Appunti sugli Index Store

Introduzione

Il tema trattato è riferito alla separazione degli indici tra primari e secondari. E' importante fare un necessario distinguo nel concetto di “primari” usato in questa sede.
L'accezione canonica prevede che si intenda primario, o *primary key*un indice che, combinato con un indice secondario, o *foreing key*, consenta di costituire legami logici tra record distinti, legami necessari all'integrità referenziale così come alla coerenza del data base. Ad essi si aggiungono quelle chiavi, pur sempre primarie, che entrano in gioco per qualsivoglia verifica di univocità e/o serialità.
Nello scenario di *eXtraWay* queste sono sì chiavi primarie ma ad esse vanno affiancate altre chiavi finalizzate al corretto comportamento di un'applicazione anche in assenza di *tutte le altre chiavi possibili*.
Parleremo quindi, più propriamente, di chiavi di *primaria importanza* o di *prima grandezza*.
Dal momento che un buon esempio vale spesso più di 1000 parole pensiamo ad un'applicazione come *DocWay*. In essa le chiavi primarie sono palesemente il numero di protocollo, l'nrecord ed altre finalizzate all'univocità, ma possiamo dire che anche l'oggetto del documento possa considerarsi primario essendo la componente per la quale i documenti vengono normalmente ricercati così com'è indispensabile che si considerino di primaria importanza gli indici per mezzo dei quali si stabilisce l'effettivo diritto di ciascun utente ad accedere (ed in che modo) ad ogni singolo documento.
Realizzare un'applicazione *DocWay* in cui gli indici dei “riferimenti interni”[1)] non siano subito disponibili a seguito di inserimento e modifica significherebbe rendere del tutto inutilizzabile tale applicazione sino al completamento degli indici *secondari*.
Ecco quindi perché parlare di indici di *primaria importanza o grandezza* ed indici di *secondaria importanza o grandezza*. Per brevità, comunque, ci si riferirà ad essi sempre come indici *primari* ed indici *secondari* assumendo che il lettore abbia chiaro il distinguo qui compiuto.

Glossario

Oltre al necessario distinguo tra indici primari ed indici secondari appena effettuato, ricordiamo anche le seguenti definizioni.

Termine	Significato
Vocabolario, Canale o Indice	raggruppamento di tutte le chiavi che afferiscono allo stesso indice e quindi appartengono in origine allo stesso elemento o attributo.
Chiave, termine o parola	singolo termine estratto dal documento (da un attributo o da un elemento) che appartiene ad un vocabolario.
Index Manager	Strumento per la gestione cumulativa degli indici
Index Store	Repositori per una determinata categoria di indici. Può contenere uno o più indici. Esisterà un Index Store per gli indici primari ed uno o più Index Store per gli indici secondari raggruppati secondo i criteri che si riterranno più opportuni e senza alcun vincolo legato alla struttura archivio o documento.

Lo stato dell'arte

Attualmente la costituzione degli indici prevede due scenari: Indici off-line / indici on-line

A loro volta gli indici on line sono suddivisi in due scenari: Inserimento (incrementale), modifica/cancellazione (differenziale).

Affinità: Indici off-line sono affini all'inserimento in quanto entrambe sono incrementali. Ne consegue che son affini modifica e cancellazione (diff.).

L'indicizzazione di uno o più documenti è comunque suddivisa in due parti: Preparazione e Scarico.

Sistema indici fatto con 3 files: IDX → Btree con le chiavi. Nello stesso BTree sono presenti più vocabolari, riconoscibili in base ai primi 4 byte di ogni chiave, e per ciascun vocabolario un numero imprecisato di chiavi. A ciascuna chiave viene assegnato un valore numerico progressivo ed univoco che è il numero di chiave. VCB → File di record a dimensione fissa, uno per ogni chiave “puntato” dai valori estratti dall'IDX. Ciascun record consente di identificare una catena di riferimenti (posizione e dimensione nel file REF) indicando per essa quandi documenti seleziona, quante vole la parola è presente in tali documenti, qual'è l'ultimo documento e l'ultima iword in esso. REF → Buddy File con catene di riferimenti sotto forma di catene delta. Presenta chiavi con iword (delta doc con una o più delta iwords ed un separatore etc. etc. etc.) o chiavi senza iword (sequenza di delta doc).

Preparazione: Si estraggono da elementi ed attributi dei documenti tutte le parole (tokenizzazione). A ciascuna di esse si accoppia la sua posizione (iword) se richiesta (attualmente non viene calcolata solo per le chiavi “single”). Si alimentano quindi in memoria un BTree con tutte le chiave estratte, ed un sistema di catene di strutture che si puntano l'un l'altra e servono ad identificare per una parola data tutte le sue posizioni (se e dove richiesto).

Scarico: avviene quando è terminata l'operazione (normalmente on line) o quanto ho esaurito la risorsa (tipicamente off line). Tra diversi casi.

Appunti Sparsi

Di seguito alcuni appunti sparsi che mancano ancora di un ordine mentale:

L'identificazione del vocabolario di appartenenza di una chiave, che attualmente si fa incrociando i dati che vengono dai percorsi di chiave salvati nel file IDX e da quello che c'è nel file di configurazione dell'archivio, dev'essere compito della classe

dell'IndexManager che, dato un percorso XML, deve dirmi quale'è il codice di vocabolario e quindi sapere in quale IndexStore finiranno questi vocabolari/chiavi.

Il trattamento delle chiavi dichiarative va estratto dall'attuale IDX e gestito in files a parte che rappersentino una unordered-map o un XML o quello che ci verrà in mente. Questo ci consente di abbattere i limiti dimensionali sulla lunghezza del percorso e qui nesting levels. (Quest'implementazione può attendere, i primi test possono essere fatti con le chiavi così come sono ora).

