PROGETTO TAGD

ANALISI E MODELLAZIONE DI UN DBMS

Andrea Di Paola, George Lazarica, Alessio Marinucci | 16/06/2023

Introduzione

La relazione descriverà lo studio di un benchmark TPC-H e la procedura di modellazione di un sistema simulativo che rispetti i requisiti suggeriti dal progetto consegnato dal Professore Stefano Iannucci. La realizzazione del progetto ha previsto più attività, tra le più importanti:

- Lo studio prestazionale di un sistema reale.
- La realizzazione di un primo modello analitico e successivamente simulativo, creati a partire dai risultati ottenuti dal benchmark.
- Confronto tra risultati ottenuti dal modello simulativo e i risultati del sistema reale.
- Modellazione e dimensionamento del sistema simulativo seguendo i requisiti del progetto.

STRUMENTI DI LAVORO

Al fine di realizzare le varie attività abbiamo utilizzato un sistema MacBook Air M2 con sistema operativo MacOs Ventura 13.4.

Per isolare l'ambiente di lavoro abbiamo sfruttato la tecnologia Container sfruttando **Docker**. In particolare, il Professore Stefano Iannucci ci ha fornito un'immagine per costruire e avviare il container che esegue un server Postgres sul quale abbiamo creato il database di lavoro (lo abbiamo chiamato tpch1). La raccolta dei dati prestazionali è stata facilitata dal tool **PgAdmin**, anch'esso eseguito attraverso un'estensione Docker.

Riguardo la modellazione del sistema simulativo, abbiamo usufruito del software open source **Java Modelling Tools.**

SVOLGIMENTO

1)La prima attività è stata la più sostanziosa poiché ha previsto l'esecuzione del benchmark TPC-H sul Mac, l'acquisizione dei dati sfruttando pgadmin e la successiva elaborazione di essi.

Per effettuare il test delle prestazioni abbiamo sfruttato la suite TPC-H che si avvale di uno script python il quale compie tre operazioni: la prima fase di preparazione dei dati nella quale vengono generati sulla base di una scala di grandezza definita al momento di avvio dello script; la seconda fase di caricamento dei dati nel database creato anzi tempo sul server postgres in esecuzione nel container; la terza ed ultima fase di effettiva esecuzione del benchmark, durante la quale vengono eseguite 22 query e viene specificato il fatto che esse siano solo di lettura (nel nostro caso) e il numero di stream concorrenti.

È importante specificare il fatto che, in questo primo passo, abbiamo effettuato il benchmark configurando il container docker per utilizzare una sola CPU del sistema reale, come da requisito del progetto, e 4 Gb di Ram per questioni di stabilità del sistema. Il limite di Ram ci ha costretto a lavorare con un fattore di scala dei dati generati di 1 GB.

Svolte le tre fasi del benchmark TPC-H, Abbiamo acquisito i dati grazie all'estensione *pg_stat_statements* di pgAdmin. Esportando i dati come foglio di calcolo, siamo riusciti a isolare le informazioni relative alle sole queries del benchmark riuscendo a calcolare i Service demands dei centri CPU e Disk di ogni query nel seguente modo:

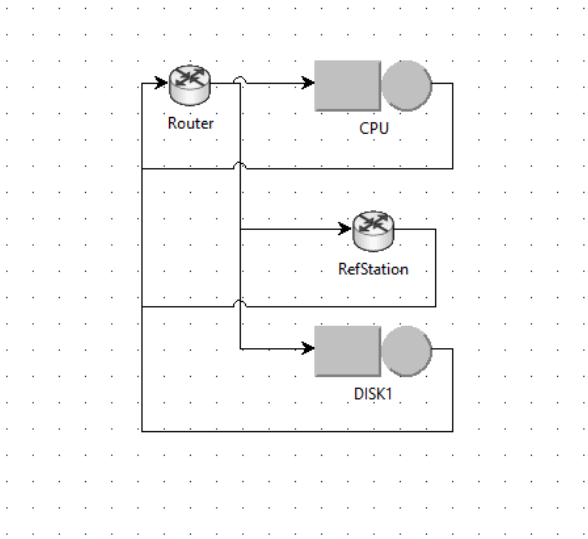
- Service Demand Disk = tempo di lettura dei blocchi / num. di chiamate della query
- Service Demand CPU = tempo di esecuzione medio della query – Service Demands Disk

