Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

New Generation Databases

ML4DB

Applicazioni del machine learning per l'ottimizzazione automatica dei DBMS

Gruppo:
Regy Cekerri
Ciro Maccarone
Luca Liberatore

Professoressa:
Claudia Diamantini

A.A. 2022/2023

Indice

Introduzione

Il machine learning e l'intelligenza artificiale, grazie alla loro crescente popolarità, stanno alimentando con successo un ampio spettro di applicazioni, tra cui il riconoscimento vocale, la classificazione delle immagini, l'elaborazione del linguaggio naturale e molte altre.
Il machine learning si concentra sullo sviluppo di algoritmi e modelli computazionali in grado di apprendere da dati e migliorare le proprie prestazioni nel tempo senza essere esplicitamente programmati. In altre parole, il machine learning consente ai computer di acquisire conoscenza e compiere previsioni o prendere decisioni basate su esperienze passate o dati forniti. Questo processo di apprendimento è spesso basato su statistiche, matematica e algoritmi che consentono ai computer di riconoscere pattern, estrarre informazioni significative e adattarsi a situazioni nuove o complesse.
Tra le diverse branche del machine learning, il reinforcement learning ha recentemente guadagnato notevole attenzione. Il reinforcement learning si concentra sulla formazione di agenti o sistemi intelligenti in grado di prendere decisioni sequenziali per massimizzare una ricompensa in un ambiente specifico. Nel reinforcement learning, un agente interagisce con un ambiente e prende decisioni basate sulle informazioni disponibili e sulla sua conoscenza corrente. L'obiettivo dell'agente è imparare a prendere le decisioni migliori in modo da massimizzare una ricompensa cumulativa nel lungo termine. L'agente esplora l'ambiente, sperimenta diverse azioni e riceve feedback in forma di ricompense o punizioni. Il reinforcement learning è ampiamente utilizzato in applicazioni in cui la sequenza di decisioni è cruciale, come nel controllo dei robot, nei giochi, nell'ottimizzazione di strategie, nella gestione di risorse, nella guida autonoma e in molti altri campi.
Anche un'ulteriore branca del machine learning ha guadagnato di recente un ruolo cruciale, grazie all'incremento dei dati e alla crescente potenza di calcolo. Si tratta del deep learning, che consente di replicare il funzionamento del cervello umano attraverso la creazione di reti neurali complesse e profonde.

In sintesi, i sistemi che adottano il machine learning possiedono la capacità di apprendere dai dati, una caratteristica che manca alla maggior parte dei database tradizionali. Questi sono infatti generalmente statici e hanno delle limitate opzioni di adattamento. Ciò è in contrasto con quanto avviene nel mondo reale, soprattutto nell'ambito del cloud, dove i carichi di lavoro delle query evolvono nel tempo e la distribuzione dei dati varia in diverse situazioni applicative.
Nonostante i primi tentativi di integrazione del machine learning nei database risalgono a molti decenni fa, soltanto negli ultimi anni si è assistito a dei progressi significativi sia nell'ambito accademico che in quello industriale. Tale integrazione ha l'obiettivo principale di rendere i database in grado di configurarsi, ottimizzarsi, monitorarsi e ripararsi in modo autonomo.
Come mostrato in figura 1, il machine learning può essere utilizzato in diversi modi per migliorare i database:

- ottimizzazione dei parametri di configurazione;

- indicizzazione automatica: il machine learning può essere utilizzato per creare indicizzazioni automatiche dei dati; questo permette un accesso più rapido ai dati e una migliore organizzazione delle informazioni all'interno del database;
- ottimizzazione delle query: il machine learning può essere impiegato per migliorare le prestazioni delle query identificando automaticamente i piani di esecuzione più efficienti; ciò consente di accelerare il recupero dei dati e ridurre il tempo di risposta;
- predizione della domanda: il machine learning può essere utilizzato per prevedere le future esigenze del database, come il carico di lavoro, in modo da allocare risorse in modo ottimale;
- previsione delle prestazioni: il machine learning può essere utilizzato per prevedere le prestazioni del database in base alle variazioni del carico di lavoro, consentendo una pianificazione anticipata per scalare o ottimizzare il database di conseguenza;
- riduzione della manutenzione manuale;
- rilevamento delle anomalie: l'uso del machine learning per il rilevamento delle anomalie aiuta a identificare comportamenti non conformi o accessi non autorizzati al database, migliorando la sicurezza.

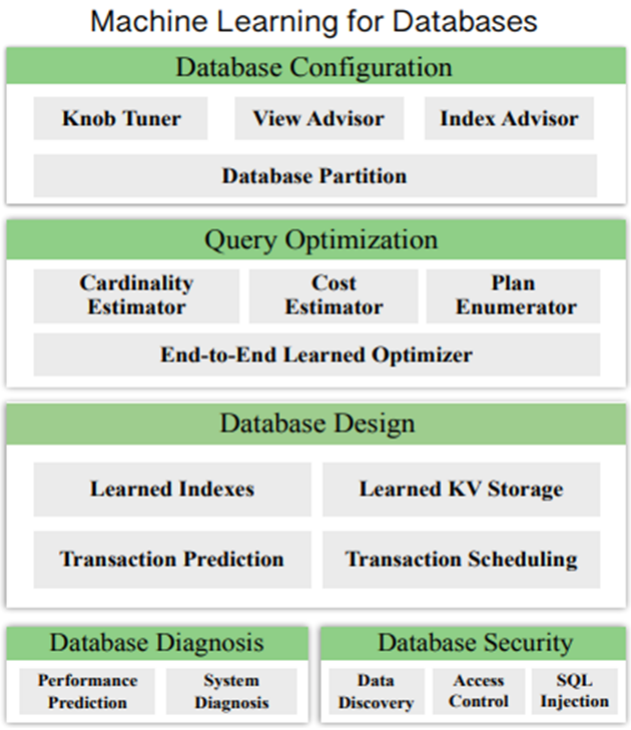


Fig. 1: Panoramica sui metodi di integrazione del machine learning nei database.

Capitolo 1

Ottimizzazione automatica dei parametri di configurazione del database

I parametri di configurazione controllano numerosi aspetti dei DBMS, tra cui le connessioni degli utenti, l'ottimizzazione delle query e la gestione delle risorse sottostanti. Le diverse combinazioni di valori di tali parametri influenzano in modo significativo la robustezza del sistema, le prestazioni e l'utilizzo delle risorse. In generale, l'ottimizzazione dei parametri di configurazione ha lo scopo di regolare con giudizio i valori dei parametri in modo da raggiungere alcuni obiettivi. Ad esempio, impostare correttamente i parametri di configurazione che influenzano l'ottimizzatore nei database relazionali consente di generare piani di query efficaci e di ottimizzare le prestazioni, così come regolare i parametri di configurazione relativi alla gestione della memoria del database consente di evitare frequenti operazioni su disco e di migliorare l'efficienza dell'accesso ai dati.
In passato, l'ottimizzazione dei parametri di configurazione richiedeva agli amministratori di database di testare manualmente combinazioni tipiche dei parametri. Questo approccio risultava estremamente time-consuming e non era adatto a gestire un grande numero di istanze di database. Per risolvere questi problemi, i ricercatori stanno lavorando alla progettazione di metodi in grado di stabilire automaticamente i parametri di configurazione ottimali per ogni diversa situazione.

1.1 Sfide nell'ottimizzazione dei parametri di configurazione

L'ottimizzazione dei parametri di configurazione non è esente da problematiche (un'illustrazione è mostrata in figura 2):

- bisogna stabilire quali sono gli obiettivi della regolazione dei parametri;
- bisogna stabilire quali parametri devono essere regolati;
- bisogna stabilire come e con cosa regolare i parametri.

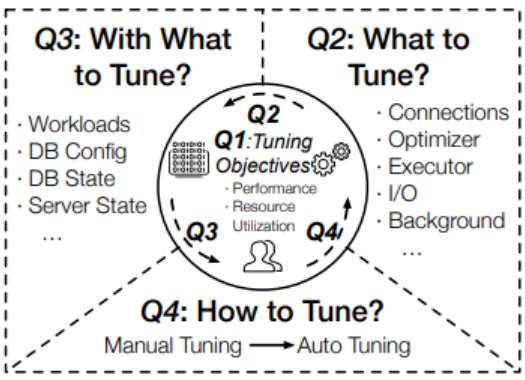


Fig. 2: Problematiche principali nell'ottimizzazione dei parametri di configurazione.

1.1.1 Quali sono gli obiettivi della regolazione dei parametri?

Gli obiettivi della regolazione dei parametri presentano due aspetti. Da un lato, i database devono essere configurati in modo da ottenere prestazioni elevate, in termini di throughtput e di latenza. D'altra parte, i database devono utilizzare al meglio le risorse e presentare dei costi di manutenzione ridotti, senza dover sacrificare le prestazioni.
Un metodo di regolazione dei parametri viene generalmente valutato sotto quattro aspetti:

- prestazioni (quanto bene il metodo raggiunge gli obiettivi in un dato scenario);
- overhead (quanto tempo o risorse di sistema richiede il metodo per raccomandare la configurazione);
- adattabilità (quanto bene il metodo raggiunge gli obiettivi in nuovi scenari);
- sicurezza.

1.1.2 Quali parametri devono essere regolati?

I parametri di configurazione di un DBMS sono centinaia e la loro regolazione può richiedere molto tempo e molte risorse di sistema. È quindi fondamentale selezionare i parametri più importanti che possono influenzare in modo significativo le prestazioni. Tuttavia, vi sono delle problematiche:

- non è semplice stabilire quanto la regolazione di un parametro impatti sulle prestazioni (testare ogni singolo parametro richiede molto tempo);
- alcuni parametri sono tali per cui la loro regolazione può incrementare le prestazioni in alcuni scenari, ma deteriorarle in altri (è necessario quindi stabilire un compromesso).

1.1.3 Con cosa si devono regolare i parametri?

Con i parametri appropriati, un altro problema è come selezionare le feature di regolazione (ad esempio, i carichi di lavoro, lo stato del database, gli ambienti hardware), che possono riflettere i comportamenti di esecuzione del database e potenzialmente influenzare le prestazioni della regolazione. È difficile caratterizzare i requisiti della regolazione con delle feature adeguate. In primo luogo, le feature di regolazione si trovano in uno spazio dimensionale elevato. Sebbene numerose feature possono riflettere lo stato del database, diverse di esse catturano aspetti di regolazione simili, rendendo complesso il processo di filtraggio delle feature ridondanti. In secondo luogo, molte feature di regolazione appartengono a domini diversi; dunque, è difficile combinarle nello stesso dominio.

1.1.4 Come si devono regolare i parametri?

Dopo aver filtrato la maggior parte dei parametri e delle feature non rilevanti, lo spazio di configurazione può essere ancora grande. Poichè esistono varie combinazioni dei parametri, la loro regolazione è un problema NP-difficile.
Le problematiche principali sono le seguenti:

- anche se il numero dei parametri è stato ridotto in modo significativo, la maggior parte di questi varia in un intervallo continuo; dunque, lo spazio di configurazione è troppo grande per essere enumerato;
- valutare le prestazioni della regolazione è costoso e complesso.

I metodi di regolazione tradizionali sono inadatti per i seguenti motivi:

- non riescono distinguere tra vari scenari applicativi reali;
- non riescono a comprendere i complessi legami tra la regolazione dei parametri e le performance.

I nuovi approcci devono quindi, oltre ad ottimizzare efficacemente la configurazione di un database, rendere tali sistemi capaci di adattarsi a più scenari applicativi.

1.2 Panoramica sull'ottimizzazione dei parametri di configurazione

Come illustrato nella figura 3, un workflow generale per l'ottimizzazione dei parametri di configurazione comprende quattro moduli principali:

- la selezione dei parametri di configurazione;
- la selezione delle feature;
- i metodi di regolazione;
- le tecniche di trasferimento.

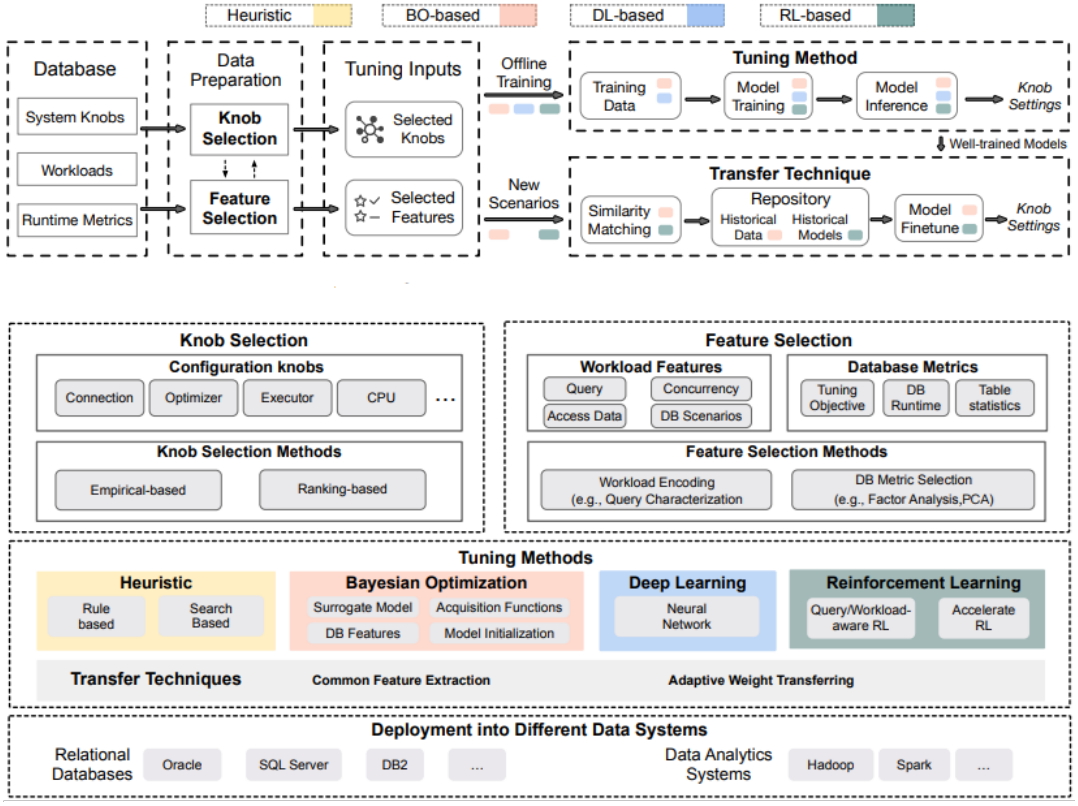


Fig. 3: Workflow generale per l'ottimizzazione automatica dei parametri di configurazione.

