SAE Storage System

SDCC Project, Storage nel Cloud Continuum

Simone Nicosanti
Facoltà di Ingegneria Informatica
Unversità degli Studi di Roma
Tor Vergata
Velletri, Roma
snicosanti@gmail.com

Andrea De Filippis
Facoltà di Ingegneria Informatica
Unversità degli Studi di Roma
Tor Vergata
Lenola, Latina
andreadefilippis00@gmail.com

Edoardo Manenti
Facoltà di Ingegneria Informatica
Unversità degli Studi di Roma
Tor Vergata
Velletri, Roma
manenti000@gmail.com

***Abstract*—Il sistema SAE sfrutta l'edge continuum per migliorare la latenza, riscontrata durante l'accesso a una piattaforma di storage Cloud come il servizio AWS S3, attraverso un meccanismo di caching presente all'interno degli edge.**

I. Introduzione

Il sistema di storage SAE è un sistema che punta a sfruttare l'edge continuum per migliorare la latenza, che in genere è piuttosto elevata, degli accessi al Cloud. Il dominio applicativo considerato per lo sviluppo del sistema è quello in cui un'organizzazione, che possiede numerosi dispositivi IoT (e.g. sensori), vuole effettuare l'accesso e il salvataggio dei dati accumulati sulla piattaforma di storage Cloud AWS S3. In particolare, il sistema permette di ridurre le attese dei client grazie a un sistema di caching presente tra i client e il Cloud. Perciò, siamo in presenza di una rete di dispositivi IoT in ambito locale che mantiene in modo persistente i propri dati sul Cloud tramite il servizio S3.

II. Architettura del Sistema

Il sistema è composto da un server registry, un load balancer, una rete non strutturata di nodi edge, diversi nodi client e dal servizio di storage Cloud AWS S3.

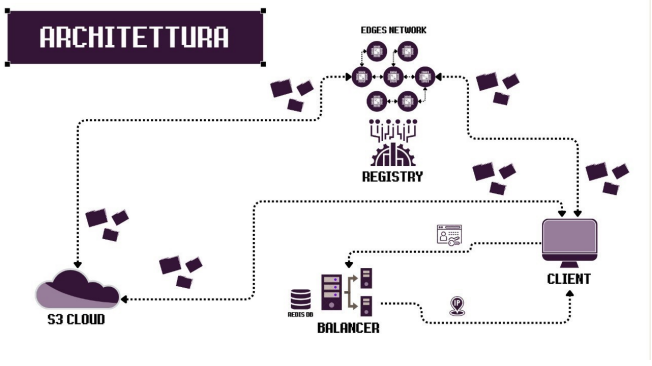

Fig. 1. Architettura del sistema

A. Server Registry

Il server Registry ha la responsabilità di "guidare" i nodi edge all'interno della rete non strutturata. Uno dei suoi compiti è quello di gestire le connessioni dei nodi edge, comunicando i collegamenti quando un nuovo nodo entra nella rete: la rete viene costruita seguendo un approccio di tipo Erdős-Rényi, in cui ogni nodo che entra nella rete ha una probabilità p di connettersi con gli altri nodi già presenti; è stato scelto questo modello poiché permette la costruzione di reti con un diametro limitato e un coefficiente di clustering facilmente configurabile. Ovviamente, per fare in modo che ci sia sempre almeno un collegamento, un nuovo nodo viene connesso casualmente ad un dei nodi della rete prima di far partire i meccanismi del modello Erdős-Rényi.

Grazie a un servizio di Heartbeat, il server registry può rilevare eventuali nodi caduti, trovare componenti connesse e rimodellare la rete di conseguenza. Infatti, è presente un meccanismo di ricerca e risoluzione delle partizioni all'interno della rete. Tramite Heartbeat il server conosce quali nodi sono attualmente attivi nel sistema e periodicamente fa partire una scansione su questi nodi, chiedendo ad ognuno quali sono i nodi adiacenti di cui ha contezza; raccolte queste informazioni, il Registry costruisce il grafo della rete: la costruzione del grafo della rete è fatta "per eccesso", ovvero se il nodo edge1 dice che ha per adiacente edge2, ma edge2 non ha tra la lista di adiacenti edge1, allora comunque la connessione è considerata bidirezionale (si veda in II-C la connessione tra edge). L'algoritmo usato per rilevare le partizioni di rete è composto dai seguenti passi:

1) Tutti edge sono inizializzati come non visitati.
2) Partendo da un edge non visitato, si effettua una visita in profondità e si marcano come "visitati" tutti gli edge a cui riesce ad arrivare.
3) Se ci sono ancora edge non visitati, si riprende dal punto 2.

