWSStaticCredentialsProvider(cloud))
5      .withRegion("eu-west-2")
6      .build();
7
8  static AmazonIdentityManagement __iam_cloud =
       AmazonIdentityManagementClientBuilder.standard()
9      .withCredentials(new AWSStaticCredentialsProvider(cloud))
10     .withRegion("eu-west-2")
11     .build();
12
13 static AWSLambda __lambda_cloud = AWSLambdaClientBuilder.standard()
14     .withCredentials(new AWSStaticCredentialsProvider(cloud))
15     .withRegion("eu-west-2")
16     .build();
17
18 static AmazonS3 __s3_cloud = AmazonS3Client.builder()
19     .withCredentials(new AWSStaticCredentialsProvider(cloud))
20     .withRegion("eu-west-2")
21     .build();
```

*Listing 3.5: Codice generato per l'ambiente AWS.*

### Ambiente su Cloud - Microsoft Azure

Come già specificato, l' utilizzo dei servizi a disposizione sul **Cloud di Microsoft Azure** risulta complesso e macchinoso a causa delle problematiche relative alle SDK fornite ed alla documentazione particolarmente scarna e spesso obsoleta. Per ovviare a tali problemi si è scelto di utilizzare il sistema di REST API attraverso le chiamate *HTTP*. La posibilità di integrare servizi di Microsoft Azure nel codice Java generato dal compilatore FLY è possibile grazie alla libreria AzureClient sviluppata presso l'ISISLab [22], che fornisce un'unica interfaccia, agevolando l'utilizzo dei servizi. Per gestire i servizi di Azure con JavaScript è necessario utilizzare delle chiamate *HTTP* e, nello specifico, vengono impiegate le librerie *axios* [8] e *qs* [41].Rimane comunque indispensabile il servizio Azure AD [9] per l'ottenimento del *token* di autorizzazione per poter effettuare tali chiamate.
Per utilizzare un ambiente di esecuzione su **Cloud Azure** è necessaria la dichiarazione di una variabile di `type="azure"`, che necessita di una serie di parametri aggiuntivi, i quali corrispondono alle caratteristiche necessarie al lancio delle funzioni Serverless con Azure Function [11]:

- **dati dell'account Azure**: sono necessari `client_id`, `tenant_id`, `secret_key` e `subscription_id`;
- **regione**: id della regione su cui lanciare i servizi;
- **linguaggio**: linguaggio di programmazione in cui tradurre ed eseguite le funzioni Azure. Può essere `nodejs12.x` per JavaScript o `python` per Python;
- **thread**: numero di istanze concorrenti delle funzioni;
- **time**: tempo limite per l'esecuzione di ogni funzione.

```
1 var cloud = [type="azure", clientID="client_id", tenantID="tenant_id",
      secret_key="secret_key", subscriptionID="subscription_id", region
    ="France Central", language="nodejs12.x", threads="10", time="300"
    ]
```

*Listing 3.6: Dichiarazione di un ambiente Azure.*

Il codice nel Listato 3.7 include la creazione di una variabile istanza della libreria AzureClient, che risulta necessaria per avviare le procedure relative ai servizi Azure. Nello specifico, il metodo `init()` si occupa di istanziare i servizi indispensabili per l'esecuzione di qualsiasi programma FLY, grazie le informazioni presenti nella dichiarazione dell'ambiente di esecuzione. Il

metodo `createFunctionApp(...)`, invece, consente la creazione dell'istanza del servizio Serverless di Azure per eseguire le funzioni.

```
cloud = new AzureClient("client_id",
    "tenant_id",
    "secret_key",
    "subscription_id",
    __id_execution+"",
    "France Central");

cloud.init();

cloud.createFunctionApp("FLYappcloud","nodejs12.x");
```

*Listing 3.7: Codice generato per l'ambiente Azure.*

### 3.5.2 *Channel*

Per dichiarare un `type="channel"`, cioè il canale di comunicazione utilizzato da FLY, è necessaria la specifica dell'ambiente di esecuzione su cui dovrà funzionare, come si evince dal Listato 3.8. Per fare ciò si usa la parola chiave `on` e, in base all'ambiente specificato, il compilatore genererà del codice differente in modo che il canale sfrutti la gestione delle code fornita dall'ambiente stesso. AWS utilizza il servizio AWS Simple Queue Service (SQS) [1], mentre Azure sfrutta i metodi della libreria AzureClient.
La logica del linguaggio FLY prevede che ad uno dei thread dell'ambiente locale venga associato il task di lettura della coda, in modo che resti in ascolto e, all'intercettazione dei messaggi, questi vengono immediatamente scritti sul canale sul quale è possibile leggerli.