1.3 Selezione dei parametri di configurazione

1.3.1 Parametri di configurazione

In senso lato, per parametri di configurazione si intendono i parametri del database i cui valori possono essere regolati. Più formalmente, ad ogni parametro di configurazione è associato un valore V_k, che può essere regolato all'interno di un intervallo di valori (V_{min}, V_{max}) o di un insieme di valori candidati $\{V_1, ..., V_k\}$. A seconda del valore associato, le prestazioni del database possono migliorare, rimanere invariate o peggiorare.
Tipicamente, un DBMS contiene centinaia di parametri di configurazione, ognuno dei quali produce diversi effetti sulle prestazioni del sistema. Tali parametri possono essere categorizzati in base all'area del DBMS impattata (degli esempi sono riportati in figura 4):

- controllo degli accessi;
- ottimizzazione delle query;
- esecuzione delle query;
- processi in background;

- gestione delle risorse.

	Category	Functionality	Example (Postgres)	Example (MySQL)
1	Access Control	Connections	max_connections	innodb_thread_concurrency
		Transactions	deadlock_timeout	innodb_table_locks
2	Query Optimizer	Query Plan	join_collapse_limit	rewriter_enabled
		Cost Values	seq_page_cost	join_buffer_size
3	Query Executor	Persistence	full_page_writes	replica_pending_jobs_size_max
4	Background Processes	Logging	log_rotation_size	binlog_cache_size
		Others	checkpoint_timeout	innodb_log_file_size
5	Resource (CPU)	CPU Usage	max_files_per_process	innodb thread concurrency
6	Resource (Memory)	Memory Space	shared_buffers	innodb_buffer_pool_size
7	Resource (Disk)	Disk IO/Caches	temp_file_limit	max_sort_file_size

Fig. 4: Esempi di parametri di configurazione.

Ogni parametro ha inoltre un costo di regolazione differente: alto, se richiede il riavvio del database, medio, se richiede delle modifiche significative dei meccanismi sottostanti, oppure basso.

1.3.2 Metodi per la selezione dei parametri di configurazione

Esistono varie tecniche per la selezione dei parametri di configurazione, tra cui le strategie empiriche e le strategie basate sul ranking, che mirano a filtrare i parametri non importanti e a ridurre lo spazio di configurazione.
Le strategie empiriche richiedono che gli esseri umani selezionino i parametri di configurazione in base a documenti di configurazione o all'esperienza empirica. Il vantaggio delle strategie empiriche è che sono relativamente semplici e più affidabili, poiché ogni parametro di configurazione viene verificato manualmente. Tuttavia, presentano alcune limitazioni:

- è difficile per gli esseri umani identificare le complesse correlazioni tra i vari parametri di configurazione;
- affidarsi semplicemente all'esperienza può portare ad ignorare parametri di configurazione potenzialmente efficaci sulle prestazioni del sistema;
- le regole euristiche seguite potrebbero non funzionare bene per scenari complessi;
- l'esperienza nella regolazione potrebbe non essere al passo con i frequenti aggiornamenti di versione dei DBMS.

Le strategie empiriche effettuano la selezione senza valutare la relazione tra i parametri di configurazione e le prestazioni. D'altra parte, le strategie basate sul ranking ordinano i parametri di configurazione in base al loro impatto sulle prestazioni del database e selezionano quelli con un impatto elevato. Tali strategie comprendono:

- un primo passo, in cui vengono raccolti dei campioni di configurazione;
- un secondo passo, in cui vengono classificati i parametri di configurazione.

Anche le strategie basate sul ranking non sono esenti da problematiche:

- trovare le relazioni tra i parametri di configurazione e le prestazioni non è affatto semplice;
- le prestazioni della regolazione possono essere influenzate dall'ommissione di alcuni parametri importanti;
- si hanno delle maggiori difficoltà in scenari dinamici.

1.4 Selezione delle feature

Diverse feature possono rappresentare le caratteristiche del database e fornire importanti indicazioni sui requisiti di regolazione. In linea di massima, le feature possono essere distinte in:

- feature del carico di lavoro;
- feature metriche del database.

Esistono differenti tecniche di selezione per queste due categorie di feature: per codificare le feature del carico di lavoro possono essere utilizzate delle tecniche di discretizzazione, mentre per la selezione delle feature metriche possono essere utilizzati degli algoritmi di clustering.

1.5 Metodi di regolazione

Il problema della regolazione dei parametri di configurazione può essere formalizzato in un problema di ottimizzazione che, dato un carico di lavoro e una serie di parametri regolabili, seleziona le impostazioni ottimali dei parametri in base a obiettivi e vincoli predefiniti. Poiché tale problema è NP-difficile, un unico metodo di regolazione non può funzionare bene in tutti gli scenari. In particolare, i metodi di regolazione esistenti possono essere classificati in quattro categorie:

- metodi euristici;
- metodi basati sull'ottimizzazione bayesiana;
- metodi basati sul deep learning;
- metodi basati sul reinforcement learning.

Una linea temporale di questi metodi di regolazione è illustrata nella figura 5.

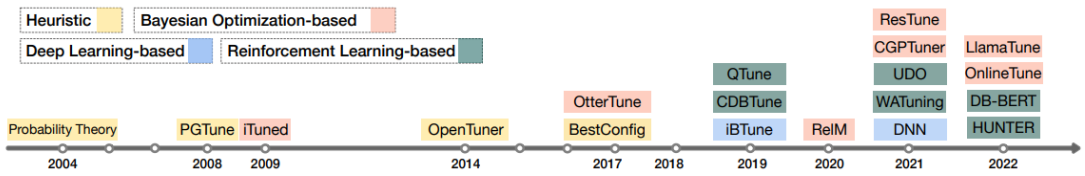


Fig. 5: Linea temporale dei metodi di regolazione dei parametri.

1.5.1 Metodi euristici

I metodi euristici possono essere classificati a grandi linee in:

- metodi basati sulle regole;
- metodi basati sulla ricerca.

I metodi basati sulle regole si ispirano alla regolazione manuale e si basano sulla teoria della probabilità. In particolare, per determinare se una regola può essere utilizzata per un determinato carico di lavoro, si ragiona sulle informazioni a disposizione. Se si ritiene che le prestazioni possano essere migliorate, allora è possibile utilizzare questa regola.
Il vantaggio principale dei metodi basati sulle regole è che sono semplici e veloci. Tuttavia, poichè la regolazione dei parametri è influenzata da molti fattori, questi metodi potrebbero non trovare la configurazione ottimale attraverso queste semplici regole. Inoltre, non possono adattarsi alle variazioni del carico di lavoro e dei dati.
D'altra parte, i metodi basati sulla ricerca impiegano un metodo di ricerca gerarchica per regolare i parametri. In primo luogo, suddividono lo spazio dei parametri in diversi sottospazi, eseguono un carico di lavoro per ciascun sottospazio e selezionano il sottospazio con le migliori prestazioni. In seguito, cercano il vicino di questo sottospazio modificando i valori dei parametri.
Sebbene non possano garantire una configurazione globalmente ottimale dei parametri, i metodi basati sulla ricerca sono in grado di individuare una configurazione che si avvicina all'ottimalità. Inoltre, rispetto ai metodi basati sulle regole, i metodi basati sulla ricerca non si basano sull'esperienza storica e possono adattarsi a diversi DBMS. Tuttavia, presentano alcuni svantaggi non trascurabili:

- possono richiedere molto tempo;
- possono ignorare, per via del campionamento, alcuni parametri importanti.

1.5.2 Metodi basati sull'ottimizzazione bayesiana

L'ottimizzazione bayesiana è un algoritmo sequenziale iterativo basato su un modello. In particolare, tale tecnica utilizza un modello surrogato per approssimare la funzione obiettivo e aggiorna sequenzialmente questo modello con i nuovi dati osservati attraverso le iterazioni. In genere, l'ottimizzazione bayesiana inizializza il modello surrogato con alcuni dati campionati. Ad ogni iterazione, una funzione di acquisizione è utilizzata per decidere dove campionare i nuovi dati con la politica di esplorazione-sfruttamento.
Nel caso del problema della regolazione dei parametri, la politica di esplorazione-sfruttamento nell'ottimizzazione bayesiana può essere illustrata come segue:

- sfruttamento: si apportano delle leggere modifiche ai valori dei parametri in base alla configurazione corrente;
- esplorazione: si esplorano alcuni valori dei parametri in base allo spazio di configurazione inesplorato.

I metodi basati sull'ottimizzazione bayesiana utilizzano le impostazioni dei parametri campionati e le loro prestazioni come punti di partenza per avviare il processo di regolazione. Dunque, ripetono le fasi di modellazione e aggiornamento finché non viene trovata una configurazione dei parametri soddisfacente o non viene raggiunta la condizione di terminazione.

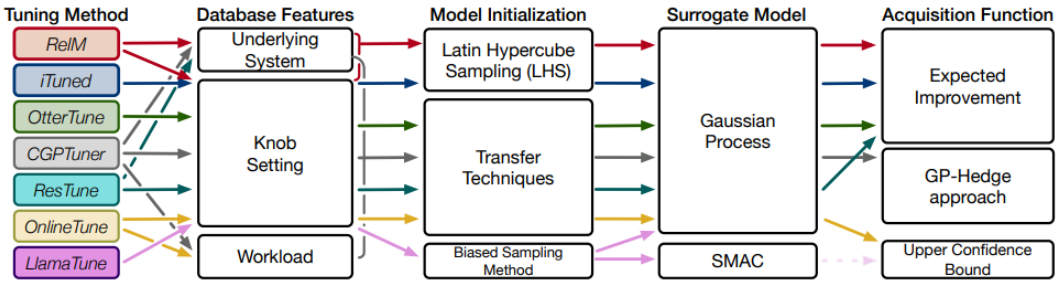


Fig. 6: Workflow generale dei metodi basati sull'ottimizzazione bayesiana.

Come mostrato nella figura 6, i metodi basati sull'ottimizzazione bayesiana si differenziano per:

- le feature che utilizzano;
- il modo in cui inizializzano il modello surrogato e la tipologia di quest'ultimo;
- la funzione di acquisizione che utilizzano.

I metodi basati sull'ottimizzazione bayesiana possono trovare una configurazione dei parametri di alta qualità attraverso la strategia di esplorazione-sfruttamento e generalmente surclassano i metodi euristici. Tuttavia, tali metodi, nel caso in cui vi siano degli ampi spazi di configurazione, non riescono a scalare efficientemente e cadono nella subottimalità.

1.5.3 Metodi basati sul deep learning

Oltre ai metodi basati sull'ottimizzazione bayesiana, anche i metodi basati sul deep learning possono stimare e migliorare iterativamente le prestazioni della regolazione. Tali metodi utilizzano una rete neurale profonda come modello di regolazione, la quale è composta da due strati nascosti (completamente connessi), ognuno dei quali possiede 64 neuroni e utilizza la ReLU come funzione di attivazione. Inoltre, per evitare la subottimalità, aggiungono un rumore gaussiano alla rete neurale per controllare la possibilità di esplorare configurazioni non viste. Rispetto ai metodi basati sull'ottimizzazione bayesiana, quelli basati sul deep learning sono in grado di stimare efficacemente le prestazioni della regolazione e non hanno bisogno di eseguire ripetutamente carichi di lavoro per valutare le impostazioni selezionate, il che può ridurre l'overhead della regolazione. Tuttavia, i metodi basati sul deep learning richiedono un gran numero di campioni di allenamento preparati per addestrare bene il modello di regolazione.

1.5.4 Metodi basati sul reinforcement learning

Nell'ambito del reinforcement learning, vi sono sei moduli:

- le azioni;
- le ricompense;
- l'agente;
- l'ambiente;
- la politica;
- lo stato.

I metodi basati sul reinforcement learning non fanno altro che mappare il problema della regolazione dei parametri in questi sei moduli. In particolare:

- le regolazioni dei parametri sono considerate come le azioni;
- le variazioni delle prestazioni dopo le regolazioni sono considerate come le ricompense;
- il modello di regolazione è considerato come l'agente;
- il database è considerato come l'ambiente;
- il modello di valutazione della regolazione è considerato come la politica;
- le metriche di esecuzione sono considerate come lo stato.

In modo iterativo, l'agente raccomanda azioni di regolazione basate sulle caratteristiche dello stato e aggiorna la politica di regolazione in base alla ricompensa per ottimizzare le prestazioni del database.
Pertanto, come mostrato nella figura 7, ci sono due sfide principali per quanto riguarda i metodi basati sul reinforcement learning: come progettare la politica per trovare configurazioni dei parametri di alta qualità e come progettare lo stato per riflettere le statistiche di esecuzione del database.

Method	State	Policy
CDBTune [89]	Runtime metric	Deep Deterministic Policy Gradient
QTune [52]	Runtime metric, Predicted metric change	Double-State Deep Deterministic Policy Gradient
WATuning [26]	Weighted runtime metric	Deep Deterministic Policy Gradient using pre-trained model
UDO [82]	Performance metric	Delayed Hierarchical Optimistic Optimization (a mcts algorithm)
DB-BERT [77]	Runtime metric	Double Deep Q-Network algorithm using pre-trained language model
HUNTER [9]	Runtime metric	Deep Deterministic Policy Gradient

Fig. 7: Metodi basati sul reinforcement learning.

I metodi basati sul reinforcement learning non sono soggetti a limitazioni sulla dimensione dello spazio di configurazione. Tuttavia, questi richiedono un overhead di regolazione molto più elevato rispetto ai metodi basati sull'ottimizzazione bayesiana e le loro prestazioni dipendono in larga misura dalle impostazioni dei parametri inizialmente campionati.

1.6 Tecniche di trasferimento

Negli scenari reali, i carichi di lavoro possono cambiare dinamicamente. Pertanto, non esiste un'unica impostazione di un parametro che possa essere ottimale per diversi carichi di lavoro. Tuttavia, i metodi analizzati potrebbero non essere efficientemente trasferibili a nuovi carichi di lavoro. In altre parole, ogni volta che il carico di lavoro cambia, questi metodi devono essere addestrati da zero, il che richiede molto tempo e spreca molte risorse di sistema.
Per adattarsi ai carichi di lavoro dinamici, le tecniche di trasferimento mirano a migliorare l'efficienza della regolazione migrando le conoscenze apprese da attività di regolazione storiche a nuove attività di regolazione. Attualmente, sono state proposte tre tecniche di trasferimento, che possono migliorare l'adattabilità dei metodi di regolazione dei parametri per i carichi di lavoro dinamici, tra cui:

- la mappatura del carico di lavoro;
- l'incorporazione del carico di lavoro appreso;
- l'ensemble di modelli.

Queste tecniche, seppur con approcci distinti, condividono alcune premesse fondamentali. Innanzitutto, presuppongono che sia possibile rendere un modello di regolazione versatile ed adattabile, estraendo direttamente le feature comuni da diversi carichi di lavoro. Inoltre, si basano sull'assunzione che migliorare il processo di individuazione di un nuovo modello di regolazione per un particolare carico di lavoro è possibile quando si dispone di informazioni relative all'addestramento di diversi modelli di regolazione su varie tipologie di carichi di lavoro. Nello specifico, il nuovo modello può essere sviluppato identificando i carichi di lavoro storici più simili a quello attuale. Di conseguenza, è possibile utilizzare come punto di riferimento sia i modelli di regolazione corrispondenti a tali carichi di lavoro, sia il risultato di un ensemble di tutti i modelli, dove i pesi vengono assegnati in base alla similarità.