Nel .conf.xml bisogna dichiarare gli IndexStores. Per ciascuno di essi, salvo valori di default, può essere indicato l'ammontare di risorse da usare on line e off line per b+tree e catene riferimenti. Approcciamo per default. Ci sono un IS primario ed uno secondario di default. Tutte le chiavi dichiarate primarie afferiscono al primo, tutte le chiavi non diversamene dichiarate afferiscono al secondo. Per fare altri IS specifici vanno configurati espressamente e gli vanno associate delle chiavi...

Ciascun IS fa quello che attualmente fa l'indicizzazione in senso generale, ovvero accoglie una chiave e la sua Iword, la pone nel BTree, compone la catena di strutture finalizzata alla realizzzione delle catene di riferimenti e all'atto dello scaricamento fa una delle 3 operazioni previste: a) Realizzazione della coppia di files lot/lop; b) Scarico incrementale; c) Scarico differenziale.

L'attuale BTree in memmoria pe ril reperimento rapido delle chiavi e delle loro caratteristiche potrebbe essere serializzato su disco (e rimosso solo quando si rifanno gli indici) in modo che non debba essere ricalcolato dinamicamente ogni volta che un server apre un archivio.

Le chiavi secondarie possono essere elaborate con operazioni che vadano direttametne a scirvere una serie di ToDo files di una dimensione nota che un thread a parte provvederà a consumare. Questro thread avrebbe quindi facoltà di operare sulle chiavi secondarie bloccandone una alla volta.

La separazione degli indici tra primari e secondari può essere solo logica e quindi non richiedere effettivamente files di indici distinti. Quello che è essenziale è che le operazioni incrementali su singoli documenti potrebbero essere effettuate anche direttamente, mente le differenziali e le incrementali massive devono operare direttamente sulle chiavi primarie (stesura delle catene dei riferimenti in modo incrementale ovvero stesura di LOT e LOP) mente quelle secondarie avvalersi di un ToDo file, uno per archivio, in cui registare per ogni documento e per ogni chiave la posizione da aggiungere o togliere.
Il thread (del master?) preposto a consumare tali ToDo files procede in modo lineare, e quando il file è completo lo rimuove [2)] intervenendo sulle singole chiavi, bloccandole una per una, singolarmente, riducendo ai minimi termini gli impatti sulle attività da svolgere.
Si possono quindi immaginare una batteria di files per gli indici primari, una, ed una sola, per gli indici secondari oppure più batterie distinte per essi così come tutti potrebbero convivere senza grandi complicazioni nello stesso sistema di indici. Le chiavi nel btree si possono inserire senza problemi in fase di indicizzazione, lasciando nel ToDo file solo le tracce delle operazioni da compiere sulle chiavi (identificate per numero e non per chiave). Va beh, bisogna ancora pensarci sopra...

Se il ToDo file lo pensiamo un po' come una catena di riferimenti, come sono ora fatti i files LOT e LOP, possiamo immaginare una cosa. Se la chiave l'abbiamo già messa nel BTree, nel nostro To Do dobbiamo indicare il numero del documento poi il numero della chiave e tutti i delta positivi e negativi.
Diciamo che potremmo replicare la coppia “documento-chiave” ad ogni chiave anziché indicarla una tantum all'inizio del trattamento di un documento, in questo modo il thread che consuma q uesto file potrebbe anche elaborare una chiave alla volta interrompendosi quando vuole. Viene poi il momento delle delta iwords. Se la chiave non ha iwords bisogna poter dichiarare se va agguinta o tolta... ed a questo voglio pensare ancora, ma se ha le iwords, allora bisogna poter indicare nella catena dei numeri positivi e negativi.
Immaginiamo di dover sostituire la posizione 27 con la posizione 28, quindi avremo -27+28. Ora... 27 è 0x1b mentre 28 è 0x1C. Se li shifto a sinistra di un bit ottengo 0x36(54) e 0x38(56). Ora diciamo che il bit meno significativo funge da bit del segno. Trasformerò quindi il mio 0x36(54) in 0x37(54+1) che vuol dire “27 negativo”. Lo faccio seguire dal mio 0x38(56) che vuol dire “28 positivo” e posso scrivere una rapida catena di riferimenti con delta negativi!!

Idee del 21/04/2010

- Per non perdere prestazioni in ricerca in adiacenza, non è ragionevole separare la catena dei riferimenti ai documenti di ciascuna chiave dalla sua corrispondente catena di riferimenti alle singole iwords. Tali catene devono continuare ad essere mescolate.
- In considerazione del punto precedente, l'ottimizzazione che può applicarsi in modifica senza che essa abbia impatti negativi (ed anzi potrebbe risultare vantaggiosa) in ricerca, è quella di spezzare catene di riferimenti relativamente lunghe in catene più brevi che si puntino l'un l'altra in modo che intervenendo in modifica su una precisa porzione gli spostamenti necessari si limitino ad essa. In questo caso l'idea era di creare aree di Buddy File che si referenziano l'un l'altra e che, raggiunta una soglia dimensionale nota, si splittano alla bisogna creando una “deviazione” della catena che le unisce.
- La soluzione appena vista non nasconde problemi di manutenzione quindi si può agire secondo questa logica: per ciascuna chiave di qualsiasi vocabolario esiste una (o più di una, vedremo poi perché) corrispondete chiava in altro B+Tree. Tale chiave è composta dal numero di chiave evinto dal B+Tree principale, e dal primo documento puntato da una porzione di

catena di riferimenti[3]. Le catene brevi avranno un unica chiave in questo BTree mente le catene di dimensioni più ampie ne avranno più di una. Ciascuna chiave descrive quale intervallo di documenti viene trattato dalla catena puntata. I Dati associati alla chiave saranno posizione e dimensione in un buddy file (il .ref) dove si trova tale catena e, conseguentemente, verranno modificati (specialmente nella size) ogni volta che si compie un ritocco a tale chiave.