```
var ch = [type="channel"] on cloud
```

*Listing 3.8: Dichiarazione di un Channel su ambiente Cloud.*

### 3.5.3 Script di deploy e undeploy delle funzioni FLY

L'esecuzione delle funzioni Serverless su ambiente Cloud richiede che alcune procedure di deploy ed undeploy vengano eseguite mediante gli strumenti di Command Line Interface forniti dai provider, per cui i comandi variano in base al provider scelto. Tuttavia, queste procedure non si differenziano particolarmente tra di loro, poiché per tutte sono necessari dei file in formato JSON per la specifica del ruolo e delle autorizzazioni con cui eseguire la funzione e per dichiarare le dipendenze del codice. FLY prevede la costruzione

del pacchetto per l'esecuzione delle funzioni Serverless prima del deploy su Cloud. Nello specifico, il pacchetto contiene sia i documenti JSON, sia le librerie necessarie per l'esecuzione, che vengono installate mediante il gestore di pacchetti *npm* [35] per JavaScript e *pip* [47] per Python.
Il codice contenuto negli script può essere suddiviso nelle sezioni descritte di seguito.

- **Verifica di parametri e requisiti** Inizialmente, lo script si occupa di verificare la presenza dei parametri necessari all'interazione con l'ambiente di esecuzione, per poi assegnarli ad apposite variabili. Successivamente si effettuano dei controlli per verificare che i pacchetti necessari siano installati sulla macchina, come ad esempio le CLI.

- **Inizializzazione dei servizi** Attraverso la CLI viene inizializzato l'ambiente del provider scelto utilizzando i dati di accesso forniti dall'utente. Specificamente, AWS necessita di un'istanza del servizio IAM [5], mentre Azure richiede semplicemente il login.

- **Creazione del virtual env - Python** Nel caso venga scelto Python come linguaggio per la funzione Serverless, è necessario creare un *virtual env*, cioé un ambiente virtuale che Python utilizza per installare librerie e script.

- **Creazione del progetto locale** Grazie ai comandi *bash*, lo script crea il progetto FLY, sia per quanto riguarda le cartelle che i file.In questa fase che vengono creati i file JSON per la configurazione del servizio FaaS, dei permessi e delle autorizzazioni.

- **Traduzione della funzione FLY** La funzione FLY viene tradotta nel linguaggio scelto e scritta in un file all'interno del progetto.

- **Deploy della funzione** In base al all'ambiente scelto si effettuano le procedure di deploy. AWS richiede l'inserimento delle librerie necessarie all'esecuzione nel pacchetto che verrà caricato su Cloud. Lo script utilizza i gestori di pacchetti *npm* o *pip* per la loro installazione e, successivamente, crea un file *.zip* contenente l'intero progetto. Nel caso di superamento dei limiti di dimensione imposti da AWS, si utilizza il servizio di archiviazione S3 [2] per il caricamento del file. Il deploy per Azure è meno macchinoso, poiché è necessario specificare le librerie da installare all'interno di un file di requisiti, in modo da mantenere la dimensione del progetto molto contenuta.

- **Undeploy della funzione** Una volta eseguite le funzioni FLY, lo script di undeploy si occupa di eliminare le istanze dei servizi creati in precedenza. Anche questa fase prevede l'utilizzo della CLI per AWS, mentre per Azure lo script di undeploy contiene solo i comandi per effettuare il logout, in quanto per l'undeploy dei servizi è necessario utilizzare i comandi della libreria AzureClient.

### 3.5.4 Sincronizzazione

Per eseguire le funzioni FLY su Cloud è necessario un sistema di sincronizzazione che possa accertarsi del fatto che tutte le funzioni FLY abbiano terminato l'esecuzione. L'implementazione di questo meccanismo si basa su una coda di terminazione sulla quale ogni funzione invia un messaggio una volta completata. Grazie all'uso di un contatore il sistema può accertarsi del fatto che tutte le funzioni siano terminate. L'implementazione di tale coda prevede la stessa logica vista per i `channel`, per cui anche questo oggetto è gestito dal programma stesso.

# Capitolo 4

# Algoritmi Genetici

## 4.1 Introduzione agli algoritmi genetici

Tra problemi risolubili attraverso il calcolo scientifico, i **problemi di ottimizzazione multi-obiettivo** sono da sempre ritenuti complessi ed onerosi in termini di risorse richieste per la computazione. Tali problemi sono caratterizzati da molteplici vincoli da rispettare ed obiettivi da raggiungere per poter trovare una soluzione che può essere ottima, nel caso sia la migliore soluzione in assoluto per il problema considerato, oppure ottimale nel caso in cui tale soluzione non sia la migliore, pur soddisfacendo tutti i vincoli del problema. Data la complessità dei problemi di ottimizzazione multi-obiettivo, è sorta la necessità di algoritmi che siano in grado di risolverli in un tempo accettabile dall'utenza. Il crescente interesse verso questa categoria di problemi ha portato alla nascita della **Programmazione genetica**, ovvero dell'insieme di metodologie di programmazione che traggono ispirazione dall'evoluzione biologica e che consentono lo sviluppo di algoritmi ad-hoc per svolgere un determinato compito.
Gli algoritmi sviluppati grazie alle tecniche di programmazione genetica prendono il nome di **Algoritmi Genetici**. Questa tipologia di algoritmi risulta concettualmente semplice, grazie alla similitudine dei metodi utilizzati e della nomenclatura delle componenti con il funzionamento e la nomenclatura dei processi biologici descritti nella Teoria dell'Evoluzione di Charles Darwin, risultando inoltre molto valida per la risoluzione di problemi di ottimizzazione multi-obiettivo [21].
Nello specifico, le componenti principali degli algoritmi genetici sono:

- **Gene**: componente di base del corredo genetico di un individuo;

- **Individuo**: particolare istanza del problema considerato, i cui geni rappresentano la configurazione di una possibile soluzione. Gli individui che raggiungono gli obiettivi del problema possono essere considerati come potenziali soluzioni;
- **Popolazione**: insieme di individui su cui eseguire la computazione;
- **Valore di fitness**: valore che rappresenta la valutazione assegnata ad un individuo, sulla base di un'analisi del suo corredo genetico, rispetto ai vincoli ed agli obiettivi del problema.

## 4.2 Struttura di un algoritmo genetico

Gli algoritmi genetici sono composti da alcune funzionalità di base, ognuna delle quali effettua una particolare operazione sulla popolazione. Nello specifico, le funzionalità sono:

- **Inizializzazione** - istanzia la prima generazione di individui;
- **Fitness** - funzionalità che si occupa di valutare gli individui;
- **Selezione** - funzionalità che permette di scegliere gli individui che potranno accedere al crossover;
- **Crossover** - permette di mescolare il corredo genetico di due individui, detti genitori.
- **Mutazione** - permette al corredo genetico di un individuo di subire mutazioni casuali.

In questa sezione verrà descritta la struttura di base di un algoritmo genetico e saranno descritte nello specifico le funzionalità principali che lo compongono.

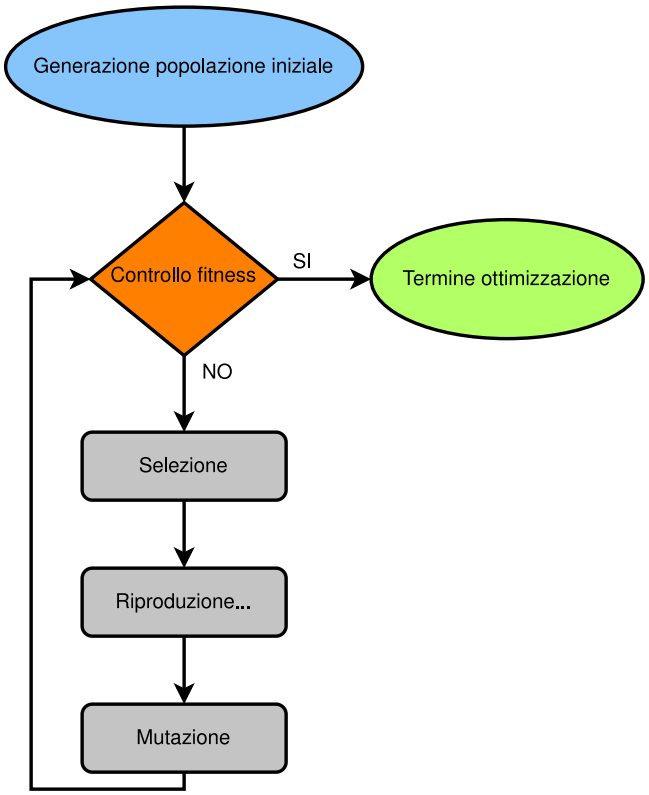


*Figura 4.1: Diagramma di flusso di un semplice algoritmo genetico.*

In generale, il funzionamento di un algoritmo genetico è riassumibile nelle seguenti fasi:

1. **Fase di inizializzazione**, in cui si genera la popolazione iniziale;
2. **Fase di valutazione**, in cui ad ogni individuo viene associato un valore di fitness;
3. **Fase di selezione**, in cui vengono scelti i migliori individui della popolazione in base al valore di fitness;
4. **Fase di crossover**, in cui il corredo genetico degli individui selezionati e suddivisi in coppie viene unito per creare un nuovo insieme di individui;

5. **Fase di mutazione**, in cui il corredo genetico dei nuovi individui subisce mutazioni casuali;

6. **Fase di verifica dei risultati**, in cui viene verificato se tra gli individui sia presente una soluzione al problema. Nel caso la soluzione esista l'algoritmo termina, altrimenti vengono ripetute tutte le fasi precedenti, tranne quella di inizializzazione.

Questa struttura ciclica dona particolare versatilità agli algoritmi genetici, consentendo agli sviluppatori di adattarli alla risoluzione dei problemi più disparati. Ad esempio, si può scegliere di terminare l'esecuzione dopo un certo numero di generazioni in cui non viene trovata nessuna soluzione, rispettando eventuali vincoli temporali per l'esecuzione dell'applicazione su cui si sta lavorando, oppure è possibile reinizializzare la popolazione nel caso ci si discosti troppo dagli obiettivi del problema.

### 4.2.1 Inizializzazione

L'inizializzazione avviene solitamente all'inizio dell'esecuzione di un algoritmo genetico e prevede la generazione di una popolazione iniziale, composta da un numero fisso di individui deciso dall'utente. La generazione dell'individuo può avvenire in maniera casuale o pseudo-casuale, a seconda delle esigenze specifiche del caso e, inoltre, può prevedere una prima valutazione atta ad re-inizializzare gli individui che non rispettano i vincoli del problema o che hanno un valore di fitness insufficiente rispetto ad un valore limite deciso dall'utente.

### 4.2.2 Fitness

L'assegnamento dei valori di fitness agli individui avviene solitamente in due momenti: una tantum appena dopo la fase di inizializzazione ed ogni volta che viene completata l'inserimento di nuovi individui nella popolazione, ovvero successivamente alla fase di mutazione. La funzione che ha il compito di generare i valori di fitness ha un ruolo centrale negli algoritmi genetici, in quanto su di essa si basa la capacità di quest'ultimo di fornire come risultato una soluzione valida. La funzione di fitness, a differenza delle altre funzionalità, deve essere sviluppata in modo specifico per il problema che si vuole risolvere, poiché deve valutare gli individui sulla base dei vincoli e degli obiettivi specifici del problema.