Sebbene possano adattarsi a carichi di lavoro diversi, le tecniche descritte potrebbero non funzionare nei casi in cui si hanno cambiamenti nello schema, nel kernel o nell'hardware del database. In tali casi, quindi, si rendono necessarie tecniche nuove e più complesse.

1.7 Sfide e problemi aperti

L'ottimizzazione automatica dei parametri di configurazione del database presenta alcune problematiche che richiedono particolare attenzione.

In primo luogo, nonostante la selezione dei parametri di configurazione sia particolarmente importante, i metodi esistenti risultano spesso inefficaci per i seguenti motivi:

- la relazione tra i parametri e le prestazioni del database viene comunemente modellata in modo lineare, il che può portare a errori significativi;
- la correlazione tra i parametri non è tenuta in considerazione, sebbene sia un'informazione particolarmente rilevante per la regolazione.

Pertanto, sono necessari metodi di selezione che considerino sia le caratteristiche dei singoli parametri che i loro effetti combinati.
In secondo luogo, i metodi attuali per la selezione delle feature devono affrontare la sfida dell'interpretabilità, ossia è fondamentale che questi forniscano giustificazioni chiare e comprensibili alle selezioni effettuate, consentendo agli operatori del sistema di comprendere il razionale dietro le configurazioni proposte.
Un ulteriore ostacolo è rappresentato dalla dipendenza da dati costosi, come informazioni sui carichi di lavoro e lo stato del database, che potrebbero non essere sempre disponibili nella pratica. È imperativo migliorare le metodologie di regolazione in modo che siano valide anche quando i dati sono limitati. La capacità di adattamento incrementale è altresì essenziale, permettendo una regolazione efficiente in risposta a variazioni nei carichi di lavoro o nello schema del database.
In aggiunta, va tenuto in considerazione il fatto che la regolazione dei parametri raggiunge il suo scopo soltanto se è possibile valutarne le prestazioni. Tale processo è però dispendioso; di conseguenza, è cruciale sviluppare modelli predittivi in grado di stimare le prestazioni della regolazione senza la necessità di implementarla fisicamente.
Infine, la complessità di addestrare modelli di regolazione separati per scenari diversi può essere proibitiva a causa dei costi elevati. Pertanto, è essenziale esplorare soluzioni che consentano lo sviluppo di modelli singoli in grado di generalizzare efficacemente e ottenere buone prestazioni in una varietà di contesti.

1.8 OtterTune

OtterTune è un servizio automatizzato progettato per ottimizzare le configurazioni dei database mediante la regolazione dei loro parametri di configurazione. Come affermato in precedenza, tali parametri influenzano vari aspetti del comportamento del database, come le dimensioni dei buffer e le politiche di caching. La corretta regolazione di questi parametri in base al carico di lavoro, alle dimensioni dei dati e all'hardware sottostante può notevolmente migliorare le prestazioni e l'efficienza complessiva del database. Tuttavia, i moderni database dispongono di centinaia di questi parametri, rendendo difficile per gli esseri umani la loro gestione e corretta configurazione.
OtterTune è quindi un servizio di regolazione che fa uso del machine learning per ottimizzare automaticamente i parametri di configurazione di un database. Questo software è compatibile con database quali Postgres, MySQL e Oracle, operanti sia nel cloud che in data center privati. Collegare OtterTune a un database è un

processo agevole: non è richiesta l'installazione di software dedicati né la modifica dell'applicazione. Dopo aver concesso a OtterTune permessi limitati per accedere al database, il driver monitora il comportamento del database mediante l'uso di comandi SQL standard per recuperare le relative metriche. Queste metriche comprendono contatori interni mantenuti da ciascun database, come le operazioni di lettura e scrittura su disco. Inoltre, OtterTune è in grado di raccogliere metriche da fonti esterne di terze parti, come Amazon CloudWatch, per ottenere informazioni dettagliate sull'hardware, come l'utilizzo della CPU e la disponibilità di memoria.
Una rappresentazione dell'architettura interna di OtterTune è riportata nella figura 8.

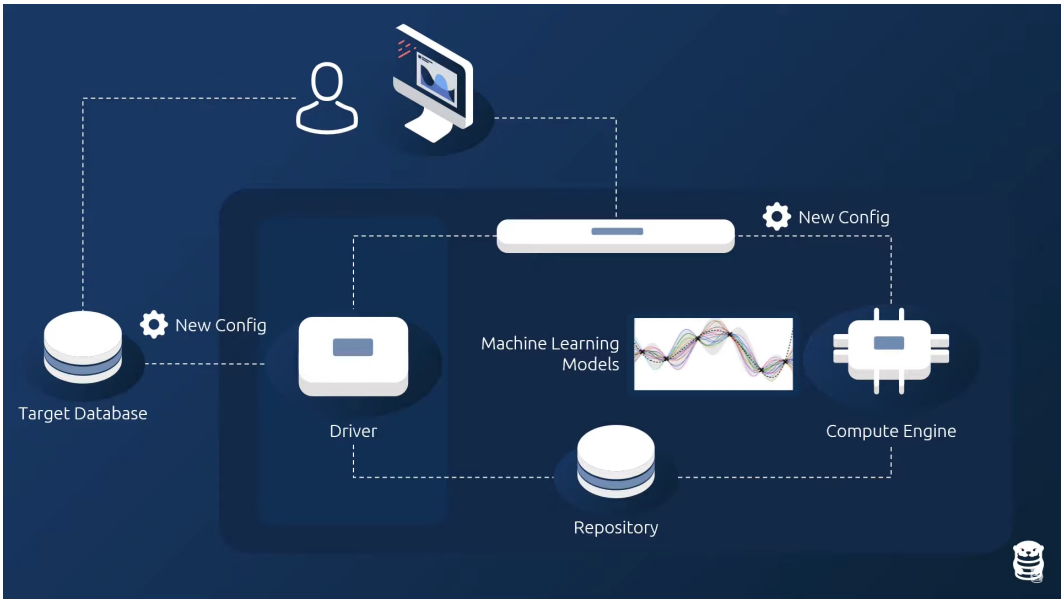


Fig. 8: Architettura interna di OtterTune.

Il driver memorizza le metriche e la configurazione del database nel proprio repository interno. In seguito, il motore di calcolo di OtterTune (Compute Engine) addestra quindi dei modelli di machine learning (in particolare, viene utilizzata l'ottimizzazione bayesiana) per predire in che modo cambieranno le prestazioni del database al variare dei parametri di configurazione. L'algoritmo di raccomandazione del Compute Engine sfrutta questi modelli per generare nuove configurazioni. Il driver installa quindi automaticamente la nuova configurazione e ne monitora attentamente gli effetti prodotti. Parallelamente, monitora continuamente il database per assicurarsi che utilizzi costantemente la configurazione ottimale.

1.8.1 Applicazione di OtterTune su Amazon RDS per PostgreSQL

Quando si esegue il provisioning di un'istanza RDS PostgreSQL su Amazon, il servizio la avvia con le impostazioni predefinite per decine di parametri di configurazione regolabili. Queste impostazioni hanno un impatto sulle prestazioni del

database, ma spesso non sono ottimali per il carico di lavoro specifico del database in questione. Ottimizzare tali impostazioni richiede una conoscenza specializzata degli aspetti interni di PostgreSQL e necessita di tempo per apportare modifiche in modo iterativo, testandone l'impatto sulle prestazioni fino al raggiungimento del livello desiderato di ottimizzazione.
Un esperimento è stato condotto al fine di valutare le prestazioni di Amazon RDS per PostgreSQL in tre situazioni differenti, nelle quali sono state utilizzate:

- le impostazioni predefinite dei parametri di configurazione di Amazon RDS;
- le raccomandazioni delle impostazioni dei parametri di configurazione fornite da PGTune (un metodo euristico per l'ottimizzazione dei parametri di configurazione);
- le raccomandazioni delle impostazioni dei parametri di configurazione ottenute attraverso gli algoritmi di machine learning di OtterTune.

Tale esperimento è stato eseguito mediante l'impiego del framework OLTP-bench e il benchmark TPC-C. Sono state raccolte le metriche di runtime dal database, e il throughput è stato successivamente calcolato basandosi su di esse. I test sono stati eseguiti su un'istanza di Amazon RDS (db.m5.4xlarge) con le seguenti specifiche hardware:

- 16 vCPU;
- 64 GB di RAM;
- 4000 IOPs.

I risultati di tale esperimento sono riportati nella figura 9.

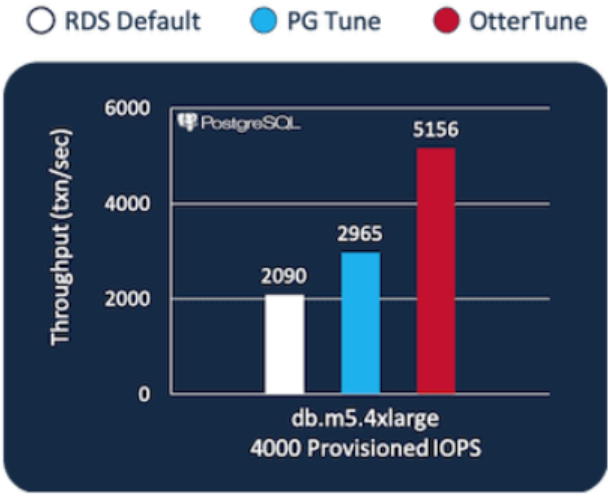


Fig. 9: Risultati in termini di throughput dell'esperimento condotto su Amazon RDS per PostgreSQL.

Come è possibile osservare, le raccomandazioni fornite da PGTune hanno prodotto un aumento del throughput del 41.9%. D'altro canto, OtterTune ha superato

notevolmente PGTune, portando il throughput a oltre tre volte quello ottenuto con quest'ultimo e vantando un incremento del 146.7% rispetto alle impostazioni predefinite di Amazon RDS.
I risultati dell'esperimento dimostrano che l'uso del machine learning per trovare le impostazioni ottimali dei parametri di configurazione ha più che raddoppiato le prestazioni del database, un risultato positivo per chi cerca prestazioni elevate.

Per coloro che mirano a ridurre i costi, pur mantenendo elevate le prestazioni del database, è stata condotta una replica del test su un'istanza di Amazon RDS (db.m5.2xlarge) di dimensioni inferiori, caratterizzata da un costo più contenuto, con le seguenti specifiche hardware:

- 8 vCPU;
- 32 GB di RAM;
- 2000 IOPs.

I risultati di tale replica dell'esperimento sono riportati nella figura 10.

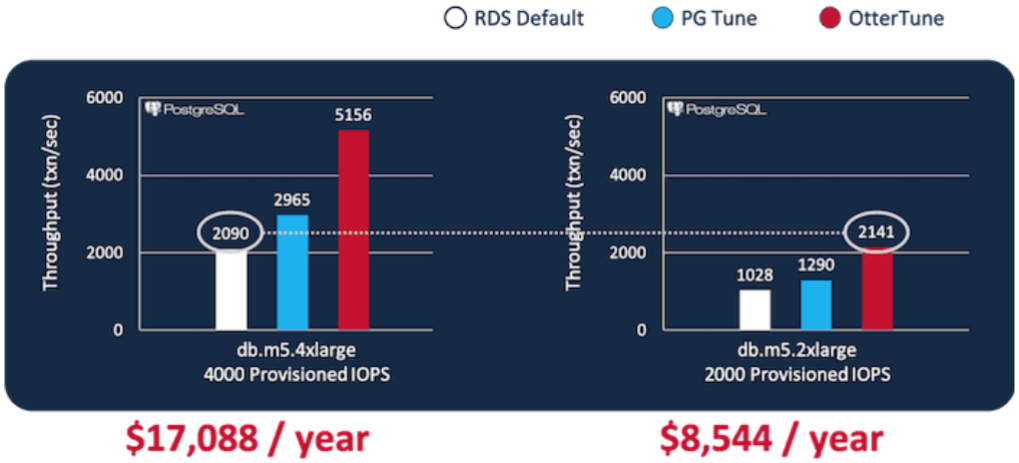


Fig. 10: Risultati in termini di throughput della replica dell'esperimento condotto su Amazon RDS per PostgreSQL.

Come è possibile osservare, OtterTune permette di ottenere le stesse prestazioni dell'istanza più grande, con le impostazioni predefinite, ma a metà del costo.

OtterTune opera attraverso un processo iterativo in cui vengono esaminate diverse impostazioni dei parametri di configurazione e le prestazioni del database sono monitorate attentamente. I modelli di machine learning di OtterTune vengono addestrati per prendere decisioni riguardo alle nuove modifiche delle impostazioni, basandosi sul carico di lavoro osservato e sugli obiettivi di ottimizzazione. Questo approccio è paragonabile a quello di una figura umana esperta, ma viene eseguito in modo più rapido ed automatizzato. Infatti, nella replica dell'esperimento, sono state necessarie diverse iterazioni per identificare le prestazioni ottimali, con un tempo totale di 2 ore dall'inizio alla fine (a tale scopo, si osservi la figura 11).

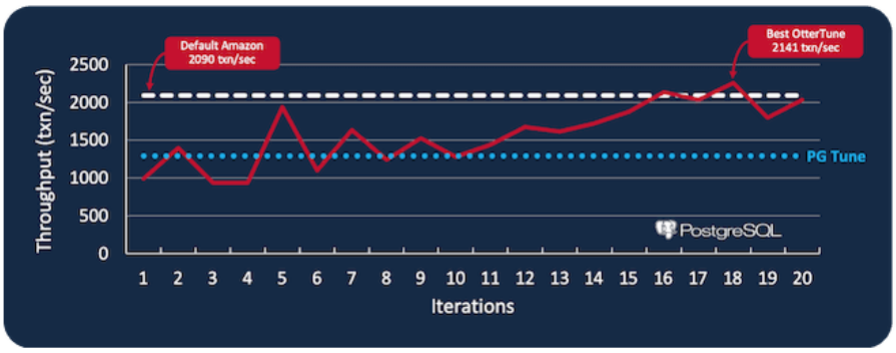


Fig. 11: Esempio di come il processo iterativo alla base di OtterTune porti ad un incremento delle prestazioni.

Inoltre, OtterTune continua costantemente a monitorare il carico di lavoro del database, regolando automaticamente le impostazioni dei parametri di configurazione qualora si verifichino cambiamenti nel carico di lavoro.

1.9 CDBTune

CDBTune è un sistema di regolazione automatica dei parametri di configurazione, particolarmente adatto per i database cloud, che sfrutta il reinforcement learning per identificare le configurazioni ottimali. In particolare, adotta una strategia per tentativi ed errori per apprendere le impostazioni dei parametri di configurazione con un numero limitato di campioni per realizzare l'addestramento iniziale, il che allevia la difficoltà di raccogliere campioni massicci di alta qualità. CDBTune adotta un meccanismo di reward-feedback al posto della regressione tradizionale, accelerando la velocità di convergenza del modello e migliorando l'efficacia della regolazione online.

