ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

ELABORAZIONE DI STATISTICHE NEL RESOURCE SHARING TRA BIBLIOTECHE

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
ZAID CHEIKH IBRAHIM

Correlatori:
Dott. RABIH KAHALEH
Dott.ssa SILVANA MANGIARACINA
Dott. ALESSANDRO TUGNOLI

Sessione II - Primo appello
Anno Accademico 2023-2024

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

ELABORAZIONE DI STATISTICHE NEL RESOURCE SHARING TRA BIBLIOTECHE

Relatore:
Chiar.mo Prof.
FABIO VITALI

Presentata da:
ZAID CHEIKH IBRAHIM

Correlatori:
Dott. RABIH KAHALEH
Dott.ssa SILVANA MANGIARACINA
Dott. ALESSANDRO TUGNOLI

Sessione II - Primo appello
Anno Accademico 2023-2024

"Education never ends, Watson. It is a series of lessons, with the greatest for the last."

Sherlock Holmes in *"His Last Bow"*,
Sir Arthur Conan Doyle

Abstract

In questa tesi viene illustrato il percorso di studio, progettazione e integrazione di Elasticsearch, un motore di ricerca utilizzato per indicizzare e analizzare grandi quantità di dati in modo efficiente e veloce, per l'elaborazione di dati statistici in Talaria, un software open source per lo scambio di documenti tra biblioteche. Talaria è un'applicazione web basata su Laravel, React e MySQL. L'obiettivo del presente lavoro è quello di rendere più efficiente l'elaborazione di statistiche sui dati generati dalle richieste di documenti tra biblioteche. Grazie all'integrazione di Elasticsearch, Talaria può mostrare le statistiche in modo più rapido e accurato rispetto al predecessore, NILDE.

La dissertazione descrive come è stato progettato e implementato Elasticsearch in Talaria, partendo dalla configurazione del backend, sviluppato in Laravel, fino alla realizzazione del frontend con React, che consente di visualizzare statistiche avanzate in modo interattivo. I test, eseguiti sia su dati reali che fittizi, hanno evidenziato miglioramenti significativi nelle prestazioni rispetto alla precedente architettura, basata esclusivamente sugli strumenti di indicizzazione e ricerca messi a disposizione da MySQL.

Infine vengono proposti alcuni sviluppi futuri, come l'ampliamento della gamma di statistiche offerte e l'uso del machine learning per analisi statistiche più avanzate.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

In questa tesi si propone l'utilizzo di Elasticsearch, un motore di ricerca basato su Apache Lucene e utilizzato per ricercare, indicizzare, immagazzinare e analizzare grandi quantità di dati in modo efficiente e veloce, per l'elaborazione di dati statistici nel resource sharing tra biblioteche.
Il caso di studio preso in esame tratta l'elaborazione dei dati statistici prodotti in un sistema per lo scambio di documenti tra biblioteche.
Elasticsearch è stato implementato e integrato in un *Resource Sharing Management System* di nuova generazione: Talaria, un software open source sviluppato dal CNR di Bologna, basato su Laravel, React JS (con React Redux + Saga) e MySQL.
Le statistiche sono indispensabili per monitorare l'andamento dei servizi di resource sharing tra biblioteche, sia a livello globale che per ogni biblioteca che utilizza il sistema. I dati delle transazioni (ossia le richieste e le forniture dei documenti tra biblioteche) dipendono dal numero di biblioteche aderenti al sistema, che può avere valenza nazionale o internazionale, e dal numero medio di transazioni giornaliere per biblioteca. I dati elaborati consistono nelle statistiche aggregate per anno o per mese. Le transazioni possono essere analizzate secondo varie dimensioni: lo stato della transazione, il tempo impiegato, il tipo di utenza, l'ente di appartenenza, la nazione ecc...
Si stima che in un sistema di resource sharing nazionale, come ad esempio NILDE, operativo in Italia da 15 anni, che oggi conta 1000 biblioteche e un volume di scambi di circa 1000 richieste giornaliere, il volume dei dati trattati può essere dell'ordine di qualche milione di record. Per questo si rende necessario avvalersi di

strumenti efficaci che permettano l'elaborazione dei dati statistici in tempo quasi reale. Elasticsearch, grazie al modo di indicizzare i documenti in maniera efficiente e veloce, è un ottimo strumento per l'elaborazione di statistiche di una grande mole di dati, come può essere il numero di transazioni dell'intero sistema.
Il primo capitolo è dedicato alla descrizione dello scenario in cui si inquadra il lavoro di tesi. Viene prima illustrata la situazione del resource sharing tra biblioteche in Italia e a livello mondiale con un focus sull'elaborazione delle statistiche. Successivamente viene descritto Talaria e ne viene confrontata l'architettura e le funzionalità con il precedente software NILDE. Infine viene introdotto Elasticsearch nel suo complesso, partendo dalla sua architettura e funzionamento, e presentando i suoi vantaggi e svantaggi.
Nel secondo capitolo vengono illustrate le scelte progettuali per l'integrazione di Elasticsearch in Talaria.
Il terzo capitolo illustra le fasi implementative effettuate. Il codice sviluppato per il nuovo modulo delle statistiche di Talaria è disponibile su GitHub[1].
Infine, si conclude con l'analisi dei risultati ottenuti, la valutazione dei benefici che questa tecnologia ha apportato a Talaria e i possibili sviluppi futuri.

[1] `https://github.com/Zaid1710/talaria-statistics`

1 Stato dell'arte

1.1 Resource sharing tra biblioteche

I primi esempi di resource sharing interbibliotecario (o *Interlibrary Loan*, ILL) risalgono al VIII secolo nell'Europa occidentale. Durante il Medioevo, i monasteri erano ricchi di manoscritti che venivano scambiati tra monaci e suore in continuazione. L'aumento di richiesta di manoscritti portò un incremento della produzione e nel XII secolo vennero creati i primi cataloghi. Gli scambi effettuati, però, erano tutti geograficamente vicini.
Nel Rinascimento le biblioteche in Italia, Francia e Inghilterra erano ricchissime di documenti e molti studiosi cercavano di prendere in prestito manoscritti. Nacque quindi una prima forma di prestito bibliotecario internazionale di documenti. Circa un secolo dopo nacquero anche i cataloghi individuali delle biblioteche per facilitare la ricerca delle collezioni.
Lo studioso francese Peiresc provò a creare uno scambio interbibliotecario tra la *Bibliothèque nationale de France* a Parigi e le biblioteche Vaticane e Barberini a Roma. Il Cardinale Barberini, allora nipote del Papa, aiutò in ogni modo Peiresc offrendo la possibilità di evadere ogni richiesta di prestito della *Bibliothèque nationale de France*. Purtroppo l'idea di Peiresc fu rifiutata a causa delle politiche francesi che vietavano il trasporto di manoscritti.
Solo verso la fine del XIX secolo si ricominciarono ad effettuare scambi interbibliotecari internazionali. Infatti, negli Stati Uniti fu suggerita una prima versione di un sistema di scambio interbibliotecario tra le biblioteche pubbliche.
In Europa, il governo austriaco dichiarò che le biblioteche locali potevano prestare

a biblioteche estere senza bisogno dell'autorizzazione da parte del governo.
Fino alla Prima Guerra Mondiale gli Stati Europei praticarono una specie di scambio internazionale interbibliotecario di un volume di circa 6500 - 12.500 scambi annuali. Dopo lo stop causato dalla prima Guerra Mondiale la Germania e l'Inghilterra ripresero lo scambio interbibliotecario tra paesi europei e intercontinentali.
Nel 1927 fu fondata la *International Federation of Library Associations* (IFLA) per lo scambio di idee e la promozione della cooperazione internazionale. Nel 1935 venne richiesto all'IFLA di pubblicare un regolamento per standardizzare lo scambio di documenti e fu pubblicato l'anno seguente.
La seconda Guerra Mondiale distrusse le pratiche dello scambio interbibliotecario e l'IFLA, nel 1954, emanò nuove regole che portarono una rapida crescita dello scambio internazionale. Agli inizi degli anni '70 in Italia non si rilevava un numero altissimo di richieste: 1750 richieste di prestito di materiale dal Consiglio Nazionale dall'estero di cui solo 550 furono evase e 375 inviate all'estero. La situazione migliorava nettamente verso la fine degli anni '70 con un numero di richieste evase di 30.000. [MI07]
Oggi l'IFLA ha una sezione dedicata al Document delivery (DD) e Resource sharing (RS) il cui obiettivo è quello di estendere e migliorare il Document delivery nazionale e internazionale attraverso l'uso delle tecnologie e la cooperazione interbibliotecaria. Quando si parla di ILL si intendono gli scambi interbibliotecari di documenti sia *returnables*, come ad esempio i libri, che devono essere restituiti alla fine del prestito, che *non-returnables*, come ad esempio le fotocopie di capitoli di libri o articoli, che non necessitano di essere restituiti. Il DD tratta solo gli scambi dei documenti *non-returnables*. Al giorno d'oggi questo concetto si amplia includendo anche gli strumenti messi in atto dalle biblioteche per migliorare la collaborazione, ad esempio i cataloghi delle biblioteche, in un unico grande concetto: il Resource sharing (RS).

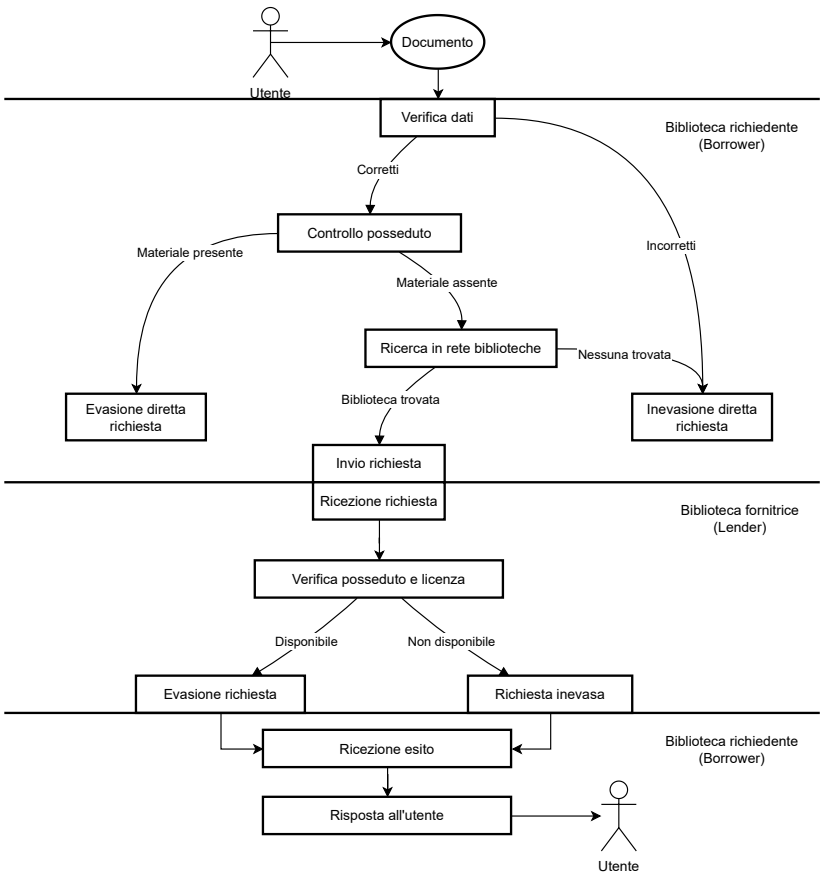


Figura 1.1: Illustrazione del processo base di RS

Resource sharing in Italia: NILDE

Network interlibrary document exchange (NILDE)[1] è una piattaforma web per la gestione delle richieste di Interlibrary loan (ILL) progettata e sviluppata nel 2001 dalla biblioteca del CNR dell'Area di ricerca di Bologna.
NILDE è la rete di ILL più grande in Italia, con circa un migliaio di biblioteche e 100.000 utenti attivi.
Il software NILDE supporta l'intero flusso di lavorazione delle richieste di Document delivery (DD), partendo dalla richiesta dell'utente alla propria biblioteca, la

[1] `https://nilde.bo.cnr.it/`

gestione della richiesta, la ricerca e l'invio ad una biblioteca fornitrice da parte della biblioteca dell'utente (richiesta di *borrowing*), l'evasione o inevasione della richiesta da parte della biblioteca fornitrice (per la quale la stessa è una richiesta di *lending*) e termina con la notifica all'utente di evasione o inevasione della richiesta, come schematizzato nella Figura 1.1.
NILDE per gli utenti assolve alla funzione di un *reference manager*, ovvero un utente può inserire i riferimenti bibliografici manualmente o automaticamente da database conformi allo standard OpenURL e richiedere un documento alla propria biblioteca con un semplice click [NA05]. OpenURL è un collegamento progettato per trasportare informazioni specifiche, come metadati di una risorsa, verso un server che interpreta i dati ricevuti e restituisce all'utente la risorsa richiesta. Si tratta di uno standard aperto, non proprietario, sviluppato dal NISO[2]: un'organizzazione senza scopo di lucro riconosciuta dall'ANSI[3] per la definizione di standard. Un principio di NILDE è la reciprocità e la cooperazione tra pari, ovvero, tra tutte le biblioteche. Per questo motivo è presente una classifica basata sull'indicatore di reciprocità tra le richieste di borrowing e quelle di lending. La classifica suggerisce all'utente quale biblioteca fornitrice è la più adatta a ricevere la richiesta per mantenere la reciprocità tra biblioteche. [MT12]

