Università degli Studi di Trieste - Dipartimento di Ingegneria e Architettura

Sintesi estesa di "Transazioni e Analisi Cloud-Native in SingleStore"

Enrico Piccin - IN0501089

Anno Accademico 2023/2024

Tesi presentata per il conseguimento della laurea in
Ingegneria Elettronica e Informatica

1 Sintesi estesa

Il team di SingleStore[1] ha osservato che la sempre più crescente necessità di persistere grandi moli di dati e di accedervi con frequenza e bassa latenza richiede l'elasticità offerta da soluzioni in cloud. Alla luce di ciò, il DBMS S2DB di SingleStore nasce proprio come database cloud-native, progettato appositamente per supportare pesanti ed eterogenei carichi di lavoro transazionali (OLTP) e analitici (OLAP).
Essendo un database distribuito che implementa l'horizontal partitioning, S2DB è organizzato in cluster di shard: i nodi aggregatori coordinano l'esecuzione delle query, mentre i nodi foglia contengono specifiche partizioni di dati determinate dalle shard-key. Lo studio sperimentale condotto dagli autori ha evidenziato che tale approccio, oltre ad ottimizzare la scalabilità e il load balancing tra gli shard, favorisce anche maggiore rapidità nell'auto-healing in caso di failover.
Più specificatamente, la memorizzazione dei dati in S2DB è strutturata in due fasi, illustrate in Figura 1: i nuovi dati vengono prima memorizzati localmente nei nodi di ogni cluster e poi, solo successivamente, trasferiti in modo asincrono su un BLOB (Binary Large Object) storage separato, riducendo, così, la latenza di archiviazione. Il BLOB storage, infatti, è strutturato affinché sia scalabile indipendentemente dall'elaborazione dei nodi, risultando in uno spazio di archiviazione fortemente estensibile e capace di conservare molteplici versioni dei dati nel tempo: tale cronologia può essere impiegata da S2DB per effettuare un Point In Time Restore (PITR) per riportare il database ad uno stato passato senza la necessità di un backup relativo.
Il valore aggiunto riconosciuto a S2DB consiste, tuttavia, nella creazione out-of-order, per ogni cluster, di più repliche di una stessa partizione memorizzate su più nodi: tale meccanismo garantisce resilienza a seguito del guasto di un nodo, ma anche la possibilità di eseguire letture/scritture su tali repliche con tempi brevi e prevedibili, senza dover attendere il completamento di eventuali transazioni che interessano i medesimi dati o il caricamento degli stessi sul BLOB storage.