1.9.1 Meccanismo di funzionamento di CDBTune

CDBTune addestra innanzitutto un modello basato su alcuni dati di addestramento. Successivamente, in caso di una richiesta di regolazione online, il sistema sfrutta il modello per raccomandare le impostazioni dei parametri di configurazione. Inoltre, CDBTune ha la capacità di aggiornare tale modello.

I dati di addestramento sono rappresentati da una quadrupla, <q a s r>, in cui:

- q: indica un insieme di carichi di lavoro delle query;
- a: rappresenta un insieme di parametri di configurazione e dei loro valori durante l'elaborazione dei carichi di lavoro;
- s: denota lo stato del database, che consiste in un insieme di 63 metriche, durante l'elaborazione dei carichi di lavoro;
- r: rappresenta le prestazioni durante l'elaborazione dei carichi di lavoro.

Poiché il problema della regolazione dei parametri di configurazione di un DBMS, che mira a trovare la soluzione ottimale in uno spazio continuo, è NP-difficile, si adotta il reinforcement learning come modello di addestramento. Il reinforcement learning adotta una strategia per tentativi ed errori per addestrare il modello, consentendo di esplorare più configurazioni ottimali, riducendo il rischio di cadere in un ottimo locale. Si noti che il modello viene addestrato una volta offline e successivamente utilizzato per regolare i parametri di configurazione del database per ogni richiesta di regolazione da parte degli utenti.

I dati di addestramento possono essere raccolti in due modalità:

- cold start:
  - a causa della mancanza di dati storici all'inizio del processo di addestramento offline, si utilizzano strumenti standard di test dei carichi di lavoro (come Sysbench) per generare una serie di carichi di lavoro di query;
  - successivamente, si esegue ogni carico di lavoro sul database cloud e si ottiene la quadrupla iniziale;
  - infine, si utilizza la strategia per tentativi ed errori per addestrare la quadrupla e ottenere più dati di addestramento;
- addestramento incrementale:
  - durante il successivo utilizzo pratico di CDBTune, per ogni richiesta di regolazione da parte dell'utente, il sistema acquisisce continuamente informazioni di feedback dalla richiesta dell'utente in base alle configurazioni consigliate;
  - aggiungendo gradualmente altri dati sul comportamento reale degli utenti al processo di addestramento, CDBTune rafforza ulteriormente il modello e ne migliora l'accuratezza delle raccomandazioni.

Se un utente desidera ottimizzare il proprio database, deve semplicemente inviare una richiesta di regolazione a CDBTune. Una volta ricevuta la richiesta, CDBTune raccoglie il carico di lavoro delle query dell'utente (q) relative agli ultimi 150 secondi circa, ottiene la configurazione corrente dei parametri (a) ed esegue il carico di lavoro delle query nel database cloud per generare lo stato corrente (s) e le prestazioni (r). Successivamente, utilizzando il modello ottenuto dall'addestramento offline, esegue la regolazione online. Infine, i parametri corrispondenti alle prestazioni migliori nella regolazione online sono consigliati all'utente. Se il processo di ottimizzazione termina, è necessario aggiornare anche il modello di reinforcement learning e il pool di memoria.
Le differenze tra l'ottimizzazione online e l'addestramento offline sono principalmente due. In primo luogo, non si utilizzano più dati simulati; al contrario, si replica il carico di lavoro corrente dell'utente per condurre test di stress sul database cloud al fine di perfezionare il modello. In secondo luogo, l'ottimizzazione

termina quando l'utente raggiunge prestazioni soddisfacenti con miglioramenti rispetto alla configurazione iniziale o quando il numero di passi di ottimizzazione raggiunge il massimo predefinito.

1.9.2 Architettura di CDBTune

Nella figura 12 è illustrata l'architettura di CDBTune.

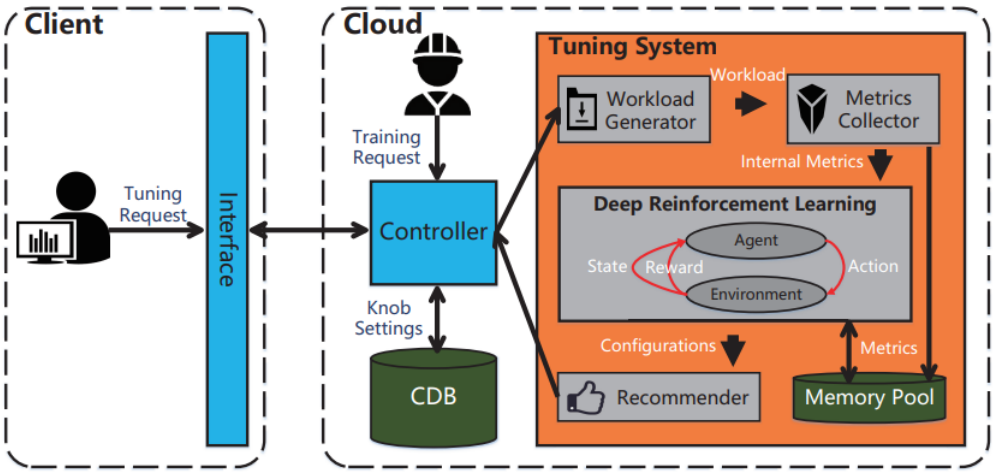


Fig. 12: Architettura di CDBTune.

Il riquadro tratteggiato a sinistra rappresenta il client, dove gli utenti inviano le proprie richieste al server attraverso l'interfaccia locale. L'altro riquadro tratteggiato rappresenta il sistema di ottimizzazione, in cui il controller, all'interno della piattaforma cloud distribuita, gestisce le informazioni tra il client, il database cloud e CDBTune. Quando l'utente avvia una richiesta di ottimizzazione o l'amministratore del database avvia una richiesta di addestramento tramite il controller, il workload generator esegue uno stress test sulle istanze del database cloud che devono essere ancora regolate, simulando carichi di lavoro o riproducendo i carichi di lavoro dell'utente. Allo stesso tempo, il metrics collector raccoglie ed elabora le metriche correlate. I dati elaborati vengono immagazzinati nel pool di memoria e alimentati rispettivamente nella rete neurale profonda basata sul reinforcement learning. Infine, il recommender produce le configurazioni dei parametri che saranno impiegate sul database cloud.
Di seguito, i singoli moduli sono analizzati più nel dettaglio:

- workload generator:
  - come descritto in precedenza, tale modulo svolge principalmente due compiti, ossia generare il carico di lavoro di test standard e riprodurre il carico di lavoro reale dell'utente corrente;
  - a causa della mancanza di campioni durante l'addestramento iniziale, si possono utilizzare strumenti standard di test del carico di lavoro come Sysbench;

  - dopo aver accumulato una certa quantità di dati di feedback dall'utente e aver raccomandato le configurazioni, si utilizza il meccanismo di riproduzione del generatore dei carichi di lavoro per raccogliere i record SQL dell'utente in un periodo di tempo e poi eseguirli nello stesso ambiente, in modo da ripristinare i dati sul comportamento reale dell'utente;
  - in questo modo, il modello è in grado di cogliere con maggiore precisione lo stato reale delle istanze del database dell'utente e di consigliare configurazioni migliori;

- metrics collector:
  - quando si esegue l'ottimizzazione di un database cloud in seguito a una richiesta di regolazione, vengono raccolti ed elaborati i dati delle metriche che possono catturare più aspetti del comportamento di runtime del database cloud in un determinato intervallo di tempo;
  - poichè le 63 metriche rappresentano lo stato corrente del database e vengono fornite al modello di reinforcement learning sotto forma di vettori, è necessario fornire istruzioni semplici per il loro utilizzo;
  - per quanto riguarda le metriche esterne, come la latenza e il throughput, si procede campionando ogni 5 secondi e calcolando la media dei risultati campionati per determinare la ricompensa, che rappresenta come cambieranno le prestazioni del database dopo aver applicato le impostazioni dei parametri di configurazione);

- recommender:
  - quando il modello di reinforcement learning emette le configurazioni consigliate, tale modulo genera i corrispondenti comandi di impostazione dei parametri e inoltra la richiesta di modifica delle configurazioni al controller;
  - dopo aver acquisito l'approvazione dell'amministratore del database o dell'utente, il controller implementa le suddette configurazioni sulle istanze del database cloud;

- pool di memoria:
  - tale modulo è impiegato per memorizzare i campioni di addestramento;
  - in generale, ciascun campione di esperienza contiene quattro categorie di informazioni: lo stato del database corrente (s_t), il valore di ricompensa calcolato attraverso la funzione di ricompensa tramite metriche esterne (r_t), i parametri del database da eseguire (a_t) e lo stato del database dopo l'esecuzione delle configurazioni dei parametri (s_{t+1});
  - un campione di esperienza può essere descritto come (s_t, a_t, r_t, s_{t+1}) ed è denominato transizione.

1.9.3 Implementazione del reinforcement learning all'interno di CDBTune

Nella figura 13 è mostrato il diagramma di interazione dei sei elementi chiave relativi al reinforcement learning e viene illustrata la corrispondenza tra questi sei elementi e l'ottimizzazione dei parametri di configurazione del database.

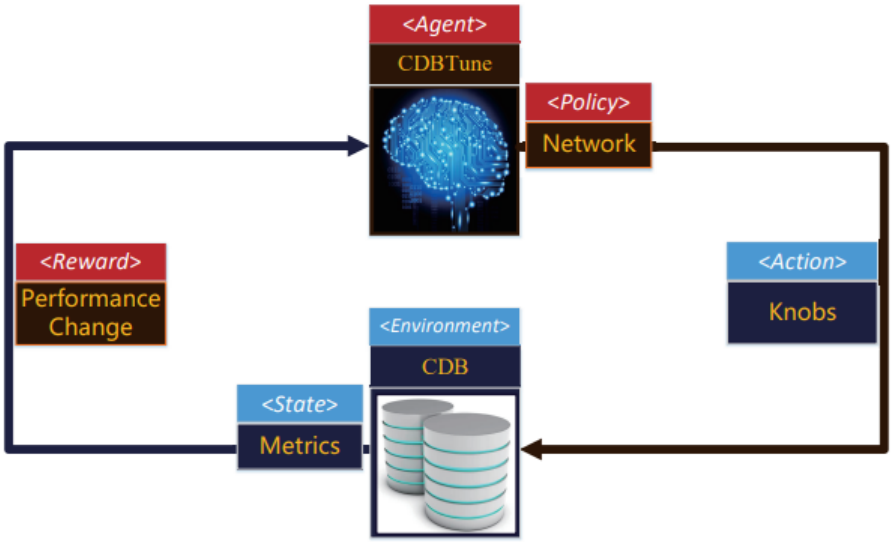


Fig. 13: Corrispondenza tra gli elementi chiave relativi al reinforcement learning e la regolazione dei parametri di configurazione di CDBTune.

Di seguito sono descritti tali elementi:

- agente: può essere visto come il sistema di ottimizzazione CDBTune, che riceve la ricompensa e lo stato dal database cloud e aggiorna la politica per guidare la regolazione dei parametri di configurazione per ottenere una ricompensa più alta (prestazioni più elevate);
- ambiente: rappresenta il target della regolazione, in particolare un'istanza del database cloud;
- stato: rappresenta lo stato attuale dell'agente, ovvero le 63 metriche;
- ricompensa: è uno scalare (descritto come r_t), che rappresenta il cambiamento delle prestazioni dopo e prima che il database cloud abbia eseguito le nuove configurazioni dei parametri consigliati da CDBTune;
- azione: proviene dallo spazio delle configurazioni dei parametri (spesso descritto come a_t) e corrisponde ad un'operazione di regolazione dei parametri;
- politica: definisce il comportamento di CDBTune in un determinato momento e in un determinato ambiente, che è una mappatura dallo stato all'azione; in altre parole, dato uno stato del database cloud, se viene richiamata un'azione (cioè la regolazione dei parametri), la politica mantiene lo stato successivo applicando l'azione allo stato originale.

Nel processo di ottimizzazione della configurazione di un DBMS mediante il reinforcement learning, l'ambiente da ottimizzare è rappresentato dal database cloud. Il modello di reinforcement learning implementato in CDBTune assume il ruolo di agente, il quale, analizzando lo stato attuale del database, propone nuove configurazioni al fine di migliorarne le prestazioni. Una volta applicate queste nuove configurazioni al database, le sue condizioni cambiano e si ottengono metriche che riflettono il suo comportamento. Queste metriche costituiscono il feedback, evidenziando l'efficacia delle nuove configurazioni. L'agente adatta le proprie raccomandazioni in base a questo feedback, cercando costantemente di suggerire impostazioni sempre più ottimali. Questo processo iterativo si ripete finché l'agente acquisisce competenza nel proporre configurazioni altamente adatte al database. Al termine del processo, si ottengono le impostazioni ottimali che massimizzano le prestazioni complessive del database.

1.9.4 Prestazioni di CDBTune

Per poter valutare l'efficienza di CDBTune si è reso necessario confrontarne le prestazioni con altri metodi di ottimizzazione, come OtterTune, BestConfig e quella manuale ad opera degli amministratori dei database.
Nella figura 14 è riportato un confronto tra i vari metodi di ottimizzazione in termini di velocità nel soddisfare una richiesta di regolazione dei parametri di configurazione.

Tuning Tools	Total Steps	Time of One Step (mins)	Total Time (mins)
CDBTune	5	5	25
OtterTune	5	11	55
BestConfig	50	5	250
DBA	1	516	516

Fig. 14: Confronto tra CDBTune ed altri metodi di ottimizzazione in termini di velocità nel soddisfare una richiesta di regolazione dei parametri di configurazione (in particolare, sono confrontati i passi di regolazione e il tempo impiegato per ciascun passo e in totale).

Come è possibile notare, per ogni richiesta di ottimizzazione, OtterTune impiega 55 minuti, BestConfig circa 250 minuti, gli amministratori dei database 8.6 ore, mentre CDBTune soltanto 25 minuti.
Nella figura 15 sono riportati i risultati di alcuni esperimenti condotti su un database cloud, attraverso i quali i metodi di ottimizzazione sono stati confrontati in termini di throughput e latenza, al variare del numero di passi di regolazione.