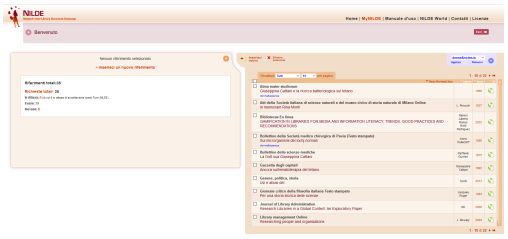


Figura 1.2: Homepage di NILDE lato utente

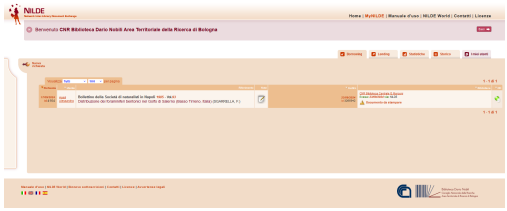


Figura 1.3: Homepage di NILDE lato biblioteca

[2] https://niso.org/
[3] https://www.ansi.org/

Resource sharing a livello internazionale: Progetto HERMES e RSCVD

Resource Sharing in the Time of COVID-19 (RSCVD)[4] è un servizio open source di RS internazionale tra biblioteche volontarie che si impegnano a garantire un accesso universale ed equo alle informazioni soprattutto durante periodi di crisi, come quello del COVID-19. Il progetto nacque nel 2020 dalla commissione IFLA DDRS durante il periodo del COVID-19 per facilitare il Document delivery digitale durante il lockdown. Le richieste erano visualizzate in un foglio di calcolo condiviso e i bibliotecari potevano fornire i documenti digitalmente e gratuitamente. Inizialmente il foglio di calcolo era molto basilare e ridotto all'indispensabile in quanto è stato ideato in circa un mese. Successivamente fu migliorato con l'aggiunta di nuove funzionalità, come ad esempio una data di scadenza della richiesta, dopo la quale veniva nascosta. Nel 2021 l'iniziativa RSCVD fu finanziata dal programma

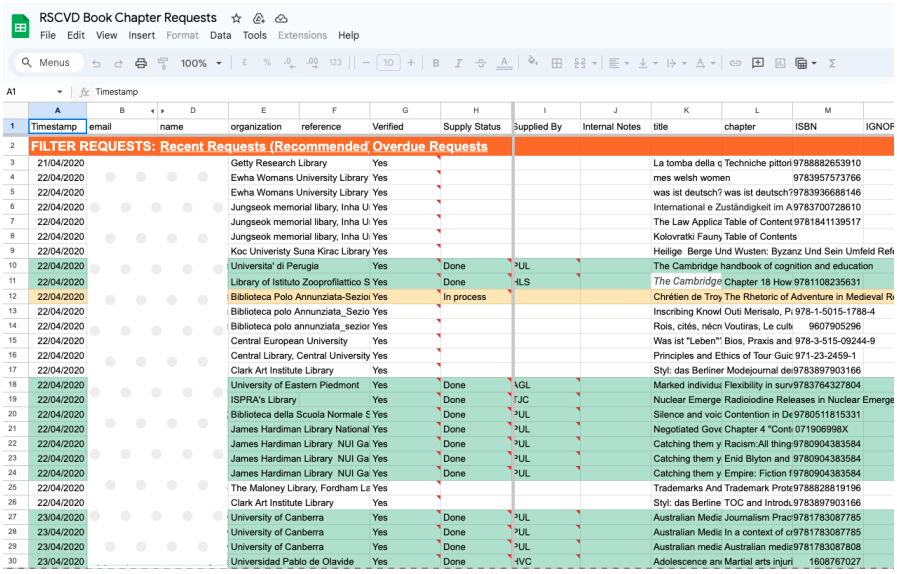


Figura 1.4: Foglio di calcolo[5]

europeo Erasmus Plus attraverso il progetto HERMES per sostenere RSCVD in

[4] `https://rscvd.ifla.org/`

[5] Fonte: `https://video.cnr.it/w/ePc6zWWWSAbDofkPzFudfR?start=53m38s`, visitato il giorno 07/10/2024.

futuro, oltre al periodo COVID-19. [LMM23]
Il progetto HERMES: *Strengthening digital resource sharing during COVID and beyond*[6] è un progetto europeo svoltosi dal 2021 al 2023 sotto il coordinamento della Biblioteca del CNR dell'area di Ricerca di Bologna. Gli obiettivi del progetto HERMES includevano il potenziamento dell'iniziativa RSCVD tramite azioni sinergiche finalizzate a migliorare l'accesso e la disponibilità di testi accademici e scientifici, oltre a formare i bibliotecari nell'uso delle tecnologie digitali per RS, al fine di ottimizzare il sistema internazionale di DD. Un ulteriore obiettivo era lo sviluppo del software open source per Resource sharing, Talaria.

Elaborazione di statistiche nel Resource sharing

Le statistiche nel mondo del Resource sharing interbibliotecario sono fondamentali per valutare l'efficienza del servizio e migliorare l'esperienza degli utenti. Attraverso dati come il numero di richieste ricevute, i tempi di risposta e il tasso di successo le biblioteche possono monitorare la qualità del servizio e identificare le aree di miglioramento.
Uno studio delle statistiche presso la biblioteca universitaria dell'Indiana (IUPUI) ha evidenziato un aumento costante delle richieste di ILL di risorse open access dimostrando che, nonostante la disponibilità dei contenuti open access, molti utenti continuano a preferire l'utilizzo dell'ILL per accedere a questi materiali invece che cercarli in autonomia. [BA15]
Nel corso degli anni dal 2005 al 2009, il numero di richieste di articoli attraverso NILDE ha visto una crescita costante. Anche il numero di biblioteche partecipanti è aumentato e ciò ha comportato un incremento del 26% del numero medio di richieste gestite da ciascuna biblioteca, con un aumento significativo dell'attività di ILL, nonostante la massiccia diffusione negli stessi anni di risorse elettroniche come le riviste online ed ebooks disponibili per gli utenti delle biblioteche. Un'analisi più approfondita delle richieste rivela che, sebbene molte riviste siano richieste solo occasionalmente, una piccola percentuale di titoli è responsabile della maggior parte delle transazioni di ILL. Infine, l'analisi dei dati effettuata su migliaia

[6] `https://www.hermes-eplus.eu/`

di richieste di prestito in Italia ha sottolineato una crescente domanda di articoli pubblicati più di recente, implicando che i ricercatori sono interessati ai materiali più aggiornati. [BM11]
Nel campo del Document delivery internazionale non esistono dati statistici concreti basati su metrice tradizionali, ma sono disponibili informazioni rilevanti grazie a sondaggi periodici, come quello eseguito dall'American Library Association (ALA) attraverso la sua sezione RUSA STARS[7]. Il sondaggio del 2023, condotto a livello globale, ha raccolto informazioni da istituzioni in oltre 85 paesi. Il 72% delle biblioteche intervistate da dichiarato di prendere in prestito materiali da altre nazioni, mentre il 71% ha affermato di prestare a livello internazionale. I risultati dimostrano inoltre che le biblioteche richiedano sia materiali *returnables* (come libri) che *non-returnables* (come articoli digitalizzati), con una prevalenza per il secondo tipo. [CO24]

1.2 Resource sharing management systems: Talaria

Talaria è un *resource sharing management system* open source di nuova generazione ideato dalla Biblioteca del CNR dell'area di Bologna. Lo sviluppo di Talaria è stato un obiettivo del progetto HERMES (dalla mitologia greca: Hermes era il dio dei messaggeri e le Talaria erano i suoi calzari alati). Dal 2023 Talaria è diventata la nuova piattaforma software del servizio di RS internazionale RSCVD. Il software è in continua evoluzione, i nuovi aggiornamenti vengono rilasciati dal CNR di Bologna e messi a disposizione di tutta la comunità. [MA22]

Descrizione e architettura

Talaria è un software altamente personalizzabile per supportare la condivisione delle risorse in una comunità di biblioteche.
Talaria è eseguito all'interno di un'infrastruttura containerizzata tramite Docker,

[7] `https://www.ala.org/rusa/sections/stars`

che garantisce un ambiente isolato e riproducibile, semplificando la gestione delle dipendenze e migliorando la portabilità tra diversi ambienti di sviluppo e produzione.
Talaria utilizza esclusivamente tecnologie open source: il backend è sviluppato in PHP usando il framework open source Laravel mentre il frontend di Talaria è sviluppato come *Progressive Web App* (PWA) per rendere la navigazione scorrevole e dinamica grazie all'utilizzo del framework di JavaScript ReactJS. Infine per la gestione e l'archiviazione dei dati viene utilizzato il DBMS MySQL. Tutte le interazioni tra backend e frontend sono gestite attraverso API RESTful, che permettono al frontend di inviare richieste e ottenere risposte dal backend attraverso chiamate HTTP, utilizzando i metodi standard come GET, POST, PUT e DELETE.

Funzionalità

Talaria permette agli utenti la creazione di un account personale. Una volta creato, è richiesta la registrazione di una nuova biblioteca o l'associazione a una già esistente nel sistema. Una biblioteca può operare come richiedente (*borrower*), fornitrice (*lender*) o entrambi. Il flusso base di una richiesta è quello evidenziato nella Figura 1.1. Gli utenti possono richiedere un documento alla propria biblioteca inserendo solo i suoi identificativi, come il DOI o l'ISBN, oppure compilando un form con tutte le informazioni necessarie, dette metadati bibliografici. Se il documento richiesto è disponibile in Open Access, Talaria fornisce un link diretto per accedervi. Le biblioteche possono inviare richieste di documenti (*borrowing*) a una specifica biblioteca o all'intera comunità di Talaria (richieste orfane). Una biblioteca che riceve una richiesta (*lending*) oppure ne accetta una dall'elenco delle richieste orfane, può evadere o inevadere la richiesta. In caso di inevasione, la biblioteca richiedente può scegliere di inoltrare (*forward*) una nuova richiesta dello stesso documento a un'altra biblioteca o all'intera comunità. Ogni richiesta è associata a un documento ed è caratterizzata da vari campi che forniscono agli operatori le informazioni sul tipo di materiale richiesto (articolo, rivista, parte di libro, tesi, carta geografica, manoscritto), sui dati bibliografici che identificano il

documento, sulla biblioteca richiedente e fornitrice, sulle date di richiesta e accettazione, sulle modalità di evasione o inevasione ecc..., come illustrato nella Figura 1.5. [MA23b]

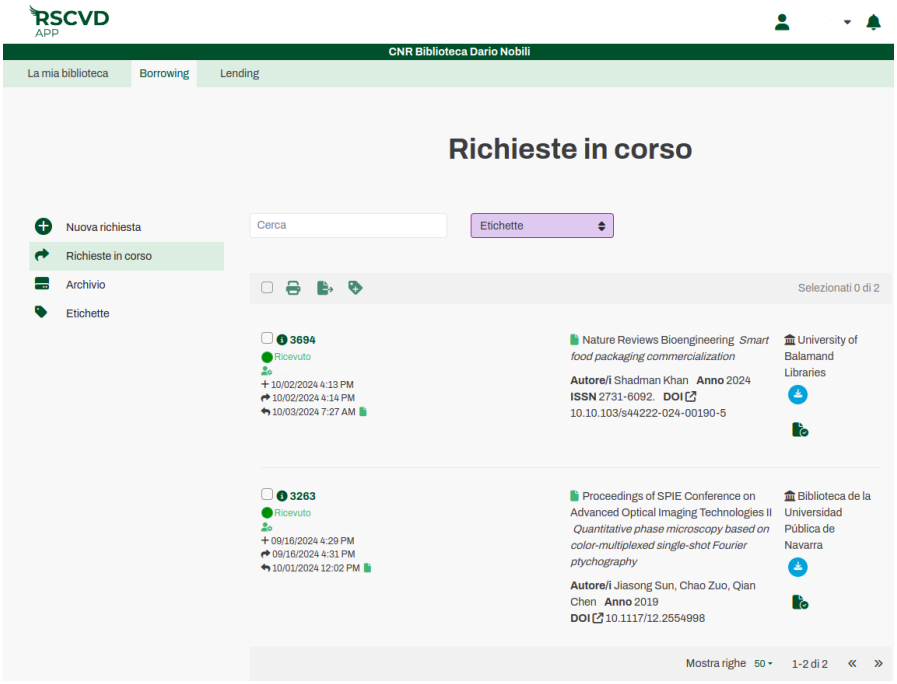


Figura 1.5: Sezione Richieste di *borrowing* in corso di RSCVD

Sono presenti inoltre due stati di lavorazione della richiesta: *borrowing status* e *lending status*, ognuna viene aggiornata a ogni "step" di lavorazione della richiesta. Un obiettivo di Talaria sarà l'interoperabilità con altri sistemi di gestione di RS attraverso l'utilizzo del protocollo ISO 18626, che permette il collegamento tra diverse reti di biblioteche nazionali e internazionali. Inoltre, la piattaforma è stata concepita come un sistema modulare e personalizzabile, consentendo alle biblioteche di adattare le funzionalità in base alle proprie esigenze [MA22].

Confronto tra le tecnologie usate in Nilde e in Talaria

NILDE, sviluppato con tecnologie più tradizionali, come PHP, XHTML, CSS e AJAX, è stato costruito come un sistema monolitico e chiuso. Questo ha funzionato per molto tempo, ma ha limitato la sua capacità di adattarsi alle nuove

sfide dell'interoperabilità. Nel tempo, NILDE si è evoluto all'introduzione di API REST, che permettono di connettere NILDE con altri sistemi, semplificando processi come l'importazione automatica di riferimenti bibliografici e la gestione delle licenze tramite l'integrazione con archivi come ALPE. Tuttavia, il suo design originale lo rende meno flessibile rispetto alle nuove tecnologie e più macchinoso nella gestione dei dati in tempo reale [TAM17].
Talaria, d'altra parte, è stato progettato da subito per essere moderno, flessibile e scalabile. Sviluppato con Laravel per il backend e ReactJS per il frontend, utilizza PHP e JavaScript in modo molto più dinamico, permettendo un'esperienza utente fluida e reattiva. La sua capacità di interfacciarsi con altri sistemi tramite il protocollo ISO 18626 lo renderà una soluzione perfettamente predisposta alla collaborazione internazionale. Talaria, quindi, con il suo design modulare e la possibilità di essere utilizzato su qualsiasi dispositivo rappresenta una risposta di nuova generazione.