- L'ipotesi ventilata ha i suoi vantaggi anche in ricerca in quanto con i più recenti interventi alla pola predittiva, sarà possibile stabilire a priori quale intervallo di documenti può soddisfare la selezione. Così facendo è altamente probabile che una data combinazione di chiavi consenta, per le più lunghe di esse, di accedere solo alla porzione che serve, saltando a piè pari tutte le precedenti con un evidente risparmio in termini di operazioni di lettura ed attività di interpretazione della catena dei riferimenti.
- E' stata anche discussa la possibilità di evitare il file .ref ponendo i blocchi di catene di riferimenti direttamente nel BTree destinato a mantenere queste chiavi particolari. Questa soluzione richiederebbe che il Btree usi chiavi a dimensione variabile (come quello “vecchio”) ma impone, in un simile scenario, che ogni intervento effettuato sulla chiave comporti la sua “sostituzione”, attività impropria per un BTree. Questo in un BTree con chiave dimensione variabile e Dati associati a dimensione fissa. L'ideale sarebbe invece un BTree con chiavi a dimensione fissa e Dati associati di dimensione variabile. La possibilità che essi cambino spesso suggerisce però di mantenere tali dati, di fatto, in un Buddy File esterno e limitarci a puntare ad essi con dimensione e posizione, come già detto precedentemente.
- In quest'ottica potrebbe apparire non necessario il record del VCB. Di fatto, però, è comunque necessario, per poter compiere in modo performante le attività incrementali, conoscere l'ultimo documento e l'ultima iword in esso. Per altre ragioni è invece necessario conoscere il numero di documenti puntati ed il numero di presenze della parola in essi. D'altro canto manca proprio nel VCB il numero del primo documento puntato che invece sarebbe comunque strategico (per risolvere la pola predittiva prima ancora di caricare la catena di riferimenti e poter quindi caricare solo quella opportuna e/o per evitare del tutto il caricamento di una catena di riferimenti per le chiavi ove il primo e l'ultimo documento coincidano sia se sono senza iwords sia se le presenze sono pari ad 1 avendo già l'ultima iword).

Questi ragionamenti portano a ritenere che:

1. Il VCB possa perdere le informazioni riferite a posizione e dimensione della catena di riferimenti per guadagnare, al contrario, il numero del primo documento referenziato (che sarebbe utile già ora anche non cambiando le carte in tavola).
2. Le catene di riferimenti molto lunghe sono spezzate in parti di dimensione nota, ciascuna non superiore ad una soglia stabilita. Le catene possono essere splittate e potrebbero anche scendere al di sotto della loro dimensione ottimale ma il puntamento ad esse avviene per mezzo di un sistema di chiavi in un BTree anziché un sistema di riferimenti da un blocco ad un altro.
3. All'atto dello spezzamento di una catena, la divisione in due deve tener conto di quale parte ospiterà la modifica da compiere e dove collocare un documento (e relative iwords) che dovessero risultare a cavallo tra lo spartiacque del “mezzo nodo”. Questo potrebbe portare a compiere un split di un nodo di dimensione 'X' sia in due nodi di pari dimensione, sia in due nodi che siano uno di dimensione 'X' e l'altro di dimensione 'X/2'.
4. In base a quanto detto precedentemente, può risultare vantaggioso compiere lo splitting di un nodo di dimensione 'X' in un nodo di pari dimensione ed uno di dimensione 'X/2' migrando nel primo i 2/3 del contenuto del blocco attuale e nell'altro solo 1/3. Lo sfrido complessivo sarà comunque inferiore a quello che si avrebbe nel gestire due nodi di dimensione 'X'. L'esigenza di compiere ulteriori modifiche nella stessa “zona” di documenti troverà posto a sufficienza portando a nuovo split del nodo maggiore se le modifiche insisteranno su tale area ovvero ad una crescita naturale e non dolorosa del nodo più piccolo sino al raggiungimento della dimensione 'X' precedentemente dichiarata. In quest'ottica può risultare conveniente, assumendo che la crescita proceda normalmente verso la fine dell'archivio, che lo splitting dia ai documenti più vecchi di un blocco la nuova zona di dimensione 'X/2' ed ai documenti più nuovi la dimensione 'X'. In caso di attività incrementale, però, conviene sempre che lo split porti a sinistra un nodo molto pieno, molto prossimo a saturazione, ed a destra, dove le cose continueranno a crescere, una parte potenzialmente piccola.
5. Analogamente, quando ad un blocco si tolgono dei riferimenti ad iwords o documenti tali da condurre la sua dimensione al di sotto della metà della soglia del blocco (evento comunque molto raro), anziché richiedere un nodo di dimensione minore e spostare l'intero contenuto in esso, può essere sensato mantenere il puntamento al blocco corrente identificandolo per una nuova dimensione, riferita alla potenza di 2 inferiore, e liberare la “seconda metà” del blocco stesso come se fosse appartenente, appunto, alla categoria di blocchi inferiore. Questo può avere benefici impatti in quanto a spostamenti delle catene.
6. I Buddy Files mantengono normalmente un indicatore dello spazio liberato che serve a dire quanta parte dell'archivio risulti non utilizzata (se pure utilizzabile). Potrebbe essere utile, quando si fanno interventi sui blocchi del Buddy File, mantenere aggiornato un indicatore di sfrido che ci dica quanta parte del file è non utilizzata ed inutilizzabile a meno che non si intervenga sui contenuti. Questo può risultare un valido indicatore di quando sia opportuno/consigliabile/necessario compiere un compattamento. Questo dato incrociato col numero di catene che risultano spezzate può dare il polso della situazione. Diversamente da questo ragionamento si può pensare che il record del VCB mantenga un contatore dello spazio effettivo coperto dalla somma di tutte le catene di riferimenti per quella chiave e che, contemporaneamente, mantenga un contatore dello sfrido assoluto. Questo può consentire, chiave per chiave, la dove lo sfrido assoluto supera, più o meno sensibilmente, lo sfrido che si avrebbe se la catena fosse espressa in un unico blocco di dimensione adeguata, di stabilire se quella chiave, e solo quella, può avvantaggiarsi da un'attività di ricompattamento.
7. Condizione mista: per le chiavi la cui catena di riferimenti non supera la dimensione massima dei singoli blocchi di catene non è necessario fare il doppio passaggio dal secondo B+Tree che offre il puntamento alla catena. Essa, ragionevolmente, può competere direttamente al VCB stesso, saltando un passaggio. Ciò comporta che il VCB non deve perdere

l'informazione inerente posizione e dimensione ma le deve organizzare diveramente.