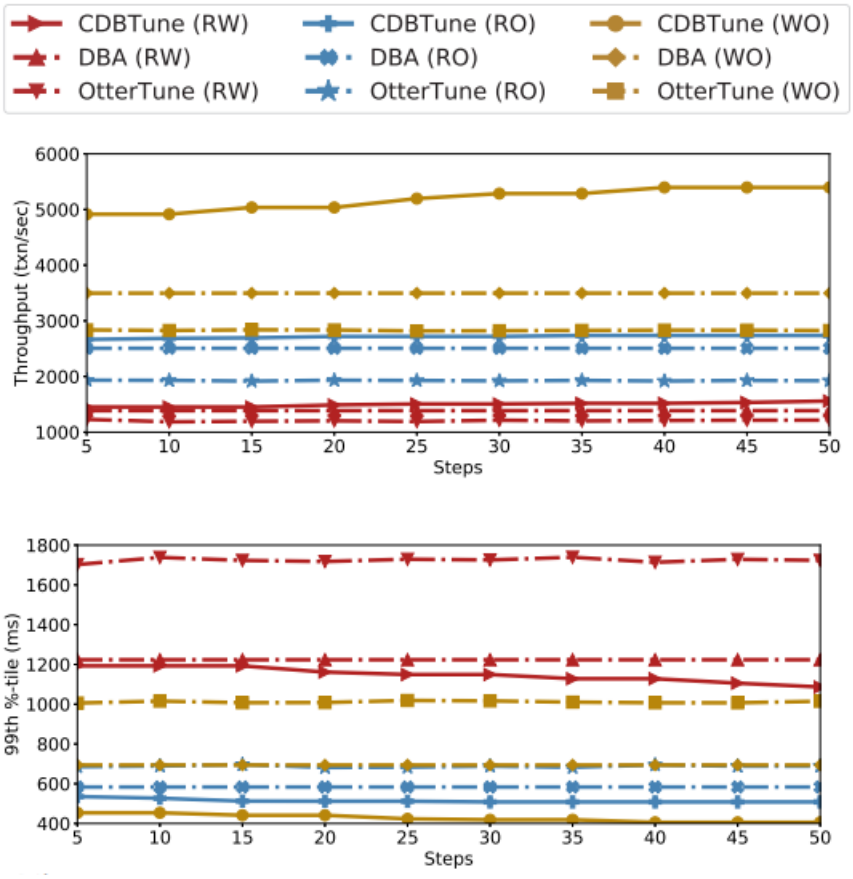


Fig. 15: Confronto tra le prestazioni di CDBTune e quelle di altri metodi di ottimizzazione all'aumentare del numero di passi di regolazione.

Come è possibile notare, confrontato con gli altri metodi di ottimizzazione, CDB-Tune raggiunge delle ottime prestazioni, anche in un numero limitato di passi di regolazione, dimostrandosi altamente efficiente.
Infine, nella figura 16 sono riportati dei risultati di ulteriori esperimenti condotti su un database cloud, attraverso i quali si dimostra il modo in cui le prestazioni di CDBTune aumentano all'aumentare del numero dei parametri regolati.

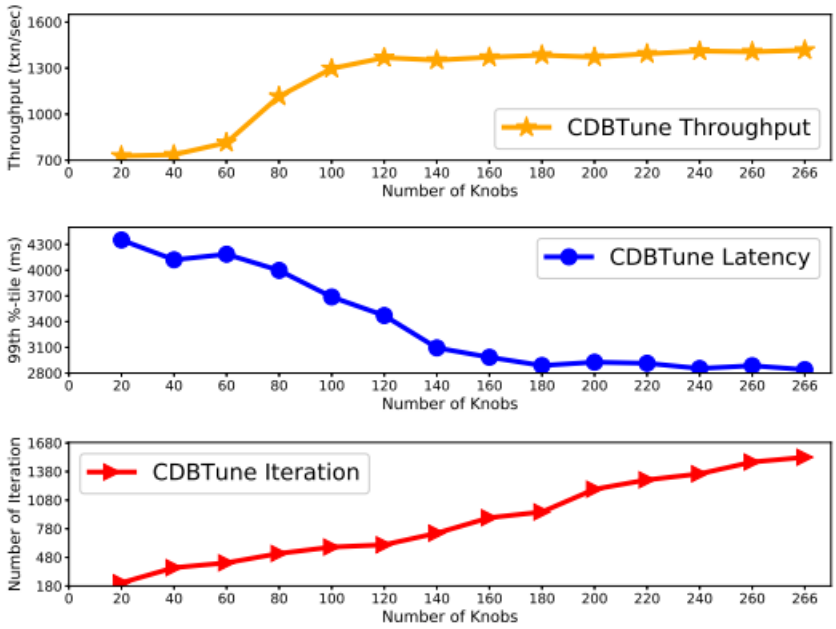


Fig. 16: Prestazioni di CDBTune all'aumentare del numero di parametri regolati.

Capitolo 2

Partizionamento automatico di database cloud

Le piattaforme cloud, come Amazon Web Services o Microsoft Azure, offrono come servizio diverse soluzioni DBMS scale-out pronte all'uso per carichi di lavoro di tipo OLAP. Utilizzando questi servizi, i clienti possono facilmente distribuire un database, definirne lo schema, caricare i dati e quindi interrogarlo attraverso un cluster di macchine. Mentre l'approvvigionamento è di solito completamente automatizzato, molte decisioni di progettazione che tradizionalmente venivano prese dall'amministratore del database rimangono uno sforzo manuale. Ad esempio, nel Data Warehouse di Azure, ma anche in Amazon Redshift, i clienti devono scegliere un attributo di partizionamento di una tabella per suddividere orizzontalmente tabelle di grandi dimensioni su più macchine. Sebbene sia un compito non banale, il partizionamento del database può migliorare notevolmente le prestazioni dei carichi di lavoro analitici, poiché le query SQL ad alta intensità di dati possono essere distribuite su più macchine.
Esiste già un ampio lavoro per automatizzare la progettazione fisica dei DBMS distribuiti, compreso il partizionamento dei dati. Questi consulenti formalizzano il problema come un problema di ottimizzazione e quindi si basano su modelli di costo per stimare il tempo di esecuzione delle query per diversi partizionamenti.
Tuttavia, questo approccio non è adatto ai fornitori di cloud, poiché consentono tipicamente ai clienti di distribuire le loro soluzioni DBMS su diverse piattaforme hardware, il che rende il problema dell'acquisizione di modelli di costo esatti una sfida a sé stante. In secondo luogo, anche se il modello di costo viene messo a punto per una determinata piattaforma hardware, le stime dei costi dell'ottimizzatore sono spesso notoriamente imprecise, il che porta a progetti di partizionamento non ottimali se si utilizzano gli approcci di progettazione automatizzati esistenti.

Un'alternativa consiste nell'utilizzare il deep reinforcement learning per realizzare un consulente per il partizionamento del cloud come servizio. Poiché un agente impara per tentativi ed errori, il deep reinforcement learning ha il vantaggio di non basarsi sul fatto che sia disponibile un modello di costo accurato. Piuttosto, adottando diversi partizionamenti e osservando i tempi di esecuzione delle query, si apprendono i trade-off per i vari carichi di lavoro. Una volta addestrato, il consulente può essere interrogato per ottenere un partizionamento del carico di lavoro osservato.

2.1 Un approccio di partizionamento di database cloud attraverso il reinforcement learning

L'idea di base è quella di addestrare un agente di reinforcement learning per ogni utente del database cloud, il quale apprende i vari trade-off nell'uso di diversi partizionamenti per un dato schema di database e per diversi carichi di lavoro. L'apprendimento di questi trade-off è interessante perché i modelli di costo sono

notoriamente imprecisi e quindi sovrastimerebbero o sottostimerebbero i vantaggi di certi partizionamenti. Una panoramica di tale approccio è mostrata nella figura 17.

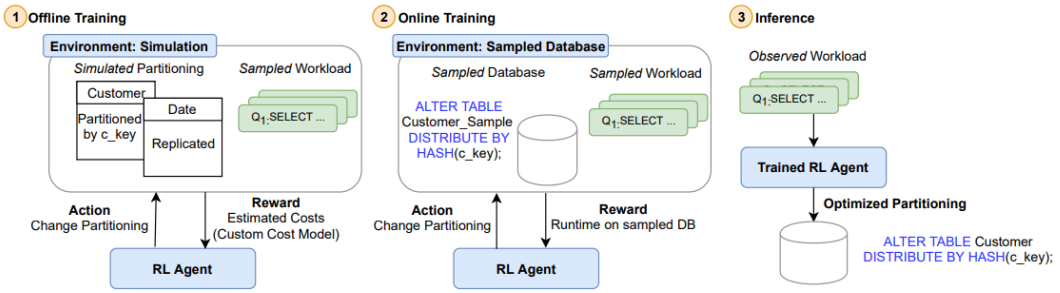


Fig. 17: Panoramica dell'approccio di partizionamento di database cloud attraverso il reinforcement learning.

Per utilizzare tale approccio, l'utente deve solo fornire il DBMS (schema e dati) e un carico di lavoro campione che rifletta l'insieme delle query tipiche di un carico di lavoro di produzione. Sulla base di queste informazioni, un agente di deep reinforcement learning è allenato in due fasi, la prima offline e la seconda online. In seguito all'addestramento, l'agente di deep reinforcement learning può essere usato nel DBMS di produzione per decidere quali partizionamenti distribuire monitorando il carico di lavoro effettivo. Le variazioni del carico di lavoro possono indurre l'agente a suggerire nuovi partizionamenti più adatti.

2.1.1 Addestramento del modello di reinforcement learning

In generale, gli agenti di deep reinforcement learning imparano interagendo con un ambiente, scegliendo delle azioni e osservando le ricompense che cercano di massimizzare. Nella configurazione in esame, l'ambiente è il DBMS, che l'agente manipola con azioni che modificano il partizionamento delle singole tabelle. Durante la fase di addestramento, l'agente impara a minimizzare il tempo di esecuzione di un determinato carico di lavoro costituito da un misto di query rappresentative. Quindi, l'agente impara gli effetti dei diversi partizionamenti sulle latenze delle singole query.
Più dettagliatamente, nella fase di addestramento offline, l'agente interagisce esclusivamente con una "simulazione" del database dell'utente. Poiché la rete è tipicamente il collo di bottiglia dei join distribuiti, si è sviluppato un modello di costo semplice ma generico, incentrato sull'overhead di rete necessario per rispondere a una query dato un certo partizionamento. In combinazione con i metadati (schema e dimensioni delle tabelle) riguardanti il database dell'utente, si possono stimare i costi delle query in base a un partizionamento nella simulazione. Queste stime vengono utilizzate come ricompense per l'agente. Sebbene non sia preciso, questo metodo consente all'agente di trovare già un partizionamento ragionevole con un carico di lavoro di produzione.
In una fase di addestramento online opzionale, l'agente non interagisce solo con una simulazione, ma con un vero database. Tuttavia, invece di utilizzare l'intero

database, si utilizza solo un campione dei dati per accelerare questa fase dell'allenamento. Il vantaggio di questa fase è che non dipende più dall'accuratezza del semplice modello di costo incentrato sulla rete, consentendo di misurare i tempi di esecuzione delle query sul database campionato per calcolare le ricompense dell'agente. Di conseguenza, l'agente apprende con maggiore precisione gli effetti dei partizionamenti.
Una volta completato l'addestramento, l'agente può essere utilizzato per prendere le decisioni riguardanti i partizionamenti effettivi. Come input, richiede un carico di lavoro, cioè quali query sono state inviate in una certa finestra temporale. Sulla base di questo carico di lavoro, l'agente suggerisce dei partizionamenti che vengono distribuiti sul database reale dell'utente. In molti casi, l'agente può essere utilizzato per suggerire un partizionamento ottimizzato senza un ulteriore addestramento, il quale però diventa necessario nel caso in cui lo schema del database cambi o si verifichino classi di query completamente nuove in un carico di lavoro.

2.1.2 Valutazione sperimentale: impostazione degli esperimenti

Per poter valutare i vantaggi dell'utilizzo di consulenti di partizionamento appresi per database con schemi di varia complessità, sono stati condotti diversi esperimenti. In particolare, sono analizzati i seguenti aspetti dell'approccio considerato:

- prestazioni in seguito all'addestramento offline;
- miglioramento dovuto all'addestramento online;
- adattabilità ai dati e ai carichi di lavoro;
- confronto con altri approcci.

Il consulente di partizionamento è stato valutato su tre schemi di database e carichi di lavoro diversi per complessità:

- come caso più semplice, si è utilizzato lo Star Schema Benchmark (SSB), il quale si basa su TPC-H e riorganizza il database in uno schema a stella puro con 5 tabelle (1 tabella di fatti e 4 tabelle di dimensioni) e 13 query, oltre al relativo carico di lavoro;
- come secondo database si è utilizzato TPC-DS, il quale ha uno schema molto più complesso di 24 tabelle (7 tabelle di fatti e 17 tabelle di dimensioni) e 99 query, oltre al relativo carico di lavoro;
- per verificare la capacità del consulente di gestire schemi più complessi non basati su uno schema a stella, come terzo database si è utilizzato il benchmark TPC-CH, il quale è la combinazione dello schema del benchmark TPC-C con query analitiche dello schema TPC-H.

Per tutti i benchmark si è utilizzato un fattore di scala pari a 100.

I partizionamenti per i diversi schemi analitici sono stati valutati su due sistemi di

database. Per dimostrare che l'approccio è in generale applicabile sia ai database distribuiti su disco che a quelli in memoria, si è utilizzato PostgresXL 10R1.1 (un popolare database distribuito open-source su disco) e System-X (un database distribuito commerciale in-memory). Per l'esecuzione dei database in una configurazione distribuita, si è utilizzato CloudLab, un'infrastruttura scientifica per la ricerca sul cloud computing. Per gli esperimenti, sono stati utilizzati cluster di dimensioni diverse, da 4 a 6 nodi. Ogni nodo è stato configurato per utilizzare 128 GB di memoria principale DDR4, due CPU Intel Xeon Silver 4114 a 10 core e un'interconnessione a 10 Gbps. Il consulente di partizionamento è costruito utilizzando reti neurali implementate in Keras. In particolare, la rete neurale per l'approssimazione delle funzioni Q ha utilizzato 2 strati nascosti con 128 e 64 neuroni, rispettivamente. Si è utilizzata la funzione di attivazione standard ReLU in ogni strato e una funzione lineare per l'uscita (per rappresentare il valore Q), una combinazione comune per il deep reinforcement learning. Nella figura 18 è riportata una panoramica di tutti i migliori iperparametri identificati per l'addestramento. L'unico iperparametro modificato per i diversi database è stata la quantità di episodi utilizzati per addestrare il modello. Poiché SSB ha una quantità di tabelle e query significativamente inferiore, gli agenti di deep reinforcement learning sono stati addestrati solo per 600 episodi invece che per 1200 episodi per TPC-DS e TPC-CH.

Parameter	**Value**
Learning Rate	$5 \cdot 10^{-4}$
τ (Target network update)	10^{-3}
Optimizer	Adam
Experience Replay Buffer Size	10000
Batch Size for Experience Replay	32
Epsilon Decay	0.997
t_{max} (Max Stepsize)	100
Episodes	600/1200
Network Layout	128-64
γ (Reward Discount)	0.99

Fig. 18: Iperparametri utilizzati per l'addestramento dell'agente di deep reinforcement learning.

2.1.3 Valutazione sperimentale: prestazioni in seguito all'addestramento offline

Per ogni database menzionato in precedenza, si è addestrato un agente di deep reinforcement learning dedicato attraverso un apprendimento offline, cioè utilizzando il semplice modello di costi incentrato sulla rete. Nella figura 19 è riportata la media del tempo di esecuzione totale di tutte le query per cinque esecuzioni per i partizionamenti suggeriti dal consulente in esame e per le baseline.

