Traduttore Vocale Simultaneo

Progetto Sistemi distribuiti e Cloud Computing

Francesco Pasceri 204963

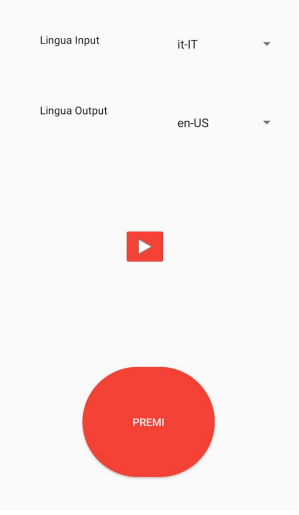
Simone Giorgio 163926

Descrizione progetto

Lo scopo del progetto è quello di costruire una piattaforma per la traduzione del parlato con gestione del carico delle richieste mediante Java RMI.
La piattaforma è costituita dai client, i quali richiedono il servizio di traduzione, e dai server, i quali comunicano con i servizi Google integrati.
I servizi Google utilizzati per attuare l'idea sono:

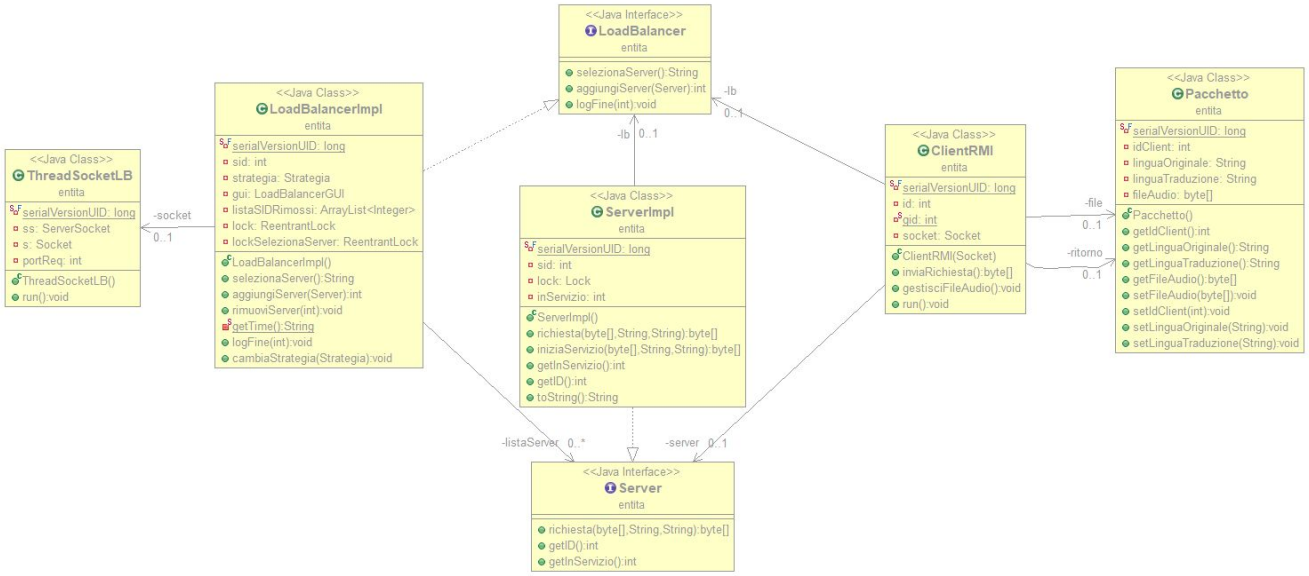
• *Speech-to-Text API*: il quale riceve in ingresso un file audio e restituisce il testo trascritto nella lingua selezionata;
• *Translate API*: traduttore del testo in ingresso nella lingua selezionata come output;
• *Text-to-Speech API*: questo servizio svolge la funzione duale al primo.

L'utente si interfaccia alla piattaforma mediante un'*applicazione Android* che permette la registrazione del parlato e si occupa di comunicare con un server. Quest'ultimo riceve il file audio che viene processato dai vari servizi e consegna all'utente la sua versione tradotta.