2) Per raggiungere l'obiettivo di creare un modello simulativo che sia verosimile alla macchina presa in esame durante il progetto, abbiamo identificato due classi di workload: una rappresentata dalla query quasi ad uso esclusivo di CPU (CPU intensive) e la seconda a maggior uso di Disco (Disk intensive). Dividendo il service demand della cpu per il service demand del disco abbiamo ottenuto un rapporto che mette in risalto l'utilizzazione di una risorsa rispetto all'altra. In particolare, un valore del rapporto molto grande indica un uso intensivo della CPU e, viceversa, un valore del rapporto prossimo a zero indica un uso maggiore del Disco.

Di seguito i valori raccolti delle due query distinte:

Nome Query	D_CPU	D_DISK	CPU/IO
16.sql	2693,30592	57,6226825	46,7403773
5.sql	1773,82349	1271,65649	1,39489202

I dati raccolti sono necessari per la realizzazione di un modello Simulativo, implementato attraverso java modelling tools. Siamo riusciti, quindi, a creare un modello chiuso con due centri di lavoro, che rappresentano CPU e Disco con service demands prima individuati, e abbiamo modellato il sistema con le due classi di workload: CPU intensive e Disk intensive.



Al fine di confrontare il modello simulativo con il sistema, abbiamo modificato il test di query svolto dallo script python così da eseguire solo le query di interesse (cpu intensive e disk intensive). In questo modo, i test ci hanno indicato il tempo totale di esecuzione di uno specifico numero di chiamate alla query. Dividendo il numero di chiamate per il tempo totale di esecuzione, abbiamo ottenuto il throughput del sistema, al netto del possibile rumore introdotto dalla presenza di altri processi in esecuzione.

I valori ottenuti sono i seguenti, distinti per dati raccolti dal test sulla macchina e dall'esecuzione di una simulazione del modello realizzato:

Classe Query	Throughput Sistema	Throughput Modello
CPU intensive	0,35 req/sec	0,36 req/sec
Disk intensive	0,31 req/sec	0,32 req/sec

Osserviamo come il sistema modellato è piuttosto fedele al sistema reale.

3) Avanzando nelle analisi del confronto tra il sistema reale e il modello simulativo, abbiamo svolto nuovamente un test sulle due classi di queries impostando il numero di job concorrenti pari a 2. Nel modello simulativo per introdurre le due stream concorrenti è stato necessario modificare le "customer classes" specificando il valore 2 sul numero di job.

I risultati sono anche stavolta conformi tra sistema e modello:

Classe Query	Throughput Sistema	Throughput Modello
CPU intensive	0,35 req/sec	0,37 req/sec
Disk intensive	0,38 req/sec	0,43 req/sec

4) Prima di dimensionare il modello secondo le specifiche di progetto, è stato necessario svolgere alcuni ultimi test di confronto sistema-simulazione. Stavolta abbiamo modificato il numero di CPU messe a disposizione dal sistema Mac per il container Docker di lavoro passando da una singola CPU a due. Di conseguenza sul modello abbiamo apportato lo stesso cambiamento aumentando il numero di server nella stazione CPU. Seguendo le indicazioni del Professore Stefano Iannucci abbiamo svolto una simulazione approfondita da un'analisi "what if" per predire le prestazioni del sistema simulato in un intervallo di query concorrenti: nel nostro caso l'intervallo varia da 1 a 5. Ciò che abbiamo fatto è stato confrontare i risultati ottenuti dalla simulazione con quelli ottenuti sulla macchina eseguendo lo script python specificando a più riprese il numero di stream concorrenti da 1 a 5.