Il caso delle statistiche: da NILDE a Talaria

Per l'elaborazione di statistiche in NILDE, di cui vengono mostrati alcuni esempi nelle Figure 1.6 e 1.7, il sistema utilizza un metodo che non risulta particolarmente ottimizzato, basato su query eseguite su tabelle ausiliarie appositamente create. Questo approccio implica che, invece di operare direttamente sulle tabelle principali del database, i dati statistici vengono elaborati attraverso tabelle aggiuntive. Sebbene questo metodo consenta di raccogliere informazioni utili per l'analisi di richieste di DD [MB16], la sua implementazione comporta una maggiore complessità e riduce l'efficienza del sistema, soprattutto in termini di velocità e scalabilità. Il presente lavoro di tesi ha consentito lo studio, la progettazione e la realizzazione di statistiche in Talaria mediante Elasticsearch, allo scopo di superare i limiti evidenziati e di proporre una soluzione ottimale al problema delle statistiche. Nel prossimo capitolo viene presentato lo strumento Elasticsearch, mentre i capitoli successivi sono dedicati alla progettazione e all'implementazione della soluzione proposta.

[8] Consultabili al link `https://nildeworld.bo.cnr.it/it/statistiche/generali`

Figura 1.6: Statistiche generali di NILDE[8]

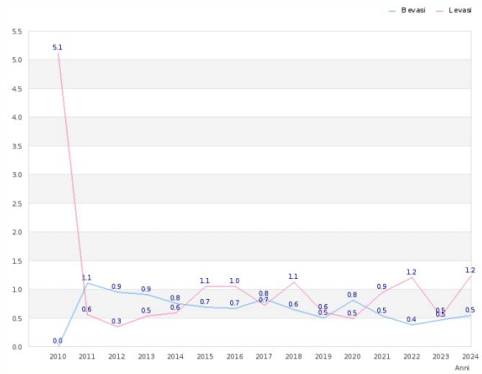


Figura 1.7: Statistiche specifiche di NILDE - Tempo medio di giacenza delle richieste gestite dalla Biblioteca del CNR dell'area di Bologna

1.3 Elasticsearch per l'elaborazione di dati statistici

Elasticsearch è un motore di ricerca open source, basato su Apache Lucene e progettato per gestire grandi volumi di dati in tempo reale. Apache Lucene è una libreria open source scritta in Java, utilizzata per l'indicizzazione e la ricerca full-text di dati testuali. La ricerca full-text è una tecnica per la ricerca di parti di testo all'interno del contenuto di un documento.
La principale funzione di Elasticsearch è quella di indicizzare e cercare dati in modo estremamente veloce e semplice tramite un'interfaccia RESTful.

Descrizione e architettura

Elasticsearch è un sistema distribuito e scalabile sviluppato con Java e basato su Lucene. Il suo punto di forza è la capacità di gestire grandi volumi di dati in tempo reale. Il punto centrale di Elasticsearch è il cluster, ovvero un insieme di nodi che lavorano insieme per portare a termine operazioni. In ogni cluster ci sono vari nodi, che rappresentano istanze di Elasticsearch e che svolgono ruoli specifici. Alcuni nodi sono detti master e sono responsabili della supervisione del cluster: si occupano, ad esempio, di monitorare lo stato di salute dei nodi, di decidere dove allocare i dati e di gestire la configurazione complessiva. Poi ci sono nodi dati, che immagazzinano effettivamente i dati e rispondono alle query.
Un elemento chiave nell'architettura di Elasticsearch è il concetto di indice, che contiene i documenti. Ogni documento è memorizzato in formato JSON e viene diviso in porzioni più piccole, dette shard, che possono essere distribuite tra diversi nodi. Questa frammentazione consente di sfruttare la natura distribuita del sistema, bilanciando il carico su più macchine. Gli shard, infatti, sono vere e proprie copie indipendenti del database che possono essere sparse su più nodi, facilitando la scalabilità. Ogni shard è, a sua volta, un piccolo indice Lucene autonomo e ciò permette di eseguire operazioni di ricerca su ognuno di essi in parallelo. Per garantire che i dati siano sempre disponibili, anche in caso di malfunzionamenti, Elasticsearch prevede la creazione di repliche degli shard. Questo meccanismo assicura che se un nodo va offline, le repliche degli altri nodi possano continuare a rispondere alle query, evitando interruzioni del sistema. [MA23a; WE20]
Elasticsearch viene considerato un database non relazionale (NoSQL) grazie alla sua capacità di archiviare, indicizzare e recuperare grandi quantità di dati in modo efficiente. A differenza dei database relazionali tradizionali, che utilizzano tabelle con rigide relazioni, Elasticsearch adotta un modello flessibile basato su documenti. Ogni documento in Elasticsearch viene memorizzato come un documento JSON, che può avere campi diversi e non seguire uno schema rigido. Elasticsearch ha quindi un approccio differente ai concetti ACID e al teorema CAP, due principi fondamentali nella progettazione e gestione di sistemi di archiviazione dei dati. Il termine ACID si riferisce a quattro proprietà chiave che i database relazionali

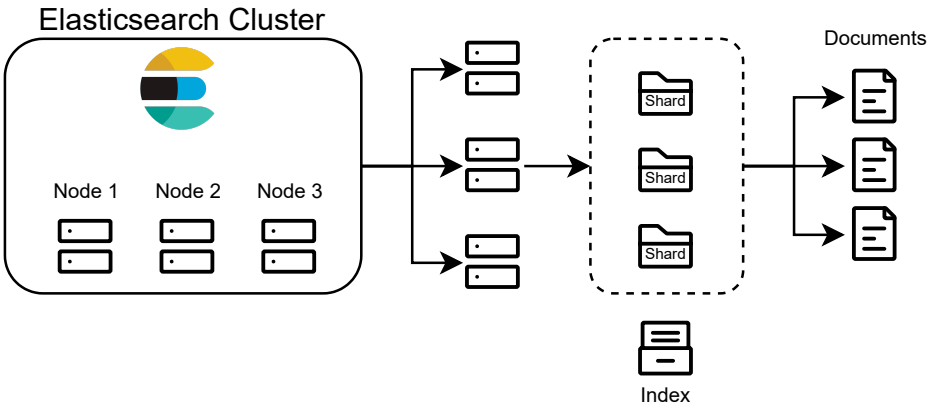


Figura 1.8: Architettura di un cluster multi-nodo in Elasticsearch

garantiscono durante l'esecuzione di transazioni:

- Atomicità (*Atomicity*): Le operazioni di una transazione devono essere completate tutte o nessuna.
- Coerenza (*Consistency*): Ogni transazione deve portare il sistema da uno stato valido a un altro stato valido.
- Isolamento (*Isolation*): Le transazioni simultanee non devono interferire tra di loro.
- Persistenza (*Durability*): Una volta confermata una transazione, i suoi effetti devono essere permanenti anche in caso di errori o crash del sistema.

Elasticsearch, non segue strettamente il modello ACID, ma offre alcune garanzie simili.
Il teorema CAP (o teorema di Brewer) [BR12] afferma che un sistema distribuito, come Elasticsearch, non può garantire contemporaneamente tutte e tre le seguenti proprietà:

- Coerenza (*Consistency*): Tutti i nodi del sistema vedono gli stessi dati nello stesso momento.

- Disponibilità (*Availability*): Il sistema risponde a tutte le richieste, anche in caso di errori.
- Tolleranza alle partizioni (*Partition Tolerance*): Il sistema continua a funzionare nonostante alcuni malfunzionamenti.

Elasticsearch sceglie un compromesso tra disponibilità e tolleranza alle partizioni sacrificando la coerenza. Questo significa che, in caso di guasto, il sistema continuerà a funzionare anche se alcuni nodi non riescono a comunicare tra di loro. Tuttavia in queste situazioni è possibile che alcuni dati non siano immediatamente coerenti tra tutti i nodi, il che introduce una coerenza finale (*eventual consistency*), ovvero che i dati si sincronizzeranno nel tempo.
Grazie a Lucene, Elasticsearch può gestire efficacemente la creazione di indici invertiti, fondamentali per ricerche rapide. L'indice invertito funziona creando una mappa che associa ogni termine presente nei documenti a una lista di documenti in cui quel termine appare. Ciò consente di recuperare i documenti in cui una parola è presente in modo molto veloce rispetto a una scansione lineare dei dati. Prima di creare un indice invertito, però, i documenti vengono normalizzati: i dati vengono analizzati e *tokenizzati*, ovvero vengono resi in minuscolo e vengono rimosse le *stop words*, ovvero le parole poco rilevanti per la ricerca, secondo il processo mostrato nella Figura 1.9[9]. A scopo dimostrativo, si supponga di avere 3 documenti:

- *Document 1*: The QuIcK brown fox
- *Document 2*: The quick BROWN dog
- *Document 3*: The lazy dOg

Il corrispettivo indice invertito sarà come dimostrato nella Tabella 1.1.

[9]Fonte: `https://dev.to/aws-builders/a-learning-journey-with-elk-stack-4om4`, visitato il giorno 25/09/2024.

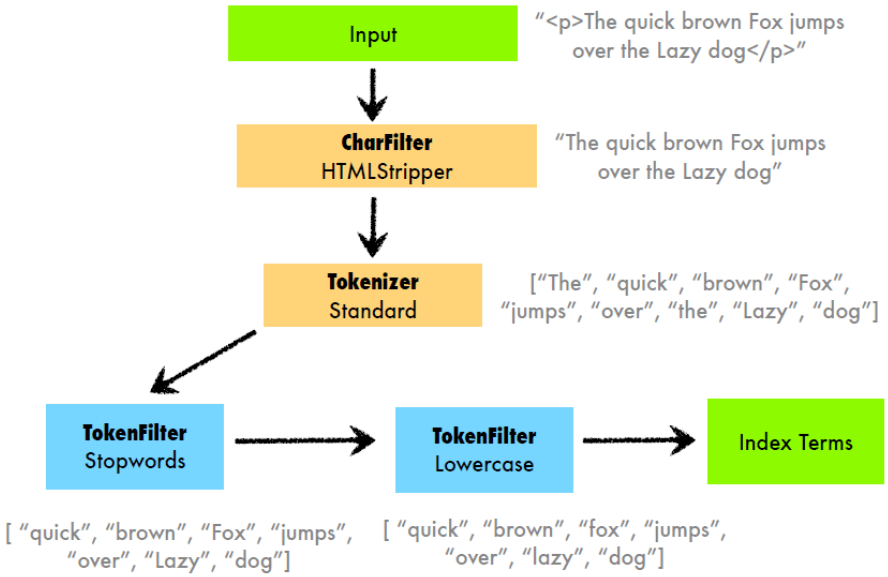


Figura 1.9: Processo di normalizzazione dei documenti

Term	Freq	Documents
quick	2	1,2
brown	2	1,2
fox	1	1
dog	2	2,3
lazy	1	3

Tabella 1.1: Indice invertito

Quando viene richiesta una query, Elasticsearch cercherà nell'indice invertito i termini di ricerca e ordinerà i documenti idonei secondo un rank di rilevanza calcolato tramite l'algoritmo Okapi BM25.
Data una query Q contenente le parole $q_1, \ldots, q_n$ lo score del documento D è:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)} \tag{1.1}$$

Dove $f(q_i, D)$ è il numero di occorrenze di q_i nel documento D, $|D|$ è la lunghezza del documento D in parole, avgdl è la lunghezza media dei documenti. k_1 e b sono parametri liberi scelti per ottimizzare la query con valori predefiniti $k_1 \in [1.2, 2.0]$ e $b = 0.75$. La funzione di *inverse document frequency* $\text{IDF}(q_i)$ calcola il peso di una parola q_i all'interno del documento.

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right) \tag{1.2}$$

Dove N è il numero di documenti e $n(q_i)$ è il numero di documenti che contengono q_i.
Grazie a questi algoritmi, Elasticsearch riesce ad essere molto veloce nella ricerca di grandi volumi di documenti rispetto ad altri database NoSQL [GGS15; ZM21; AAİ16], come mostrato nella Figura 1.10.