1. La dimensione è quella reale della catena se abbiamo una catena sola ed è la sommatoria delle catene negli altri casi.
2. La posizione ha senso solo se c'è una sola catena. In presenza di più catene essa è nulla ed indica la necessità di accedere al B+Tree secondario.
3. Il contatore di sfrido può essere mantenuto in ambo i casi, per ragioni statistiche, anche se in presenza di una sola catena esso si deriva per differenza tra dimensione catene a relativa potenza di 2.

8. Nel BPT é possibile implementare una funzione che rinomini le chiavi vecchie in modo da riciclarle, invece che doverle mantenere inutilizzate e dover creare un nuovo indice di chiave e il corrispettivo elemento del VCB; questo procedimento è utilizzabile in compattamento.
    1. Per il BPT delle catene manterrà il numero dell'ultimo documento indirizzato (per facilitare la ricerca, si presta all'accesso “bt_read_key_greater”)

Nota supplementare sulle catene da scorrere in ordine inverso, idea che si affianca a quella delle catene di riferimenti spezzate in blocchi e può dirsi alternativa ad essa

Una catena di riferimenti diviene molto facilmente gestibile e modificabile, con tempi più contenuti specialmente quando la catena è lunga, se siamo in grado di accedere ad essa in senso inverso, ovvero dalla coda, visto che le modifiche interesseranno prevalentemente documenti più recenti. In questo modo una catena lunga viene modificata dalla coda e non dovendola scandire tutta dalla testa.
L'ipotesi prevede che ogni lotto riferito ad una chiave in un documento sia composto da un *Delta* Doc, una batteria di *Delta* IWord ed un *Delta* lenght o size che consenta di fare salti indietro al documento precedente nella catena.
Sapendo qual'è l'ultimo documento della catena e valutando il suo *Delta* Doc, con l'ausilio del *Delta* lenght siamo in grado di saltare al documento precedente.
La sequenza ha senso se la vediamo così:

DeltaDoc + *DeltaIWord + DeltaLenght

Per consentire la navigazione in senso opposto, è necessario essere in grado di leggere la *Delta* lenght in senso opposto e questo è un problema. Se si riesce a smarcare la problematica di interpretare una *Delta* lenght leggendola da destra verso sinistra, allora questa soluzione sarebbe percorribile. Parimenti, se la cosa fosse possibile, potremmo anche supporre di scegliere di navigare una catena di riferimenti in una direzione o in quella opposta a seconda del punto che stimiamo di dover raggiungere conoscendo l'estremo inferiore e superiore.
Ipoteticamente, se fosse possibile riconoscere i *Delta* Doc ed i *Delta* IWord da destra verso sinistra, allora sarebbe possibile anche navigare la catena senza avvalersi dei *Delta* lenght, ovvero senza fare salti indietro ma riconoscendo l'equivalente dell'attuale terminatore. In tal caso il superamento delle IWords verrebbe fatto senza valutarle nel vero senso della parola, mentre il superamento del *Delta* Doc servirebbe proprio ad identificare il documento precedente.

Migrazione dallo stato corrente a quello degli Index Store

Di seguito le attività svolte in molteplici punti da parte del server e come esse potrebbero essere svolte nell'architettura IndexManager/IndexStores.

Apertura dei files idx, vcb e ref. Attualmente compito della ApriDb_ApriArc() in apri_db.c. La funzione assume di poter aprire tutti i files o solo una porzione degli stessi sulla base di alcuni *flags* che riceve ed elabora ma di fatto non si fa mai alcun accesso agli archivi senza provvedere ad aprire tutti i files coinvolti quindi questa *finezza* può essere ignorata.
Non è più competenza della componente server. Il server istanzia l'IndexManager il quale, sulla base della configurazione caricata stabilisce quali IndexStore istanziare. Essi apriranno a loro volta i files essendo i soli candidati a farvi accesso. Questo comporta la necessità che l'IndexManager riceva il file di profilo in forma DOM ovvero che lo carichi autonomamente. Di fatto tutte le componenti in gioco, IndexManager ed IndexStore oltre ovviamente al resto del server, dovrebbero poter accedere alla configurazione archivio, ciascuno per il proprio scopo.
Caricamento del contenuto del file di profilo attualmente compito della LoadArcProfile() in prof_db.cpp.
Allo stato delle cose i parametri caricati interessano tanto la mera gestione dell'archivio quanto la gestione dei suoi indici. Tra le voci degne di nota abbiamo: i separatori per l'indicizzazione, il tempo di flush degli indici (che dovrà/potrà essere spostato come competenza agli IndexStores), modalità di stesura della chiavi di tipo data che comprendano anche il tempo ed il frazionamento in anno e anno e mese, presispozione per il legame tra canali di ricerca e corrispondenti canali del thesaurus.
Caricamento della modalità di indicizzazione dei canali esplicitati, attualmente compito della DbSw_PreloadIndexMode in dbsw.c.

Apertura dei files idx, vcb e ref. Attualmente compito della ApriDb_ApriArc() in apri_db.c. La funzione assume di poter aprire tutti i files o solo una porzione degli stessi sulla base di alcuni *flags* che riceve ed elabora ma di fatto non si fa mai alcun accesso agli archivi senza provvedere ad aprire tutti i files coinvolti quindi questa *finezza* può essere ignorata.