La procedura descritta è stata eseguita sia per la classe di workload CPU intensive sia per quella Disk intensive.

Tabella dati raccolti per cpu intensive:

N. Query concorrenti	Throughput Sistema	Throughput Modello
1	0,36 req/sec	0,39 req/sec
2	0,73 req/sec	0,80 req/sec
5	0,74 req/sec	0,80 req/sec

Tabella dati raccolti per disk intensive:

N. Query concorrenti	Throughput Sistema	Throughput Modello
1	0,33 req/sec	0,34 req/sec
2	0,52 req/sec	0,55 req/sec
5	0,70 req/sec	0,73 req/sec

Nell'osservare i risultati ottenuti dalla simulazione, abbiamo notato come i risultati del throughput, a partire da tre queries concorrenti, non superano un certo valore poiché il valore dell'utilizzazione è prossimo al 100%. Inoltre, notiamo come, diversamente dal punto 3, il passaggio da una query a due queries concorrenti cambi di molto il throughput (specialmente per la cpu intensive) perché in quest'ultimo test il sistema può usufruire di due cpu e quindi processare insieme le due queries differentemente dal caso precedente in cui una query rimaneva in attesa.

4a) Stabilito dai confronti effettuati che il modello simulativo sia affidabile, abbiamo modellato il sistema in modo tale che il tempo di risposta medio sia inferiore ai 30

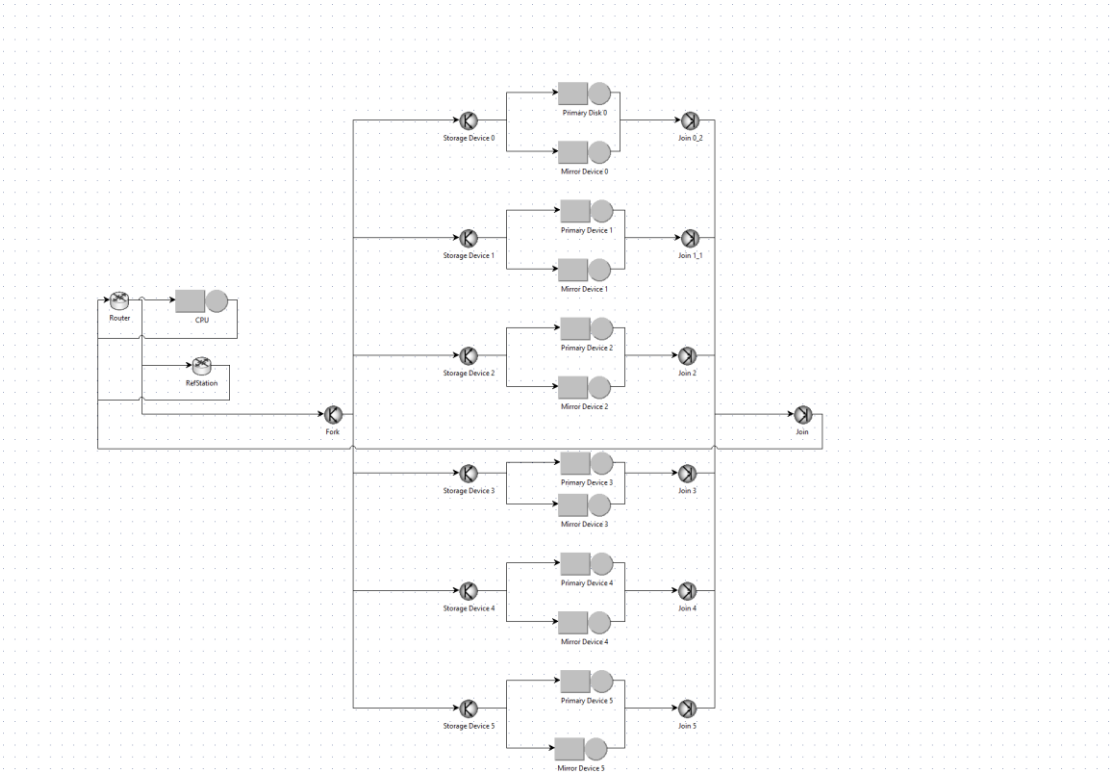
secondi per un numero di query concorrenti massimo non superiore a 20 e assicurandoci che l'utilizzazione sia nell'intervallo 60%-70%. Inoltre, come da requisito, abbiamo implementato il sistema di storage con un RAID 10.