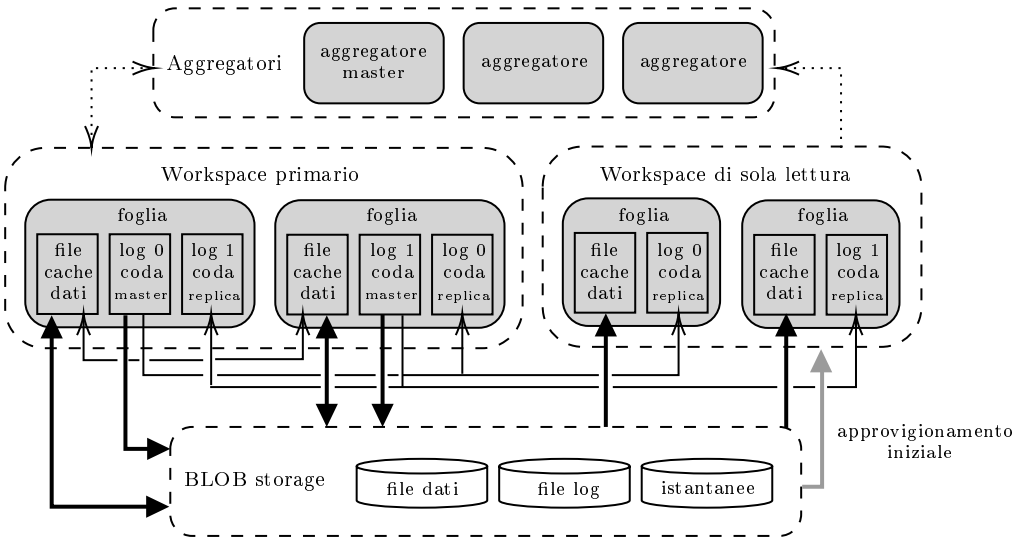

Figura 1: Architettura cluster con BLOB storage separato

Conservando in cache locali entità informative ad accesso frequente (IOPS elevato) S2DB riesce, infatti, a produrre un notevole aumento delle prestazioni per i carichi di lavoro analitici e transazionali che eseguono molte letture e scritture simultanee. Si concretizza, così, la netta separazione tra l'archiviazione e l'elaborazione, attività che avvengono in modo concorrente e indipendente, minimizzando gli aggiornamenti delle tabelle sul cloud.
A fronte, però, delle efficienti prestazioni evidenziate, gli autori denotano la maggiore complessità di gestione di tale meccanismo di archiviazione e accessibilità che, inevitabilmente, va a scapito della sua flessibilità: l'aggiunta o la rimozione di nodi, infatti, richiede un attento spostamento dei dati locali non ancora presenti nel BLOB storage per garantirne la persistenza e disponibilità.
Un'altra grande innovazione attribuita a S2DB consta nel suo schema di archiviazione dei dati a tabelle unificate che, indipendentemente dalle operazioni che andranno ad interessare ciascuna tabella, favorisce ugualmente tanto carichi OLTP che OLAP. Ciò è reso possibile attraverso una combinazione interna di due diverse modalità di archiviazione dei dati

- per righe, con una indicizzazione strutturata per mezzo di una gerarchia di liste concatenate (skip list): ogni nodo della skip list crea e gestisce un elenco di versioni della medesima riga per favorire e velocizzare le letture/scritture concorrenti.
  Periodicamente, inoltre, viene eseguito un backup dello stato serializzato delle tabelle memorizzate per righe al fine di limitarne i tempi di ripristino in caso di failover.

- per colonne[2], i cui dati vengono strutturati in segmenti, ognuno dei quali persiste un sottoinsieme disgiunto di righe. Ciascuna colonna viene compressa secondo delle codifiche che consentono una lettura efficiente di un dato specifico senza decodificare tutte le righe: viene sottolineato, infatti, che S2DB supporta l'interrogazione delle tabelle memorizzate per colonne sia vettorialmente che a livello codificato, ottenendo considerevoli accelerazioni nelle operazioni di filtraggio e aggregazione, interagendo direttamente con i dati compressi e utilizzando istruzioni SIMD quando opportuno.
  Tali prestazioni si concretizzazione anche mediante la creazione di serie ordinate di segmenti secondo una specifica chiave di ordinamento: processi in background eseguono frequentemente riorganizzazioni e fusioni dei segmenti costituenti tali serie allo scopo di accelerare le operazioni di IO.

È indubbio che tali performance siano il risultato di un'attenta progettazione del motore di archiviazione, costruito appositamente per carichi di lavoro HTAP[3] (Hybrid Transactions and Analytics Processing), eliminando congiuntamente l'onere per gli utenti di scegliere il layout dei dati adatto al loro particolare carico di lavoro.
La decisione dei progettisti SingleStore di basare l'archiviazione su LSM-tree (Log Structured Merge Tree) e su più livelli incontra nativamente le esigenze di carichi di lavoro OLAP, ma la vera sfida descritta dagli autori, è stata quella di rendere tale approccio efficiente anche per i casi d'uso OLTP, senza sacrificare le prestazioni OLAP: con tale fine, a differenza di altre implementazioni LSM-tree, in cui le cancellazioni e gli aggiornamenti di dati richiedono l'unione di tutti i livelli dell'albero, in S2DB, invece, tali operazioni vengono rappresentate per mezzo di un vettore di bit memorizzato come metadati di ogni segmento, rendendo, così, più prestante il filtraggio delle righe interessate.
Sarkar e Papon, nel loro studio[4], osservano, però, che se da un lato la soluzione di S2DB riduce al minimo l'accesso al disco, dall'altro introduce, per ogni segmento, la necessità di gestire sezioni critiche e di scongiurare il blocco dovuto a modifiche concorrenti sugli stessi metadati: tali proprietà in S2DB vengono, di fatto, garantite trasferendo le righe da cancellare/modificare, attraverso transazioni autonome e out-of-order, in tabelle di transizione che forzano modifiche transazionali in mutua esclusione.
Chi scrive evidenzia, inoltre, che la struttura LSM-tree ben si integra con la creazione di indici secondari a due livelli adottata in S2SB: per ogni segmento, infatti, viene costruito un inverted index per mappare gli offset delle righe contenute nel segmento stesso; in aggiunta a ciò, viene utilizzato un global index per mappare i valori della colonna indicizzata agli id dei segmenti tramite hash table, insieme ad un riferimento al relativo inverted index, come mostrato nella Figura 2:

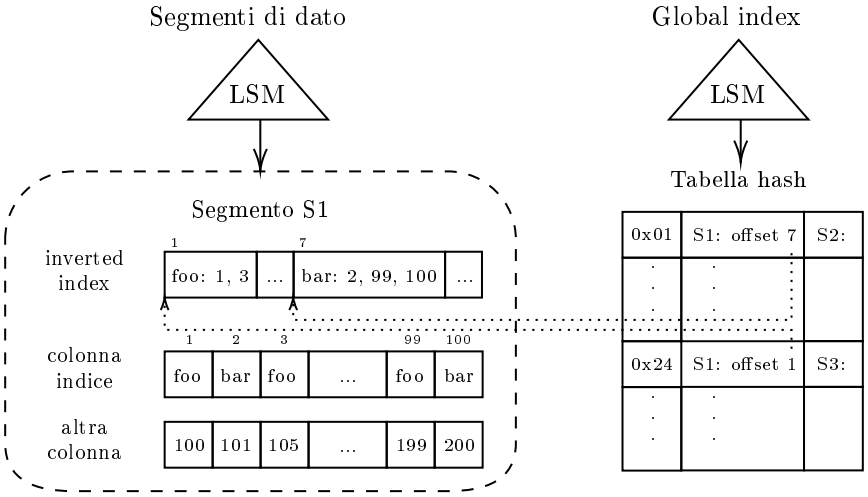

Figura 2: Struttura dell'indice secondario a due livelli

Gli autori puntualizzano che la complessità computazionale di tale approccio vincola, però, S2DB a supportare attualmente solo indici secondari non ordinati: se così non fosse, l'implementazione del global index come LSM-tree non ordinato non sarebbe indipendente dai segmenti di dati stessi, proprietà cruciale per evitare che gli indici secondari non diventino un punto di contesa durante le scritture. Dall'altro lato, però, l'indicizzazione descritta consente un'agevole applicazione dei vincoli di unicità, senza forzare alcuna chiave di ordinamento o la replicazione dei dati: all'inserimento di un nuovo record, viene controllato l'indice secondario per scongiurare la presenza di duplicati.
Tali caratteristiche, infine, sono funzionali all'esecuzione adattiva delle query, un meccanismo che consente di adottare dinamicamente la combinazione di metodi di accesso ai dati più conveniente per carichi HTAP: l'individuazione dei segmenti da processare, infatti, utilizza principalmente le strutture dell'indice secondario globale mediante un probing sulle relative chiavi; laddove il numero di chiavi da ricercare sia elevato per le dimensioni della tabella, S2DB disabilita dinamicamente tale approccio, eseguendo un probing sui metadati di ciascun segmento. Il team di SingleStore ha dimostrato sperimentalmente che tale automatismo risulta particolarmente efficace nell'esecuzione di JOIN tra tabelle, in cui il numero di chiavi utilizzate per il probing può avere grandi variazioni.
Successivamente, avviene il filtraggio dei segmenti atto alla circoscrizione delle righe da leggere: il filtro può interessare i dati compressi, oppure già decodificati, valutando singolarmente ogni clausola o aggregandole insieme e, quando opportuno, basandosi sull'indice secondario. Per selezionare la strategia di filtraggio ottimale, S2DB calcola il costo di ogni metodo di processamento del filtro e combina tale risultato con la probabilità che ogni riga soddisfi il predicato del filtro stesso (selettività del filtro[5]), riordinando dinamicamente le clausole di filtraggio.
Per certificare le significative performance di S2DB, alla luce di quanto esposto, gli autori hanno raccolto eloquenti risultati sperimentali sulla base dei benchmark TPC-H, un benchmark OLAP, e TPC-C, un benchmark OLTP[6]. Il confronto è avvenuto con altri due data warehouse in cloud, denominati CDW1 e CDW2, e un database operativo in cloud, denominato CDB, prodotti leader del settore.
I test per TPC-H hanno previsto interrogazioni con e senza cached data, ottenendo risultati tra S2DB, CDW1 e CDW2, raccolti nella Tabella 1, fortemente competitivi che dimostrano come la soluzione SD2B sia altamente all'avanguardia:

Prodotto	**Prezzo orario cluster**	**Media geometrica TPC-H**	**Media geometrica TPC-H**	**Throughput TPC-H (QPS)**
S2DB	€ 16.50	8.57 s	€ 3.92×10^{-2}	0.078
CDW1	€ 16.00	10.31 s	€ 4.58×10^{-2}	0.069
CDW2	€ 16.30	10.06 s	€ 4.55×10^{-2}	0.082
CDB	€ 13.92	Nessun risultato in 24 ore		

Tabella 1: Riepilogo dei risultati di TPC-H (1TB)

Per TPC-C, è stata eseguita la simulazione della gestione di un'attività di vendita e distribuzione di prodotti a magazzino, registrando il throughput a livelli di carico incrementali e dimostrando la capacità di S2DB di scalare linearmente, come esposto nella Tabella 2:

Prodotto	**vCPU**	**Dimensione (magazzini)**	**Throughput (tpmC)**	**Throughput (% max)**
CDB	32	1000	12.582	97.8%
S2DB	32	1000	12.556	97.7%
S2DB	256	10000	121.432	94.4%

Tabella 2: Risultati TPC-C (fino al limite di 12.86 tpmC/magazzino)

Appare, quindi, manifesto l'impatto favorevole dello schema di archiviazione dei dati a tabelle unificate di S2DB, specialmente se comparato con il formato di archiviazione orientato alle righe

di CDB, il quale ha ottenuto prestazioni di ordini di grandezza peggiori su TPC-H rispetto a S2DB, CDW1 e CDW2. A differenza di CDW1 e CDW2, che mancano della presenza di vincoli di unicità e di ricerche efficienti e concorrenti, S2DB è capace di adattarsi anche a carichi TPC-C, a dimostrazione che la soluzione di SingleStore è in grado di soddisfare i requisiti di carico di lavoro che in precedenza richiedevano l'uso di più sistemi di database specializzati.

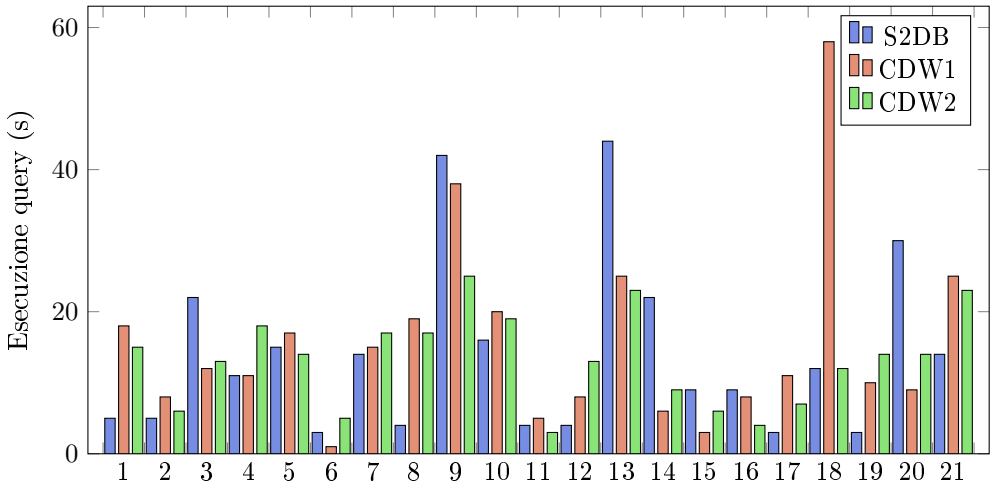

Figura 3: Tempi di esecuzione delle query TPC-H da 1 TB (più basso è meglio)