Per correggere la rete il server collega casualmente due edge appartenenti ognuno ad una delle partizioni. In caso più di due partizioni l'algoritmo è eseguito in modo iterativo: se ci sono tre partizioni, (A, B, C), allora A $\rightarrow$B $\rightarrow$C.

Il Registry è stato scritto in Golang.

B. Load Balancer

Il Load Balancer è responsabile di due aspetti: agisce sia da servizio di Login al sistema, sia come Entry Point per i client. Per motivi di interoperabilità e manutenibilità, i servizi

sono esposti usando gRPC, in modo che client scritti in diversi linguaggi possano accedere ai servizi senza problemi.

Il Client inizia il meccanismo di login inviando una richiesta contenente *Username* e *Password*. Successivamente, il Load Balancer accede ad un'istanza Redis, in cui sono contenute le credenziali degli utenti. Se l'autenticazione ha successo, allora si risponde con la coppia (Indirizzo IP, Porta) relativi all'edge da contattare.

La scelta dell'edge da restituire al client passa dal servizio di balancing offerto dal Load Balancer, basato su semplici concetti:

- Per conoscere gli edge disponibili, questi implementano un meccanismo di Heartbeat verso il Load Balancer. Inoltre, nel messaggio di heartbeat viene allegato il carico attuale che l'edge ha in quel momento.
- Il Balancer mantiene internamente una struttura dati in cui associa ad ogni edge il *Carico Corrente*.
- Ogni volta che viene ricevuto un Heartbeat, viene aggiornato il carico nella struttura dati del Load Balancer come la media pesata tra il carico ricevuto nel messaggio di Heartbeat dall'edge e il carico noto al Load Balancer. In questo modo si riesce a evitare che il carico noto al Load Balancer risulti inconsistente: ciò potrebbe accadere se il client non contatta l'edge che gli è stato assegnato (in tali casi il Load Balancer incrementerebbe il carico associato all'edge senza che quest'ultimo abbia ricevuto la richiesta).
- Quando l'edge termina il lavoro che gli è stato assegnato, invoca il servizio *NotifyJobEnd* del Balancer: l'invocazione del servizio causa il decremento del carico associato all'edge nel Load Balancer.
- La scelta dell'edge con cui rispondere al Client è fatta scegliendo quello con carico minore al momento della richiesta; a parità di carico se ne sceglie uno casualmente.

Da sottolineare, infine, che il Load Balancer è collocato all'interno dell'azienda insieme alla rete di edge; perciò, è molto vicino a tutti gli edge e ciò permette di avere basse latenze.

Il Balancer è stato scritto in Golang.

C. Edge

Quando l'edge si attiva, prepara per prima cosa l'esposizione dei servizi, per poi contattare il Registry per ottenere la lista dei vicini e connettersi ad essi. Finita questa fase di inizializzazione, l'edge esegue tre operazioni periodiche:

- Heartbeat al Registry e Balancer, per comunicare che è attualmente attivo.
- Ping verso i vicini, per avere contezza dei vicini attualmente attivi. Un vicino viene rimosso dalla lista dei vicini attivi solo passato un certo numero di ping fatti senza successo.
- Notifica del proprio filtro di Bloom ai vicini. Il filtro di Bloom viene usato per eseguire in modo più efficace le operazioni di lookup e di notifica dell'eliminazione di un file (vedere sezione IV e in particolare IV-B e IV-C).