All'aumentare delle richieste di servizio da più client possono sorgere problemi di sovraccarico del server che si occupa di gestire lo svolgimento della funzione. Per gestire tale problematica, una soluzione è incrementare il numero dei server, supponiamo k, e adottare un modello di piattaforma con nodo "load balancer", il quale si occupa di distribuire il carico delle richieste secondo una politica scelta (server più scarico, round robin etc etc). In questa configurazione l'utente richiede il servizio comunicando con il load balancer che fornisce il nome del server da utilizzare.
Solo a seguito della ricezione del nome del server, l'utente può utilizzare il servizio.

1. Struttura del servizio



Come raffigura il diagramma il progetto è costituito da 3 entità principali:

- Server: implementa l'interfaccia ServerInterface ed estende UnicastRemoteObject permettendo l'esposizione di alcuni suoi metodi tra i quali il metodo richiesta(byte[], String, String), il quale consente ai Client di inoltrare la richiesta del servizio.
  Per gestire le richieste in entrata viene adottato un metodo di sincronizzazione con ReentrantLock e gestione FIFO della coda.
  Una volta registrato dal LoadBalancer e ricevuto il suo ID, effettua la rebind sull'rmi registry per esporre i suoi servizi;

- LoadBalancer: come il server implementa la propria interfaccia remota ed estende UnicastRemoteObject.
  La sua funzione è quella di restituire ad ogni Client richiedente il riferimento al Server.
  Il paragrafo successivo si occupa dell'analisi del suo funzionamento all'aumentare dei Server e delle richieste;

- ClientRMI: questo thread è creato dal ThreadSocketLB per svolgere la singola richiesta.
  Durante la creazione effettua la lookup sull'rmi registry per individuare il LoadBalancer e successivamente per collegarsi al server remoto ed inoltrare il file audio da elaborare;

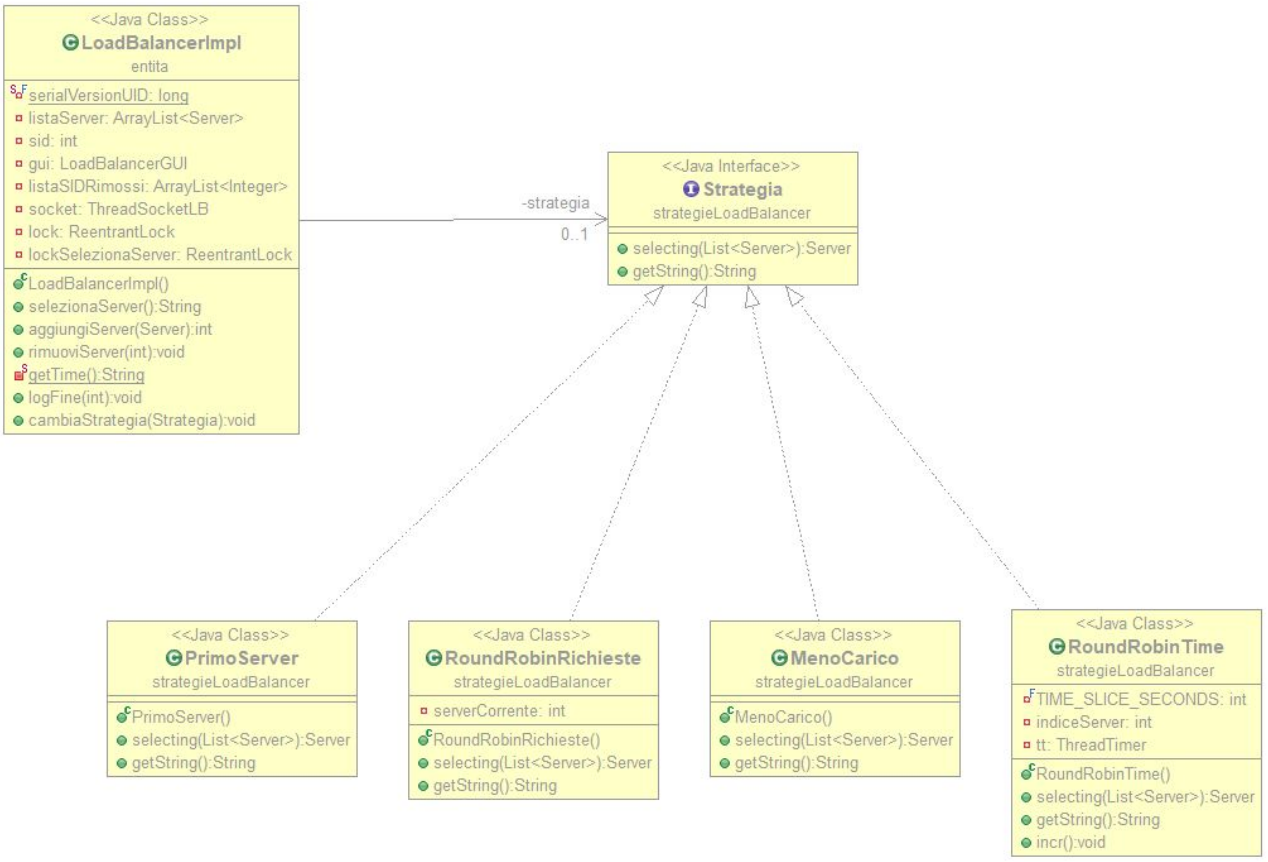
A completare lo scenario le classi ThreadSocketLB e Pacchetto. La prima crea il socket per consentire la comunicazione con l'applicazione Android sia per l'invio del file audio che della sua versione tradotta.
La classe Pacchetto (serializzabile) viene utilizzata come messaggio dall'applicazione e contiene l'audio e altri dati.