Per quanto riguarda la CPU, la nostra analisi è stata questa: 2 job concorrenti processate da 2 cpu causano un'utilizzazione del sistema pari al 100%, quindi 20 cpu processerebbero 20 queries concorrenti tenendo sempre un'utilizzazione al massimo; seguendo questa proporzione, per avere un'utilizzazione compresa tra il 60%-70% bisogna aumentare il numero di cpu quasi del doppio.

In particolare, abbiamo raggiunto il 64,5% di utilizzazione con 30 CPU.

Per quanto concerne il disco, applicando lo stesso ragionamento usato per la cpu abbiamo raggiunto un'utilizzazione pari al 64% grazie a 12 dischi.

L'implementazione dello storage attraverso sistema RAID 10 è costituito da 6 sottosistemi RAID *zero* ciascuno composto da un disco primario e un disco di mirroring (RAID *uno*)



L'utilizzazione dei singoli dischi RAID è 74% e questo valore evidenzia una differenza rispetto all'utilizzo dei 12 dischi modellati in precedenza. Questo poiché le richieste in arrivo non vengono ripartite per ogni disco, ma vengono completamente inoltrate ad ogni disco raid con la differenza che il job da effettuare è solo una parte della richiesta intera.

5) Nell'ultima fase di analisi e modellazione del sistema, il progetto ha richiesto un cambiamento riguardo la definizione del workload che passa da numero di job concorrenti ad un tasso di arrivi. Questo cambiamento ha reso il modello, costruito precedentemente, un sistema aperto.

Sono state definite le 5 classi di queries suggerite dal Professore Stefano Iannucci: due (A e B) caratterizzate da service demand cpu intensive, due (D e E) caratterizzate da service demand disk intensive, una (C) caratterizzata da tempi di servizio della query con rapporto CPU/IO più bilanciato. Nel nostro caso la classe C coincide proprio con la query disk intensive poiché il disco del sistema Mac preso in analisi per il progetto ha un disco altamente prestante che raramente viene messo in difficoltà rispetto all'uso della CPU.

Tutte le classi introdotte hanno lo stesso valore di tasso di arrivi esponenziale, fatta eccezione per la query di tipo C che segue un andamento di tipo Bursty caratterizzato da una distribuzione probabilistica.

L'analisi ha richiesto lo studio del valore dei tempi di risposta e dell'utilizzazione rispettando i vincoli del progetto: un'utilizzazione compresa tra 60%-70% e che il tempo di risposta di ciascuna query non deve superare in media il tempo di risposta della query più lunga a sistema scarico.

Distinguendo sempre il caso cpu intensive e disk intensive, abbiamo modellato il sistema scegliendo un numero di dischi e di cpu che rendessero un'utilizzazione conforme ai requisiti.

Per la CPU, abbiamo utilizzato 35 unità ottenendo valore di utilizzazione pari a 61%.

Per lo storage, abbiamo inserito 12 unità per ottenere utilizzazione pari a 69%.

Per quanto riguarda i tempi di risposta delle varie query, abbiamo notato che la query più lunga impiega 2.9 secondi per essere processata. Un commento, che si può fare riguardo alla query di classe C, è che la simulazione impiega più tempo per concludersi perché la classe C è caratterizzata da un andamento probabilistico che presenta dei picchi di valori più complessi e lunghi da calcolare.