L'edge espone due tipi di servizi:

- Servizi esposti ai client: soddisfa le richieste relative a tutte le operazioni presenti nel sistema (download, upload, delete) utilizzando la rete di edge oppure accedendo direttamente alla piattaforma di storage Cloud AWS S3. Per la comunicazione tra edge e client è stato usato il servizio gRPC per migliorare la manutenibilità del sistema e non vincolare il client ad uno specifico linguaggio di programmazione.
- Servizi esposti ad altri edge o al Registry: Il servizio principale usato dal Registry è quello di richiesta dei vicini attuali, mentre i servizi principali usati dagli altri edge sono quelli di ping, ricerca del file, notifica di cancellazione e dei filtri di Bloom. Questi servizi sono realizzati usando GoRPC, eccetto per l'invio dei file tra edge implementati usando gRPC per sfruttare la modalità di comunicazione stream.

Il nodo edge, inoltre, pone in atto un meccanismo di Caching dei file che permette di evitare l'accesso al sistema Cloud per operazioni di download.

Per l'uso dei filtri di Bloom è stata usata la libreria Golang *github.com/tylertreat/BoomFilters*.

L'edge è stato scritto in Golang.

D. Client

Il client ha la responsabilità di interfacciarsi con il sistema e di fornire una UI per gli utenti. In particolare, il client comunica, inizialmente, con il Load Balancer per ottenere l'indirizzo di un edge e, successivamente, con l'edge scelto per soddisfare la richiesta. Nel caso in cui l'ottenimento di un edge risultasse impossibile, il client ripiegherebbe contattando il servizio S3 direttamente (vedere sezione VI). Internamente, il client ha anche il compito di mascherare all'utente la suddivisione in chunk dei file.

Il client è stato scritto in Python.

E. Storage Cloud

Come storage Cloud è stato usato il servizio S3 di AWS. Per interfacciarsi con questo servizio dall'edge sono state usate:

- Le librerie AWS di Golang per l'interfacciamento fatto dall'edge.
- La libreria *boto3* di Python per l'interfacciamento fatto dal Client.

III. Aspetti implementativi preliminari

A. Cache

La Cache è la componente che in ogni edge gestisce il caching dei file, ovvero del salvataggio in locale dei file e della loro gestione. La Cache entra in gioco in diversi scenari:

- Quando l'edge scarica un file da S3 per una richiesta di dowload oppure da un client per una upload: la Cache controlla che il file rispetti i limiti di grandezza imposti e, se è così, scrive il file in locale. Quando aggiunge un file in locale, se non ha abbastanza spazio libero, ordina i file già presenti in base alla loro popolarità (numero

di volte che sono stati richiesti dall'ultima volta che sono entrati in Cache) e seleziona i file con popolarità minore fino a quando la dimensione dei file selezionati non è abbastanza grande da permettere il salvataggio del nuovo file. A questo punto, la Cache controlla che non ci siano delle cancellazioni superflue tra i file selezionati: supponiamo che ci siano N file molto piccoli in fondo alla lista ordinata per priorità e che dopo di essi ci sia invece un file di grandi dimensioni; in tal caso, se la Cache deve salvare un nuovo file di medie dimensioni (maggiore della somma delle dimensioni di quelli piccoli), eliminerà soltanto il file di grandi dimensioni, minimizzando il numero di file eliminati. In questo modo si ottimizza l'utilizzo dello spazio dei vari edge.

- Quando viene eseguito il controllo periodico dei file scaduti: ad ogni file che entra in Cache viene associato un timestamp di entrata, superato il quale il file viene considerato "scaduto". Periodicamente viene eseguito un controllo per eliminare i file scaduti dalla Cache (anche se non si trovano in fondo alla coda di popolarità). In questo modo limitiamo la possibilità che un edge mantenga nella rete file cancellato o modificato sul Cloud.
- Quando un file è richiesto in download e avviene un cache-hit: viene incrementata la popolarità del file. Ciò accade anche quando la richiesta del file arriva da un altro edge.
- Quando deve essere calcolato il filtro di Bloom dell'edge: la Cache è la componente che gestisce tutti i file, quindi è sua responsabilità anche quella di calcolare il filtro di Bloom locale da inviare ai vicini dell'edge.

In caso di riavvio di un edge, la Cache eseguirà una procedura di recupero in cui scansiona tutti i file presenti sull'edge e ritorna così in funzione.