Questa funzione viene chiamata per compiere il caricamento delle chiavi configurate per una data primary_node dell'archivio in esame. Se l'archivio conta numerose/numerosissime chiavi per ogni unità informativa, quest'operazione più risultare gravosa, anche per questo scopo si è pensato di compierla solo su richiesta.
Di fatto quest'atteggiamento non ha più molto senso in quanto il caso in cui un archivio sia composto da numerosi diversi tipi di unità informative ma non ne venga compiuto l'integrale caricamento sino a quando non se ne fa uso non trova più reale applicazione[4)].
I caricamento integrale di tutte le chiavi di tutte le U.I. è ora affidato alla DbSw_IndexModeFullPreloader(), sempre in dbsw.c, dd è ragionevole che sia quella l'obiettivo dell'attività di caricamento.
Qui sorge una questione: l'IndexManager necessita di compiere questo caricamento per saper collocare sui diversi IndexStore le diverse chiavi ma lo stesso riguarda ed interessa l'IndexStore che deve sapere che cosa può e deve fare di tale chiave. I due devono “comunicarsi” quest'informazione.
C'è dell'altro... Se ad un'unità informativa viene associato un IndexStore per le sue chiavi secondarie è naturale che anche le chiavi sottintese affluiscano ad esso. Se però ci sono due IndexStore per le chiavi secondarie di quell'unità informative, le sottintese dove affluiscono?
Ad ogni modo queste informazioni servono spessissimo anche al motore di indicizzazione in quanto deve sapere come costituire le chiavi. Pari discorso va fatto in fase di ricerca. Entrambe questi compiti, quindi, andrebbero assolti da parte dell'IndexManager così da lasciare il server del tutto ignaro del resto?
Il server, in questo caso, si occuperebbe esclusivamente della persistenza dei dati, demandando all'IndexManager ogni ulteriore attività, compresi indici on e off line.

Che sia così o meno, le informazioni sulle chiavi devono poter permeare sino ai livelli superiori dove operazioni quali la consultazione dei vocabolari devono poter agire in modo diretto richiamando l'IndexManager senza ulteriori complicazioni. Perché questo avvenga ed abbia senso, però, qualsiasi chiave esposta al mondo esterno dall'IndexManager deve risultare univoca, indipendentemente da quale sia l'IndexStore cui essa appartiene.

Identificazione delle caratteristiche di una chiave, attualmente compito della DbSw_GetFieldIndexModeEx() in dbsw.c

Questa funzione serve al di sopra dell'IndexManager se ricerca ed indicizzazione sono svincolati da esso. Se invece sono nel suo “core” allora l'esposizione al mondo esterno di una simile funzione, contestualmente alla visitazione dell'elenco delle chiavi, tutte, in costituzione elenco (Vds. catxml.c) ha senso solo in un limitato set di casi.
Viene gestita una cache che tiene traccia di tutti gli XPath per i quali è stata calcolata correttamente una chiave. Questa cache potrebbe essere riversata su disco ed annullata quando il timestamp del file .conf.xml d'archivio risulta più recente della stessa. Ciò renderebbe il loading della configurazione dell'archivio estremamente più rapida. Del resto, se la cache c'è ed è valida, allora vuol dire che come minimo tutte le chiavi del profilo vi sono state caricate, anche assumendo che non sia stata compiuta, dal client, alcuna richiesta in materia di struttura chiavi archivio. La cache può essere anche annullata, etichettandola come platform dependent, quando la piattaforma non combacia [5)].

Gestione degli Alias (differente dalla gestione degli Also), attualmente compito della DbSw_GetFieldAlias() in dbsw.c

La funzione viene evocata sia in fase di composizione dello schema delle chiavi (catxml.c) da inviare al client sia in ricerca quando si deve tradurre un campo che non ha un proprio vocabolario in un altro ed ovviamente in indicizzazione, quando si deve creare/acquisire la chiave di un canale che è così configurato.

Gestione della mappa, attualmente compito della DbSw_GetFieldMapMode() in dbsw.c

Ci si riferisce ai prunable branches ed alle complicazioni che ruotano attorno ad essi. Purtroppo è ampiamente immerso nella configurazione delle chiavi ma non afferisce ad esse dal momento che si parla espressamente di attività riguardanti il trattamento delle unità informative.

Gestione degli alias di ricerca (e non solo), attualmente compito della DbSw_Preloader() in dbsw.c

Questa funzionalità non interessa l'IndexManager e può essere tenuta al di fuori di esso. Il *Search Alias* è un capriccio, se vogliamo chiamarlo così, che appartiene ed interessa solo chi fa ricerche. Effettivamente potrebbe anche “star dentro”, ma è tutto da vedere. Lo vedo come un inutile appesantimento (come del resto l'ho sempre reputato per quanto risulti una semplificazione applicativa).

Gestione dei “morsetti”

Questa tecnica interessa sia la parte di server che si occupa di indici sia quella che non se ne occupa. Ciò comporta che sarebbe inutile separarle del tutto. O la parte “dati” comunica i morsetti alla parte “indici” oppure le due parti devono agire autonomamente sul file di configurazione per trarne ciascuno la propria configuraione pertinente.

Gestione Univocità

Combinazione mista. La regola di univocità è propria della gestione dei dati ma in qualche modo ha senso anche nella gestione degli indici, specie se si assume che un valore seriale è nativamente univoco quando viene assegnato. Effettivamente non è compito di un motore di indicizzazione preoccuparsi di fare verifiche di univocità ma certo sta a lui eseguirle

Gestione delle chiavi condizionate

Sarebbe il caso di rimuoverlo dalla versione 23 a favore dell'uso di XSLT che, di fatto, è infinitamente più preciso e versatile nella definizione di queste chiavi fatte espressamente per E.Rendina.

Chiavi Blind

Viene registrata la chiave nel btree con al seguito i suoi dati di configurazione. Naif ma da non ignorare del tutto.

