Andrea_Tolve_SRS

Progetto: CDN Distribuito con RMI e DHT
Autore: Andrea Tolve
Data: 10/01/2026

---

1. Introduzione

1.1 Scopo

Questo documento definisce i requisiti per un sistema **Content Distribution Network (CDN)** che utilizza **Java RMI** per la comunicazione tra server distribuiti e una **Distributed Hash Table (DHT)** per la gestione decentralizzata della mappatura tra contenuti e Edge Server.

1.2 Obiettivi

- Distribuire contenuti statici in modo efficiente tra server geograficamente distribuiti.
- Implementare una **DHT** per la mappatura decentralizzata dei contenuti.
- Utilizzare algoritmi di caching e load balancing per ottimizzare le prestazioni.

1.3 Definizioni e Acronimi

- **CDN:** Content Distribution Network
- **RMI:** Remote Method Invocation
- **DHT:** Distributed Hash Table
- **Edge Server:** Server distribuiti che memorizzano in cache i contenuti.
- **Origin Server:** Server centrale che contiene i contenuti originali.
- **Load Balancer:** Componente che distribuisce le richieste degli utenti.

---

2. Descrizione Generale

2.1 Prospettiva del Prodotto

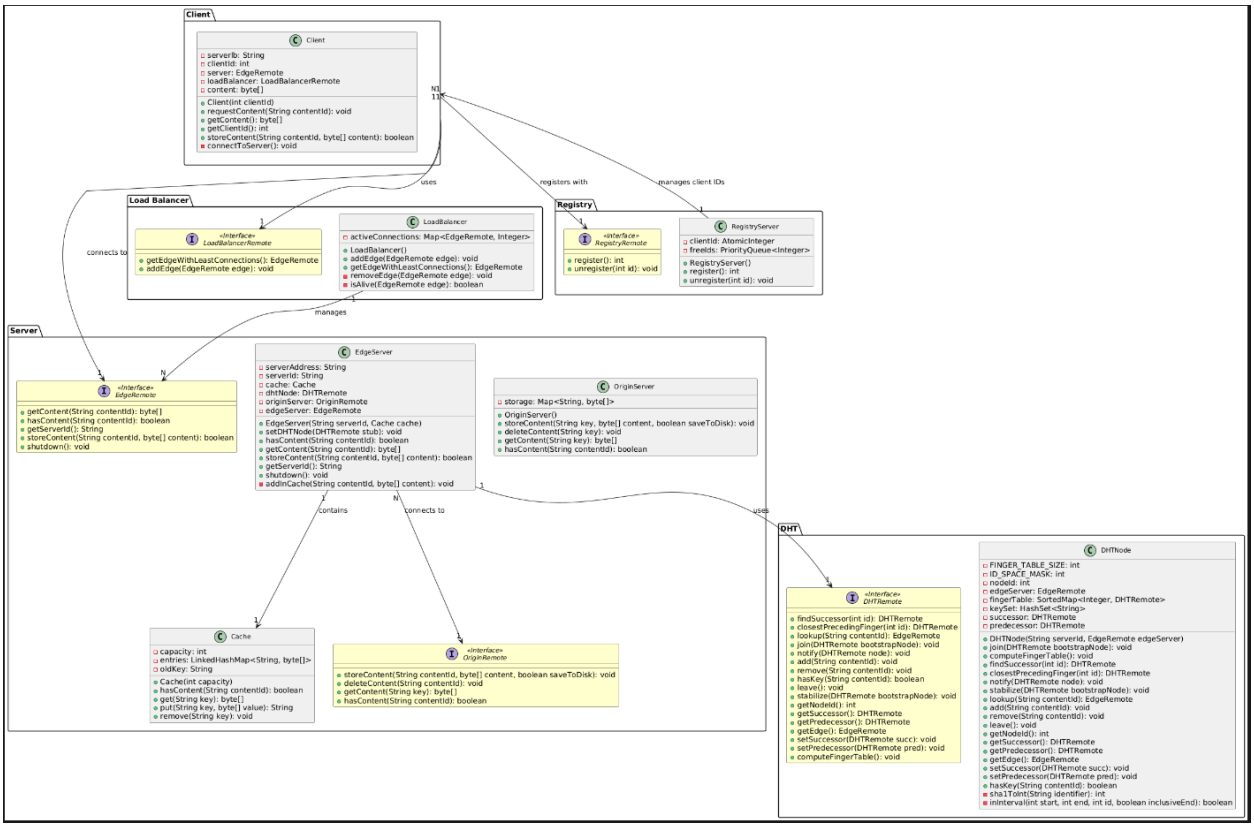
Il progetto nasce come parte del corso di **Algoritmi Distribuiti** e mira a sviluppare un CDN semplificato per la distribuzione di contenuti statici. Il sistema utilizza **nodi asimmetrici** con ruoli distinti:

- **Client**: Utenti che richiedono contenuti.
- **Load Balancer**: Smista le richieste degli utenti agli Edge Server.
- **Edge Server**: Memorizzano in cache i contenuti e rispondono alle richieste.
- **Origin Server**: Memorizza i contenuti originali.
- **DHT**: Gestisce la mappatura decentralizzata dei contenuti tra gli Edge Server.