B. Ridirezione dei Flussi

Lo spostamento dei file all'interno del sistema viene fatta combinando canali Golang e stream gRPC su cui vengono inviate delle porzioni (chunk) di file. Abbiamo:

- Sorgenti: si tratta della sorgente da cui riceviamo i chunk. Nel nostro contesto può essere il Client, un Edge oppure lo storage Cloud.
- Input Stream: si tratta dello stream su cui i chunk vengono caricati dalla sorgente.
- Output Stream: si tratta dello stream su cui i chunk vengono inviati verso la destinazione.
- Canale Golang: viene usato per ridirezionare i chunk tra più componenti.

Il meccanismo di ridirezione segue la seguente logica (vedere figura 2):

1) La sorgente invia i chunk sull'Input Stream.
2) Un thread (in figura 2 rappresentato dalla freccia verticale lunga) si occupa della lettura dei chunk dall'Input Stream e della loro ridirezione alle componenti interessate.
3) Ogni componente interessata possiede un canale Golang sul quale riceve i chunk ridirezionati ed effettua le sue operazioni su di essi (e.g. la cache che salva localmente il file).

Nel nostro caso come storage Cloud è stato usato S3, il quale permette di leggere o scrivere su un qualsiasi oggetto che implementa l'interfaccia *ReadAt* o *WriteAt*; per poter trattare S3 come sorgente o destinazione nel modo specificato, sono stati costruiti degli oggetti custom che implementassero queste interfacce e che leggessero/scrivessero dai/sui canali.

Questo meccanismo di stream e canali garantisce una maggiore flessibilità ed estensibilità nella ridirezione dei chunk: qualora si volesse aggiungere una nuova destinazione dei chunk (come ad esempio un nuovo storage Cloud su cui salvare i file), sarebbe sufficiente aggiungere un canale Go per la ridirezione.

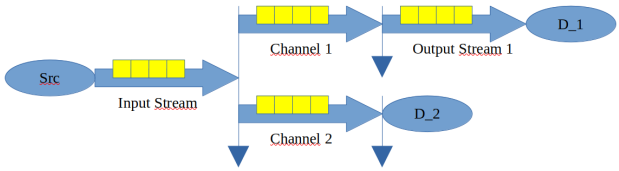

Fig. 2. Rappresentazione grafica degli stream e della ridirezione

IV. Casi d'Uso

Per prima cosa il Client comunica con il Balancer fornendo le sue credenziali. In risposta, il Balancer invia dei metadati per contattare uno degli edge attivi nel sistema. Se non ci fossero edge attivi nel sistema, oppure se il Balancer risultasse non raggiungibile, il Client contatterebbe direttamente il server S3 usando la libreria *boto3*. In caso di accesso fatto con successo, il Client contatta l'Edge.

A. Upload

Quando viene eseguita l'operazione di Upload, il client invia i chunk del file all'interno di uno stream gRPC e specifica la dimensione del file all'interno dei metadati associati alla comunicazione: questo permette all'edge contattato di sapere se il file supera o meno la soglia di Cache. Se il file può essere inserito in Cache i chunk vengono ridiretti, oltre che verso S3, anche verso la Cache (la ridirezione avviene come specificato in III-B).

L'operazione è considerata completata quando il file è stato ridiretto completamente verso S3: nel caso dell'upload quindi il sistema funziona solo come ponte verso S3 e, se possibile, mette il file in cache per velocizzare le operazioni di download successive.

B. Download

La ricerca del file in seguito alla ricezione di un'operazione di Download avviene su tre livelli (fare riferimento alla figura 3 per l'activity diagram della *Download*):

1) Ricerca del file in locale.
2) Ricerca del file nella rete.
3) Ricerca del file su S3.

Quando il file è trovato in una delle sorgenti, questo viene diviso in chunk che vengono ridirezionati usando uno stream gRPC. Se il file è presente nella Cache locale dell'edge, allora i chunk vengono ridiretti direttamente verso il Client.