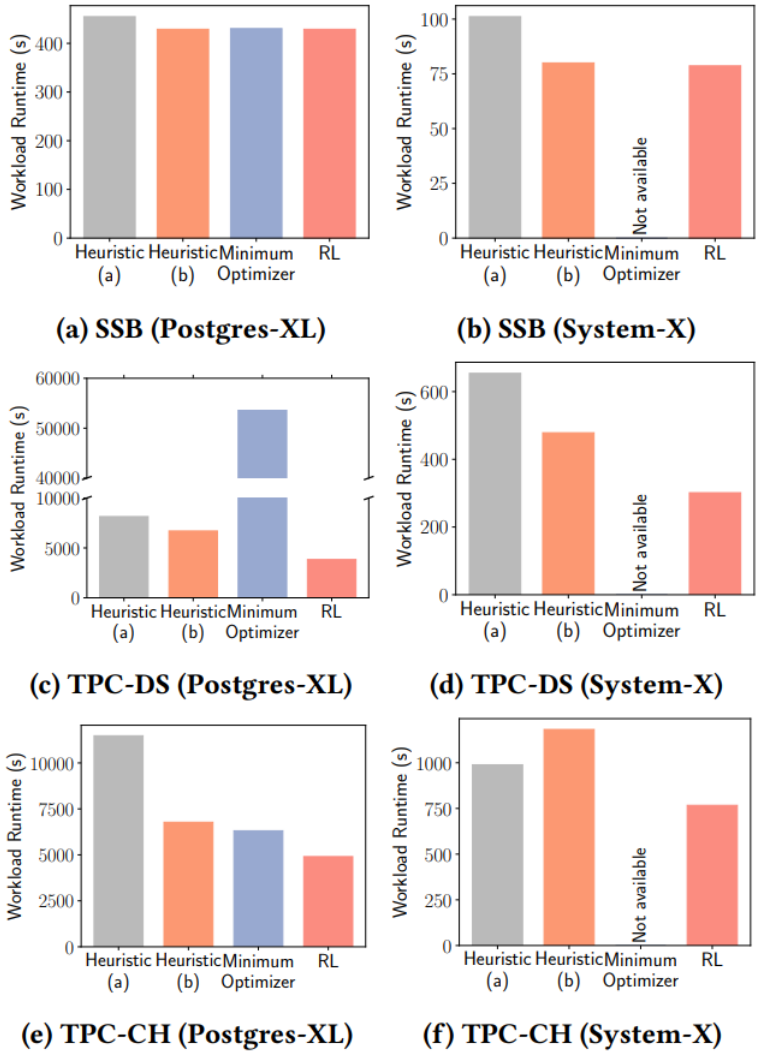


Fig. 19: Confronto tra il reinforcement learning (tramite l'addestramento offline) e le baseline.

Per il benchmark SSB, il consulente suggerisce di partizionare la tabella dei fatti con la tabella delle dimensioni più grande per Postgres-XL (esattamente come la seconda euristica). Per System-X, il consulente suggerisce inoltre di partizionare la tabella delle dimensioni `Part` in base alla sua chiave primaria, con un miglioramento minimo dei tempi di esecuzione.
Per TPC-DS, che è uno schema più complesso composto da diverse tabelle di fatti con dimensioni condivise, gli agenti di deep reinforcement learning trovano soluzioni superiori non ovvie. In questo caso, i miglioramenti sono più significativi, riducendo il tempo di esecuzione rispetto alla prima euristica di circa il 50%. Sia per Postgres-XL che per System-X, gli agenti di deep reinforcement learning propongono di partizionare le tabelle dei fatti con una tabella di dimensioni medie, ad esempio `Item`. D'altra parte, il partizionamento con i costi minimi dell'ottimizzatore per PostgresXL porta a un partizionamento non ottimale. Ciò è dovuto all'elevata complessità delle query che porta a stime errate dei costi.
Per TPC-CH, che utilizza uno schema significativamente più complesso rispetto

a SSB e TPC-DS, poiché non simile a uno schema a stella, l'agente di deep reinforcement learning, rispetto alle due euristiche, propone partizioni migliori. In particolare, adotta due soluzioni differenti tra Postgres-XL e System-X.

2.1.4 Valutazione sperimentale: miglioramento dovuto all'addestramento online

Un ulteriore esperimento è stato condotto per valutare se gli agenti di deep reinforcement learning addestrati online sono superiori a quelli addestrati esclusivamente offline. In questo caso, l'esperimento è stato incentrato sullo schema più complesso, ossia TPC-CH, e Postgres-XL per analizzare la fase online aggiuntiva che sfrutta i tempi di esecuzione effettivi invece delle stime dei costi. Per l'addestramento online si è perfezionato l'agente di deep reinforcement learning già precedentemente avviato con il semplice modello di costi incentrato sulla rete offline.
I tempi di esecuzione delle query di benchmark utilizzando i partizionamenti suggeriti sull'intero database TPC-CH sono mostrati nella figura 20.

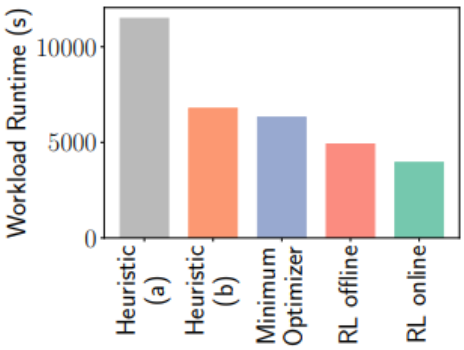


Fig. 20: Confronto tra il reinforcement learning (tramite l'addestramento online) e le baseline.

Il partizionamento suggerito dall'agente addestrato online è superiore del 20% a quello dell'agente addestrato offline, oltre ad essere completamente differente.
Se eseguita in modo ingenuo, la fase di addestramento online richiede molto tempo, rendendo quindi necessarie delle ottimizzazioni. Per questo esperimento, è stato eseguito l'addestramento solo con tutte le ottimizzazioni (tranne i timeout) attivate. Tenendo traccia delle query che verrebbero eseguite due volte senza Runtime Caching, nonché della frequenza con cui una tabella verrebbe ripartita senza Lazy Repartitioning e di quanto tempo si potrebbe risparmiare con un particolare Timeout, è stato possibile determinare il risparmio dovuto alle ottimizzazioni. Come si può vedere nella figura 21, ogni ottimizzazione riduce significativamente il tempo di esecuzione e il miglioramento maggiore si ottiene con il Lazy Repartitioning.

Optimizations	Training Time	Speedup
None	4621h	-
+ Runtime Cache	1160.4h	4.0
+ Lazy Repartitioning	60h	19.3
+ Timeouts	33.4h	1.8
+ Offline Phase	13.3h	2.5

Fig. 21: Riduzione del tempo di addestramento dovuto alle ottimizzazioni.

L'ultima ottimizzazione confronta il tempo di addestramento di un agente che è stato avviato in una fase offline con un agente inizializzato in modo casuale.

2.1.5 Valutazione sperimentale: adattabilità ai dati e ai carichi di lavoro

Ulteriori esperimenti sono stati condotti per dimostrare l'adattabilità di un agente di deep reinforcement learning a dati e carichi di lavoro mutevoli.
Nel primo esperimento si è valutato quanto l'agente addestrato fosse robusto in caso di variazioni dei dati. In particolare, si è utilizzato lo schema TPC-CH e si è addestrato il consulente sull'intero database. Successivamente, si è aggiornato il DBMS, caricando fino al 60% di nuovi dati nello schema TPC-CH. Nella figura 22 sono mostrati i risultati ottenuti utilizzando il consulente addestrato online, confrontati con quelli relativi a tutte le altre baseline.

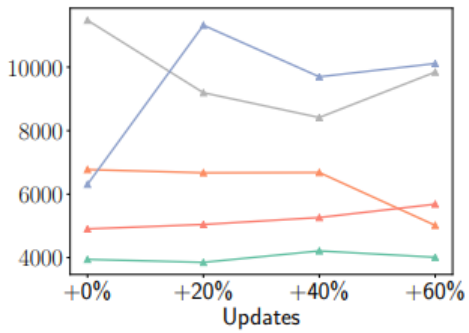


Fig. 22: Confronto tra il reinforcement learning (rappresentato in verde e in rosso, a seconda del tipo di addestramento) e le baseline, al variare dei dati.

Come si può notare, il partizionamento individuato dal consulente è sempre il migliore anche per tassi di aggiornamento relativamente elevati, fino al 60%. Tuttavia, se il database cambia in modo significativo, è necessario riqualificare il consulente.
Nel secondo esperimento si è valutato invece quanto il consulente addestrato sia capace di individuare partizionamenti ottimali per diversi insiemi di query. A tal fine, si è addestrato un agente di deep reinforcement learning con l'approccio naïve per diverse frequenze di carico di lavoro per lo schema TPC-CH. Inoltre, è stato addestrato anche un comitato di esperti del sottospazio. Dopo l'addestramento

di entrambi gli approcci, nella figura 23 è riportata la percentuale di partizioni corrette per due diversi cluster di carichi di lavoro.

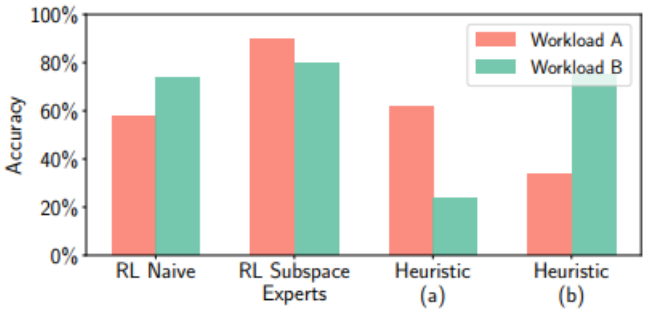


Fig. 23: Miglior partizionamento individuato da diversi approcci per carichi di lavoro diversi (più alto è il valore meglio è).

Come si può notare, l'accuratezza può essere significativamente migliorata quando si utilizzano esperti del sottospazio, superando tutti gli altri approcci. Quindi, è vantaggioso dividere il problema di individuazione del partizionamento ottimale per un dato carico di lavoro in sottoproblemi che vengono poi risolti dal modello esperto dedicato. Ciò è dovuto alla nota tecnica di utilizzo di insiemi di modelli di machine learning per migliorare le prestazioni.

Per come è definito il consulente, se si presentano query completamente nuove che hanno un impatto significativo sul tempo di esecuzione del carico di lavoro e non hanno una query simile nell'insieme di query rappresentative, è necessario un addestramento incrementale, il che è significativamente più veloce dell'addestramento di un nuovo agente da zero. Nell'ultimo esperimento si valuta quindi l'overhead di addestramento aggiuntivo se vengono introdotte queste nuove query, dimostrando che è decisamente più economico rispetto all'addestramento da zero.
Nella figura 24 è mostrato il tempo di addestramento incrementale rispetto al tempo necessario per addestrare un agente di deep reinforcement learning da zero, a seconda del numero di query ulteriori aggiunte in seguito all'addestramento iniziale.

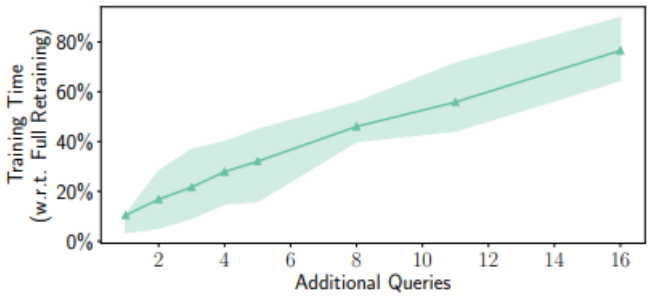


Fig. 24: Tempo di addestramento aggiuntivo (rispetto al riaddestramento completo) assieme ai quantili (25% e 75%).

Come si può notare, l'overhead dell'addestramento incrementale è molto più basso rispetto all'addestramento di un consulente di partizionamento da zero.

Capitolo 3

Ottimizzazione automatica delle query

3.1 Introduzione all'ottimizzazione delle query

L'ottimizzazione delle query è una caratteristica comune nei sistemi di gestione di database relazionali, NoSQL e a grafo. Il suo obiettivo è determinare il modo più efficiente per eseguire una determinata query, considerando i vari piani di esecuzione possibili.
In genere, l'ottimizzatore di query non è accessibile direttamente agli utenti: una volta che le query vengono inviate al server del database e analizzate dal parser, vengono poi passate all'ottimizzatore di query, dove avviene l'ottimizzazione.
Una query, che può essere semplice o complessa, rappresenta una richiesta di informazioni da un database. Il risultato di una query viene generato elaborando le righe di un database in modo da ottenere le informazioni richieste. Poiché le strutture dei database sono complesse, nella maggior parte dei casi, e soprattutto per le interrogazioni non molto semplici, i dati necessari per un'interrogazione possono essere raccolti da un database accedendovi in modi diversi, attraverso strutture di dati differenti e in ordini distinti. Ogni metodo richiede in genere tempi di elaborazione diversi, i quali, per una stessa interrogazione, possono variare notevolmente, da una frazione di secondo a ore, a seconda del metodo scelto. Lo scopo dell'ottimizzazione delle query, che è un processo automatizzato, è quello di trovare il modo di elaborare una determinata query nel minor tempo possibile. Sebbene trovare il piano di query ottimale tra tutte le possibilità sia spesso impraticabile, l'ottimizzazione cerca di approssimare il miglior risultato confrontando diverse alternative ragionevoli e fornendo, in un tempo accettabile, un piano "sufficientemente buono" che si discosta poco dal risultato ottimale.

Nell'ambito dei DBMS basati su SQL, per poter eseguire le query, il motore di database deve analizzare le istruzioni per determinare il modo più efficiente di accedere ai dati necessari ed elaborarli. Questa analisi è gestita da un componente denominato Query Optimizer. I dati di input per il Query Optimezer includono la query stessa, lo schema del database (definizioni delle tabelle e degli indici) e le statistiche del database. Il Query Optimizer elabora uno o più piani di esecuzione di query, talvolta definiti piani di query o piani di esecuzione. La scelta del piano di query da parte del Query Optimizer avviene utilizzando un set di euristiche per bilanciare il tempo di compilazione e ottimizzare la ricerca di un piano di query valido.
Un piano di esecuzione di query è la definizione di:

- la sequenza di accesso alle tabelle di origine:
  - in genere, il server del database può utilizzare molte sequenze diverse per accedere alle tabelle di base e quindi compilare il set di risultati;
  - ad esempio, se un'istruzione `SELECT` fa riferimento a tre tabelle, il server del database può accedere prima alla tabella A, utilizzare i dati della

tabella A per estrarre le righe corrispondenti dalla tabella B e quindi utilizzare i dati della tabella B per estrarre i dati dalla tabella C;

  - altre sequenze possibili di accesso alle tabelle utilizzabili dal server del database sono le seguenti: C, B, A o B, A, C o B, C, A o C, A, B;

- i metodi utilizzati per estrarre i dati da ogni tabella:

  - per accedere ai dati di ogni tabella sono in genere disponibili metodi diversi, come l'utilizzo di indici per righe specifiche o l'analisi completa della tabella;
  - la scelta dipende dalla necessità di dati specifici e dalle condizioni della query;

- i metodi utilizzati per eseguire calcoli e le modalità per filtrare, aggregare e ordinare i dati da ogni tabella:

  - diversi metodi possono essere impiegati per eseguire calcoli sui dati, aggregare e ordinare i risultati in base alle condizioni specificate nella query;
  - ad esempio, l'utilizzo di clausole come `GROUP BY` o `ORDER BY` influenza il modo in cui i dati sono elaborati e restituiti, mentre le clausole come `WHERE` o `HAVING` possono essere utilizzate per filtrare i dati in base a criteri specifici.