2.2 Topologia

- Architettura **client-server distribuita**.
- **Client → Load Balancer → Edge Server → (DHT/Origin)**.
- Gli Edge Server interagiscono con la DHT per identificare il nodo responsabile di una chiave.
- L'Origin Server viene contattato solo se nessun Edge Server responsabile o contenente la risorsa è disponibile.

2.3 Architettura del Sistema



3. System Features

3.1 Requisiti Funzionali

Origin Server

ID	Requisito	Descrizione
RS1	Memorizzazione contenuti	L'Origin Server deve memorizzare i contenuti originali (immagini, video, file di testo).
RS2	Fornitura contenuti	L'Origin Server deve fornire i contenuti agli Edge Server su richiesta.

Edge Server

ID	Requisito	Descrizione
ES1	Caching contenuti	Ogni Edge Server deve memorizzare in cache i contenuti richiesti dagli utenti.
ES2	Algoritmo di	L'Edge Server deve implementare un algoritmo di

ID	Requisito	Descrizione
	caching	caching.
ES3	Gestione richieste	L’Edge risponde alle richieste dei client, interrogando DHT o Origin Server se necessario.
ES4	Comunicazione con DHT	L’Edge interagisce con la DHT per determinare il nodo responsabile di una chiave.
ES5	Aggiornamento DHT	L’Edge aggiorna la DHT solo quando riceve un contenuto dall’Origin Server, se è il responsabile della risorsa.

Load Balancer

ID	Requisito	Descrizione
LB1	Distribuzione richieste	Il Load Balancer deve distribuire le richieste degli utenti agli Edge Server in base ad un criterio (es. vicinanza, carico).
LB2	Algoritmo di bilanciamento	Il Load Balancer deve implementare un algoritmo di bilanciamento.

DHT

ID	Requisito	Descrizione
DHT1	Mappatura contenuti	La DHT mantiene la mappatura tra contenuti e Edge Server responsabili.
DHT2	Ricerca contenuti	La DHT deve permettere la ricerca degli Edge Server che memorizzano un determinato contenuto.
DHT3	Aggiornamento mappatura	La DHT aggiorna la mappatura solo quando un nodo diventa responsabile di un nuovo contenuto dall’Origin Server o lascia la rete.
DHT4	Gestione nodi	La DHT deve gestire l’aggiunta e la rimozione dinamica degli Edge Server.