### 4.2.3 Selezione

La fase di selezione permette di scegliere, secondo un criterio definito dall'utente, un numero di individui da far partecipare al crossover. Solitamente, gli individui selezionati sono quelli con i migliori valori di fitness, tuttavia sono celebri anche metodologie di selezione che prevedono di creare alcune pool di individui presi in modo casuale, tra cui scegliere i migliori. La selezione si rivela particolarmente importante per ridurre il numero di generazioni necessarie all'algoritmo genetico per la risoluzione del problema. Una funzione di selezione ben progettata, infatti, consente di selezionare i migliori individui nella popolazione, aumentando di conseguenza la probabilità che venga generata una soluzione al problema durante la fase di crossover.

### 4.2.4 Crossover

La fase di crossover prevede la combinazione del corredo genetico di due individui, detti genitori, da cui vengono generati uno o più individui figli. Come le altre fasi, anche il crossover prevede una componente casuale, ovvero il punto di taglio degli individui genitori. L'operazione di taglio permette di creare porzioni di geni, la cui combinazione diventa poi il corredo genetico del nuovo individuo. Chiaramente, più il numero di tagli è elevato, più porzioni di geni vengono create, aumentando il numero di individui diversi generabili tramite una singola operazione di crossover, tuttavia aumentando i costi computazionali.

### 4.2.5 Mutazione

La fase di mutazione consente di introdurre mutazioni casuali nel corredo genetico degli individui. Solitamente la mutazione viene effettuata in maniera casuale solo sui nuovi individui, secondo alcuni valori definiti dall'utente, che aumentano o riducono le probabilità che tali mutazioni avvengano. Una fase di mutazione ben progettata può consentire di ridurre, anche notevolmente, i tempi di esecuzione. Benché questa funzionalità venga solitamente considerata poco importante, il suo ruolo risulta quindi fondamentale, in linea con quanto avvenuto con l'evoluzione delle specie, in cui la mutazione ha da sempre giocato un ruolo centrale.

## 4.3 Implementazione di un algoritmo genetico in FLY

Questo progetto di tesi nasce da due necessità principali: fornire degli esempi di algoritmi complessi agli sviluppatori che intendono approcciarsi a FLY e verificare che il compilatore FLY sia in grado di completare correttamente la compilazione di algoritmi complessi. Essendo FLY un linguaggio ancora in fase di sviluppo, infatti, gli esempi di algoritmi implementati in tale linguaggio sono poco numerosi e caratterizzati da una bassa complessità. Sviluppare algoritmi complessi consente, inoltre, di facilitare l'individuazione di eventuali errori all'interno del generatore FLY e quindi permetterne la risoluzione.
Per portare a compimento gli obiettivi preposti è necessario quindi:

- Sviluppare un algoritmo genetico di base in FLY, che consenta di migliorare e correggere, dove necessario, le funzionalità del compilatore;
- Sviluppare un algoritmo genetico che risolva un problema complesso, adattando l'algoritmo genetico di base.

## 4.4 Scelte implementative

Per l'implementazione degli esempi di algoritmi genetici in linguaggio FLY è stata necessaria un'attenta fase di progettazione atta a definire le caratteristiche principali dell'algoritmo di base e le modalità con cui quest'ultimo debba interagire con il Cloud. Nello specifico, durante la fase di progettazione sono stati definite le principali varianti delle funzioni da utilizzare per la realizzazione delle funzioni caratteristiche di un algoritmo genetico e quali di queste debbano essere eseguite su ambiente Cloud.
Nello specifico, le varianti scelte per le funzionalità sono:

- **Selezione - Tournament selection**, ovvero una tipologia di selezione basata su dei tornei tra individui scelti casualmente tra la popolazione. Una volta creati i tornei, gli individui partecipano ad una serie di prove virtuali, il cui vincitore, detto anche campione, può accedere al crossover;
- **Crossover - One point crossover**, particolare tipologia di crossover che prevede di effettuare un solo taglio in un punto casuale del corredo genetico dei genitori. Ogni coppia di genitori può quindi generare al massimo due nuovi individui;

- **Mutazione - Gaussian mutation**, particolare tipologia di mutazione completamente basata sulla probabilità. Nello specifico, sono previste almeno due valori scelti dall'utente che permettono rispettivamente all'individuo di accedere alla mutazione e al gene considerato di mutare. Durante l'esecuzione di questa funzionalità, vengono generati dei valori casuali che, in caso siano superiori ai valori limite impostati, permettono alle operazioni di proseguire. Nel caso l'individuo riesca ad accedere alla mutazione, si procede ad analizzare ogni singolo gene di quest'ultimo ed a generare valori casuali che, nel caso siano superiori al valore limite, permetteranno al gene di mutare.

Per l'esecuzione su ambiente Cloud è stata scelta la funzione per il calcolo della fitness, dati gli elevati costi computazionali che essa comporta. In questo modo è possibile suddividere la popolazione in gruppi da assegnare alle varie istanze della funzione e migliorare così le prestazioni generali dell'algoritmo.

### 4.4.1 Fasi dello sviluppo

Lo sviluppo degli algoritmi genetici è stato suddiviso in cinque fasi, seguendo un modello di sviluppo incrementale che ha portato da un comune algoritmo genetico scritto in linguaggio Java ad un algoritmo eseguibile in modo parallelo su ambiente Cloud, scritto in linguaggio FLY.
Le fasi dello sviluppo sono:

- **Progettazione** - fase iniziale in cui sono state scelte le varianti delle funzioni da implementare, oltre che le strutture dati necessarie al funzionamento dell'algoritmo. In questa fase è stato sviluppato anche il diagramma di flusso presente nella Figura 4.1 come base da cui partire per lo sviluppo, poi modificato nelle fasi successive;

- **Sviluppo dell'algoritmo in Java** - durante questa fase l'algoritmo è stato scritto in Java secondo le scelte effettuate in fase di progettazione dell'algoritmo. Durante questa fase sono state inserite alcune modifiche al sopracitato diagramma di flusso, in modo da ottenere migliori prestazioni. Nello specifico, è stato scelto di valutare tutta la popolazione subito dopo l'inizializzazione della popolazione e, successivamente, valutare soltanto i nuovi individui, in modo da non dover valutare l'intera popolazione ad ogni generazione. Inoltre, è stato scelto di avere una popolazione a numero fisso, sostituendo i peggiori individui della popolazione con i migliori individui generati durante il crossover;

- **Conversione dell'algoritmo in FLY** - durante questa fase l'algoritmo è stato riscritto in linguaggio FLY, utilizzando i costrutti e le caratteristiche peculiari del linguaggio, mantenendo però le medesime caratteristiche e prestazioni dell'algoritmo scritto in Java;
- **Passaggio da fitness sequenziale a fitness parallela** - fase in cui è stata modificata la funzione di fitness per consentirne l'esecuzione parallela su ambiente SMP. Grazie all'astrazione fornita da FLY, le modifiche apportate in questa fase hanno permesso alla funzione di essere eseguita anche su ambiente Cloud;
- **Adattamento dell'algoritmo all'esecuzione Serverless** - fase finale dello sviluppo in cui sono state inserite le istruzioni necessarie per recuperare i risultati della fitness su ambiente Cloud e di assegnare questi ultimi ai rispettivi individui all'interno della popolazione locale.

Alla fine delle cinque fasi è stato prodotto un diagramma di flusso aggiornato, come è possibile vedere dalla Figura 4.2.


*Figura 4.2: Diagramma di flusso aggiornato.*

### 4.4.2 Implementazioni aggiuntive

Per consentire la corretta compilazione ed esecuzione degli algoritmi genetici in linguaggio FLY è stato necessario modificare alcune funzionalità del compilatore FLY. Tali modifiche hanno avuto il duplice obiettivo di consentire l'implementazione di algoritmi complessi ed esosi di risorse, come appunto gli algoritmi genetici, e di permettere il riconoscimento e la correzione

di eventuali errori già presenti all'interno del compilatore. Le modifiche apportate hanno riguardato soprattutto il modo in cui il generatore Java gestisce il codice delle funzioni FLY e delle varie risorse ad esse collegate. Altre modifiche minori hanno riguardato la correzione di piccoli errori che impedivano la corretta generazione del codice Javascript da parte del relativo generatore e la gestione delle strutture dati, specificamente delle matrici e degli array, da parte del generatore Java. Inoltre, la codifica delle matrici da fornire in input alle funzioni FLY su ambiente Cloud è stata aggiornata in modo da prevedere un campo indicante il numero progressivo della porzione di matrice passata all'istanza della funzione Serverless. Ciò ha consentito di riordinare facilmente i risultati ottenuti dalla funzione di fitness, in modo da poterli assegnare correttamente ai rispettivi individui.

## 4.5 Algoritmo genetico di base

Per poter sviluppare un algoritmo genetico è necessario avere un problema da risolvere. L'algoritmo genetico di base implementato in questo lavoro di tesi consente di risolvere il problema di massimizzazione del vettore binario, meglio conosciuto nella letteratura scientifica come **One Max Problem**. Si tratta di un problema molto semplice e, per tale motivo, molto utilizzato come esempio introduttivo ai concetti relativi all'intelligenza artificiale. Lo scopo del problema è trovare un vettore composto unicamente da 1, mentre il corredo genetico di un individuo può essere rappresentato da una sequenza di lunghezza fissa composta dai caratteri 0 e 1. Questo problema è stato scelto poiché si presta particolarmente bene alla risoluzione mediante algoritmi genetici. Inoltre, la semplicità dell'algoritmo genetico ottenuto in linguaggio FLY risulta essere un ottimo esempio per gli sviluppatori che vogliono approcciarsi a tale linguaggio, oltre che particolarmente adatto per essere utilizzato come base per lo sviluppo di altri algoritmi genetici, in linea con gli obiettivi del progetto.

### 4.5.1 Codice dell'algoritmo

In questa sezione verranno mostrate alcune porzioni di codice relative all'algoritmo genetico implementato.