La doppia mappa, idee del 05/10/2010 e giorni successivi

La doppia mappa assolve a due scopi.
Consentire le attività transazionali e consentire, in sede di indicizzazione off-line non bloccante, che le modifiche risultino eseguite come rimozioni del vecchio documento ed inserimento in coda del documento che lo modifica. In questo modo, con la doppia mappa, l'ordine naturale dei documenti rimane invariato e le applicazioni, che conoscono solo i numeri di documenti (logici) della mappa *esterna*, non notano la differenza.

Il concetto di doppia mamma si esprime quindi appunto con una mappa *esterna* di numeri logici di documenti, che sarà nota principalmente alle applicazioni, ed una mappa *interna* di numeri fisici di documenti.
In un momento nel tempo, ad un documento della mappa *esterna* potrà corrispondere più di un documento della mappa *interna*[6)].
Per contro, ad un documento della mappa *interna* corisponde uno ed uno soltanto dei documenti della mappa *esterna*.

Quest'analisi viene condotta prendendo come assunto che, da ora in avanti, si farà sì che i comandi esposti dal server siano (ovvero debbano divenire) *stateless*. Questo comporta che parlando ad esempio di transazioni, esse devono essere in tutti gestite dal server.
Vediamo le complicazioni che ne derivano. Un utente apre una transazione e fin qui tutto regolare. Nel momento in cui volesse aprirne un'altra essa potrebbe essere o una transazione in tutto distinta dalla precedente, ovvero una transazione “innestata” in essa che dev'essere condotta a compimento ovvero rigettata prima di quella che la “contiene”. Dal momento che l'utente non deve avere l'onere di riconoscere e ricordare la propria transazione si esclude automaticamente la possibilità che tale transazione sia “parallela e distinta” dalla precedente. Essa può solo essere una transazione innestata.
Questo non toglie che più utenti possano avere ciascuno il proprio set di transazioni innestate e che esse non debbano pestarsi i piedi reciprocamente.
Se è vero che questa considerazione può essere implementata senza troppe complicazioni (e l'implementazione dovrà tenerne conto) assumiamo come obiettivo iniziale di consentire una sola transazione per utente e che quindi non si preveda neppure il caso delle transazioni innestate.

Tesi #1: Un documento coinvolto in una transazione non può essere coinvolto in nessun'altra. Di fatto si potrebbe consentire a più utenti di intervenire distintamente sullo stesso documento iniziale, creando versioni separate dello stesso, ma il primo che fa “commit” inficia il lavoro di tutti gli altri. Ci pare più ragionevole negare la possibilità di intervenire su un documento a monte, anziché lasciar lavorare e scoprire solo a valle se l'insieme dei passi compiuti può essere completato o meno. Insorge una possibilità, anche se remota, che va presa in esame. Se una tranazione usa un documento, nel senso che ne usa i contenuti, pur non intervenendo sullo stesso, esso potrebbe a sua volta far parte di una transazione di altro utente. Se essa va a buon fine o meno per l'altro utente, i valori estratti dal documento per portare a termine la prima transazione potrebbero non corrispondere a quelli del documento al termine delle due distinte transazioni.

Suggerimento #1: Quando una transazione si avvale di un documento anche solo per utilizzarne il contenuto è buona norma caricarlo in modo bloccato così da farlo appartenere alla propria transazione e garantirsi di conseguenza che non subisca modifiche prima che la stessa sia stata completata.

Tesi #2: Se ci sono transazioni innestate, il commit sulla più interna comporta il commit di tutte le altre, visto che lo stesso si basa sui documenti già inseriti o modificati da quelle a monte. Il rollback, invece, agisce ovviamente solo sulla più interna.

Abbiamo detto quindi che come prima implementazione non ci saranno più di una transazione per utente, ne innestate ne distinte[7)], si ha comunque l'esigenza che la consultazione della mappa *interna* consenta di identificare correttametne ciascun documento appartenente ad una transazione ed identificare, nella catena che essi compongono[8)], quale sia la “versione” del documento di nostra competenza.
Per ottenere un simile comportamento, la mappa interna dovrà prevedere:

- Per il documento originario, ovvero l'inizio della catena:
  - Id della Transazione se ce n'è una presente[9)].
  - Il puntamento all'ultima versione del documento nella transazione[10)] attuale.
  - Volendo, il puntatore al prossimo documento della catena di transazione che corrisponderà solitamente con l'ultimo.
- Per ogni documento della catena di transazione:
  - Id della Transazione[11)].
  - Il puntatore al precedente documento della catena di transazione.
  - <color grey>Volendo, il puntatore al primo documento della catena di transazione che corrisponderà solitamente al documento di cui al passo precedente</grey>

Tesi #3: La mappa *esterna* conosce sempre e comunque il primo anello della catena di una transazione, ovvero il documento originario in modo che, raggiunto quello, si possa procedere sino al termine della catena di transazione. Puntare al primo anziché all'ultimo ci consente di percorrere la catena se e solo se siamo l'utente interessato da quella transazione. Per tutti gli altri l'accesso è diretto.

All'apertura di una transazione viene assegnato un ID di transazione a quell'utente[12)]. Di fatto non interviene ancora alcuna modifica nella mappa interna.
Da questo momento, ogni inserimento o modifica che l'utente compie si traduce in un intervento sulla mappa *interna* e su quella *esterna*.
In caso di inserimento, il documento viene creato tanto nella mappa *esterna* quanto in quella *interna*[13)]. In questo caso, nella mappa interna, il primo e l'ultimo documento della catena di transazione sono lo stesso record, non ci sarà riferimento a documenti precedenti e seguenti ma l'ID di transazione sarà valido.

In caso di modifica si genera un nuovo documento nella mappa *interna*, si modifica il documento originario della mappa *interna* in modo che referenzi il successivo (e vice versa) e si colloca opportunamente l'ID di transazione nei due record.