2. Scenario con strategia di selezione dei server

Come precedentemente anticipato supportiamo il caso in cui per far fronte a numerose richieste vengono impiegati più Server.

Il LoadBalancer si occupa di instradare le richieste ai vari Server cercando di garantire un carico equo tra di essi. Utilizzando il pattern Strategy abbiamo la possibilità di cambiare la politica di instradamento delle richieste.

Le politiche sviluppate sono: Primo Server, Round Robin (Singola Richieste o TimeSlice), Meno Carico.



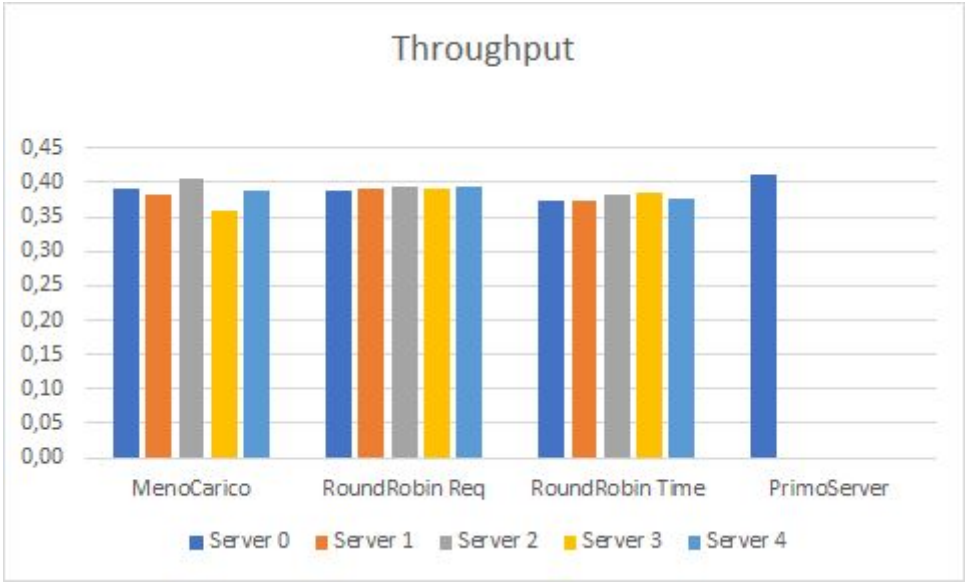
Le quattro strategia messe a disposizione del Load Balancer per gestire l'instradamento lavorano in maniera diversa: Primo Server, si utilizza nella fase in cui si vuole indirizzare l'intero carico verso un solo Server ed è la strategia scelta di default; Meno Carico, la si può utilizzare nelle situazioni in cui vi è necessità di garantire una equa distribuzione del carico di lavoro su ogni Server; Round Robin Time è la politica basata su delle "*slice*" temporali ad ognuna delle quali è assegnato uno dei Server che regolarmente si alternano come via scelta per indirizzare le richieste Client; infine, Round Robin Richieste, la quale è simile alla sua

versione temporale ma si differenzia nel meccanismo di rotazione che non è più basato sul tempo ma sulle singole richieste, per cui ad ogni richiesta è scelto il Server successivo.