Se il file non è presente nella Cache, allora viene avviata una ricerca all'interno della rete di edge. La ricerca viene eseguita contattando i vicini positivi al controllo dei filtri di Bloom. I vicini che non posseggono il file inoltreranno la richiesta allo stesso modo. Per limitare la diffusione di messaggi è stato imposto un limite sul numero di vicini contattabili.
La ricerca del file all'interno della rete di edge avviene nel seguente modo:

1) Vengono controllati i filtri di Bloom dei nodi vicini. Il messaggio di ricerca viene mandato primariamente agli edge i cui filtri di Bloom hanno dato riscontro positivo; se il numero di nodi positivi ai filtri è minore del numero massimo, allora vengono mandati messaggi di richiesta anche ad alcuni edge scelti casualmente tra quelli negativi ai filtri. Il motivo di questo "complemento" è che i filtri di Bloom permettono agli edge di conoscere solo ciò che si trova ad un hop di distanza, ma non oltre; qualora il file si trovasse a due hop, ma fosse legato all'edge tramite un altro edge con filtro negativo, il file si troverebbe nella rete, ma potrebbe non essere trovato.
2) Viene aperta una socket per attendere la callback da parte di un qualunque nodo edge che possiede il file e si fa partire un timer di attesa.
3) Se si riceve una risposta positiva sulla socket, allora si richiede il file all'edge che ci ha risposto (che chiamiamo *Owner* per distinguerlo). La richiesta avviene tramite gRPC e restituisce uno stream di chunk che viene ridirezionato al client tramite lo stream gRPC aperto durante il soddisfacimento della sua richiesta.
4) Se scade il timer e non è stata ricevuta risposta (oppure è stata ricevuta ma si è verificato un errore nella ridirezione) allora si contatta direttamente S3.

Un nodo che riceve una richiesta di lookup inserisce la richiesta in una Cache degli ID delle richieste (per evitare di rielaborare più volte la stessa richiesta) e, se possiede il file, invia al richiedente l'indirizzo su cui contattarlo; altrimenti, se non lo possiede, inoltra la richiesta ai suoi vicini decrementando il TTL associato alla richiesta. La risposta quindi viene mandata direttamente al richiedente evitando di ripercorrere il cammino di andata della richiesta: è stata presa questa scelta perché permette di ridurre la latenza e perché siamo in un contesto in cui la Cache, per come progettata, può essere molto dinamica (vedere sezione III-A), quindi salvare l'informazione su chi possiede un certo file ad un certo istante non risulta vantaggiosa.

Se non si è riusciti a sfruttare il sistema di caching del sistema per inviare il file, allora si ripiega direttamente su S3. Anche in questo caso il file viene inviato al Client usando dei chunk, creati usando la classe *Downloader* della libreria di AWS di Golang.

Quando un file viene scaricato dall'edge (a prescindere che sia scaricato da un altro edge oppure da S3) per essere ridirezionato verso il Client, il file viene salvato nella Cache locale dell'edge se non supera un certo valore di soglia (vedere III-A). Questo perché è preferibile ridurre i tempi di accesso ad S3 per file piccoli piuttosto che per file grandi. Inoltre, in questo modo gli edge possono contenere un numero alto di file e avere maggiori cache-hit. La dimensione del file viene ottenuta diversamente in base al tipo di sorgente dei chunk:

- Ottenimento da un altro Edge (Owner): viene inviata dall'Owner al richiedente all'interno della risposta.
- Ottenimento da S3: viene richiesta prima di inviare la richiesta ad S3, in modo da sapere anche se il file esiste o meno.

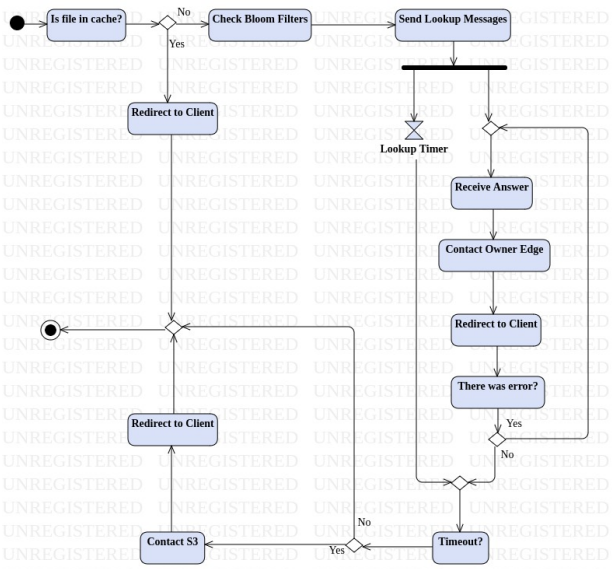

Fig. 3. Activity Diagram della *Download*

C. Delete

Quando l'edge riceve una richiesta di cancellazione del file vengono effettuate le seguenti operazioni:

1) Viene rimosso il file dalla Cache, se presente.
2) Viene inviata la richiesta di eliminazione ad S3 e si attende la conferma dell'operazione.
3) Viene propagata la cancellazione del file all'interno della rete: la notifica è inoltrata a tutti i vicini con filtro di Bloom positivo. Nella propagazione non è presente un TTL in modo da informare quanti più nodi possibili, limitando il flooding grazie ai filtri di Bloom.