Commit: su documenti inseriti, si modifica la sola mappa *interna* rimouvendo l'ID di transazione ed elimando le informazioni non più utili e finalizzate a descrivere la catena di un solo documento. Su documenti modificati si modifica la mappa *esterna* perché punti all'ultimo documento della catena di transazione (che viene ripulito come nel caso precedente), poi da esso si procede a ritroso per annullare (ossia cancellare) tutti i documenti precedenti della catena sino al documento originario.

Rollback: Sui documenti inseriti essi vengono scartati dalla mappa *esterna* e rimossi, quindi cancellati, resi non più ricercabili. Sui documenti modificati, la mappa *esterna* continua a puntare correttamente il documento originario, che viene epurato dalla informazioni di transazione, e di seguito, partendo dal successivo o dall'ultimo della catena di transazione, tutte le “versioni” del documento in esame vengono rimosse.

Nota: anziché toccare gli indici quando si cancella un documento, metterlo nella lista nera. Diversamente dagli altri questo non verrà mai rimosso.

Doppia mappa, Transazione, Indici Fuori Linea, Archivio Cluster(Arbiter, Dispacther)

Introduzione

La **doppia mappa**, o per meglio dire la coppia mappa logica e mappa fisica, è stata ideata per mantenere più versioni dello stesso documento. Un server può avere più di un documento a seguito:

- di una transazione, un utente apre una transazione modificando il documento, quindi dovrà essere visibile per tutti la vecchia versione del documento e solo per chi ha la transazione aperta il nuovo documento.

- di un indicizzazione fuori linea, il server considera i documenti modificati come nuovi documenti inseriti,e segna i documenti da modificare come cancellati (non accessibili e non oggetto di ricerca).

Per **Archivio Cluster** in questo documento un archivio frazionato su diversi Folder ovvero Macchine ovvero risorse. Le operazioni su un Archivio Cluster possono essere svolte da un **Arbiter**. Un Arbiter è entità, a cui è delegato il compito di decidere su quale frazione deve essere inserito o ricercato un documento. Un Dispachter è altresi un'entità che decide dove si trova l'Arbiter per un determinato archivio. In questa sezione si rimarrà volutamente vaghi perchè la progettazione di un Server eXtraWay, che opera in modalità cluster non è stata ancora compiuta.

Transazione modalità d'uso

Transazione: Introduzione

Una transazione sono una serie di operazioni che si devono svolgere in modo sequenziale e atomico: nessuna altra operazione che può alterare l'atomicità può essere svolta. Nell'accezione di eXtraWay: - la modifica di un documento D durante la transazione T impone che le modifiche del documento D potranno essere svolte solo nella transazione T fino alla chiusura della stessa - un documento D inserito durante la transazione T potrà essere visto solo dalle operazioni in T Una transazione può essere innestata: una transazione t1 è innestata in t2, se t2 viene aperta utilizzando l'ID di transazione di t1 e viene chiusa prima della terminazione di t1 (quindi se si chiude t1 tutte le transazioni innestate in t1 dovranno essere chiuse).

Transazione: Generazione ID

L'ID di transazione deve essere univoco rispetto a tutte l'entità coinvolte (potrebbe essere generato dalla uri). Un modo semplice per creare un ID univoco per la transazione è quello di comporre ID tenendo conto:

User
Per identificare a chi appartiene quella transazione
URI (Uniform Resource Identifier)
Nel caso la transazione del Dispacher o dell'Arbiter: per identificare la macchina che ha effettuato l'apertura della transazione (e che dovrà chiuderla)
Archivio
Per identificare a chi appartiene quella transazione
Num
(caso statefull)Per permettere ad un user di aprire più transazioni ,per lasciare più libertà all'applicativo questo valore potrebbe venire dall'applicazione stessa
Progressivo
Nel caso la transazione innestate per identificare il livello e a quale transazione appartiene

Ad esempio un ID di transazione potrebbe essere composta:

1. TRANSACTION_USER_SITE_DISPACTHER_NUM_PROGRESSIVE (a livello Dispatcher)
2. TRANSACTION_USER_SITE_ARBITER_NUM_PROGRESSIVE (a livello Arbiter)
3. TRANSACTION_USER_ARCHIVE_NUM_PROGRESSIVE (a livello di Archivio cluster)

Vedremo che poi verrà composta in altro modo per renderne facile l'uso.

Transazione Uso

Una transazione può essere aperta da una qualsiasi delle entità (Cluster,Archive,Arbiter,Dispatcher) e sarà conosciuta e visibile per tutte l'entità sottostanti. Quindi ad esempio se apro una transazione a livello Dispatcher dovrà essere inserita da quelle conosciute livello Archivio Cluster e il viceversa non è vero: una transazione aperta a livello Archivio Cluster non sarà conosciuta dal livello Dispatcher.

La procedura di creazione (apertura) prevede che: - l'ID sia creato dall'entità proprietaria - una volta creata sia passata all'entità sottostanti (per permettere di identificare le operazioni nelle sottoentità)

La transazione appartiene all'utente e all'entità in modo congiunto (se l'utente richiede l'utilizzo di una transazione da un entità diversa da quella di creazione, questa non dovrà essere visibile).

Una transazione per poter essere committata (chiusa) dall'entità proprietaria dovrà verificare che le sue sottoentità ne permettono il commit: se una sola delle sue entità risponde in modo negativo bisognerà fare il rollback di tutte le transazioni (in modo ricorsivo) di tutte le sottoentità (per garantire la coerenza delle operazioni\dati) della transazione.

Ogni entità manterrà le transazioni che afferiscono all'entità stessa. Per semplicità quest'operazione potrebbe essere fatta creando una mappa ad albero con le chiavi di transazione. Le chiavi dovranno identificare le sottoentità coinvolte al fine di effettuare un rollback: _<user>_<entita padre>_<num>_<sottoentità>_ In questo modo si può identificare facilmente quali siano le sottoentità su cui fare il commit\rollback ed a chi rispondere sul esito della transazione. A livello più basso saranno inserite delle chiavi per identificare quale documento l'utente potra modificare, visualizzare.