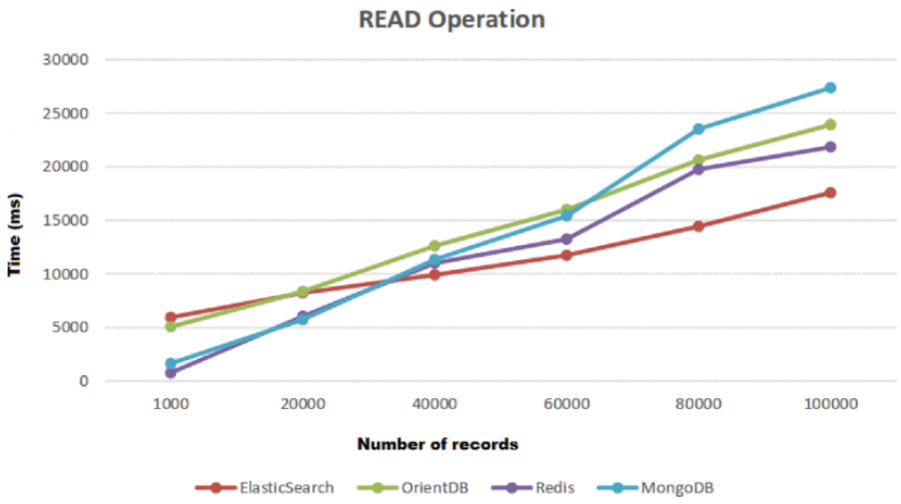


Figura 1.10: Confronto dell'efficienza di operazioni di lettura [VUL16]

Nonostante Elasticsearch sia molto rapido nella ricerca di grandi dataset in confronto a MySQL o ad altri database NoSQL, è anche doveroso citare che un indice di Elasticsearch richiede una capacità di disco maggiore rispetto a una tabella MySQL. Questo aumento di dimensione è dovuto alle differenze di progettazione dei due database, partendo dal fatto che i dati memorizzati su MySQL sono in formato binario, mentre su Elasticsearch sono in un formato binario basato su Lucene che contiene anche l'indice invertito e altri metadati ottimizzati per la ricerca

veloce, rendendo un indice Elasticsearch più pesante. Nella Figura 1.11 viene comparata la richiesta di memoria su disco di una tabella MySQL rispetto allo stesso dataset indicizzato in 5 shard su Elasticsearch. [SE18]

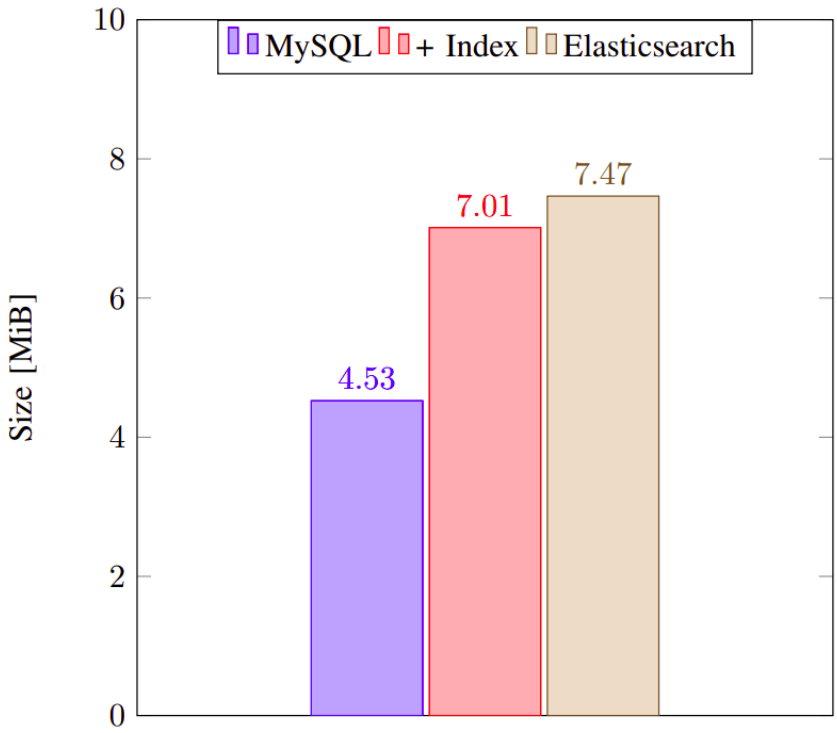


Figura 1.11: Confronto dello spazio su disco tra MySQL (+ indici) ed Elasticsearch.

Funzionalità

Elasticsearch è un motore di ricerca distribuito progettato per gestire grandi dataset. Una sua caratteristica è la capacità di eseguire ricerche full-text e query complesse in tempi rapidi. Supporta funzionalità come il fuzzy search (trova risultati anche se ci sono piccoli errori o variazioni rispetto alla parola cercata), lo stemming (riduzione di una parola alla sua forma radice) e il ranking dei risultati. Elasticsearch, essendo un database non relazionale (NoSQL), soffre della mancanza di relazioni tra gli indici. Nel modello documentale di Elasticsearch i dati sono memorizzati in documenti JSON, che contengono tutte le informazioni necessarie

e quindi non è comune mantenere relazioni tra più indici come si farebbe in un database relazionale. Per questo motivo, si tende a denormalizzare i dati, ovvero a duplicare le informazioni in più documenti anziché utilizzare join complesse. Questo approccio riduce la necessità di fare richieste multiple e ottimizza le prestazioni di ricerca. [GA20]

id	**titolo**	**id_autore**	**id_categoria**
1	1984	1	2
2	Moby Dick	2	1
3	La Fattoria degli Animali	1	3

Tabella 1.2: Tabella Libri

id	**name**
1	George Orwell
2	Herman Melville

Tabella 1.3: Tabella Autori

id	**name**
1	Avventura
2	Distopico
3	Satira

Tabella 1.4: Tabella Categorie

La tabella Libri illustrata nella Tabella 1.2 denormalizzata sarà quindi:

id	titolo	nomeAutore	nomeCategoria
1	1984	George Orwell	Distopico
2	Moby Dick	Herman Melville	Avventura
3	La Fattoria degli Animali	George Orwell	Satira

Tabella 1.5: Tabella Libri denormalizzata

Strumenti complementari

Elasticsearch fa parte di uno stack, chiamato Elastic Stack, che comprende Kibana, Logstash e Beats. Kibana è l'interfaccia grafica che permette l'esplorazione dei dati in modo intuitivo, creando delle dashboard personalizzate e interattive. Logstash è il motore che raccoglie e trasforma dati da fonti diverse, rendendo tutto più semplice e flessibile grazie alle sue pipeline. Infine, Beats è una serie di agenti leggeri che raccolgono dati in tempo reale e li inviano a Elasticsearch o Logstash. L'architettura dell'Elastic Stack è raffigurata nella Figura 1.12
Sono disponibili, inoltre, altri strumenti per supportare l'utilizzo di Elasticsearch al meglio, soprattutto quando si integra con altri sistemi, ad esempio Monstache serve per sincronizzare Elasticsearch con MongoDB [KA23] e Canal offre soluzioni per collegare e sincronizzare Elasticsearch a vari database [WSZ23].

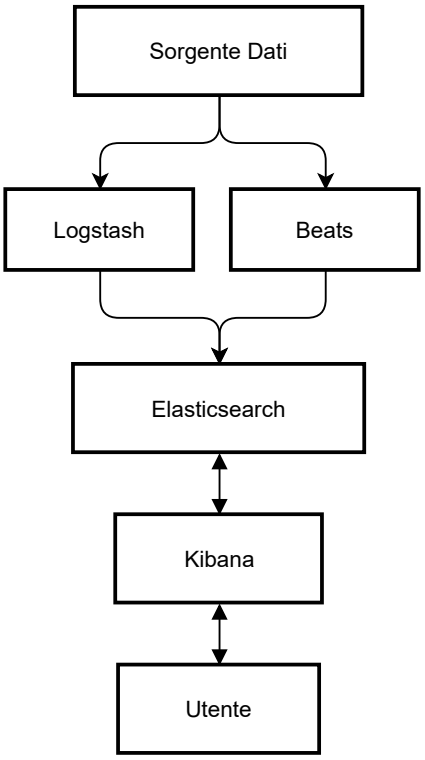


Figura 1.12: Illustrazione dell'Elastic Stack

2 Progettazione di statistiche con Elasticsearch

In questo capitolo verranno illustrate le scelte progettuali dell'implementazione di Elasticsearch in Talaria per l'elaborazione di dati statistici. Le query in Elasticsearch sono più efficienti rispetto a quelle tradizionali in MySQL per fare statistiche grazie alla loro capacità di indicizzare e cercare grandi quantità di dati in tempo reale. A differenza dell'architettura di NILDE, che per elaborare dati statistici richiede l'uso di tabelle ausiliarie, le statistiche in Talaria, con l'implementazione di Elasticsearch, offrono tempi di risposta più brevi, grazie alla possibilità di eseguire aggregazioni e analisi direttamente sui dati indicizzati. Inoltre supporta i campi script che permettono di calcolare statistiche personalizzate senza dover modificare la struttura del database, offrendo maggiore flessibilità e scalabilità.

2.1 Analisi dei requisiti

L'utilizzo di Elasticsearch ha lo scopo di elaborare le statistiche necessarie per il monitoraggio dell'efficienza del servizio di Resource sharing (RS) offerto da Talaria. Per farlo, è necessario replicare i dati rilevanti presenti nel database relazionale e mantenerli aggiornati a ogni nuovo inserimento o modifica. Questo permette di avere un sistema ottimizzato per eseguire query di ricerca e aggregazione complesse in tempi ridotti, anche su dataset di grandi dimensioni.
Le statistiche devono fornire dati sia a livello globale, aggregando tutte le biblio-

teche, sia filtrando per singola biblioteca, in base alla necessità.
Un altro obiettivo è la creazione di una dashboard interattiva, che consenta agli utenti di visualizzare in tempo reale statistiche e metriche rilevanti riguardanti l'efficienza del servizio di RS di Talaria. La dashboard dovrà offrire una panoramica chiara e intuitiva delle seguenti informazioni:

- Numero di richieste e suddivisione per tipo di materiale e stato della richiesta su specifici intervalli temporali.
- Analisi delle prestazioni, come il tempo medio di evasione di una richiesta su specifici intervalli temporali.
- Statistiche aggregate per identificare quali biblioteche inviano o ricevono più richieste e a quali altre biblioteche.
- Statistiche aggregare per identificare quali nazioni inviano o ricevono più richieste e a quali altre nazioni.
- Analisi sulla distribuzione del carico di lavoro delle biblioteche, come la media annuale di richieste effettuate e evase.
- Analisi dei titoli di rivista richiesti: per anno di pubblicazione, per licenza (open access) o per editore

L'utilizzo di Elasticsearch è particolarmente vantaggioso per gestire questi requisiti grazie alla sua capacità di gestire grandi volumi di dati in modo distribuito garantendo risposte rapide anche a query complesse e offrendo agli amministratori di sistema una visione chiara delle performance del servizio di RS.
Una delle sfide principali è assicurarsi che le statistiche siano sempre aggiornate e riflettano la situazione attuale delle richieste. Per farlo, è necessario mantenere un flusso costante di dati tra il database principale e Elasticsearch. Ogni nuova richiesta, modifica o cancellazione viene immediatamente sincronizzata, permettendo alla dashboard di mostrare dati sempre aggiornati. Questo può essere ottenuto con l'esecuzione di processi periodici o agenti esterni che si occupano di sincronizzare le informazioni, come schematizzato nella Figura 2.2. Grazie alle capacità

di Elasticsearch, il sistema può aggiornarsi continuamente senza impattare troppo sulle performance generali.

Un altro aspetto importante è la velocità delle ricerche e analisi dei dati. Elasticsearch organizza le informazioni in shard suddivisi su più nodi. Questo significa che quando si effettua una ricerca o si esegue un'aggregazione complessa il carico viene distribuito, permettendo di ottenere risposte rapide. Nel caso di Talaria, questo approccio consente di cercare tra migliaia di richieste in pochissimo tempo, anche se i dati sono voluminosi e complessi. Inoltre è possibile eseguire ricerche molto specifiche, come la durata media dei tempi di lavorazione di una richiesta di DD o il numero di richieste per materiale con uno specifico stato.

2.2 Workflow

L'integrazione di Elasticsearch in Talaria è stata pianificata con l'obiettivo di avere delle statistiche efficienti e in tempo reale, garantendo che il sistema fosse scalabile e flessibile per future espansioni. Il processo ha seguito un percorso logico e ben definito, suddiviso in diverse fasi, ognuna delle quali mirava a soddisfare requisiti specifici.

1. **Configurazione iniziale di Elasticsearch:** Essendo Talaria eseguito con Docker, è necessaria l'aggiunta delle immagini di Elasticsearch e Kibana all'interno della configurazione del Docker di Talaria. Grazie a questa tecnologia l'installazione e il setup dei server di Elasticsearch e di Kibana non vanno ad impattare con il resto del sistema in caso di problemi nella loro fase di inizializzazione. Per la configurazione iniziale di Elasticsearch si è optato per un cluster *single-node* ovvero di un singolo nodo, in quanto per la situazione attuale di Talaria è sufficiente perché non è necessario avere un sistema distribuito in quanto Talaria al momento conta poche richieste. Un aspetto importante dell'implementazione di un cluster *multi-node* è la configurazione di una *Certificate Authority (CA)*, ovvero un'entità che ha il compito di emettere e gestire i certificati digitali che vengono utilizzati per garantire la

sicurezza delle comunicazioni sulle reti, come quella di Elasticsearch in caso di più nodi.

2. **Preparazione dell'ambiente:** Il secondo passo è stato predisporre l'ambiente in modo che Elasticsearch potesse funzionare in modo integrato con il sistema esistente di Talaria. Questo ha richiesto una revisione dell'infrastruttura per assicurarsi che fosse pronta a gestire nuovi servizi. L'obiettivo principale di questa fase era quello di garantire una configurazione solida e sicura, che permettesse di avviare Elasticsearch e di collegarlo senza interruzioni al resto della piattaforma.

3. **Progettazione dell'indice di Elasticsearch:** Essendo Talaria una piattaforma che gestisce richieste di borrowing e lending tra biblioteche, è stato necessario filtrare i campi più rilevanti dal database (si veda Figura 2.4) per creare un indice di Elasticsearch che fosse tanto espressivo quanto ottimizzato.

   Per ogni campo è stato importante scegliere il tipo di dato corretto per evitare che Elasticsearch assegnasse una mappatura automatica non completamente corretta. Questo avrebbe potuto causare l'inefficienza di alcune query e risultati imprecisi.
   La struttura (o *mapping*) dell'indice di Elasticsearch sarà come quella rappresentata nella Tabella 2.1. I campi *borrowing library* e *lending library* sono oggetti che hanno la stessa struttura che è descritta nella Tabella 2.2. Il campo *reference* è un campo oggetto che ha la struttura descritta nella Tabella 2.3.

4. **Sincronizzazione dei dati:** Una volta impostato l'ambiente, la sfida successiva è stata assicurarsi che tutti i dati rilevanti, come le richieste di DD interbibliotecario, fossero indicizzati correttamente su Elasticsearch. Il punto cruciale di questa fase è stato garantire che ogni cambiamento nel sistema fosse riflesso nell'indice di Elasticsearch. In questo modo, i dati restano sempre aggiornati, permettendo agli utenti di avere un accesso continuo e immediato a informazioni corrette e attuali.