Il vincolo sui tempi di risposta è pienamente soddisfatto, in quanto la query più lunga a sistema scarico dura 33 secondi. Ciò vuol dire che il disco della macchina, utilizzata per le valutazioni, processa in maniera molto efficiente le query. Qualora il valore fosse minore di quello previsto dal modello, occorre diminuire questa variazione, per esempio, introducendo delle cpu più veloci e non aumentando il numero di cpu perché non cambierebbe nulla a livello di prestazioni.

Infine, il sottosistema di storage deve essere implementato con RAID 10, seguendo il procedimento illustrato al punto 4a, abbiamo utilizzato lo stesso numero di dischi (12). Si può sempre notare un aumento dell'utilizzazione intorno al 10% per il motivo descritto precedentemente.

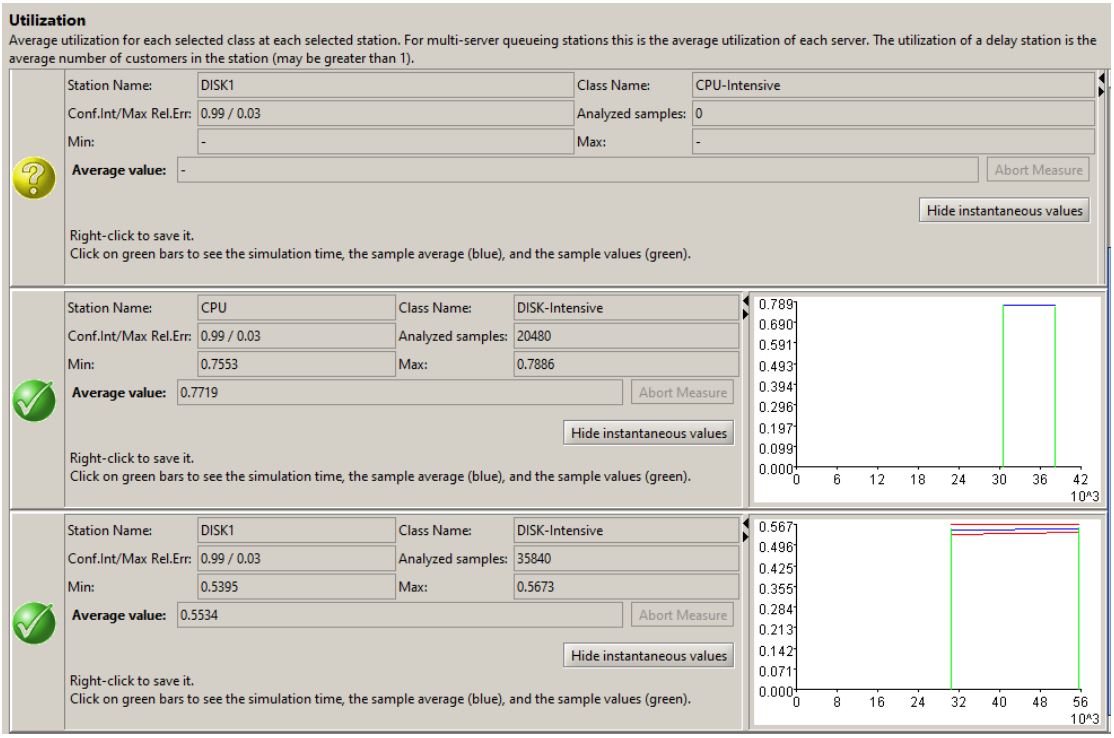
CONCLUSIONI

Vogliamo concludere questa relazione, sottolineando alcune considerazioni riguardo l'utilizzazione delle risorse che ci hanno evidenziato, nelle varie situazioni simulate, il collo di bottiglia del modello. Riconoscere ciò che limita le prestazioni del sistema significa individuare il punto critico sul quale investire risorse economiche per ottenere il massimo risultato.