```
1 func initPopulation(){
2     var newPop = Integer[dimension][individualLength]
```

```
    population= newPop
    var rand = [type="random"]

    for i in [0:dimension]{
        for j in [0:individualLength]{
            var value = 0
            value = rand.nextInt(2)

            if(value < 0){
                value = value * -1
            }

            population[i][j]= value
        }
    }
}
```

*Listing 4.1: Funzione di inizializzazione per il One Max Problem.*

La funzione di inizializzazione per il One Max Problem mostrata nel Listato 4.1 permette di istanziare la prima generazione di individui, il cui corredo genetico sarà composto esclusivamente da geni con valori casuali tra 0 e 1.

```
func fitness(populationMatrix){
    native<<<
    var resultJSON = "[" + submatrixIndex + ", " +
    __populationMatrix_rows + ", "

    for(var __i = 0; __i < __populationMatrix_rows; __i++){
        var fitnessValue = 0
        for(var __j = 0; __j < __populationMatrix_cols; __j++){
            if(populationMatrix[__i][__j] == 1){
                fitnessValue += 1
            }
        }
        resultJSON += "" + fitnessValue

        if(__i >= 0 && __i < __populationMatrix_rows - 1){
            resultJSON += ", "
        }
    }

    resultJSON += "]";
    console.log("Result: " + resultJSON)
        __data = await __sqs.getQueueUrl({ QueueName: "ch-'${id}'"}).
    promise();

        __params = {
            MessageBody : JSON.stringify(resultJSON),
            QueueUrl : __data.QueueUrl
        };

        __data = await __sqs.sendMessage(__params).promise();
```

```
29          __data = await __sqs.getQueueUrl({ QueueName: "termination-'${
    function}'-'${id}'"}).promise();
30      >>>
31  }
```

*Listing 4.2: Porzione di codice per il calcolo del valore di fitness su ambiente AWS.*

Il calcolo del valore di fitness per il One Max Problem si basa sul conteggio dei geni con valore 1 all'interno del corredo genetico dell'individuo. Gran parte del codice della funzione mostrata nel Listato 4.2, infatti, permette di esplorare la matrice estratta dalla stringa JSON passata come parametro alla funzione. Nello specifico, ciò avviene all'interno dei blocchi `for` presenti alle righe 5 e 7 con cui si gestisce l'esplorazione di ogni singola cella, inoltre alla riga 25 viene effettuato l'aggiornamento del valore di fitness in base al valore del gene esaminato.