È possibile che non tutti gli edge che posseggono il file siano raggiunti dalla cancellazione del file quando questa viene richiesta: questa situazione viene gestita mediante il meccanismo di eliminazione periodico della Cache (vedere sezione III-A).

V. Modello di Consistenza

Il modello di consistenza offerto dal sistema è consistenza finale. Questo è dato dal fatto che:

- Quando vengono caricate due versioni diverse dello stesso file, queste possono andare su due edge diversi, ma su S3 sarà sempre presente la versione del file caricata per ultima.
- Quando un file viene cancellato, è possibile che la modifica non si propaghi su tutta la rete, quindi degli edge possono ancora mantenere una copia di quel file nella Cache, ma la cancellazione viene mandata sempre su S3.
- I file che stazionano da troppo tempo nella Cache di un edge vengono periodicamente eliminati anche se non sono in fondo alla coda di popolarità; in questo modo riusciamo ad eliminare file che sono stati rimossi o modificati dal sistema, ma di cui l'edge non è stato notificato. Le cancellazioni periodiche della Cache permettono di far sì che periodicamente ogni edge debba riscaricare i file, riallineandosi con le ultime versioni.

In conclusione, un utilizzatore del sistema deve sempre considerare la possibilità che il file restituito da un edge possa essere stato in realtà cancellato o modificato su S3.

VI. Risultati Empirici

A. Contesto del testing

Le simulazioni sono state condotte in ambiente Docker, usando Docker Compose e limitando le seguenti risorse:

- Memoria, limitata usando un hard limit Docker.
- CPU, limitata usando un hard limit Docker.
- Storage, limitata usando un volume temporaneo a dimensione limitata.

Si noti che il limite sullo storage è soltanto simulato: il volume temporaneo viene infatti montato nella memoria principale, quindi le sue performance sono maggiori rispetto a quelle di un disco reale. Tutti gli edge sono stati simulati con le stesse risorse.

Le altre componenti del sistema non hanno avuto limitazioni delle risorse e il Bucket S3 usato si trovava in Virginia nella regione *us-east-1* di AWS. La grande distanza tra il punto di deployment del sistema e la posizione del Bucket ha permesso di estremizzare i risultati, mettendo in risalto la velocità del sistema di caching.

I test sono stati pensati con lo scopo di analizzare li differenze tra un uso diretto di S3 e un uso con sistema interposto. I test sono stati condotti in due modi:

- Sequenziale: in cui viene fatta al sistema una richiesta per volta per valutare la differenza tra i tempi. Questo è stato fatto sia per la upload che per la download.
- Parallelo: in cui vengono mandate al sistema più richieste insieme per valutare il comportamento in caso di carico. Questo è stato fatto soltanto per l'operazione di download.

Inoltre il test è stato condotto con i seguenti parametri di caching:

- Dimensione della Cache: 100 MB
- Dimensione massima di un file in Cache: 30 MB

I vari test sono stati condotti usando un singolo container per il client in cui vengono mandati in esecuzione diversi processi che eseguono le richieste.

B. Risultati in Upload

Analizzando il caso d'uso di upload, è evidente che non sono presenti differenze significative rispetto all'uso diretto di S3; infatti, il sistema ridireziona il file direttamente verso il Cloud. Anche se è presente un meccanismo di caching del file, questo non aggiunge ritardi significativi, in quanto viene atteso solamente il completamento del caricamento su S3 prima di notificare al client la fine dell'operazione. I risultati registrati nei test hanno confermato l'assenza di differenze (vedere figura 4 e 5).