Il processo di selezione di un piano di esecuzione è noto come ottimizzazione. Il Query Optimizer rappresenta uno dei componenti cruciali del motore di database. L'overhead derivante dall'utilizzo di Query Optimizer per l'analisi della query e la selezione di un piano è ampiamente bilanciato dall'efficienza del piano di esecuzione scelto. Un analogo può essere tracciato considerando la costruzione di una casa affidata a due imprese edili diverse: se un'impresa dedica tempo alla pianificazione, mentre l'altra inizia immediatamente la costruzione senza alcuna pianificazione, è probabile che l'impresa che ha pianificato completi il progetto per prima.
L'ottimizzazione delle query rappresenta un compromesso tra il tempo impiegato per individuare il miglior piano di query e la qualità della scelta, che l'ottimizzatore potrebbe non effettuare autonomamente. Gli ottimizzatori basati sui costi valutano l'utilizzo di risorse dei piani di query per selezionare il piano con il costo minore. Assegnano un "costo" stimato a ciascun piano, considerando vari fattori come operazioni di I/O, lunghezza del percorso della CPU, spazio buffer su disco, tempo di servizio di archiviazione, e interconnessione tra unità di parallelismo. La scelta del piano è complessa, data la vastità dello spazio di ricerca, che include diversi percorsi di accesso e tecniche di join. Esistono due tipi di ottimizzazione: logica, che genera sequenze di algebre relazionali, e fisica, che determina i mezzi per eseguire le operazioni.

3.2 Meccanismo di funzionamento alla base dell'ottimizzazione delle query

La responsabilità principale dell'ottimizzatore di query consiste nel convertire una query SQL dichiarativa in un programma eseguibile. Nello specifico, quando si utilizza un linguaggio dichiarativo come SQL, che specifica cosa fare senza dettagliare come deve essere calcolato, l'ottimizzatore svolge il compito di trasformare il concetto del "cosa" in un piano esecutivo concreto, ovvero il "come". Ciò implica la generazione di un programma eseguibile, il cui codice può essere interpretato ed eseguito dal sistema di gestione di database.
Tale processo si articola in diverse fasi, come mostrato nella figura 25.

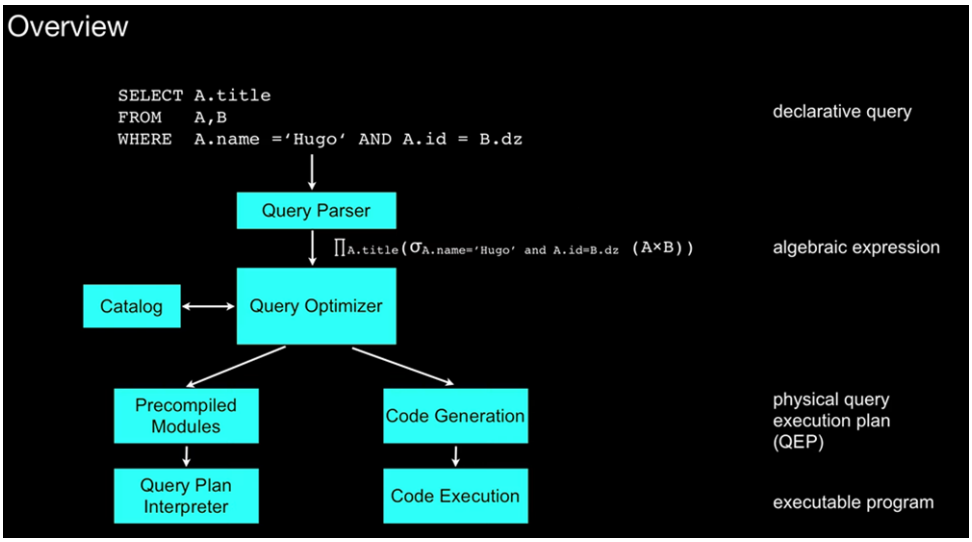


Fig. 25: Panoramica sulle fasi in cui è articolata l'ottimizzazione delle query.

Inizialmente, il "parser della query" (Query Parser) converte la query in un formato intermedio, generalmente un'espressione algebrica relazionale più complessa rispetto a un'ordinaria algebra relazionale, arricchita con proprietà specifiche. Durante questo processo, si presta particolare attenzione a non perdere informazioni cruciali provenienti dalla query SQL, annotandole durante il parsing.
L'ottimizzatore di query si basa sulle informazioni disponibili nel catalogo del database, contenente tutti i metadati, come i nomi delle tabelle, gli schemi e le dimensioni dei diversi tipi di dati utilizzati nei domini. Inoltre, il catalogo fornisce statistiche vitali per ottimizzare le query, come il numero di tuple in una specifica riga e la distribuzione dei dati in una particolare colonna.
Ci sono fondamentalmente due percorsi che l'ottimizzatore di query può seguire. Il primo, rappresentato sulla sinistra dello schema, fa affidamento su moduli precompilati, che sono piccoli frammenti di codice simili a chiamate di funzione in una libreria, e che possono essere sfruttati per generare un programma. Questi moduli sono anche chiamati operatori nel mondo dei database. Seguendo questo percorso, si prendono alcuni di questi operatori e li si assembla in un albero, dove ogni nodo rappresenta un operatore. Quello che si va a generare con questo procedimento è

chiamato piano di query (Query Plan). In sintesi, il sistema del database, essenzialmente, chiama funzioni ogni volta che esegue un'azione o passa dati in questo albero. Questo procedimento è detto di interpretazione del piano di query.
Il secondo approccio, rappresentato sulla destra dello schema, è una metodologia più tradizionale utilizzata nei database commerciali fin dall'inizio. L'idea di base è quella di generare effettivamente del codice, tramite il blocco Code Generation. Questo codice potrebbe essere in C++, un codice intermedio, istruzioni assembly o codice binario, a seconda delle preferenze. In sostanza, in questo caso si genera il codice e non si esegue, come nel caso precedente, il tipo di interpretazione basata su moduli precompilati. Successivamente si esegue il codice (Code Execution).
In generale, l'operazione del Query Optimizer appare simile a quella di un compilatore standard per linguaggi di programmazione. In questo contesto, si tratta di tradurre un linguaggio specifico, spesso non dichiarativo e fortemente legato a domini particolari, come SQL, in codice eseguibile. Tale reimplementazione è dovuta alla possibilità di sfruttare ottimizzazioni specifiche del dominio, impraticabili in un compilatore di linguaggi di programmazione generici. Quindi, il processo si adatta a informazioni e ottimizzazioni pertinenti al contesto dei database, ma che potrebbero risultare meno efficaci o non applicabili in un contesto più ampio di programmazione. Tuttavia, è importante essere consapevoli ed evitare la reimplementazione di funzionalità disponibili nei compilatori standard dei linguaggi di programmazione. Ad esempio, potrebbe essere generato del codice in C++ dal modulo Code Generation (o persino in assembly). Ciò implica che il codice C++ si integra in un compilatore, il quale svolge tutte le operazioni necessarie. Questo approccio è una pratica consigliata per delegare la responsabilità e rappresenta, in sostanza, un'applicazione che si sovrappone ai compilatori di linguaggi di programmazione esistenti, traducendo il linguaggio specifico del dominio in un programma in C++.

3.3 Problematiche nell'ottimizzazione automatica delle query

L'ottimizzazione delle query implica la trasformazione di una query SQL dichiarativa emessa dall'utente in un piano di esecuzione. Nonostante decenni di studi, l'ottimizzazione delle query rimane un problema irrisolto. Diversi lavori hanno applicato tecniche di machine learning all'ottimizzazione delle query, spesso mostrando risultati notevoli. Tuttavia, nessuna di queste tecniche è ancora pratica, in quanto presentano diverse problematiche fondamentali:

- tempi di addestramento lunghi: la maggior parte delle tecniche di machine learning proposte richiede una quantità impraticabile di dati di addestramento prima di avere un impatto positivo sulle prestazioni delle query;
- gestione del cambiamento: la maggior parte delle tecniche di machine learning proposte richiede un riaddestramento quando il carico di lavoro o lo schema cambiano;
- le tecniche di apprendimento possono superare gli ottimizzatori tradizionali in media, ma spesso si comportano in modo catastrofico nella coda;

- se gli ottimizzatori tradizionali basati sui costi sono già complessi, la comprensione dell'ottimizzazione delle query è ancora più difficile quando si utilizzano approcci black-box di deep learning;

- costi di integrazione: i precedenti ottimizzatori appresi sono ancora prototipi di ricerca, che offrono poca o nessuna integrazione con un DBMS reale, e dunque nessuno di essi supporta tutte le funzionalità standard di SQL.

3.4 Bao

Bao è il primo ottimizzatore in grado di superare le problematiche precedentemente citate, tanto da essere completamente integrato in PostgreSQL come estensione ed essere utilizzato anche esclusivamente per query specifiche.
L'idea alla base di Bao è quella di evitare di addestrare un ottimizzatore da zero. Piuttosto, si utilizza un ottimizzatore già esistente (ad esempio quello di PostgreSQL), apprendendo quando attivare o disattivare alcune delle sue funzionalità query per query. In altri termini, Bao è un componente che si aggiunge a un ottimizzatore di query esistente per migliorarne il processo di ottimizzazione, piuttosto che sostituirlo o eliminarlo del tutto.
Ad un livello più alto, Bao cerca di "correggere" un ottimizzatore di query tradizionale imparando una mappatura tra una query in arrivo e la strategia di esecuzione che l'ottimizzatore di query dovrebbe utilizzare per quella query. Queste correzioni, definite come set di suggerimenti per le query, consentono a Bao di limitare e indirizzare lo spazio di ricerca dell'ottimizzatore tradizionale.
Il funzionamento schematico di Bao può essere riassunto nei seguenti passi:

- Bao apprende un modello che prevede quali suggerimenti porteranno a buone prestazioni per una particolare query;

- quando arriva una query, il sistema seleziona un set di suggerimenti, esegue il piano di query risultante e osserva una ricompensa;

- nel corso del tempo, Bao raffina il suo modello per predire più accuratamente quale set di suggerimenti sarà più vantaggioso per una data query in arrivo.

Bao sfrutta il campionamento di Thompson, un algoritmo ben studiato ed efficiente dal punto di vista del campionamento, e, poiché utilizza un ottimizzatore di query sottostante, può adattarsi a nuovi dati e a modifiche dello schema, mantenendo una buona efficienza. Rispetto ad altri metodi di apprendimento per l'ottimizzazione di query, che devono riapprendere ciò che gli ottimizzatori tradizionali già conoscono, Bao inizia immediatamente a migliorare l'ottimizzatore sottostante, riducendo la latenza della coda anche rispetto agli ottimizzatori tradizionali. Oltre a risolvere le problematiche pratiche dei sistemi di apprendimento precedenti, Bao offre diverse caratteristiche desiderabili che erano mancanti o difficili da ottenere nei precedenti ottimizzatori tradizionali e appresi. Tra le seguenti si elencano:

- tempo di addestramento ridotto: a differenza degli altri approcci di deep learning, che possono richiedere giorni di addestramento, Bao può superare

gli ottimizzatori di query tradizionali con un tempo di addestramento molto inferiore (circa un'ora);

- robustezza alle modifiche dello schema, dei dati e al carico di lavoro;
- migliore latenza di coda;
- interpretabilità e facilità di debugging;
- bassi costi di integrazione;
- estensibilità.

Ad ogni modo, Bao ha anche degli aspetti negativi, come il fatto che sia adatto principalmente a carichi di lavoro dominati dalla coda o che contengono molte query di lunga durata di esecuzione. Perciò, è suggerita la sua disattivazione per query brevi.

3.4.1 Architettura di Bao

Ad alto livello, Bao combina un modello di convoluzione ad albero, un operatore di rete neurale in grado di riconoscere modelli importanti negli alberi dei piani di query, con il campionamento di Thompson, una tecnica per risolvere problemi contestuali di bandito a più braccia. L'architettura di Bao è mostrata nella figura 26.

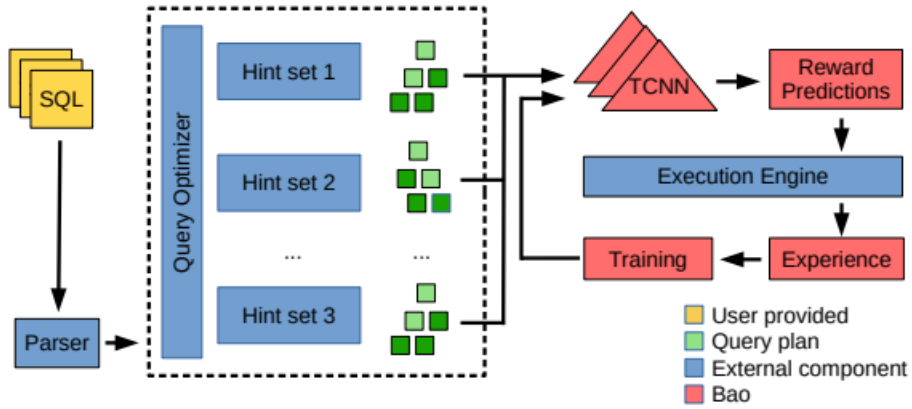


Fig. 26: Architettura di Bao.

Quando un utente invia una query, Bao utilizza l'ottimizzatore di query sottostante per produrre n piani di query, uno per ogni insieme di suggerimenti. Mentre alcuni suggerimenti possono essere applicati a una singola relazione o a un predicato, Bao si concentra solo sui suggerimenti di query che fungono da flag booleani (ad esempio, disabilitazione del loop join o forzamento dell'uso degli indici). Gli insiemi di suggerimenti disponibili per Bao devono essere specificati in anticipo. Può accadere che un insieme di suggerimenti possa essere vuoto, caso in cui si deve utilizzare l'ottimizzatore originale senza alcuna restrizione.

In seguito, ogni piano di query viene rappresentato come un albero vettoriale, ossia un albero in cui ogni nodo costituisce un vettore di feature. Questi alberi vettoriali vengono quindi elaborati dal modello di Bao, cioè dalla rete neurale convoluzionale ad albero (TCNN), che predice la qualità di ciascun piano, come ad esempio il corrispettivo tempo di esecuzione. Per accelerare il processo di ottimizzazione, i singoli piani di query possono essere generati e valutati parallelamente.
Dopo che un piano è stato selezionato, viene inviato a un motore di esecuzione delle query. Una volta eseguita la query, la combinazione del piano di query selezionato e delle prestazioni osservate viene aggiunta all'esperienza di Bao. Periodicamente, questa esperienza viene utilizzata per riqualificare il modello predittivo, creando un ciclo di feedback. Di conseguenza, il modello predittivo migliora e Bao sceglie in modo più affidabile il miglior set di suggerimenti per ogni query. Per i carichi di lavoro che non possono permettersi una regressione delle query, questa esplorazione può essere eseguita anche offline.