```
1  for i in[0:nthread]{
2      var e = ch? as String
3      e = e.replace('\\"', "")
4      e = e.replace('\\"', "")
5
6      var stIndex = 0
7      stIndex = e.indexOf(',')
8
9      var matrixPortion = 0
10     matrixPortion = e.substring(1, stIndex) as Integer
11
12     results[matrixPortion] = e
13 }
```

*Listing 4.3: Recupero e ordinamento dei risultati dell'esecuzione della funzione di fitness Serverless.*

Meccanismo particolarmente importante è quello del recupero e ordinamento dei risultati della funzione di fitness mostrato nel Listato 4.3. Grazie al numero progressivo della porzione di matrice passata come parametro, detto **submatrixIndex**, è possibile ordinare i risultati ottenuti, per poi effettuare l'aggiornamento dell'array contenente i valori di fitness per l'intera popolazione. Tale valore, di cui si osserva il recupero alla riga 10, viene inserito sia nella stringa JSON passata come parametro alla funzione di fitness, sia come primo elemento nella stringa che ne contiene i risultati dell'esecuzione.

# Capitolo 5

# K-Colorazione

L'implementazione di un esempio di algoritmo genetico basato su un problema complesso ricopre un ruolo centrale all'interno del progetto. Per raggiungere questo obiettivo è necessario che il problema sia semplice da capire e che l'algoritmo sia implementato sulla base di quello descritto nel **Capitolo 4**. Un problema che possiede le suddette caratteristiche è quello della **K-Colorazione**, la cui descrizione verrà approfondita nel **Paragrafo 5.1**. Tale problema, inoltre, è facilmente risolubile attraverso l'impiego degli algoritmi genetici.
In questo capitolo verranno descritti il problema della K-Colorazione e l'algoritmo genetico implementato, prestando particolare attenzione al codice delle principali funzioni che lo compongono.

## 5.1 Descrizione del problema

Il problema della K-Colorazione (o K-Colorabilità) di un grafo non orientato è un problema molto conosciuto in ambito informatico, specialmente in quello dell'informatica teorica. Si tratta di un problema appartenente alla classe di complessità NP, nello specifico alla classe dei problemi NP-Completi (NP-C), ovvero quella classe di problemi per cui non esiste un algoritmo eseguibile in tempo polinomiale rispetto alla grandezza dell'input. La peculiarità della classe NP-C risiede nel fatto che, nel caso si trovi un algoritmo eseguibile in tempo polinomiale per un qualsiasi problema appartenente a tale classe, allora esisterebbe un algoritmo eseguibile in tempo polinomiale per tutti i problemi NP-Completi.
Il problema della K-Colorazione consiste nel colorare i nodi di un grafo non

orientato con il minimo numero di colori, con il vincolo di non poter assegnare lo stesso colore a due nodi collegati tra loro mediante un arco [15].

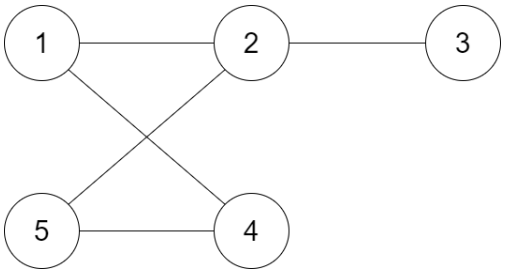


*Figura 5.1: Esempio di grafo non orientato su cui applicare l'algoritmo della K-Colorabilità.*

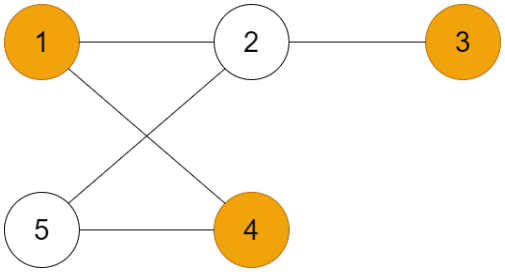


*Figura 5.2: Soluzione al problema della K-Colorabilità sul grafo mostrato nella Figura 5.1.*

Per poter generare una soluzione al problema della K-Colorazione bisogna:

1. generare un assegnamento di colori per il grafo;
2. calcolare eventuali conflitti tra le colorazioni dei nodi adiacenti;
3. scegliere il colore da sostituire nel caso si trovino conflitti;
4. ricominciare il ciclo dal calcolo dei conflitti.

Questa struttura ciclica permette di integrare facilmente l'algoritmo di risoluzione per il problema della K-Colorazione con un algoritmo genetico. Al fine di migliorare i tempi di computazione, inoltre, è possibile programmare un algoritmo in grado di fornire una soluzione ottimale al problema, anziché ottima. Questo consente, tra l'altro, di evitare costi elevati se si vuole implementare una soluzione basata sul paradigma Serverless e di fornire

rapidamente una soluzione all'utente. Questo insieme di caratteristiche rende l'algoritmo genetico per la risoluzione del problema della K-Colorabilità un ottimo esempio da fornire ai nuovi sviluppatori.

## 5.2 Implementazione

L'implementazione dell'algoritmo genetico ha previsto la modifica dell'algoritmo di base descritto nel precedente capitolo. Data la diversità dei problemi trattati, l'algoritmo di base ha subito diverse modifiche, sia per quanto concerne la struttura degli oggetti, sia per quanto riguarda la logica delle funzioni. Inoltre, al fine di minimizzare i tempi di computazione è stato introdotto un sistema di gestione delle possibili soluzioni.
In questa sezione verranno descritte nello specifico la logica dell'algoritmo, le modifiche apportate all'algoritmo di base e le aggiunte che si sono rese necessarie.

### 5.2.1 Modifiche apportate

Per poter gestire il problema della K-Colorazione è stato necessario trovare un compromesso tra le caratteristiche necessarie per la risoluzione del problema e le caratteristiche di un algoritmo genetico. Per fare ciò è stato necessario decidere a priori la logica dell'algoritmo e le eventuali funzionalità da aggiungere. L'obiettivo è quello di evitare modifiche alla logica dell'algoritmo di base, adattando le strutture dati utilizzate e le funzioni per consentire la risoluzione del nuovo problema.
Il punto di partenza è la gestione della funzione di inizializzazione della popolazione, che non dovrà più generare valori casuali tra 0 e 1, ma schemi di colorazione basati sui dati del grafo dato in input dall'utente. Nello specifico, le codifiche degli schemi di colorazione sono rappresentate da valori che vanno da 0 al numero di nodi presenti nel grafo. Il numero di nodi, infatti, rappresenta il massimo numero di colori diversi che possono essere utilizzati dall'algoritmo, facendo quindi da limite superiore ai valori generabili dalla funzione di inizializzazione.
Altro punto fondamentale è la gestione della funzione di fitness, che non dovrà più contare i geni con valore 1, ma dovrà contare i **conflitti** presenti nello schema di colorazione. Un conflitto rappresenta una violazione dei vincoli del problema, che in questo caso consiste nell'avere nodi adiacenti con lo stesso colore. La strategia di esplorazione del grafo prevede che per ogni nodo si confronti il colore di quest'ultimo con ciascuno dei nodi adiacenti,

incrementando un contatore nel caso in cui i colori corrispondano. É bene tener presente che con questa strategia di esplorazione del grafo i conflitti verranno contati due volte e che, dunque, il reale numero di conflitti equivale alla metà del valore calcolato dalla funzione di fitness.
Per consentire il calcolo del numero dei conflitti è necessario trovare una rappresentazione adeguata per i collegamenti presenti nel grafo. Una struttura dati che si presta particolarmente bene per la rappresentazione di tali collegamenti è la **matrice di adiacenza**, ovvero una matrice quadrata la cui dimensione si basa sul numero di vertici del grafo, in cui ogni cella rappresenta un eventuale collegamento tra un *vertice A* ed un *vertice B*. Se il collegamento esiste, allora la cella conterrà *1*, altrimenti conterrà *0*. Tale struttura dati è stata impiegata per consentire l'implementazione della funzione di fitness.
Ai fini della risoluzione del problema, l'individuo migliore sarà quello con il minor numero di conflitti. Questo cambia la tipologia di ottimizzazione che si desidera ottenere, che in questo caso consisterà in una minimizzazione del valore di fitness, rispetto all'algoritmo genetico di base che invece prevedeva una massimizzazione di tale valore. Inoltre, a parità di valore di fitness, l'individuo migliore sarà quello con il minor numero di colori diversi nel suo corredo genetico.
Sono state inoltre mantenute le varianti delle funzioni di selezione, crossover e mutazione, che sono state adattate alle caratteristiche del nuovo problema. A causa della diversa tipologia di ottimizzazione, la funzione di selezione favorirà gli individui con valore di fitness minore, inoltre per via dei nuovi valori che i geni possono assumere, la funzione di mutazione permetterà alterazioni casuali dei geni con valori da 0 al numero di nodi del grafo.

### 5.2.2 Strutture dati impiegate

Le modifiche apportate all'algoritmo descritto nel **Capitolo 4** hanno permesso di adattare le funzioni già implementate alle specifiche del problema della K-Colorazione. Come per le varianti delle funzioni, anche gli oggetti utilizzati sono rimasti invariati. Nello specifico, tali oggetti sono:

- **Popolazione** - matrice di grandezza ($numeroIndividui \times lunghezzaIndividuo$);
- **Individuo** - riga della matrice;
- **Fitness** - array di interi che associa il valore presente nella i-esima cella all'individuo che occupa l'i-esima riga della matrice;

- **Campioni** - matrice di grandezza ($numeroCampioni \times lunghezzaIndividuo$);
- **Nuova generazione** - matrice di grandezza ($numeroNuoviIndividui \times lunghezzaIndividuo$);

### 5.2.3 Funzionalità aggiuntive

Come già specificato, il problema della K-Colorazione è un problema NP-Completo, ovvero un problema risolubile in *tempo polinomiale non deterministico*. Ciò significa che, ad oggi, tale problema non può essere risolto in modo efficiente, ovvero trovando la migliore soluzione in un tempo accettabile per l'utente, andando in conflitto con le peculiarità del Serverless Computing. Il modello di costo delle funzioni Serverless, infatti, rende impossibile la risoluzione del problema senza incorrere in un costo finale eccessivo. Nello specifico, tale costo si basa principalmente su due fattori: tempo di esecuzione e risorse impiegate. Tuttavia, un problema NP-Completo non possiede una complessità computazionale ben definita, in quanto è impossibile stimare i tempi di risoluzione di un problema sulla base dei dati forniti in input. Di fatto, la generazione di una soluzione ottima da parte dell'algoritmo genetico potrebbe essere molto lenta e, di conseguenza, anche molto costosa.
Per ottimizzare i costi è stata sfruttata una delle caratteristiche peculiari degli algoritmi genetici, ovvero il fatto che possano trovare non solo soluzioni ottime, ma anche soluzioni ottimali. Come già specificato, una soluzione ottimale consiste in una soluzione che non viola i vincoli del problema e che non è la migliore in assoluto. Nel caso del problema della K-Colorazione una soluzione ottimale è rappresentata da un individuo con minimo numero di conflitti (valore di fitness uguale a 0) e con uno schema di colorazione che non contenga il minimo numero di colori utilizzabili per il grafo fornito in input.
Una soluzione del genere può essere generata molto prima della soluzione ottima, risolvendo di fatto il problema dei tempi e dei costi di esecuzione. Per poter fornire anche soluzioni ottimali all'utente è stato introdotto un meccanismo di controllo delle soluzioni che tiene conto, per ogni generazione, di quale sia la migliore soluzione a disposizione. Il controllo delle generazioni prevede che una volta trovata una soluzione ottimale, definita **soluzione candidata** al problema, venga inizializzato un contatore di generazioni con un valore fornito dall'utente. Questo contatore rappresenta il numero di generazioni rimanenti al termine dell'esecuzione, che avviene se e solo se la soluzione candidata non cambia fino all'esaurimento del numero di generazioni disponibili. Quando il contatore raggiunge lo 0, infatti, la soluzione

candidata diventa **definitiva** e viene fornita all'utente. Nel caso la soluzione candidata cambi durante il conteggio delle generazioni rimanenti, il contatore viene re-inizializzato al valore di partenza, ovvero quello fornito dall'utente. Grazie a questo meccanismo è possibile quindi ridurre il tempo di esecuzione a seconda delle esigenze dell'utente.

## 5.3 Codice delle funzioni

In questa sezione verrà mostrato il codice delle principali funzioni implementate.

### 5.3.1 Inizializzazione