Campo	Descrizione
id	Identificativo della richiesta
request date	Data della richiesta
fulfill date	Data dell'evasione o inevasione della richiesta
borrowing status	Stato del *borrowing*
lending status	Stato del *lending*
fulfill type	Modalità di invio del materiale
not fulfill type	Motivo dell'inevasione
forward	Indica se la richiesta è stata inoltrata
borrowing library	Biblioteca richiedente
lending library	Biblioteca fornitrice
reference	Materiale richiesto

Tabella 2.1: Struttura della richiesta di DD nell'indice di Elasticsearch

5. **Definizione delle API per le statistiche:** Una volta garantita la sincronizzazione dei dati, il passaggio successivo è stato quello di sviluppare delle API che consentissero di eseguire analisi avanzate e produrre statistiche rilevanti. Il sistema doveva essere in grado di rispondere a richieste complesse come il tempo medio di lavorazione delle richieste, la suddivisione delle richieste per materiale e altre metriche utili per misurare le performance del servizio.

6. **Visualizzazione dei dati:** Infine, è stato fondamentale presentare queste informazioni in modo chiaro e accessibile per gli utenti. Per farlo è stata creata una dashboard interattiva che contente di visualizzare le statistiche e metriche più importanti in tempo reale e filtrarle secondo le necessità.

Grazie a questo workflow, l'integrazione di Elasticsearch in Talaria è strutturata in modo da garantire un sistema di ricerca e analisi dei dati robusto, in grado di rispondere rapidamente alle richieste anche con volumi di dati significativi. Ogni fase è stata pensata per ottimizzare sia l'efficienza del sistema, sia l'esperienza utente,

Campo	Descrizione
id	Identificativo della biblioteca
name	Nome della biblioteca
country	Nazione della biblioteca (id, codice e nome)
institution	Istituzione della biblioteca (id, nome, tipo e nazione)
subject	Settore disciplinare della biblioteca (id e nome)

Tabella 2.2: Struttura della biblioteca

Campo	Descrizione
id	Identificativo del materiale
material type	Tipo di materiale
pubyear	Anno di pubblicazione
issn / isbn	Identificativo della pubblicazione, ove applicabile
oa link	Link all'articolo in Open Access

Tabella 2.3: Struttura del riferimento bibliografico oggetto della richiesta

assicurando che la piattaforma possa evolversi in modo flessibile e rispondere a esigenze future.

2.3 Progettazione del backend

Il backend di Talaria è sviluppato utilizzando il framework Laravel, scelto principalmente per l'architettura MVC (Model-View-Controller), che facilità l'organizzazione del codice. Questo approccio divide l'applicazione in tre componenti principali, ciascuno con responsabilità ben definite:

Model: Il Model è responsabile della gestione dei dati e della logica di business. Nel caso di Talaria, i Model rappresentano le entità fondamentali del sistema, come le richieste di DD, le biblioteche, gli utenti ecc...

Il Model interagisce con il database relazionale per salvare e recuperare i dati e con Elasticsearch per indicizzare e cercare le informazioni in tempo reale.

View: Le View sono responsabili della presentazione delle informazioni all'utente. Ricevono i dati dai Controller e li rendono visibili attraverso pagine web, scritti generalmente in Blade, ossia il motore di template di Laravel. Nel caso di Talaria, le View sono gestite dal frontend con React.

Controller: Il Controller è un'interfaccia tra i Model e le View. Processa le richieste inviate dagli utenti tramite le View e manipola i dati presi dai Model per inviarli alle View, pronti per la visualizzazione. In Talaria, i Controller raccolgono le richieste dal frontend e utilizzano i Model per recuperare le informazioni richieste.

L'architettura MVC garantisce una chiara separazione dei ruoli all'interno del sistema, rendendo più facile la manutenzione e estensione del codice.
Grazie a questo approccio modulare, l'integrazione di Elasticsearch per la gestione delle statistiche è stata relativamente semplice, poiché ha richiesto l'estensione dei Model relativi alla gestione delle statistiche e l'aggiunta di nuove logiche nei Controller, creando nuove API per le statistiche, senza modificare in modo significativo altre parti del sistema.
In particolar modo, sarà necessario estendere gli Observer dei Model coinvolti per la creazione delle statistiche. Gli Observer sono classi speciali di Laravel che permettono di ascoltare e gestire gli eventi che riguardano il ciclo di vita di un Model, come la creazione, l'aggiornamento e la cancellazione. Quindi ogni volta che un'azione specifica viene eseguita su un Model, l'Observer esegue automaticamente determinate operazioni in risposta, come schematizzato nella Figura 2.2
Le nuove API per le statistiche avranno quindi la seguente struttura:

`/avg-working-time`

Parametri: *anno*, *ID della biblioteca*

Descrizione: Restituisce un'aggregazione per mese della media dei giorni di lavorazione di una richiesta.

`/borrowing-requests-stats`

Parametri: *anno*, *ID della biblioteca*, *tipo di materiale*, *stato di borrowing aggregato*, *tipo di invio del materiale*, *tipo inevasione*

Descrizione: Restituisce il numero di richieste, aggregate secondo i filtri applicati.

`/requests-countries-leaderboard`

Parametri: modalità, *anno*

Descrizione: In base alla modalità scelta, restituisce una classifica che ordina le nazioni che richiedono (modalità = 0) o prestano (modalità = 1) di più.

`/requests-countries-library`

Parametri: ID della biblioteca, modalità, *anno*

Descrizione: Dato un ID di una biblioteca e la modalità, restituisce una classifica delle nazioni a cui ha richiesto o prestato di più.

`/requests-countries`

Parametri: ID della nazione, modalità, *anno*

Descrizione: Dato un ID di una nazione e la modalità, restituisce una classifica delle nazioni a cui ha richiesto o prestato di più.

Nota: I parametri scritti in *corsivo* sono opzionali, mentre quelli sottolineati sono obbligatori.
Nel flusso di lavorazione di una richiesta in Talaria, gli stati di *borrowing* e *lending* variano costantemente, seguendo il flusso rappresentato nella Figura 2.1.
Gli stati della richiesta, per fini statistici, verranno aggregati in macro categorie. Saranno presenti 7 categorie per lo stato di *borrowing* (Tabella 2.4) e 5 categorie per quello di *lending* (Tabella 2.5). Nella Tabella 2.4 i campi con ID 5 e 7 hanno la differenza, rispettivamente con ID 3 e 6, di essere richieste inoltrate, ovvero sono state ricreate.

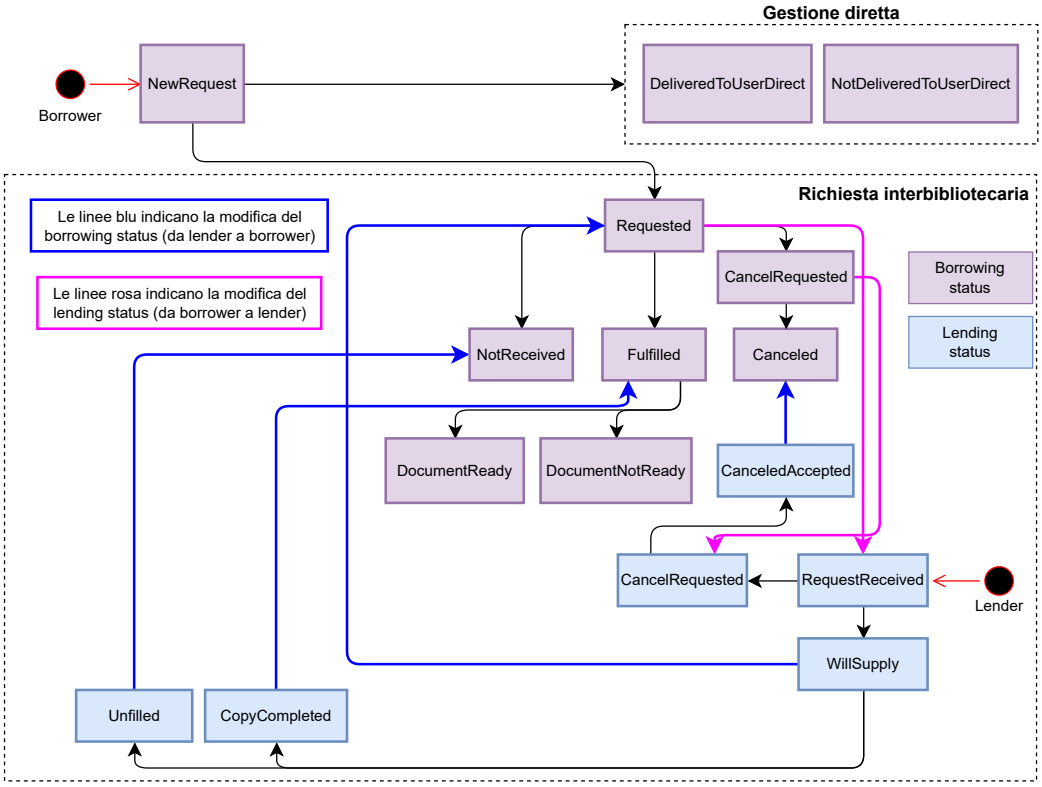


Figura 2.1: Workflow degli stati di una richiesta in Talaria

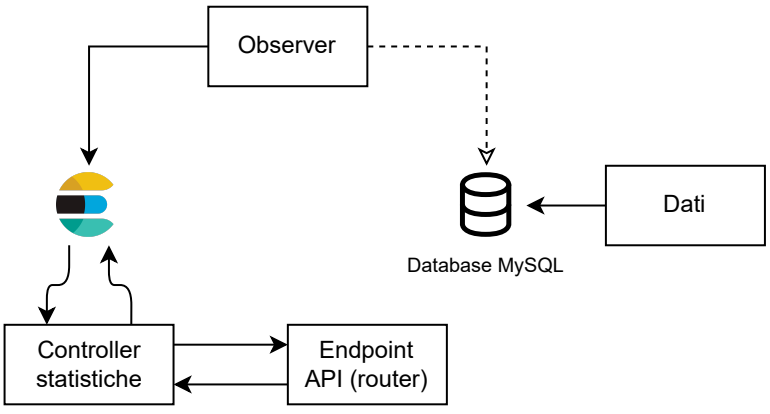


Figura 2.2: Struttura della progettazione del backend

ID aggregazione	Descrizione
1	In progress
2	Received
3	Not received
4	Canceled
5	Not received and forwarded
6	Document not ready
7	Document not ready and forwarded

Tabella 2.4: Aggregazione degli stati di *borrowing*

ID aggregazione	Descrizione
1	In progress
2	Fulfilled
3	Not fulfilled
4	Canceled
5	In progress and orphaned

Tabella 2.5: Aggregazione degli stati di *lending*

2.4 Progettazione del frontend

Il frontend di Talaria è implementato con React, un framework JavaScript che consente la creazione di interfacce utente interattive e modulari. Grazie a React, vengono implementati componenti riutilizzabili che gestiscono la visualizzazione dell'interfaccia in modo strutturato.
Per gestire i dati in modo uniforme e centralizzato viene utilizzato Redux, una libreria JavaScript che fornisce un unico luogo dove mantenere lo stato globale dell'applicazione. Redux usa il concetto di reducer per gestire le modifiche dello stato. Un reducer è una funzione che riceve lo stato attuale e un'azione e restituisce il nuovo stato aggiornato.

Per gestire operazioni asincrone ed effetti collaterali complessi, come le chiamate API, viene utilizzato il middleware Redux-Saga che consente di mantenere una separazione pulita tra la logica asincrona e i componenti. Saga rimane in attesa del lancio (*dispatch*) di azioni specifiche e in risposta esegue operazioni come il fetch di dati ed emette nuove azioni per aggiornare lo stato globale dell'applicazione.
La progettazione del frontend per le statistiche riguarderà principalmente le chiamate alle API precedentemente illustrate e lo sviluppo di grafici per la visualizzazione dei dati restituiti. Per scegliere i parametri è importante anche avere un componente che consenta all'utente di applicare i filtri personalizzati. Grazie alla natura di React, sarà sufficiente riutilizzare un componente già presente in Talaria. Per la visualizzazione dei grafici verrà utilizzato `Chart.js`, una libreria JavaScript gratuita e open source, adatta alla visualizzazione dei dati per lo scopo di Talaria. L'adattamento del frontend alle statistiche avverrà seguendo quanto illustrato nella Figura 2.3.

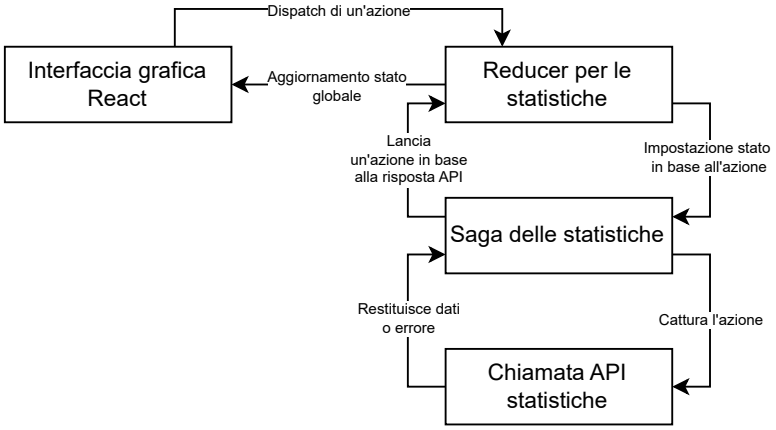


Figura 2.3: Struttura della progettazione del frontend

Il backend e il frontend lavorano in sinergia per offrire un'esperienza utente fluida. Quando un utente interagisce con la dashboard per ottenere informazioni o statistiche, il frontend invia una richiesta al backend tramite le API REST. Il backend, attraverso i suoi Controller, gestisce la richiesta, recupera i dati da Elasticsearch e restituisce le informazioni richieste.