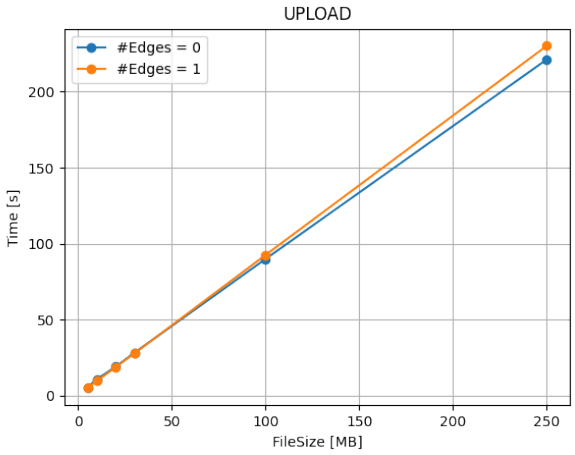

Fig. 4. Grafico della upload sequenziale

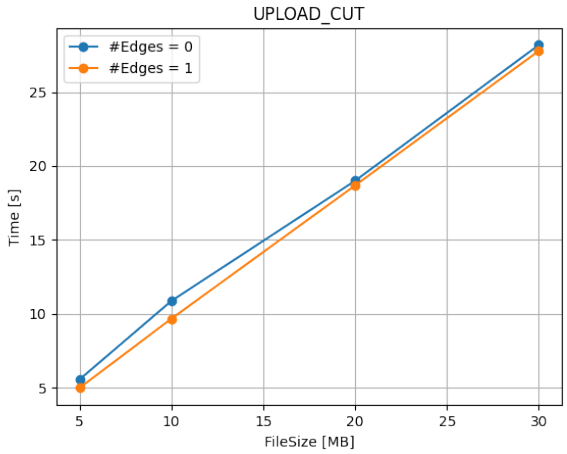

Fig. 5. Spaccato della upload sequenziale [5 MB - 30 MB]

C. Risultati in Download

Per quando riguarda la download sequenziale (vedere figura 6 e 7), notiamo che S3 presenta un andamento abbastanza lineare nei tempi di download.

Nel caso di file sufficientemente piccoli da entrare in Cache (figura 7), notiamo che l'uso del sistema abbatte notevolmente i tempi di download del file, portandoli ad essere prossimi allo zero. Il test è stato condotto con un sistema di:

- Un edge
- Cinque edge

Quando il file è troppo grande per essere inserito in Cache invece, i risultati sono paragonabili a quelli ottenuti usando direttamente S3 e crescono seguendo lo stesso andamento. L'interposizione del sistema causa un lieve aumento dei tempi di download: questo si può imputare al fatto che la presenza del sistema aggiunge latenza sia per il meccanismo di ridirezione dei chunk che è stato implementato, sia per il meccanismo di ricerca del file all'interno della rete.

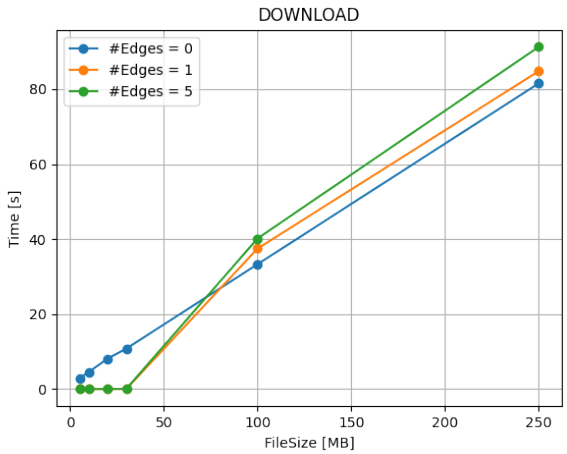

Fig. 6. Grafico della download sequenziale

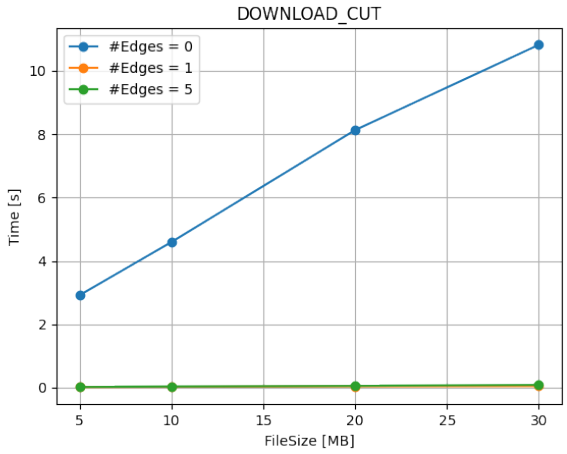

Fig. 7. Spaccato della download sequenziale [5 MB - 30 MB]