```
1  func initPopulation(){
2      var newPop = Integer[dimension][individualLength]
3      population= newPop
4      var rand = [type="random"]
5
6      for i in [0:dimension]{
7          for j in [0:individualLength]{
8              var value = 0
9              value = rand.nextInt(individualLength)
10
11             if(value < 0){
12                 value = value * -1
13             }
14
15             population[i][j]= value
16         }
17     }
18 }
```

*Listing 5.1: Codice FLY per l'inizializzazione.*

Come mostrato nel Listato 5.1, alla riga 9 la funzione di inizializzazione non genera più valori compresi tra 0 e 1, ma valori compresi tra 0 ed il numero di nodi presenti nel grafo, valore che rappresenta anche la lunghezza di un individuo.

### 5.3.2 Fitness

```
1      var resultJSON = "[" + submatrixIndex + ", " +
       __populationMatrix_rows + ", "
2
3      for(var __i = 0; __i < __populationMatrix_rows; __i++){
4          var fitnessValue = 0
```

```
5         for(var __j = 0; __j < __populationMatrix_cols; __j++){
6             var currentColor = populationMatrix[__i][__j]
7             for(var __k = 0; __k < __populationMatrix_cols; __k++){
8                 var edgeValue = edges[__j][__k]
9                 var edgeColor = populationMatrix[__i][__k]
10
11                if(__k != __j && edgeValue == 1 && currentColor ==
    edgeColor){
12                    fitnessValue += 1
13                }
14            }
15        }
16        resultJSON += "" + (fitnessValue/2)
17
18        if(__i >= 0 && __i < __populationMatrix_rows - 1){
19            resultJSON += ", "
20        }
21    }
22
23    resultJSON += "]";
```

*Listing 5.2: Frammento di codice Javascript per il calcolo della fitness.*

La funzione di Fitness mostrata nel Listato 5.2 effettua una valutazione più complessa rispetto alla funzione progettata per il One Max Problem mostrata nel Listato 4.2. In particolare, il nuovo sistema di valutazione si basa sul numero di conflitti nel pattern di colorazione, anziché sul numero di geni con valore 1.
Nello specifico, i due `for` presenti alle righe 3 e 5 consentono di analizzare ogni singolo elemento della matrice estratta dalla stringa JSON passata alla funzione. Per ogni elemento analizzato viene effettuata una scansione completa della matrice di adiacenza, in modo da verificare la sussistenza di eventuali conflitti nel pattern di colorazione. Nel blocco `if` alla riga 11 avviene la fase cruciale del confronto, in cui viene controllata la presenza del conflitto attraverso la verifica di 3 condizioni:

1. il nodo esaminato non corrisponde al nodo rappresentato dalla cella della matrice con cui viene confrontato;

2. la cella della matrice di adiacenza contiene il valore 1, ovvero sussiste un collegamento tra i nodi in esame;

3. i valori rappresentanti la colorazione dei nodi corrispondono.

Nel caso in cui tutte le condizioni siano verificate si è in presenza di un conflitto. In questo caso, il conflitto viene sommato agli altri eventualmente trovati.

Tutti i conflitti vengono contati due volte, poiché essendo il grafo non orientato, nel caso di un conflitto tra un nodo A ed un nodo B, il conteggio avverrà sia durante l'analisi dei collegamenti di A , che durante la medesima analisi per il nodo B. Per tale motivo, alla riga 16 il numero di conflitti trovati viene dimezzato.

### 5.3.3 Selezione