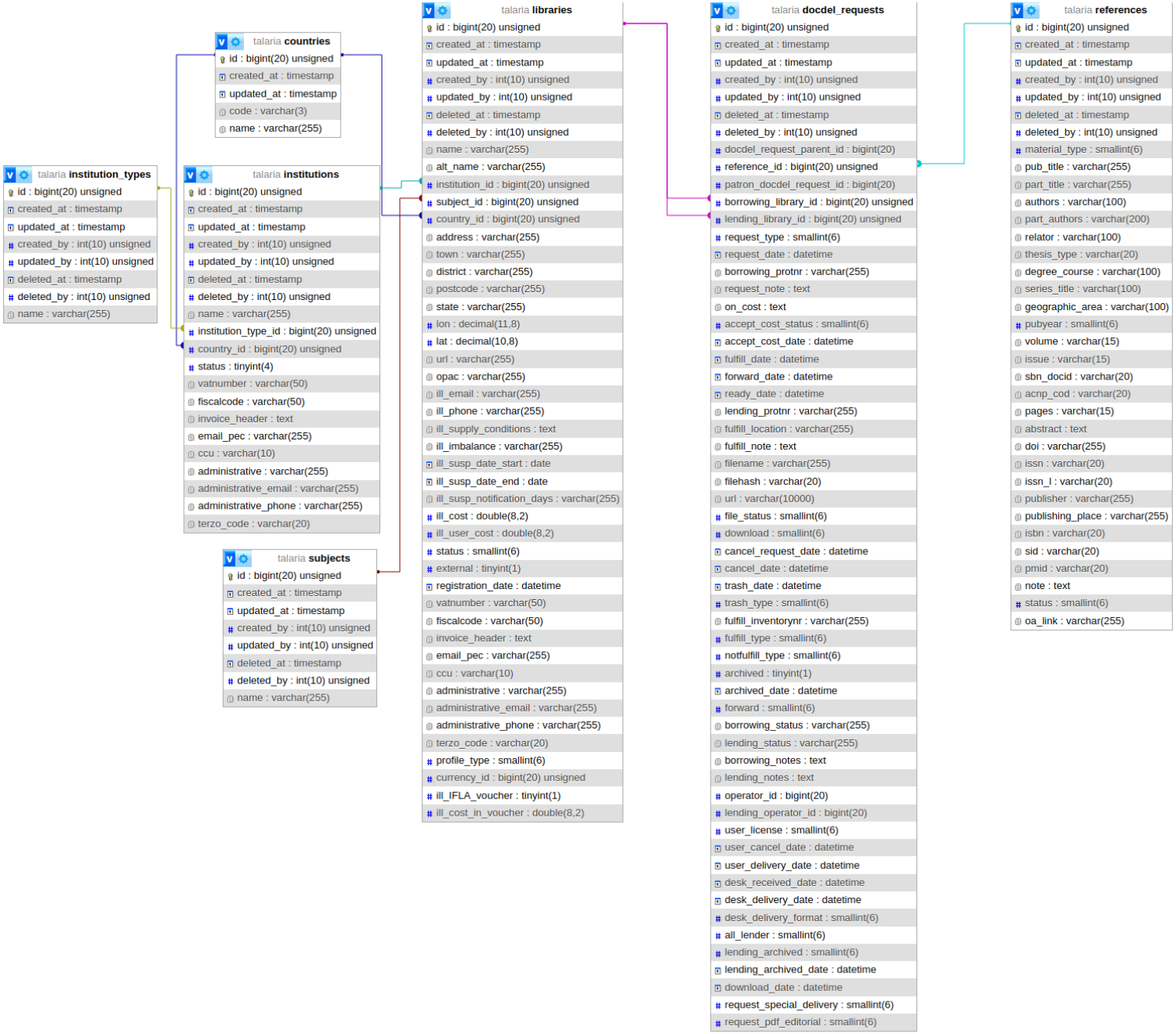


Figura 2.4: Diagramma ER della gestione delle richieste in Talaria

3 Implementazione

In questo capitolo verranno illustrati i punti focali dell'implementazione delle statistiche con Elasticsearch in Talaria. Il capitolo percorrerà il workflow di progettazione illustrato nella Sezione 2.2 e parte dall'integrazione nel backend con Laravel per concludersi con la visualizzazione dei grafici con la libreria `Chart.js`. Il codice dell'implementazione è consultabile su GitHub[1].

3.1 Implementazione del backend con Laravel

L'implementazione di Elasticsearch nel backend di Talaria ha richiesto una serie di modifiche mirate al supporto e all'installazione di Elasticsearch in Laravel. Queste modifiche hanno comportato l'installazione delle librerie necessarie per il supporto di Elasticsearch, la configurazione e il collegamento del client Elasticsearch verso Talaria e infine la sincronizzazione del database MySQL con l'indice di Elasticsearch.

Configurazione iniziale di Elasticsearch

Talaria gira all'interno di un'infrastruttura containerizzata tramite Docker, quindi per aggiungere un'istanza di Elasticsearch e Kibana è necessario modificare il file `docker-compose.yml` inserendo le rispettive immagini e le impostazioni del cluster. La versione installata di Elasticsearch e Kibana è la 8.14.3, che, al momento dello sviluppo rappresentava l'ultima versione rilasciata. Per la configurazione del primo avvio, è stato necessario creare delle credenziali di accesso

[1] `https://github.com/Zaid1710/talaria-statistics`

per l'amministratore di Elasticsearch e Kibana nel file di configurazione di Talaria usato per definire le variabili d'ambiente `.env`.
Per configurare l'accesso di Talaria a Elasticsearch si è optato per un'autenticazione tramite API key con permessi di modifica e creazione degli indici. Le API key, rispetto all'autenticazione base con username e password, permettono un controllo più flessibile sui permessi assegnati e limitano i rischi in casi di accessi non autorizzati. In caso di compromissione, l'amministratore di sistema può facilmente invalidare e rigenerare l'API key[2]. La generazione dell'API key avviene tramite una chiamata POST all'endpoint `/_security/api_key` di Elasticsearch che restituisce le informazioni della chiave generata. Sia la chiave vera e propria che l'ID della stessa sono stati inseriti nel file `.env` per separare le informazioni sensibili dal codice sorgente.

Preparazione dell'ambiente

L'implementazione di Elasticsearch nel backend di Talaria è iniziata con l'installazione della libreria ufficiale di Elasticsearch per PHP, con il comando:

```
> composer require elasticsearch/elasticsearch
```

Questo comando chiama `composer`, il gestore di pacchetti in PHP che ricerca il pacchetto `elasticsearch/elasticsearch` nel repository ufficiale di pacchetti di PHP e lo aggiunge alle dipendenze del progetto. Tale libreria permette di interfacciarsi direttamente con un server Elasticsearch e consente la manipolazione di indici e dati. Per interfacciarsi è necessario prima configurare l'accesso al server Elasticsearch. È stato creato un file di configurazione `config/elasticsearch.php` al cui interno sono presenti parametri per l'esecuzione di Elasticsearch, come ad esempio l'indirizzo dell'host e le informazioni sull'API key per l'autenticazione. Successivamente bisogna costruire l'oggetto client di Elasticsearch. Per crearlo è stato implementato un Service Provider: un componente di Laravel che gestisce la comunicazione tra l'applicazione e servizi esterni, come Elasticsearch, al fine

[2]Fonte: `https://www.elastic.co/guide/en/elasticsearch/reference/current/security-api-create-api-key.html`, visitato il giorno 08/10/2024.

di centralizzare la configurazione e l'inizializzazione del servizio. Nel file `ElasticsearchServiceProvider.php` si definisce l'oggetto di Elasticsearch come un singleton con i parametri presi dal file di configurazione precedentemente creato. Un singleton è un design pattern (modello di progettazione) che garantisce che venga creata una sola istanza della classe. L'uso del pattern singleton è motivato dal fatto che serve un'unica connessione persistente con Elasticsearch, ottimizzando l'utilizzo di risorse ed evitando la creazione di connessioni multiple e ridondanti. In alternativa, sarebbe stato possibile utilizzare il pattern Dependency Injection (DI), che prevede il passaggio della dipendenza alle classi che ne hanno bisogno al momento della richiesta, offrendo maggiore flessibilità. Con la dependency injection l'oggetto di Elasticsearch verrebbe fornito alle classi tramite il costruttore anziché essere istanziato direttamente all'interno delle stesse. Tuttavia, per lo scopo di Talaria, in cui è necessaria solo un'istanza globale e condivisa di Elasticsearch in esecuzione, il pattern singleton risulta essere più efficiente ed adeguato.
Infine bisogna registrare il Service Provider appena creato nella sezione `providers` del file `config/app.php`. Ora è possibile interagire con il server Elasticsearch tramite l'oggetto istanziato con:

```
$this->client = app('Elasticsearch\Client');
```

Progettazione dell'indice di Elasticsearch

Il passo successivo è creare l'indice di Elasticsearch e popolarlo con i dati già esistenti nel database di Talaria. È stato quindi implementato un Job apposito: `InitializeElasticsearchIndex.php`. Il Job in Laravel è un'unità di lavoro che esegue determinate operazioni, in questo caso inizializza l'indice di Elasticsearch. L'indice segue il *mapping* descritto nella Tabella 2.1 ed è necessario specificare il tipo di ogni campo per evitare la mappatura automatica inefficace di Elasticsearch. Il mapping finale sarà quindi come quello descritto nella Figura 3.1.

```
id: long
request_date: date
fulfill_date: date
borrowing_status: text (keyword)
lending_status: text (keyword)
fulfill_type: long
notfulfill_type: long
forward: long
borrowing_library: object {
  id: long
  name: text (keyword)
  country: object {
    id: long
    code: text (keyword)
    name: text (keyword)
  }
  institution: object {
    id: long
    name: text (keyword)
    country: object {
      id: long
      code: text (keyword)
      name: text (keyword)
    }
    institution_type: object {
      id: long
      name: text (keyword)
    }
  }
  subject: object {
    id: long
    name: text (keyword)
  }
}
lending_library: object {
    id: long
    name: text (keyword)
    country: object {
      id: long
      code: text (keyword)
      name: text (keyword)
    }
    institution: object {
      id: long
      name: text (keyword)
      country: object {
        id: long
        code: text (keyword)
        name: text (keyword)
      }
      institution_type: object {
        id: long
        name: text (keyword)
      }
    }
    subject: object {
      id: long
      name: text (keyword)
    }
  }
reference: object {
  id: long
  material_type: long
  pubyear: long
  oa_link: text
  issn: text (keyword)
  isbn: text (keyword)
  pub_title: text (keyword)
}
```


Figura 3.1: *Mapping* dell'indice `docdel_requests` in Elasticsearch

Una volta definito il *mapping* dell'indice, bisogna caricare i dati già presenti nel database MySQL di Talaria su Elasticsearch. La tabella principale di Talaria è `docdel_requests` che segue una struttura come quella citata nella Figura 2.4. Per denormalizzare è necessario chiamare le altre tabelle che hanno relazioni con `docdel_requests`, ovvero `references` e `libraries`. La tabella `libraries`, al suo interno, ha altre relazioni come ad esempio `countries`, `institutions` (che a sua volta ha relazioni con `institution_types`) e `subjects`. Per avere una query efficiente che restituisca tutti i dati richiesti è stato invocato il modello Eloquent di `DocdelRequest`. Eloquent è l'ORM (Object-Relational Mapping) di Laravel, permette di interagire con il database utilizzando modelli che rappresentano le sue tabelle. Con il metodo `with()` viene eseguito eager loading delle relazioni, ovvero le relazioni vengono caricare con un'unica query, invece di eseguire query aggiuntive per ogni relazione. In un contesto come quello di Elasticsearch la denormalizzazione dei dati è fondamentale per migliorare le prestazioni delle query, in quanto non è possibile effettuare operazioni di join tra indici. Grazie alla denormalizzazione, tutti i dati rilevanti vengono raggruppati in un unico documento, riducendo il numero di query necessarie e migliorando la velocità di recupero delle informazioni [MA23a]. Tuttavia, la denormalizzazione comporta un aumento di consumo dello spazio su disco, poiché i dati duplicati devono essere memorizzati in più documenti, come spiegato nella Sezione 4.1.
Grazie alle funzionalità delle API Bulk di Elasticsearch è possibile indicizzare più documenti con un'unica chiamata API. Il Job appena implementato è stato richiamato in un comando apposito (`InitializeElasticsearch.php`) in modo tale da poter inizializzare l'indice di Elasticsearch con l'utilizzo del comando

```
> php artisan elasticsearch:init
```

Sincronizzazione dei dati

Ora Elasticsearch possiede tutti i dati delle richieste preesistenti, adattati al *mapping* dell'indice. Il passo successivo è sincronizzare tutte le nuove richieste.
Tale passaggio richiede l'implementazione dell'Observer nel modello `DocdelRequest`. Il modello `DocdelRequest` è la tabella genitore di due tabelle che la ereditano:

`BorrowingDocdelRequest` e `LendingDocdelRequest` sul quale lavorano rispettivamente la biblioteca richiedente e quella fornitrice di una richiesta di ILL. Negli Observer dei modelli sopracitati la sincronizzazione con Elasticsearch avviene nella funzione `saved()`, ovvero dopo che la richiesta viene salvata sul database MySQL con successo. Dato che è possibile effettuare modifiche su una richiesta prima dell'invio, in `BorrowingDocdelRequestObserver.php` è presente un controllo che verifica se la richiesta è nuova o meno e in base a ciò il documento di Elasticsearch viene inserito o aggiornato. Durante il ciclo di vita di una richiesta, molti campi vengono aggiornati costantemente (come lo stato di *borrowing* o di *lending*) e alcuni vengono inizializzati a `NULL`, come la biblioteca fornitrice, che non viene inserita finché non prende in carico la richiesta.
Mantenere la consistenza dei dati tra MySQL ed Elasticsearch è fondamentale per garantire che le informazioni siano accurate in entrambi i sistemi. Per evitare inconsistenze è necessario considerare casi in cui la sincronizzazione potrebbe fallire. Ad esempio potrebbero verificarsi errori di rete durante l'indicizzazione di una richiesta. In questo caso, la richiesta è correttamente salvata sul database MySQL ma non su Elasticsearch. Grazie al concetto di consistenza eventuale (*eventual consistency*) nei sistemi distribuiti, è accettabile che i dati non siano immediatamente consistenti tra il database MySQL ed Elasticsearch purché la consistenza venga raggiunta dopo un certo periodo di tempo. [KL17]
Per assicurarsi che i database siano sincronizzati è possibile schedulare il Job `InitializeElasticsearchIndex` per ripopolare l'indice di Elasticsearch ogni periodo di tempo prefissato.