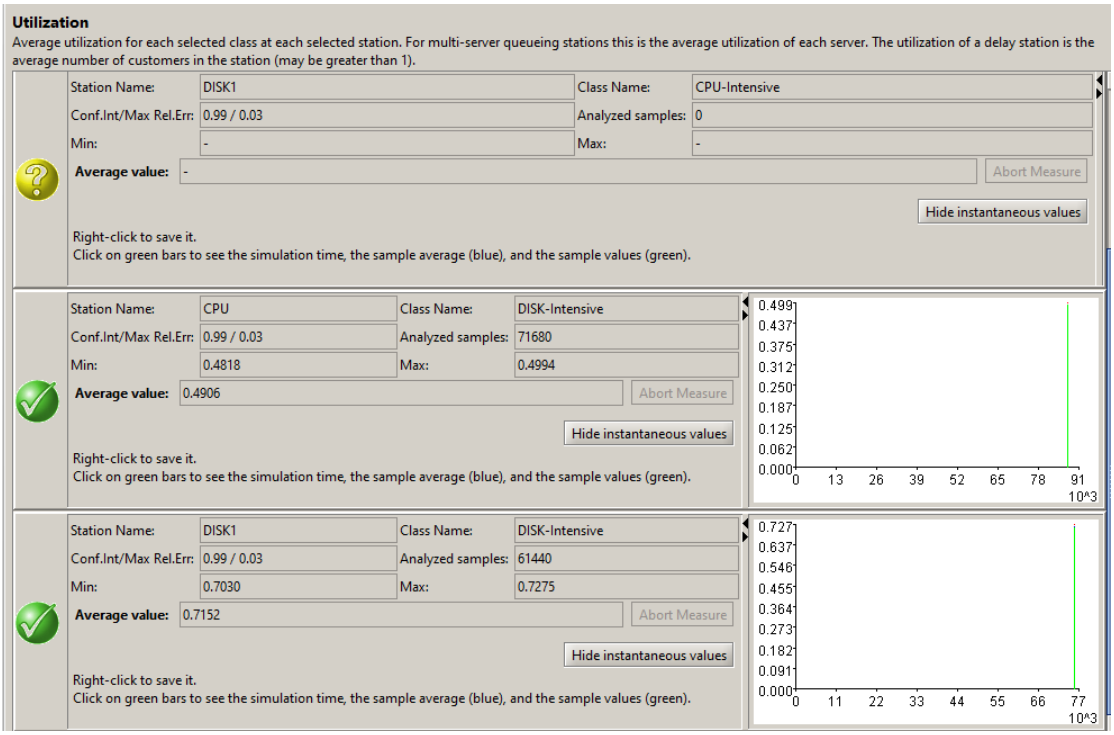
Al passo 2 e 3 dello svolgimento il sistema era limitato all'uso di una sola cpu e questo ha portato la simulazione ad evidenziare un limite forte sul processore per quanto riguarda l'esecuzione di queries cpu intensive. La classe di workload disk intensive ha reso collo di bottiglia la cpu nonostante il carico fosse sbilanciato verso il disco. Da ciò si evince che il disco del sistema preso in analisi è decisamente rapido e che il modello è limitato fortemente dall'uso di una sola cpu.

Di fatto, quando al passo 4 abbiamo introdotto un'unità cpu aggiuntiva i valori di utilizzazione sono cambiati: nel caso cpu intensive non si può avere miglioramenti visto il workload pensato per stressare il processore, nel caso disk intensive la cpu in più riesce a migliorare le prestazioni rendendo l'utilizzazione più sbilanciata verso il disco.

Di seguito è riportato il valore dell'utilizzazione della simulazione del caso di una CPU e due job disk intensive concorrenti:



Mentre per l'utilizzazione della simulazione con due CPU e due job concorrenti disk intensive:



In conclusione, riteniamo il progetto utile a comprendere i limiti dei sistemi in funzione dei carichi di lavoro. Inoltre, è stato interessante svolgere attività riguardo come si dovrebbe lavorare per creare un modello che possa simulare sistemi reali. I concetti e le metodologie apprese sono applicabili a più contesti indipendentemente dal tipo di risorsa. Il progetto è stato anche un incentivo a fare pratica con tecnologie come Docker, java modelling tools e altri strumenti che pervadono la realtà lavorativa di ogni giorno.