Per quanto riguarda le download parallele (vedere figure 8 e 9) possiamo notare come i tempi di elaborazione siano più alti rispetto al caso sequenziale: questo è probabilmente dovuto al fatto che i test vengono condotti utilizzando un solo client, quindi abbiamo un carico molto alto che ricade tutto sullo stesso container e sulla stessa banda.

Il test per download parallele con il sistema SAE è stato condotto usando dieci edge. Possiamo notare come, anche in questo caso, i tempi di download vengano abbattuti nel caso di file che possono entrare in cache, mentre si segue un andamento abbastanza lineare per file che invece non possono entrarci. Rispetto al caso sequenziale, nel caso parallelo abbiamo, anche per i file che possono entrare in cache, un andamento che tende al lineare: questo può essere dovuto ai seguenti aspetti:

- essendo il numero di edge abbastanza alto, la ricerca del file all'interno della rete può richiedere un tempo maggiore
- il file deve essere scaricato da S3 almeno una volta
- gli edge possono essere impegnati in altre operazioni

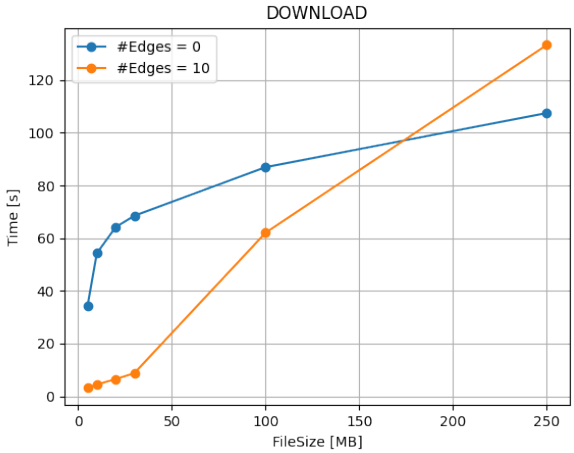

Fig. 8. Grafico della download parallela

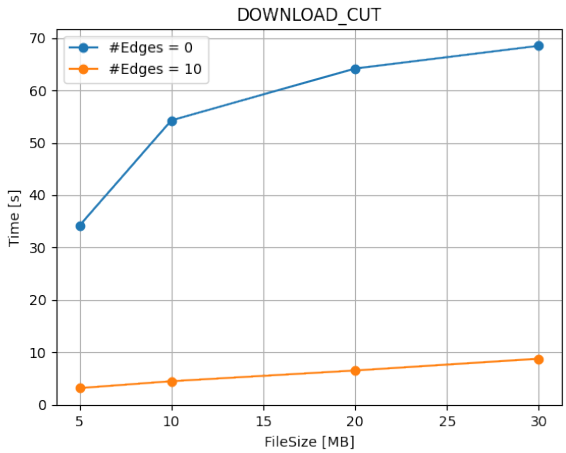

Fig. 9. Spaccato della download parallela [5 MB - 30 MB]

VII. Sviluppi futuri

Alcuni sviluppi possibili per il miglioramento del sistema sono i seguenti:

- Supporto per il versioning sui file nel sistema: attualmente il sistema non supporta un meccanismo di versioning dei file.
- Miglioramento del meccanismo di recovery sui file scaricati da Owner Edges in maniera da supportare la ripresa dello scaricamento dal punto di interruzione.
- Analisi sugli effetti delle variazioni dei parametri configurabili per la ricerca di configurazioni ottimali del sistema SAE Storage System in base al contesto di utilizzo.

VIII. Conclusioni

Il sistema SAE Storage System rappresenta un'ottima possibilità di ridurre i tempi di scaricamento di file dal Cloud. Grazie al meccanismo di caching, il sistema è ideale in un contesto in cui sono prevalenti richiesta di download di file di piccola taglia.