Transazione: Chiusura Forzata

Una transazione non chiusa è chiudibile forzatamente da un amministratore ovvero da un timeout: questo comporterà il rollback delle operazioni afferenti alla transazione. In questo si evita l'impossibilità di intervento su dei documenti per un tempo indefinito e in situazioni patologiche.

Transazione: Ricerca di Transazioni

Utilizzando una mappa con chiavi _<user>_<entita padre>_<num>_<sottoentità>_ potranno essere facilmente trovate quali sono le transazioni aperte da un utente (ed è chiaramente possibile all'entità proprietaria). Inoltre la ricerca dei documenti visibili da una transazione a livello Archivio potrà essere fatto utilizzando chiavi _<doc>_<user>_ con valore aggregato il documento da visualizzare.

Transazione Bloccanti e Seriali

L'utilizzo di un seriale\seriali di protocollo (seriali sequenziali) durante un'operazione deve essere esclusiva: poichè la transazione deve essere realizzata come una serie di operazioni che funge da singola operazione atomica implica che se una transazione chiede l'uso di un seriale durante la sua esecuzione questo diverrà bloccante per tutte le altre operazioni che ne avessero necessità: con la conseguenza che fino alla conclusione della operazione, che deve realizzarsi in modo atomico, che ha richiesto per prima il seriale nessun altra operazione potrà richiedere un seriale. Questo modo ci garantisce di operare in modo coerente con l'utilizzo dei seriali protocollo. Il blocco sarà garantito fino alla conclusione dell'operazione (in casi estremi la conclusione dell'operazione in modo negativo potrà essere svolta in modo amministrativo). L'esito negativo di un operazione porta al rollback dei seriali coinvolti nell'operazione stessa.

Doppia Mappa

Doppia Mappa Introduzione

La doppia mappa nella sua formulazione iniziale doveva mantenere copie dello stesso documento a seconda del richiedente. Nella sua formulazione più attuale si potrebbe dire ad ogni entità possiede una mappa (resource map) per la dipendenza delle sottorisorse (vedi transazioni). Questa mappa deve mantenere le proprietà d'indirizzamento (dove si trova la sottoentità), come deve essere visualizzata (nel caso dei documenti) e le dipendeze delle sottorisorse (transazioni).

Doppia Mappa Resource Map e Documenti

Nell'archivio base (che da accesso ai documenti) la resource map permette la visualizzazione di diverse copie di documenti utilizzando chiavi del tipo:

- DOC<doc>_<user>_<ID Transazione> (nel caso delle transazioni)
- DOC<doc> (per identificare il documento da rimpiazzare nella reindicizzazione)

Durante il processo di ricatalogazione per ogni modifica sul documento A verrà inserito un nuovo documento B (il risultato della modifica) e verrà segnato cancellato il documento A nella mappa dei documenti e nella Resource Map verrà inserita una chiave DOC<A> con valore B (ci riferiremo a questo sistema col nome di Jump). Nota: Le selezioni intervenute prima del completamento dell'indicizzazione andranno annullate e rieseguite (in quanto selezionano documenti che potrebbero essere cancellati).

Seriali Protocollari

Seriali Protocollari Generazione

Alcune considerazioni vanno fatte sulla generazione dei seriali. Il servizio che produce i seriali protocollari dovrebbe essere concepito come servizio esterno e il blocco di tale servizio dovrebbe essere fatto come qualsiasi altra risorsa nella mappa Resource Map. Il servizio dovrebbe essere bloccato come avviene per i documenti dell'archivio. Il servizio dovrebbe essere demandato all'entità che ha la visione completa di come quel seriale viene generato: per gli archivi cluster dovrebbe essere gestito dall'arbiter;

per i seriali protocollari multi archivio dovrebbe essere demandato al Dispatcher.

Seriali Protocollari Uso

Non ha molto senso (anche se è permesso) avere più seriali protocollari sullo stesso documento.

Seriali Protocollari Costruzione

Se il seriale protocollare può essere generato dallo stesso slave il sistema di prenotazione tramite chiave non si applica perchè questo è garantito dal semplice uso della sezione critica dell'archivio (seriale protocollare monoarchivio). Nel caso in cui l'architettura sia su più livelli verrà effettuata una richiesta come se fosse una transazione innestata sull'entità dei seriali. Dal punto di vista delle chiavi troverò nella resource map dunque:

1. (sulla macchina richiedente) _<user>_<Transizione ID>_<Macchina Seriali Protocollari>_<SerialResource>
2. (sulla macchina dei seriali) _<user>_<Richiedente>_<Transizione ID>_<SerialResource>
   <SerialResource><user>_<Richiedente>_<Transizione ID>

In questo modo potrò gestire semplicemente la generazione dei seriali.

1)
Vale a dire dei diritti
2)
ovviamente chi lo scrive, superata una certa soglia dimensionale, abbandona il file e ne crea uno nuovo e così via...
3)
Per la prima porzione, indipendentemente da quale sia l'effettivo primo documento, conviene indicare che il punto d'origine è sempre il documento numero 1 così da poterlo far crescere anche per inserimenti di chiavi riferimenti a documenti antecedenti il primo referenziato
4)
Ogni client chiede come minimo lo stato delle chiavi e per farlo provoca il caricamento di tutte le informazioni di tutte le unità informative
5)
Little & Big Endian
6)
Ciò indica che è in corso una transazione
7)
In un momento successivo sarà possibile avere transazioni di questi tipo ricorrendo ad uno scambio di ID di transazione che rende il comando *statefull*
8)
In prospettiva futura di una catena composta da più di 2 documenti
9) 11)
,
Attualmente non ha molto senso ma lo assumerà col tempo
10)
Nelle transazioni nidificate...
12)
Nella modalità *stateless* questo ID ha finalità interne e non viene tornato all'utente.
13)
Poi vedremo come sarà possibile che solo l'utente che lo ha creato lo veda