Definizione delle API per le statistiche

Ora che i dati di Elasticsearch sono sincronizzati con quelli del database MySQL, è possibile implementare le API delle statistiche. Il file `AdminStatsController.php` è il controller che gestisce la logica delle API delle statistiche. La funzione `getStatusMap($statusType)` gestisce l'aggregazione degli stati in base all'input fornito: se è "*borrowing*" allora restituisce un *mapping* basandosi sulla Tabella 2.4, altrimenti, se è "*lending*" allora il *mapping* restituito rispecchia la

Tabella 2.5.
Nel controller `AdminStatsController` sono state implementate le seguenti funzioni:

`getBorrowingStats` : Chiede alcuni parametri opzionali tra cui l' anno di ricerca, tipo di materiale del riferimento bibliografico e stato di *borrowing* e restituisce il numero di richieste che soddisfano i filtri impostati e la loro suddivisione per alcuni criteri.

`getRequestsCountriesLeaderboard` : In base a un parametro obbligatorio (`mode`), restituisce la classifica dei Paesi che richiedono (`mode = 0`) o che forniscono (`mode = 1`) di più.

`getRequestsCountriesFromLibrary` : Chiede due parametri obbligatori: `mode` e `library_id` e stila una classifica dei Paesi a cui tale biblioteca chiede (`mode = 0`) o fornisce (`mode = 1`) di più.

`getRequestsCountriesFromCountry` : Chiede due parametri obbligatori: `mode` e `country_id` e stila una classifica dei Paesi a cui tale Paese chiede (`mode = 0`) o fornisce (`mode = 1`) di più.

`getAvgWorkingTime` : Chiede un parametro opzionale: l'anno di ricerca. Restituisce, mese per mese, il tempo medio di lavorazione di una richiesta in millisecondi.

L'API descritta nella funzione `getBorrowingStats()` prende in input molti parametri opzionali: l'anno di ricerca, l'ID della biblioteca richiedente, il tipo di materiale del riferimento bibliografico, lo stato di *borrowing* aggregato, il tipo di evasione e quello di inevasione. Una volta validato l'input e generata l'aggregazione in base al *borrowing*, la query di Elasticsearch si presenta inizialmente vuota, nel campo `'must'` saranno presenti i filtri applicati. Successivamente è presente la funzione `'aggs'` che aggrega i risultati della query per tipo di materiale, stato di *borrowing* aggregato, stato di evasione o inevasione. L'API restituisce un oggetto JSON, contenente la risposta di Elasticsearch. Per analizzare l'output dell'API

è necessario creare un endpoint per quest'ultima. Il file `routes/api/admin.php` contiene tutti gli endpoint delle API per le statistiche, quindi per testare l'API è necessario fare una chiamata `GET` all'endpoint `/borrowing-requests-stats`. È possibile analizzare l'oggetto restituito dall'API suddividendolo in pezzi:

```
"took": 37,
"timed_out": false,
"_shards": {
    "total": 1,
    "successful": 1,
    "skipped": 0,
    "failed": 0
},
```

Questo primo pezzo è l'header della risposta e indica che per l'esecuzione della richiesta sono stati necessari 37 ms e non è andata in timeout. L'ultimo blocco fornisce informazioni sugli shard che hanno gestito la query, in questo caso 1 shard e ha risposto con successo.

```
"hits": {
  "total": {
    "value": 2897,
    "relation": "eq"
  },
  "max_score": null,
  "hits": []
},
```

Questa parte descrive i documenti restituiti nella loro totalità: sono stati trovati 2897 documenti e la relazione tra il numero totale di documenti e il valore mostrato in `value` è uguale. La query non ha utilizzato il punteggio di rilevanza (*score*) in quanto le query statistiche sono più orientate sull'analisi dei dati e sulle aggregazioni rispetto alle query di ricerca. Per lo stesso motivo la lista dei documenti effettivi che rispondono alla query è vuota, questo è dato da un parametro della

query (`size: 0`) che non restituisce nessun documento.
L'ultima parte della risposta dell'API riguarda le aggregazioni, dove ogni filtro viene suddiviso in `buckets`. Un `bucket` è un insieme di documenti che soddisfano una certa condizione ed è definito da una regola o criterio.

```
"by_material_type": {
  "doc_count_error_upper_bound": 0,
  "sum_other_doc_count": 0,
  "buckets": [
    {
      "key": 1,
      "doc_count": 2510
    },
    {
      "key": 2,
      "doc_count": 378
    },
    {
      "key": 3,
      "doc_count": 6
    },
    {
      "key": 5,
      "doc_count": 3
    }
  ]
},
```

Questa aggregazione chiamata `"by_material_type"` suddivide le richieste per tipo di materiale del riferimento bibliografico. La chiave `"key"` indica il tipo di materiale e il campo `"doc_count"` indica il numero di documenti in quel `bucket`. Il campo `doc_count_error_upper_bound` indica che non si sono verificati errori durante l'aggregazione, mentre `sum_other_doc_count` indica il numero di documenti

che non rientrano in nessun `bucket`.
Da questi dati, quindi, è possibile ottenere l'informazione che le richieste di ILL vertono prevalentemente verso i riferimenti bibliografici di tipo 1, ovvero articoli scientifici. Per una visione completa della risposta dell'API si consulti l'Appendice A.
Le funzioni `getRequestsCountriesLeaderboard`, `getRequestsCountriesFromLibrary` e `getRequestsCountriesFromCountry` seguono tutte la stessa filosofia d'implementazione di `getBorrowingStats`.
La funzione `getAvgWorkingTime` e rispettiva API con endpoint `/avg-working-time` sfrutta gli istogrammi, ovvero un metodo di aggregazione particolare di Elasticsearch e i campi script per calcolare la media della differenza tra `fulfill_date` e `request_date`. L'aggregazione `requests_per_month` ha un istogramma che suddivide le richieste in base a `request_date`, con intervallo mensile, nel formato `yyyy-MM` e mostra solo i mesi in cui è presente almeno una richiesta. Una volta suddivise le richieste, viene calcolato il tempo di lavorazione in millisecondi di ogni richiesta con il campo script e infine vengono aggregate ulteriormente con la funzione `avg` che calcola la media del tempo di lavorazione di una richiesta per ogni mese. Nel JSON di output dell'API, ogni mese viene descritto come di seguito:

```
{
  "key_as_string": "2023-07",
  "key": 1688169600000,
  "doc_count": 68,
  "avg_working_time": {
    "value": 13244176.470588235
  }
},
```

Da questi dati si può capire che nel mese di Luglio 2023 sono stati trovati 68 documenti e il tempo di lavorazione medio è di 13.244.176 ms, ovvero circa 3 ore e 40 minuti.

3.2 Implementazione del frontend con React

L'implementazione della dashboard per le statistiche in Talaria è strutturato secondo il flusso di React-Redux e Redux-Saga, come spiegato nella Sezione 2.4. A titolo dimostrativo, verrà preso come esempio il flusso di *BorrowingRequests*, ovvero le statistiche sulle richieste di *borrowing*.

1. **Definizione delle costanti per le azioni:** Nel file `constants.js` sono definite tutte le costanti delle azioni previste che verranno lanciate (o `dispatch`) nel corso dell'esecuzione di Talaria. Sono presenti tre costanti per ogni azione, una per il fetch, una in caso di successo e una in caso di fallimento.

   ```
   FETCH_BORROWING_REQUESTS
   FETCH_BORROWING_REQUESTS_SUCCESS
   FETCH_BORROWING_REQUESTS_FAILURE
   ```

   L'uso di costanti separate garantisce la gestione dello stato in applicazioni complesse come Talaria. Ogni fase di chiamata asincrona, come il fetch dei dati, il successo e il fallimento, devono essere gestiti in modo distinto per garantire che lo stato dell'applicazione rappresenti accuratamente la fase del ciclo di vita in cui Talaria si trova. Questa distinzione è fondamentale anche per effettuare debugging e manutenzione del codice in maniera semplificata.

2. **Implementazione del reducer:** Il file `reducer.js` contiene il reducer per le statistiche. Lo stato iniziale (`initialState`) contiene tutti i campi restituiti dalle statistiche impostati a `NULL`, con l'aggiunta di `loading` e `error`. Si passa attraverso un costrutto `switch-case` per distinguere l'azione lanciata ed eseguire l'azione di conseguenza. Ad esempio se viene lanciata l'azione `FETCH_BORROWING_REQUESTS` allora lo stato all'interno dello store di Redux deve essere di caricamento e senza errore.

3. **Definizione delle azioni:** Nel file `actions.js` sono definite tutte le azioni vere e proprie. Anche in questo caso ogni azione è suddivisa in tre funzioni:

una per la richiesta di fetch nella quale vengono passati tutti i parametri di filtro, una in caso di successo dove vengono restituiti i dati dell'API e una in caso di fallimento che restituisce l'errore riscontrato.

```
fetchBorrowingRequests
fetchBorrowingRequestsSuccess
fetchBorrowingRequestsFailure
```

Le azioni in Redux sono essenziali per mantenere la separazione delle responsabilità (*separation of concerns*) in un'architettura frontend. Ogni singola azione definisce un cambiamento dello stato dell'applicazione, permettendo di isolare la logica di aggiornamento dallo stato dei componenti che lo utilizzano. Ciò significa che i componenti non si occupano direttamente di come lo stato viene aggiornato, ma si limitano a lanciare azioni che attivano determinati cambiamenti all'interno del reducer[3].

4. **Chiamata delle API:** Nel file `apiAdmin.js` sono presenti tutte le chiamate API per quanto riguarda l'amministratore di sistema. La costante `admin_getBorrowingRequests` costruisce la richiesta HTTP in base ai parametri passati (dal parametro `options`) ed effettua una chiamata `GET` all'endpoint `/borrowing-requests-stats`.

5. **Implementazione di saga:** Il file `saga.js` contiene le funzioni generatrici che attendono il `dispatch` di un'azione e chiamano le API per restituire i dati. Nella funzione `statsSaga` si attende (con `yield`) che venga catturato un `dispatch` dell'azione `FETCH_BORROWING_REQUESTS` per chiamare la funzione `fetchBorrowingRequestsSaga`. Questa funzione prende come parametro `actions`, ovvero i parametri passati da `fetchBorrowingRequests` e li salva in un oggetto `options`. Successivamente chiama `admin_getBorrowingRequests` passandogli `options` e aspetta la risposta. Se non si sono verificati errori durante la chiamata API, i dati restituiti vengono messi

[3]Fonte: `https://medium.com/@ywongcode/separation-of-concerns-soc-design-principle-in-redux-using-redux-saga-210516b51160`, visitato il giorno 08/10/2024.

in `fetchBorrowingRequestsSuccess`, che lancia un'azione `FETCH_BORROWING_REQUESTS_SUCCESS`. Altrimenti l'errore viene messo in `fetchBorrowingRequestsFailure`, che lancia un'azione `FETCH_BORROWING_REQUESTS_FAILURE`.

La scelta di Redux-Saga rispetto ad altri middleware come Redux Thunk è motivata da alcune caratteristiche fondamentali che lo rendono più adatto nella gestione di comportamenti asincroni. Principalmente l'utilizzo di funzioni generatrici (`yield`) in Redux-Saga permette una gestione del flusso asincrono più chiara e controllata, a differenza di Redux-Thunk che utilizza `async/await`. In questo modo il codice dei Saga, in grandi applicazioni, è più leggibile e semplifica la manutenzione[4].

6. **Registrazione del reducer e di saga:** Il reducer delle statistiche implementato precedentemente viene aggiunto alla lista dei reducer già presenti in Talaria per formare un unico reducer chiamato `rootReducer` tramite la funzione `combineReducers`, che unisce più reducer per crearne uno unico. In questo modo ogni reducer può concentrarsi su una specifica parte dello stato, semplificando la gestione delle singole componenti dell'applicazione. Un'alternativa a questo approccio potrebbe essere la gestione dell'intero stato di Talaria con un unico reducer ma questo approccio renderebbe la gestione dello stato molto più complicata. Un'altra soluzione potrebbe essere la suddivisione in più store Redux, ma anche questo approccio complicherebbe la sincronizzazione tra le varie parti di Talaria, aumentando il rischio di errori[5].
   Nel file `configureStore.js` viene configurato il middleware Saga e vengono lanciati tutti i saga già presenti in Talaria, quindi è sufficiente chiamare la funzione `sagaMiddleware.run(statsSaga)` per registrare il saga delle statistiche.

7. **Creazione dei componenti e container di React:** Per ottenere i dati dall'API è sufficiente lanciare (con la funzione `dispatch()`) un'azione

[4]Fonte: `https://medium.com/@ertemishakk/redux-thunk-vs-redux-saga-b057ed1a72a3`, visitato il giorno 08/10/2024.

`FETCH_BORROWING_REQUESTS` tramite la funzione `fetchBorrowingRequests`.

Ora è possibile modificare il router del frontend per creare un percorso che mostri i nuovi componenti, tramite il file `adminRoutes.js`. Dato che Talaria supporta più lingue, è necessario inserire il testo mostrato nel file `messages.js`, che utilizza la funzione `defineMessages()` della libreria `react-intl`, che facilita la gestione dell'internazionalizzazione dell'applicazione.
L'internazionalizzazione (i18n[6]) è una componente fondamentale in Talaria, specialmente quando si tratta di gestire più lingue e culture. Le sfide principali includono la gestione delle lingue che si leggono da destra verso sinistra, come l'arabo. Un'altra sfida è legata alle performance di Talaria quando aumentano le lingue supportate e si caricano tanti file di traduzione. Una strategia di soluzione a questo problema è il caricamento lazy delle traduzioni, dove solo i file della lingua attualmente in uso vengono caricati. Oltre a questo, è presente il problema della gestione dei contenuti dinamici. Molti testi di Talaria vengono generati in base alle interazioni con l'utente o ai dati ricevuti. React Intl offre strumenti per tradurre in modo efficace testi dinamici, formattando numeri, date e valute in base alla lingua selezionata e adattandosi automaticamente alle regole culturali di ciascun paese.