```
1  func tournamentSelection(){
2      var selectedIndividual = -1
3      var rand = [type="random"]
4
5      var tournament = Integer[tournamentSize]
6
7      for j in [0:tournamentSize]{
8          var element = 0
9          element = rand.nextInt(dimension)
10
11         if(element < 0){
12             element = element * -1
13         }
14
15         tournament[j] = element
16
17         for i in [0:tournamentSize]{
18             if(i != j){
19                 if(tournament[i] == tournament[j]){
20                     tournament[j] = -1
21                 }
22             }
23         }
24
25         if(tournament[j] == -1){
26             j = j - 1
27         }
28     }
29
30     var best = 0
31     best = tournament[0]
32
33     for j in [1:tournamentSize]{
34         if(populationFitness[j] < populationFitness[best]){
35             best = tournament[j]
36         }
37     }
38
39     selectedIndividual = best
40     return selectedIndividual
41 }
```

*Listing 5.3: Codice FLY per la selezione.*

La funzione di selezione mostrata nel Listato 5.4 ha subito un solo cambiamento rispetto alla sua controparte per il One Max Problem, precisamente alla riga 34. Nel problema affrontato nel **Capitolo 4**, infatti, si vuole massimizzare il valore di fitness, mentre in questo caso si desidera l'esatto opposto, ovvero una minimizzazione. Per tale motivo, la funzione di selezione per il problema della K-Colorazione favorisce la selezione degli individui con il valore di fitness più basso.

### 5.3.4 Crossover

```
1  func onePointCrossover(){
2      var rand = [type="random"]
3      var crossoverPoint = 0
4      crossoverPoint = rand.nextInt(individualLength - 1)
5
6      while(crossoverPoint == 0){
7          crossoverPoint = rand.nextInt(individualLength - 1)
8      }
9
10     if(crossoverPoint < 0){
11         crossoverPoint = crossoverPoint * -1
12     }
13
14     var p1 = Integer[individualLength]
15     var p2 = Integer[individualLength]
16
17     for i in [0:individualLength]{
18         p1[i] = population[parent1][i]
19     }
20
21     for i in [0:individualLength]{
22         p2[i] = population[parent2][i]
23     }
24
25     var last2= Integer[individualLength - crossoverPoint]
26     var n = 0
27     for i in [crossoverPoint:individualLength]{ //Swap 1
28         last2[n] = p2[i]
29         n++
30         p2[i] = p1[i]
31     }
32
33     var first1 = Integer[crossoverPoint]
34     n = 0
35     for i in [0:crossoverPoint]{ //Swap 2
36         first1[n] = p1[i]
37         n++
38         p1[i] = p2[i]
39     }
40
41     n = 0
```

```
42      var m = 0
43      for i in [0:crossoverPoint]{
44          if(i < crossoverPoint){
45              p2[i] = first1[n]
46              n++
47          }
48          else{
49              p2[i] = last2[m]
50              m++
51          }
52      }
53
54      for i in [0:individualLength]{
55          child1[i] = p1[i]
56          child2[i] = p2[i]
57      }
58  }
```

*Listing 5.4: Codice FLY per il crossover.*

La funzione di Crossover mostrata nel Listato 5.4 è rimasta invariata rispetto alla sua implementazione per il One Max Problem.
Dopo aver generato il punto di taglio alla riga 8, la funzione di occupa della combinazione del corredo genetico degli individui genitori nei `for` presenti alle righe 17 e 21, inizializzando la prima parte degli individui figli. Il completamento del crossover avviene grazie al `for` presente alla riga 27, in cui vengono riempite le celle successive al punto di taglio per entrambi gli individui figli.

### 5.3.5 Mutazione

```
1   func gaussianMutation(){
2       var rand = [type="random"]
3
4       for i in [0:individualLength]{
5           var mutationChance = 0
6           mutationChance = rand.nextInt(2)
7
8           if(mutationChance < 0){
9               mutationChance = mutationChance * -1
10          }
11
12          if(mutationChance > 0){
13              var mutationValue = 0
14              mutationValue = rand.nextInt(individualLength)
15
16              if(mutationValue < 0){
17                  mutationValue = mutationValue * -1
18              }
19                  toMutate[i] = mutationValue
```