Client

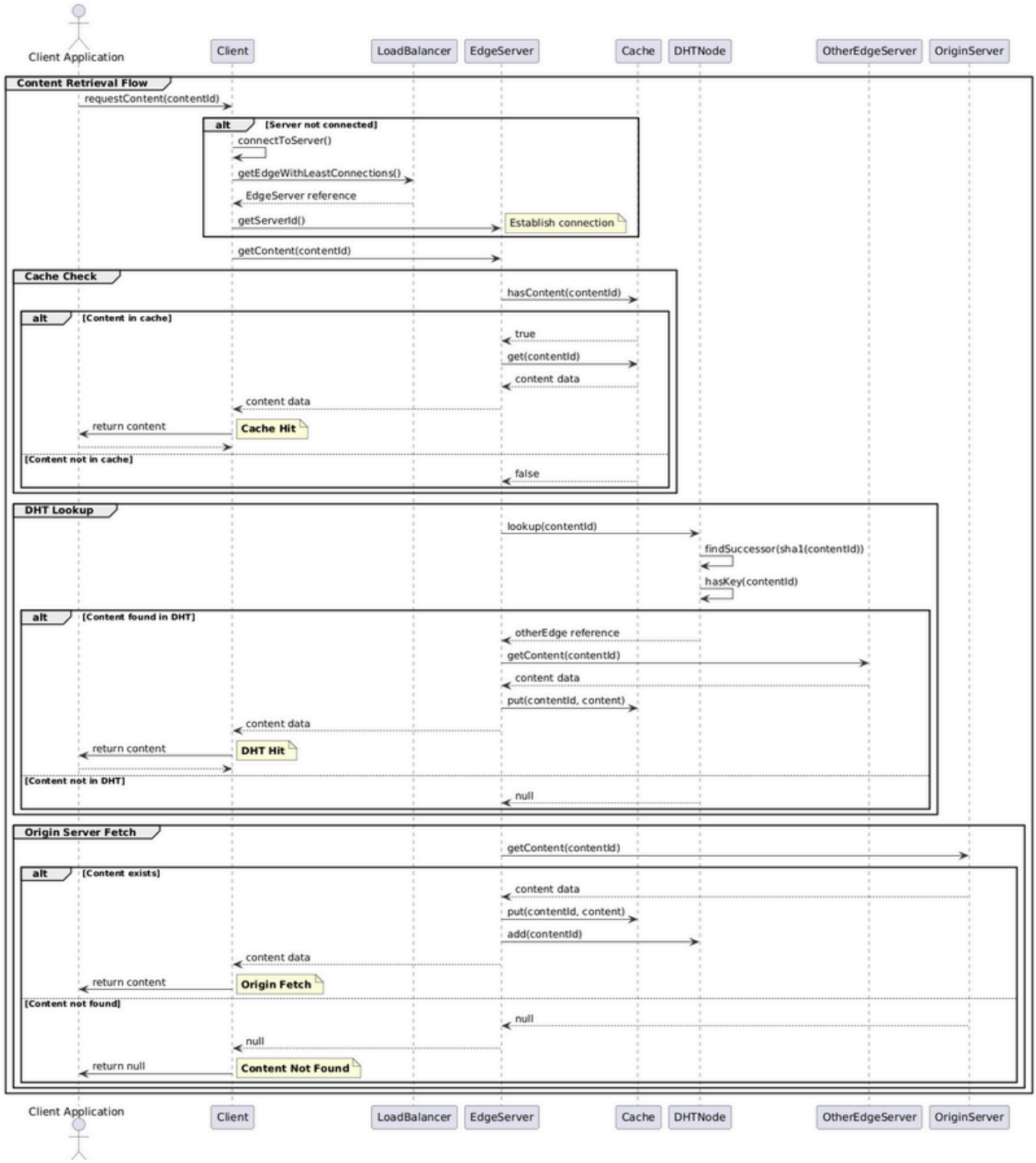
C1	Richiesta di una risorsa	Il client invia una richiesta per ottenere una risorsa.

C2	Successo di una richiesta	Il client riceve la risorsa richiesta o un messaggio di errore in caso di fallimento.
C3	Aggiunta di un contenuto	Il client può aggiungere una nuova risorsa nella rete, rendendola disponibile per gli altri client.

4. Caso d’Uso

Richiesta di un Contenuto

1. **Attore:** Utente
2. **Descrizione:** L’utente richiede un contenuto (es. un’immagine).
3. **Flusso principale:**
    - L’utente invia la richiesta al Load Balancer.
    - Il Load Balancer indirizza la richiesta a un Edge Server.
    - L’Edge Server verifica la cache:
        - Se il contenuto è in cache, lo restituisce.
        - Altrimenti, interroga la DHT per trovare l’Edge Server che ha il contenuto in cache.
            - Se la DHT indica un Edge Server che possiede il contenuto in cache, l’Edge Server destinatario della richiesta lo contatta direttamente, memorizza il contenuto ricevuto in cache e lo restituisce al Client. Se, durante l’inserimento in cache, viene rimossa una chiave di cui l’Edge Server era responsabile, aggiorna la DHT per riflettere la perdita di responsabilità.
            - Se il contenuto non esiste su altri nodi, lo richiede all’Origin Server. In questo processo, l’Edge Server può essere il responsabile della chiave e aggiornare di conseguenza la DHT.