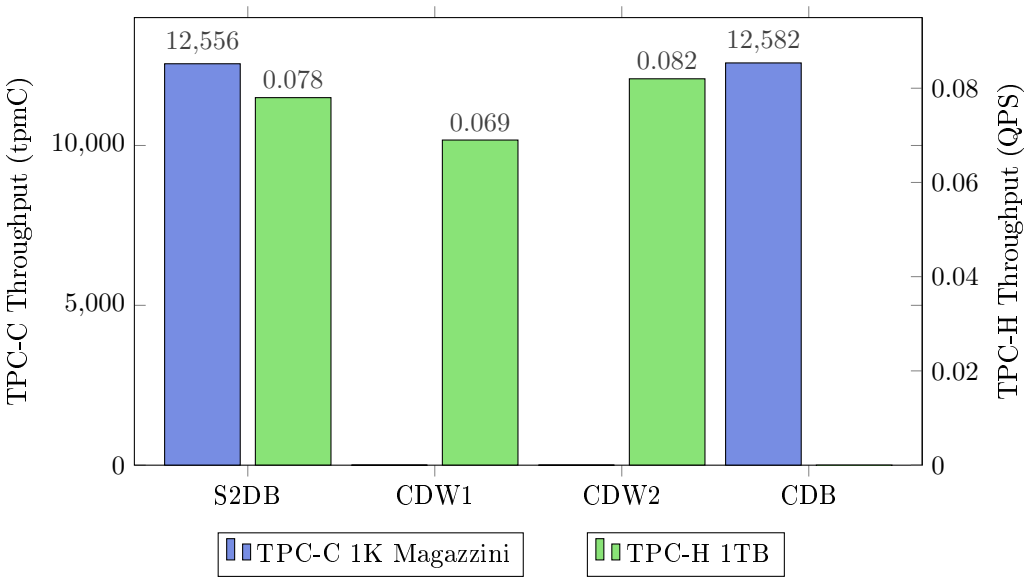

Figura 4: Riepilogo dei throughput di TPC-C e TPC-H (più alto è meglio)

Riferimenti bibliografici

[1] Adam Prout, Szu-Po Wang, Joseph Victor, Zhou Sun, Yongzhu Li, Jack Chen, Evan Bergeron, Eric Hanson, Robert Walzer, Rodrigo Gomes, Nikita Shamgunov (2022) - *Cloud-Native Transactions and Analytics in SingleStore*, https://dl.acm.org/doi/pdf/10.1145/3514221.3526055, SIGMOD '22, Philadelphia, PA, USA.

[2] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, Christof Bornhövd (2012) - *Efficient Transaction Processing in SAP HANA Database - The End of a Column Store Myth*, http://dx.doi.org/10.1145/2213836.2213946, SIGMOD '12.

[3] Guoliang Li, Chao Zhang (2022) - *HTAP Databases: What is New and What is Next*, https://dl.acm.org/doi/pdf/10.1145/3514221.3522565, SIGMOD '22, Philadelphia, PA, USA.

[4] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, Zichen Zhu, Manos Athanassoulis (2023) - *Enabling Timely and Persistent Deletion in LSM-Engines*, https://dl.acm.org/doi/abs/10.1145/3599724, pp. 1 - 40.

[5] Pranav Subramaniam (2019) - *Generating Selective Filters for Access Method and Physical Design Evaluation*, https://dl.acm.org/doi/10.1145/3299869.3300098, pp. 1853 - 1855.

[6] TPC Benchmarks Overview, https://www.tpc.org/information/benchmarks5.asp.