```
20              }
21          }
22  }
```

*Listing 5.5: Codice FLY per la mutazione.*

La funzione di mutazione mostrata nel Listato 5.5 genera valori diversi dalla sua controparte per il One Max Problem. Come per la funzione di inizializzazione, infatti, il nuovo insieme di valori generabili comprende tutti i valori tra 0 ed il numero di nodi del grafo.

# Capitolo 6

# Conclusioni

FLY è un Domain-Specific Language il cui obiettivo è permettere lo sviluppo di applicazioni di calcolo scientifico che sfruttino il paradigma multi-cloud per ottenere elevata scalabilità, alte prestazioni ed una maggiore tolleranza ai malfunzionamenti.Essendo FLY un linguaggio ancora in fase di sviluppo, gli esempi a disposizione dell'utenza sono pochi e poco complessi.
La prima parte del progetto riguarda la progettazione e l'implementazione di un semplice algoritmo genetico in linguaggio FLY. Gli algoritmi genetici sono algoritmi basati sui concetti relativi all'evoluzione delle specie, concettualmente semplici da comprendere e molto apprezzati per la risoluzione di problemi di ottimizzazione. Per consentire una corretta generazione per questa tipologia di algoritmi, è stato necessario apportare alcuni miglioramenti ai meccanismi presenti nel generatore FLY, in particolare a quelli relativi alla gestione delle matrici. Il problema che si intende risolvere con l'algoritmo genetico è il One Max Problem, un problema molto conosciuto in cui si desidera trovare un individuo il cui corredo genetico sia formato esclusivamente da 1. Una volta scelte le varianti per le funzioni caratteristiche dell'algoritmo genetico, è stato sviluppato il diagramma di flusso di quest'ultimo. In un primo momento, è stato sviluppato un algoritmo genetico per l'esecuzione in ambiente locale, che poi è stato progressivamente modificato per consentire dapprima l'esecuzione parallela della funzione di fitness e, successivamente, l'esecuzione serverless di quest'ultima su ambiente Cloud.
La seconda parte della tesi riguarda lo sviluppo di un algoritmo genetico che risolva un problema complesso. Per l'implementazione di questo algoritmo è stato scelto il problema della la K-Colorazione di un grafo non orientato. Tale problema appartiene alla classe dei problemi NP-Completi, i quali non hanno un tempo di esecuzione ben definito. L'algoritmo genetico per la

K-Colorazione è stato sviluppato modificando l'algoritmo sviluppato per la risoluzione del One Max Problem, che è stato adattato alle specifiche del nuovo problema. Per la rappresentazione del grafo è stata introdotta una matrice di adiacenza, rappresentante i collegamenti mediante archi tra i suoi vertici. Sono state apportate piccole modifiche alle funzioni di inizializzazione, selezione e mutazione, mentre la funzione di fitness è stata implementata da zero. Infine, per consentire l'esecuzione dell'algoritmo in tempi accettabili è stato introdotto un meccanismo di controllo delle generazioni che consente di terminare l'esecuzione nel caso non vengano trovate soluzioni con caratteristiche migliori a quelle attualmente disponibili tra gli individui della popolazione.
In conclusione, grazie a questo progetto di tesi sono stati forniti due nuovi esempi di algoritmi complessi agli sviluppatori che intendono approcciarsi al linguaggio FLY e sono state migliorate alcune funzionalità presenti all'interno del generatore.

# Bibliografia


[1] *Amazon Simple Queue Service - Amazon Web Services.* URL: https://aws.amazon.com/it/sqs/.

[2] *Amazon Simple Storage Service - Amazon Web Services.* URL: https://aws.amazon.com/it/s3/.

[3] *Amazon Web Services (AWS).* URL: https://aws.amazon.com/.

[4] *AWS Command Line Interface - Amazon Web Services.* URL: https://aws.amazon.com/it/cli/.

[5] *AWS Identity and Access Management - Amazon Web Services.* URL: https://aws.amazon.com/it/iam/.

[6] *AWS Lambda - Amazon Web Services.* URL: https://aws.amazon.com/it/lambda/.

[7] *AWS SDK - Amazon Web Services.* URL: https://aws.amazon.com/it/tools/.

[8] *axios - npm.* URL: https://www.npmjs.com/package/axios.

[9] *Azure Active Directory — Microsoft Azure.* URL: https://azure.microsoft.com/it-it/services/active-directory/.

[10] *Azure Command Line Interface — Microsoft Azure.* URL: https://docs.microsoft.com/it-it/cli/azure/?view=azure-cli-latest.

[11] *Azure Function — Microsoft Azure.* URL: https://docs.microsoft.com/it-it/azure/azure-functions/.

[12] *Azure Functions — Microsoft Azure.* URL: https://azure.microsoft.com/it-it/services/functions/.

[13] *Boto3 SDK AWS per Python - Amazon Web Services.* URL: https://aws.amazon.com/it/sdk-for-python/.

[14] *CAMEL.* URL: http://camel-dsl.org/.

[15] P. Cautela. ≪UN ALGORITMO GENETICO PARALLELO PER LA K-COLORABILITA≫. 2013.

[16] *Configuration functions in the AWS Lambda Console*. URL: `https://docs.aws.amazon.com/lambda/latest/dg/configuration-console.html`.

[17] Gennaro Cordasco et al. ≪Toward a domain-specific language for scientific workflow-based applications on multicloud system≫. In: *Concurrency and Computation: Practice and Experience* (2020).

[18] *Docker*. URL: `https://www.docker.com/`.

[19] *Docker Compose*. URL: `https://docs.docker.com/compose/`.

[20] Ana Juan Ferrer, David García Pérez e Román Sosa González. ≪Multi-cloud Platform-as-a-service Model, Functionalities and Approaches≫. In: *Procedia Computer Science* 97 (2016). 2nd International Conference on Cloud Forward: From Distributed to Complete Computing, pp. 63–72.

[21] Marco Di Gennaro. ≪L'evoluzione simulata: vita artificiale e algoritmi genetici≫. 2009.

[22] G. Grieco. ≪Progettazione e implementazione del supporto a Microsoft Azure per il compilatore del linguaggio FLY≫. 2019.

[23] Geir Horn e Pawel Skrzypek. ≪MELODIC: Utility Based Cross Cloud Deployment Optimisation≫. In: *2018 32nd International Conference (WAINA)* (2018).

[24] K. Hwang, G.C. Fox e J.J. Dongarra. *Distributed and Cloud Computing: From Parallel Processing to the Internet of Things*. Morgan Kaufmann. Morgan Kaufmann, 2012.

[25] *JavaScript*. URL: `https://developer.mozilla.org/it/docs/Web/JavaScript`.

[26] Eric Jonas et al. ≪Cloud Programming Simplified: A Berkeley View on Serverless Computing≫. In: *ArXiv* (feb. 2019).

[27] K. Kritikos e P. Skrzypek. ≪Towards an Optimized, Cloud-Agnostic Deployment of Hybrid Applications≫. In: *Lecture Notes in Business Information Processing* (2019).

[28] K. Kritikos et al. ≪Towards the Modelling of Hybrid Cloud Applications≫. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)* (2019).

[29] Philipp Leitner et al. ≪A mixed-method empirical study of Function-as-a-Service software development in industrial practice≫. In: *Journal of Systems and Software* (2019).

[30] *LocalStack*. URL: https://www.localstack.cloud.

[31] D.C. Marinescu. *Cloud Computing: Theory and Practice*. Elsevier Science, 2017. ISBN: 9780128128114. URL: https://books.google.it/books?id=O9smDwAAQBAJ.

[32] *Maven*. URL: https://maven.apache.org/.

[33] *Maven - POM Reference*. URL: https://maven.apache.org/pom.html.

[34] *Microsoft Azure*. URL: https://azure.microsoft.com/.

[35] *npmjs*. URL: https://www.npmjs.com/.

[36] *OpenStack*. URL: https://www.openstack.org.

[37] Dana Petcu. ≪Multi-Cloud: expectations and current approaches≫. In: *MultiCloud 2013 - Proceedings of the International Workshop on Multi-Cloud Applications and Federated Clouds* (apr. 2013), pp. 1–6.

[38] B. Peterson, G. Baumgartner e Q. Wang. ≪A Hybrid Cloud Framework for Scientific Computing≫. In: *2015 IEEE 8th International Conference on Cloud Computing* (2015), pp. 373–380.

[39] *Pulumi*. URL: https://www.pulumi.com/.

[40] *Python*. URL: https://www.python.org/.

[41] *qs - npm*. URL: https://www.npmjs.com/package/qs.

[42] *SDK AWS per Java - Amazon Web Services*. URL: https://aws.amazon.com/it/sdk-for-java/.

[43] *SDK AWS per JavaScript - Amazon Web Services*. URL: https://aws.amazon.com/it/sdk-for-node-js/.

[44] *Serverless Framework*. URL: https://www.serverless.com.

[45] Maddie Stigler. *Beginning Serverless Computing: Developing with Amazon Web Services, Microsoft Azure, and Google Cloud*. Apress, gen. 2018.

[46] Ruslan Synytsky.
*How to overcome the challanges of Gaining multi-cloud interoperability.*
URL: https://www.forbes.com/sites/forbestechcouncil/2018/10/25/how-to-overcome-the-challenges-of-gaining-multi-cloud-interoperability.

[47] *The Python Package Installer.*
URL: https://pip.pypa.io/en/stable/.

[48] *TOSCA.* URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca.

[49] Adbys Vasconcelos et al. ≪DistributedFaaS: Execution of Containerized Serverless Applications in Multi-Cloud Infrastructures≫.
In: (2019).

[50] *What is Cloud Computing?*
URL: https://aws.amazon.com/what-is-cloud-computing/.

[51] M. Wurster et al. ≪Modeling and Automated Deployment of Serverless Applications Using TOSCA≫.
In: *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)* (2018).

[52] *Xtext.* URL: https://www.eclipse.org/Xtext/.