3. Simulazioni e comparazioni

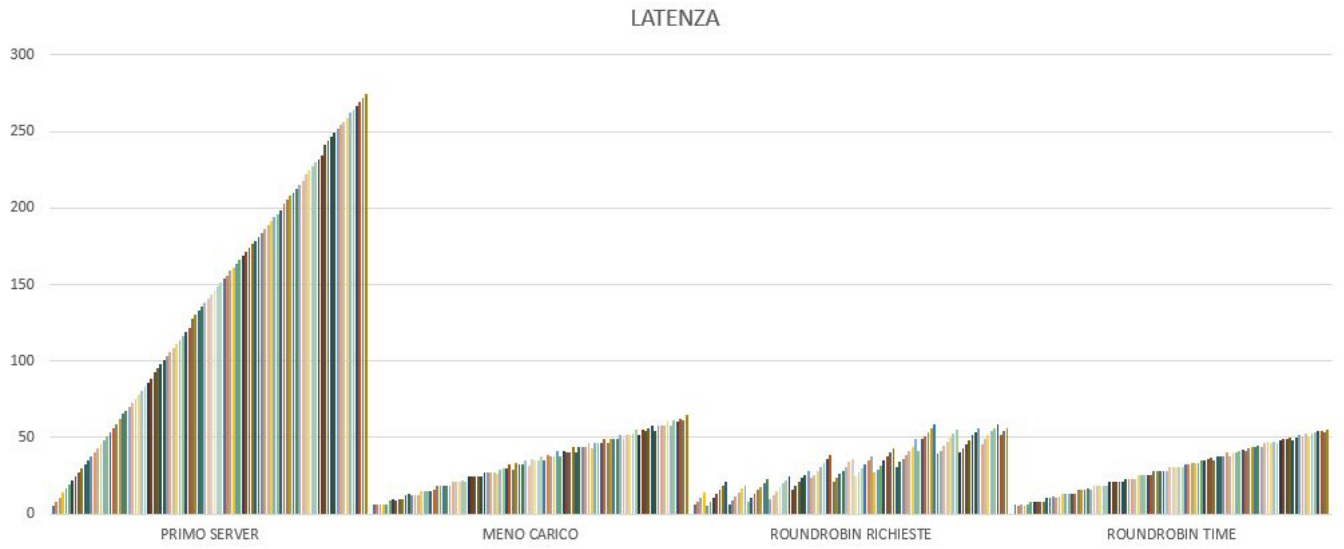
Una volta strutturate le possibili strategie, presi come riferimenti il throughput (totale Client Serviti / tempo di lavoro) e la latenza (tempo trascorso tra inizio e fine richiesta), abbiamo avviato alcune simulazioni per analizzare il comportamento della struttura.
Per effettuare le simulazioni sono stati avviati 100 client per la latenza e più di 500 per il throughput, in entrambi i casi ogni richiesta veniva formulata ogni 0,3 secondi, scelto come numero ottimale per ottenere delle variazioni nel numero di client gestiti da ogni singolo server.

3.1. Analisi Throughput



La prima analisi svolta è stata quella del throughput di ogni server nelle quattro strategia per ottenere il numero di richieste completate nell'unità di tempo e confrontare i risultati.
In relazione al grafico sopra, il throughput risultante è simile in tutti i casi. Questo è dovuto al metodo di svolgimento delle richieste, infatti, a prescindere dal tempo di attesa di ogni client, il Server smaltisce le richieste in ordine FIFO in qualsiasi strategia è adottata.
Nel caso della strategia Primo Server, ci si può aspettare un throughput minore ma, come mostrato dal grafico, si mantiene simile a quello delle altre simulazioni. Questo perché seppur il server scelto impieghi più tempo per elaborare tutte le richieste, ne completa un numero molto maggiore.

3.2. Analisi Latenza



Come ci si aspetta, la strategia con la scelta di un unico server è, anche in questo caso, quella che presenta la situazione peggiore in termini di latenza, in quanto l'intero set di richieste rimane in attesa sulla stessa coda.

Per evitare la saturazione del singolo server sono adottate le altre politiche che si alternano su 5 server. Le prestazioni, come si evince dai grafici, sono nettamente migliori: la media della latenza diminuisce in maniera significativa (da 139,89 secondi di attesa media con strategia Primo Server si passa a 29,97 secondi con Round Robin Time) e si ha un miglioramento delle prestazioni di quasi l'80% tra il caso peggiore e quello migliore, quest'ultimo rappresentato dalla strategia Round Robin Time.