La convoluzione ad albero consiste nel far scorrere dei "filtri" a forma di albero su un piano di query (simile alla convoluzione di immagini, in cui i filtri sono convolti con un'immagine) per produrre un albero trasformato della stessa dimensione. Questi filtri possono individuare schemi come coppie di join di hash o scansioni di indici su una piccola relazione. Gli operatori di convoluzione ad albero sono impilati in diversi livelli, con strati successivi in grado di imparare a riconoscere schemi più complessi, come una lunga catena di merge join o un albero fitto di operatori hash. La convoluzione ad albero sfrutta la sua capacità intrinseca di rappresentare e apprendere tali modelli, costituendo un utile bias induttivo per l'ottimizzazione delle query: in altre parole, la struttura della rete, e non solo i suoi parametri, è adattata al problema sottostante. L'architettura del modello di predizione di Bao è illustrata nella figura 27.

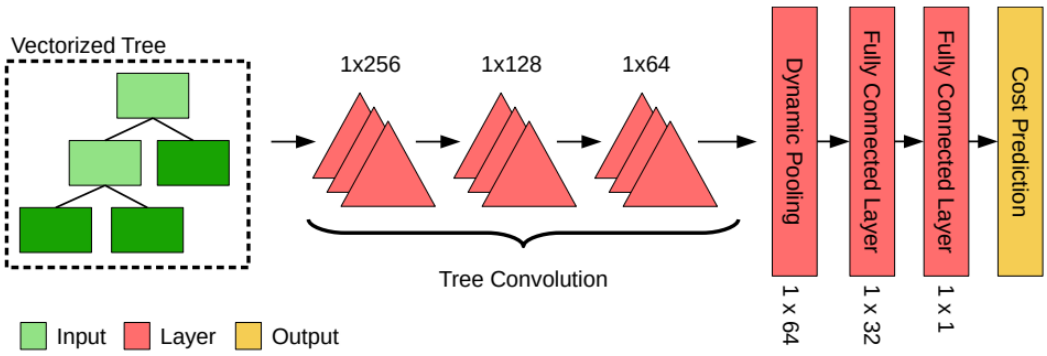


Fig. 27: Architettura del modello di predizione di Bao.

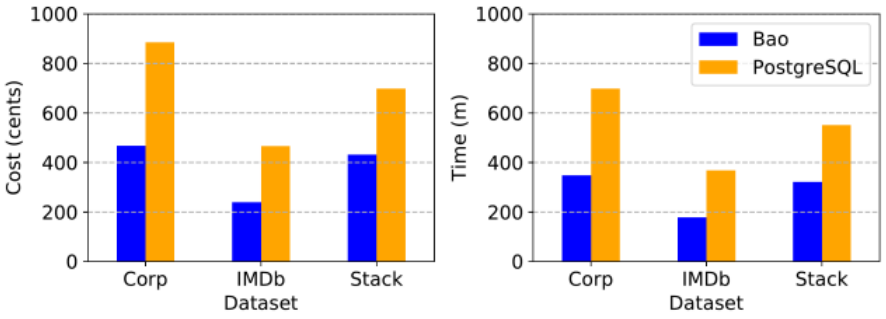
L'albero del piano di query attraversa tre strati di convoluzione ad albero. Dopo l'ultimo strato di convoluzione ad albero, si applica il pooling dinamico per appiattire la struttura ad albero in un singolo vettore. Successivamente, due strati completamente connessi convertono il pooled vector in una previsione delle prestazioni.

3.4.2 Valutazione sperimentale: impostazione degli esperimenti

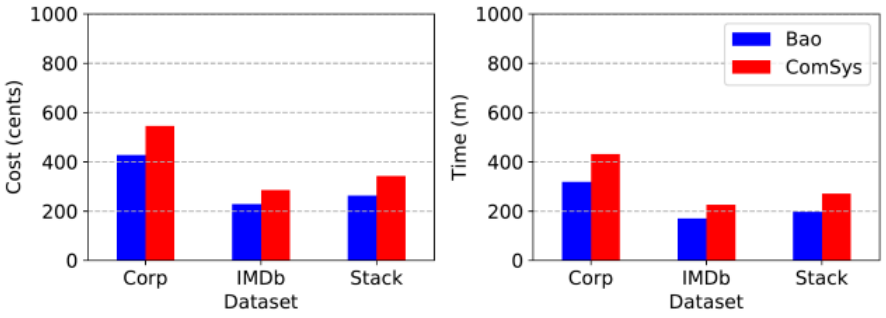
Si è confrontato Bao con PostgreSQL e con un sistema di database commerciale, ComSys, entrambi configurati e messi a punto secondo le rispettive documentazioni e le migliori pratiche. La configurazione è stata verificata da un consulente di ComSys attraverso piccoli test sulle prestazioni. In entrambi i casi, Bao è stato integrato nel database utilizzando l'ottimizzatore originale attraverso degli hint. Ad esempio, l'integrazione in ComSys sfrutta l'ottimizzatore originale di ComSys con i suggerimenti ed eseguendo tutte le query su ComSys. Se non diversamente indicato, le query sono eseguite in sequenza. Sono utilizzati 48 set di hint, ognuno dei quali impiega sottoinsiemi degli operatori di join (hash join, merge join, loop join) e degli operatori di scansione (sequenziale, con indice o solo con indice). Impostando la dimensione della finestra di lookback a k = 2000 e riaddestrando ogni n = 100 query si ottiene un buon compromesso tra il tempo della GPU e le prestazioni delle query

3.4.3 Valutazione sperimentale: costi e prestazioni nel cloud

Nella figura 28 sono riportati i costi (a sinistra) e i tempi necessari (a destra) per eseguire completamente tre carichi di lavoro su Google Cloud utilizzando una VM N1-16.



(a) Across our three evaluation datasets, Bao on the PostgreSQL engine vs. PostgreSQL optimizer on the PostgreSQL engine.



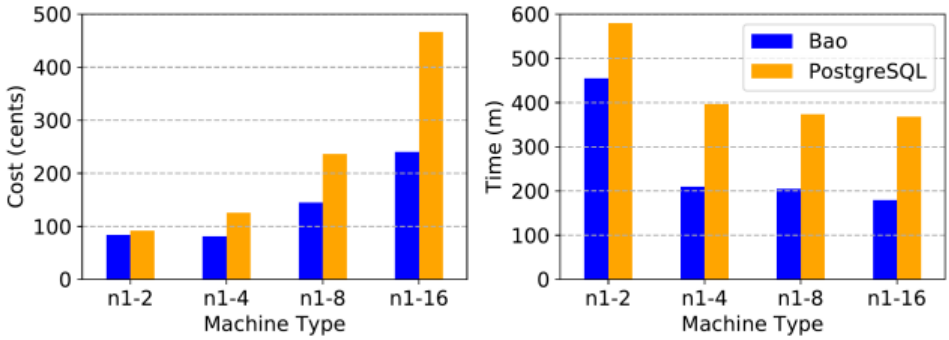
(b) Across our three evaluation datasets, Bao on the ComSys engine vs. ComSys optimizer on the ComSys engine.

Fig. 28: Costi (a sinistra) e latenze del carico di lavoro (a destra) per Bao e due ottimizzatori di query tradizionali su tre diversi carichi di lavoro su una macchina virtuale N1-16 di Google Cloud.

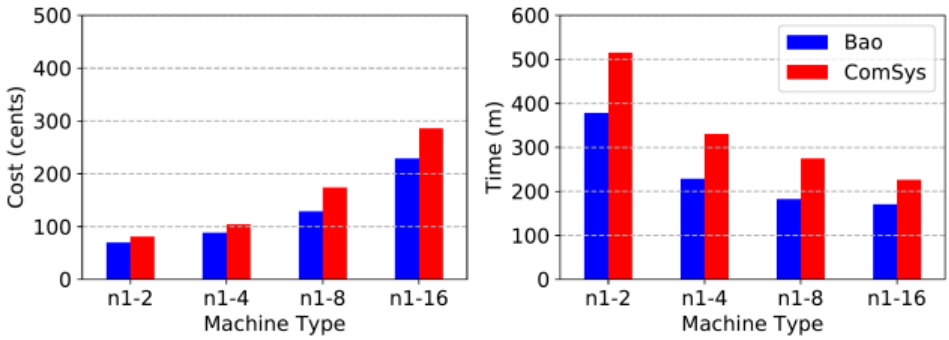
Bao supera PostgreSQL di quasi il 50% in termini di costi e latenza sui diversi set di dati. È importante notare che ciò include il costo dell'addestramento e l'allocazione della GPU alla VM. Inoltre, tutti i set di dati contengono modifiche al carico di lavoro, ai dati o allo schema, dimostrando l'adattabilità di Bao a questi scenari comuni e importanti.
Il miglioramento delle prestazioni ottenuto da Bao con il database commerciale è ancora significativo, anche se meno consistente.

3.4.4 Valutazione sperimentale: adattabilità ai tipi di hardware

In un secondo esperimento, sono state modificate le caratteristiche dell'hardware per il carico di lavoro IMDb. I corrispettivi risultati sono riportati nella figura 29.



(a) Across four different VM types, Bao on the PostgreSQL engine vs. PostgreSQL optimizer on the PostgreSQL engine.



(b) Across four different VM types, Bao on the ComSys engine vs. ComSys optimizer on the ComSys engine.

Fig. 29: Costi (a sinistra) e latenze del carico di lavoro (a destra) per Bao e due ottimizzatori di query tradizionali su quattro diverse dimensioni di VM di Google Cloud Platform per il carico di lavoro IMDb.

Nel caso di PostgreSQL, la differenza sia in termini di costo che di prestazioni risulta più significativa con tipi di VM più grandi (ad esempio, N1-16 rispetto a N1-8). Ciò suggerisce che Bao è più efficace nel regolare il proprio tuning in risposta ai cambiamenti nell'hardware rispetto a PostgreSQL.
È interessante notare che, mentre i vantaggi di Bao aumentano con dimensioni di

macchina più grandi per PostgreSQL, ciò non accade per il sistema commerciale (ComSys). Questo suggerisce che il sistema commerciale è più capace di adattarsi a diversi tipi di hardware, o forse che il sistema commerciale è "di default" tarato per tipi di macchine più grandi.

Conclusioni

La presente tesi espone il concetto dei cosiddetti "learnable database", ossia database in grado di incorporare il machine learning per acquisire conoscenza dai dati e ottimizzare le prestazioni e la gestione delle risorse. Negli ultimi anni sono stati sviluppati diversi approcci per integrare il machine learning nei database, tra cui quelli analizzati in dettaglio precedentemente, come la configurazione automatica dei parametri in base al carico di lavoro storico e ai dati delle query, il partizionamento automatico dei dati e l'ottimizzazione delle query. Oltre a tali approcci, attualmente vi sono ricerche e sviluppi relativi ad altri ambiti, come:

- la raccomandazione e l'ottimizzazione degli indici;
- la realizzazione di un'interfaccia di interrogazione, combinante il linguaggio naturale e il linguaggio SQL dei database;
- il miglioramento della sicurezza dei dati e del funzionamento del sistema.

Nonostante i risultati straordinari ottenuti a livello sperimentale, sono necessari ulteriori passaggi per rendere questi approcci accessibili e utilizzabili concretamente nei database commerciali più diffusi. In particolare, l'integrazione del machine learning nell'ottimizzazione dei database presenta diverse sfide che devono essere affrontate per garantire il successo e l'efficacia delle soluzioni implementate, tra cui:

- qualità dei dati: poiché i modelli di machine learning dipendono dalla qualità dei dati su cui vengono addestrati, la presenza di dati incompleti, rumorosi o errati può compromettere l'accuratezza e l'efficacia delle previsioni e delle ottimizzazioni;
- interpretabilità dei modelli: molti algoritmi di machine learning, specialmente quelli più complessi come le reti neurali profonde, possono essere difficili da interpretare, il che rende importante garantire che le decisioni prese dai modelli siano comprensibili e trasparenti agli utenti e agli stakeholder coinvolti;
- sicurezza e privacy dei dati: l'accesso e l'analisi dei dati sensibili da parte dei modelli di machine learning possono sollevare preoccupazioni riguardo alla sicurezza e alla privacy; dunque, è essenziale implementare misure di sicurezza robuste per proteggere i dati sensibili e garantire la conformità alle normative sulla privacy, come il GDPR;
- scalabilità: le soluzioni di machine learning per l'ottimizzazione dei database devono essere in grado di gestire volumi crescenti di dati e carichi di lavoro in modo efficiente;
- costi e risorse: l'addestramento e l'esecuzione dei modelli di machine learning possono richiedere risorse computazionali significative e competenze specializzate per la gestione e la manutenzione;

- adattamento ai cambiamenti: i modelli di machine learning devono essere in grado di adattarsi e apprendere dai cambiamenti nei dati e nei requisiti di utilizzo dei database nel tempo.

Solo attraverso un'attenta pianificazione, progettazione e implementazione è possibile superare queste sfide e realizzare pienamente il potenziale del machine learning nell'ottimizzazione dei database.

Bibliografia

Di seguito sono riportate tutte le fonti utilizzate:

[1] Survey on Learnable Databases: A Machine Learning Perspective; Benyuan Zou, Jinguo You, Quankun Wang, Xinxian Wen, Lianyin Jia; 2022: https://www.sciencedirect.com/science/article/abs/pii/S221457962100-1210?via%3Dihub

[2] Automatic Database Knob Tuning: A Survey; Xinyang Zhao, Xuanhe Zhou, Guoliang Li; 2023: https://ieeexplore.ieee.org/document/101060-50

[3] OtterTune - Carnegie Mellon Database Group: https://db.cs.cmu.edu-/projects/ottertune/

[4] OtterTune explained in 5 minutes; Andy Pavlo; 2021: https://otter-tune.com/blog/ottertune-explained-in-five-minutes

[5] An End-to-End Automatic Cloud Database Tuning; Ji Zhang et al.; 2019: https:// dbgroup.cs.tsinghua.edu.cn/ligl/papers/sigmod19-cdbtune.pdf

[6] Learning a Partitioning Advisor for Cloud Databases; Benjamin Hilprecht, Carsten Binnig, Uwe Röhm; 2020: https://dl.acm.org/doi/10.1145-/3318464.3389704

[7] Query optimization: https://en.wikipedia.org/wiki/Query_optimization

[8] Panoramica del piano di esecuzione - SQL Server: https://learn.micro-soft.com/it-it/sql/relational-databases/performance/execution-plans?view=sql-ser-ver-ver16

[9] 14.500 Query Optimizer Overview; Jens Dittrich; 2014: https://-www.youtube.com/watch?v=ZqKTBrolJ00&list=PLC4UZxBVGKtcZgLCrIUenu-ano53pbPpf1&index=7

[10] Bao: Making Learned Query Optimization Practical; Ryan Marcus et al.; 2021: https://people.csail.mit.edu/tatbul/publications/bao_sigrec22.pdf