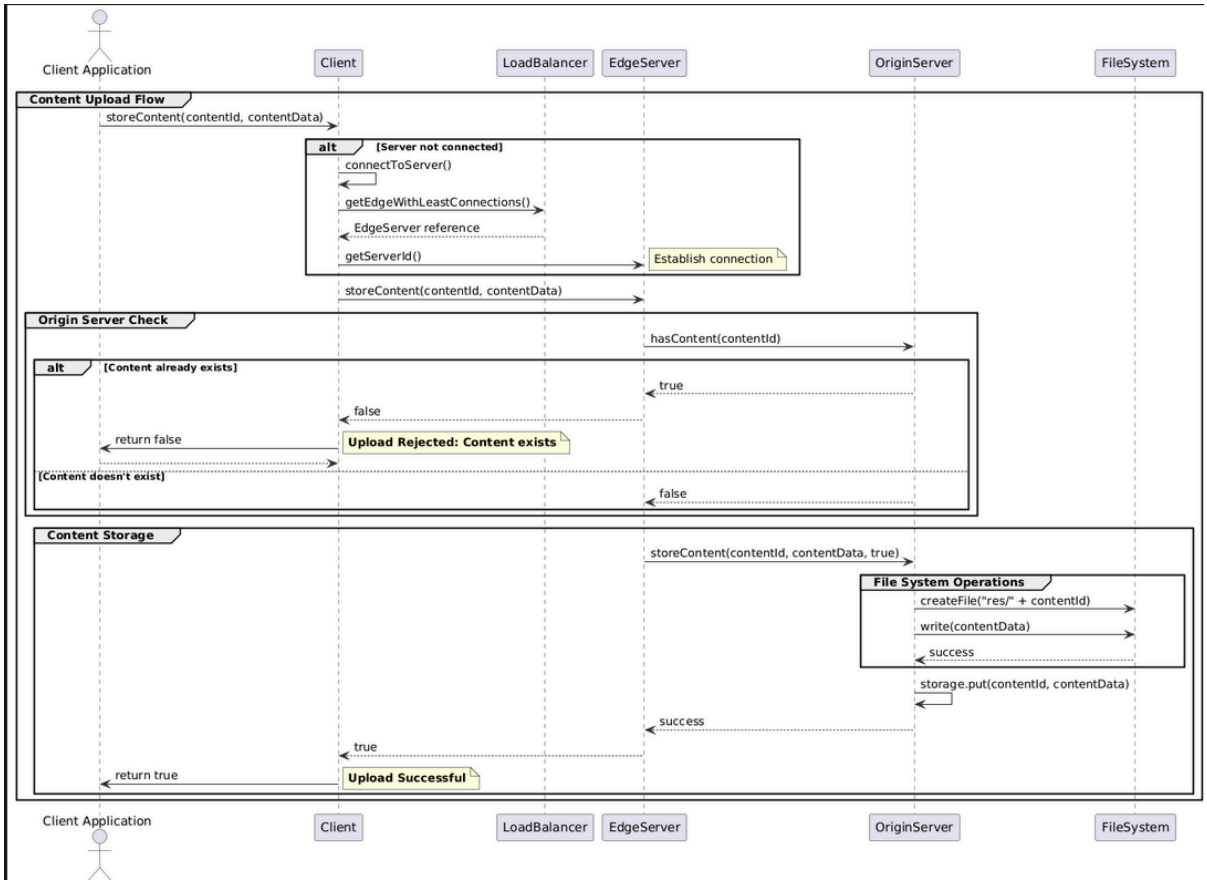


Caricamento di una Risorsa

1. **Attore:** Utente
2. **Descrizione:** L'utente invia un contenuto (es. un file) all'**Origin Server** per memorizzarlo nel sistema CDN. Il contenuto viene validato e, se conforme, salvato nello storage dell'Origin Server.

3. Flusso Principale:

- L’utente invia una richiesta di caricamento contenente il file all’Edge Server.
- L’Edge Server riceve la richiesta e la inoltra all’Origin Server.
- L’Origin Server verifica il contenuto:
  - Se il file è nuovo e valido:
    - Il contenuto viene salvato nello storage dell’Origin Server.
    - L’Origin Server restituisce un messaggio di successo all’Edge Server.
  - Se il file esiste già o è invalido:
    - L’Origin Server restituisce un messaggio di errore (ad esempio: "File già esistente" o "Errore di lettura").
- L’Edge Server inoltra la risposta (successo o errore) al Load Balancer.



5. Vincoli e Ipotesi

- Il sistema sarà implementato in **Java** utilizzando **RMI** per la comunicazione tra i componenti.
- Gli Edge Server saranno simulati su una macchina locale.
- L'algoritmo di caching scelto è **LRU (Least Recently Used)**, selezionato per la sua semplicità di implementazione e per la capacità di favorire il riutilizzo dei contenuti più recenti.
- Il sistema utilizzerà il **Least Connection Method** per distribuire le richieste. Questo algoritmo seleziona l'Edge Server con il minor numero di connessioni attive, garantendo un bilanciamento equo del carico e prevenendo il sovraccarico di singoli nodi.
- I contenuti saranno file statici (immagini, video, file di testo).
- Si assume affidabilità totale.

---

6. Future Estensioni Possibili

- Algoritmi di caching alternativi (es. **LFU**).
- Considerazione della **vicinanza geografica** nell'algoritmo di load balancing.
- Gestione dei **fallimenti** e tolleranza ai guasti.
- Possibilità di **aggiornamento dei contenuti esistenti**.