Esempi di grafici e statistiche realizzati

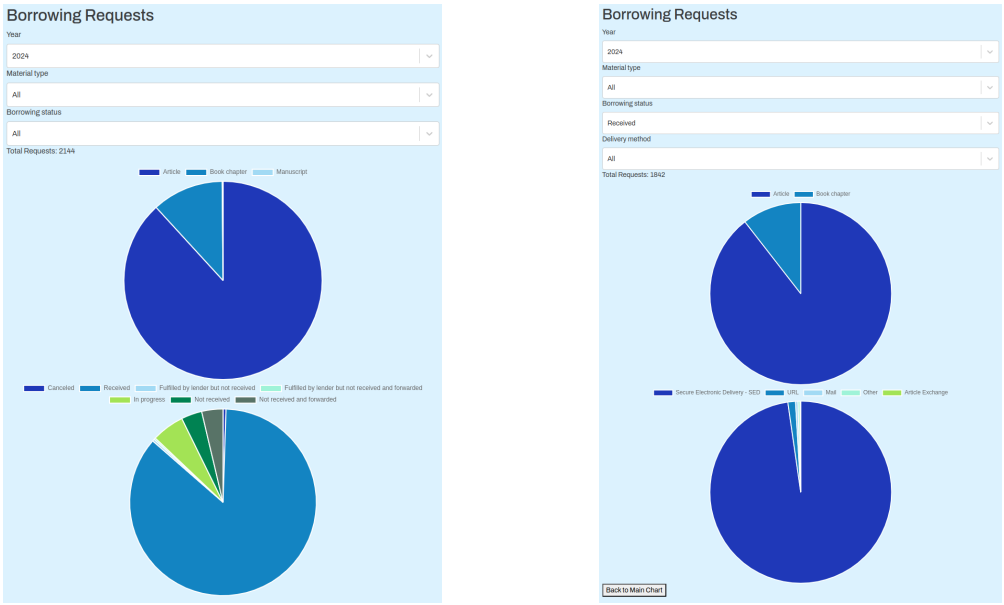
I grafici sono stati realizzati con la libreria open source `Chart.js`. `Chart.js` è stata scelta rispetto ad altre librerie di visualizzazione dei grafici, come ad esempio `D3.js` perché offre un'implementazione più semplice e immediata per la creazione di grafici comuni, rispetto a `D3.js` che permette maggiore flessibilità e personalizzazione, ma sono più complicati da implementare[7].
Per la realizzazione dei grafici a torta, come quelli mostrati nelle Figure 3.2a e 3.2b viene utilizzato il componente `Pie` della libreria `react-chartjs-2`. Un altro punto

[5]Fonte: `https://redux.js.org/usage/structuring-reducers/using-combinereducers`, visitato il giorno 08/10/2024.

[6]18 è il numero di lettere tra la 'i' e la 'n' in *internationalization*, creando "i18n".

[7]Fonte: `https://www.createwithdata.com/d3js-or-chartjs/`, visitato il giorno 08/10/2024.

di forza di `Chart.js` è il suo supporto nativo per l'interattività, come la possibilità di cliccare su una fetta di un grafico a torta per mostrare ulteriori dettagli, come avviene nella Figura 3.2b, dove viene mostrato un sottografico che appare quando si preme sulla fetta delle richieste con stato "Received" e mostra l'ulteriore suddivisione di tali richieste per metodo di evasione. Tale suddivisione è disponibile anche per le richieste con stato "Not Received" e ne mostra i motivi di inevasione. Questa funzionalità migliora l'esperienza utente, consentendo una navigazione intuitiva tra i dati e la visualizzazione di informazioni dettagliate solo quando necessario, senza intasare l'interfaccia e disorientare l'utente.
L'implementazione di funzionalità interattive presentano delle complessità tecniche, come la gestione di eventi che reagiscono alle interazioni dell'utente. La gestione di questi comportamenti interattivi è semplificata dalla libreria `react-chartjs-2`, riducendo la necessità di gestire manualmente i listener. Un listener, o *event listener*, è una funzione che si attiva quando viene effettuata una particolare azione, come ad esempio un click di un bottone o, in questo caso, di una fetta di grafico.



(a) Grafico generico (b) Grafico specifico richieste "Received"

Figura 3.2: Grafici della sezione "Borrowing Requests"

Per rappresentare il tempo medio di lavorazione di una richiesta, infine, viene utilizzato il componente `Bar` della libreria `react-chartjs-2` che mostra per ogni mese il tempo medio di lavorazione in giorni, come mostrato nella Figura 3.3.

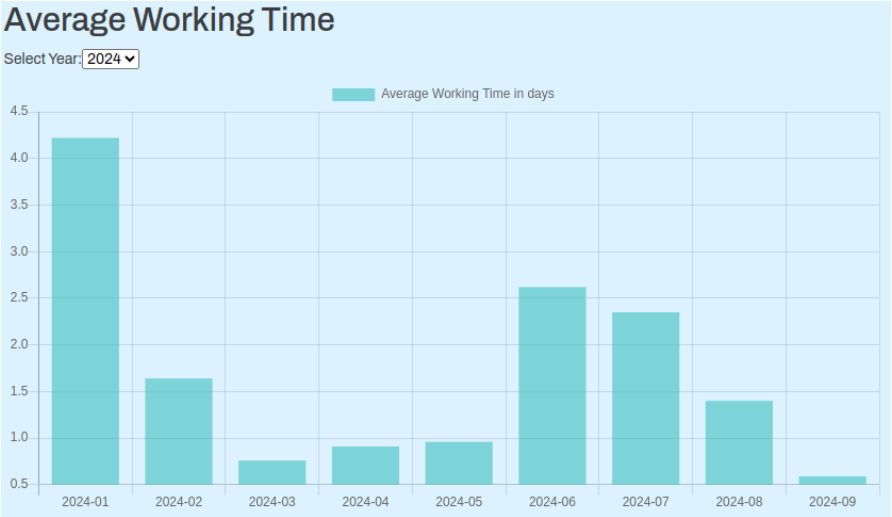


Figura 3.3: Sezione "Get average working time"

4 Valutazione

In questo capitolo si analizzano i vantaggi e gli svantaggi dell'integrazione di Elasticsearch in Talaria. Sono stati effettuati due test basati sulla previsione del carico di lavoro di Talaria. Tale previsione è basata sull'operatività di NILDE. In 15 anni di servizio, NILDE ha raggiunto circa 2 milioni di richieste, 2.5 milioni di riferimenti bibliografici e 2000 biblioteche registrate.
I test si sono effettuati, quindi, in due fasi: la prima si è concentrata sulla verifica di correttezza dei dati utilizzando il dataset reale di RSCVD, che al momento del test contava circa 3000 richieste, mentre la seconda ha riguardato l'efficienza del sistema, sfruttando dati fittizi di dimensioni variabili, da 10.000 fino a 1 milione di richieste. Infine si sono analizzate anche le limitazioni riscontrate.

4.1 Analisi dei risultati

Per verificare la correttezza delle statistiche è stata utilizzata una copia del database in produzione di RSCVD e i dati sono stati confrontati con le statistiche già esistenti, che vengono elaborate manualmente ogni mese e pubblicate sul sito web[1]. I test hanno confermato che Elasticsearch è in grado di gestire correttamente i dati reali, restituendo risultati accurati e coerenti.
Dopo aver verificato la correttezza dei dati con il dataset di RSCVD, l'attenzione si è spostata sulla valutazione delle prestazioni di Elasticsearch. Per simulare scenari più complessi e testare il sistema su grandi volumi di dati, sono stati creati dataset fittizi di dimensioni variabili, da 10.000 fino a un massimo di 1 milione di

[1]Consultabili al link `https://rscvd.ifla.org/rscvd-stats/`.

documenti. Per ogni dataset sono state lanciate 30 query uguali ed è stata calcolata la media del tempo di esecuzione sia in Elasticsearch che in MySQL nell'esecuzione della stessa query, al variare della dimensione del dataset.

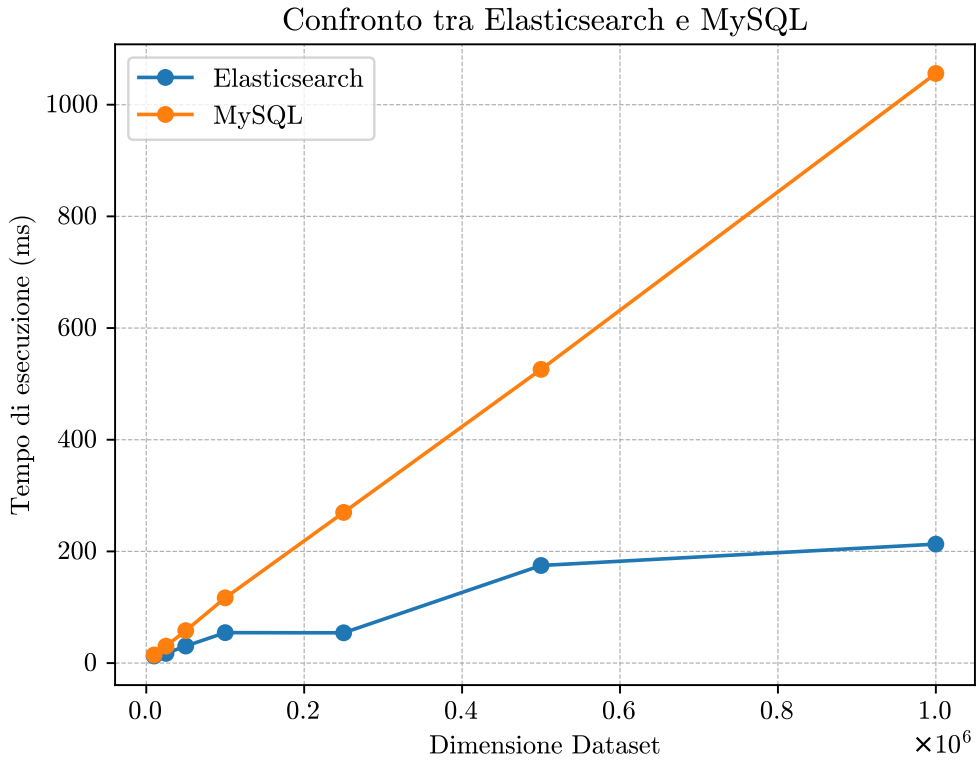


Figura 4.1: Confronto del tempo di esecuzione della query `avg-working-time`

La query in questione è `avg-working-time`, che convertita in linguaggio SQL è risultata come segue:

```
1 SELECT
2   DATE_FORMAT(request_date, '%Y-%m') AS request_month,  --
      Raggruppamento per mese
3   AVG(TIMESTAMPDIFF(SECOND, request_date, fulfill_date) * 1000) AS
       avg_working_time  -- Calcolo del tempo medio in millisecondi
4 FROM
5   docdel_requests
6 WHERE
7   fulfill_date IS NOT NULL
```

```
8     AND request_date IS NOT NULL
9   GROUP BY
10    DATE_FORMAT(request_date, '%Y-%m')  -- Raggruppamento per mese
11  HAVING
12    COUNT(*) > 0;  -- Filtraggio dei bucket con almeno un documento
```

Nella Figura 4.1 viene illustrato il confronto tra Elasticsearch e MySQL. È possibile notare subito una notevole differenza della crescita del tempo di esecuzione al variare della dimensione del dataset tra Elasticsearch e MySQL: quest'ultimo soffre di una crescita lineare, mentre Elasticsearch sembra avere un andamento asintotico, con crescita meno ripida che tende a 0 all'aumento costante del dataset. Si può notare, inoltre, l'efficienza di Elasticsearch comparata a quella di MySQL sui dataset di grandi dimensioni: con 1.000.000 di richieste, Elasticsearch è 5 volte più performante di MySQL. Un altro importante vantaggio di Elasticsearch rispetto a MySQL riguarda la semplicità con cui è possibile scrivere e gestire query complesse. Mentre in MySQL le operazioni di aggregazione richiedono query articolate e talvolta inefficienti, Elasticsearch offre strumenti che ne semplificano la gestione, come ad esempio l'utilizzo di campi script e aggregazioni.
Nella Sezione 3.1 è stato evidenziato il problema dell'*eventual consistency*. Per ovviare a questo problemaW è possibile effettuare l'indicizzazione di Elasticsearch con l'ausilio di job schedulati in periodi fissati. Si è voluta quindi valutare l'efficienza dell'indicizzazione di Elasticsearch al variare della dimensione del dataset, quest'ultimo veniva caricato solamente sul database MySQL e poi indicizzato su Elasticsearch, con l'utilizzo del job `InitializeElasticsearchIndex` definito precedentemente. Per non sovraccaricare la memoria di Elasticsearch, ogni indicizzazione è stata suddivisa in sotto-batch da 2500 documenti alla volta. Il valore 2500 è stato scelto perché all'istanza di Elasticsearch in esecuzione sono stati allocati 2GB di RAM come indicato nel file `docker-compose.yml` la riga `"ES_JAVA_OPTS=-Xms2g -Xmx2g"` indica le opzioni di configurazione Java e il parametro `-Xmx2g` specifica la dimensione massima della memoria utilizzabile, in questo caso 2GB. Un valore superiore a 2500 manda in crash il sistema per esaurimento di memoria. La Figura 4.2 illustra il tempo di esecuzione del Job. Si nota una crescita lineare al

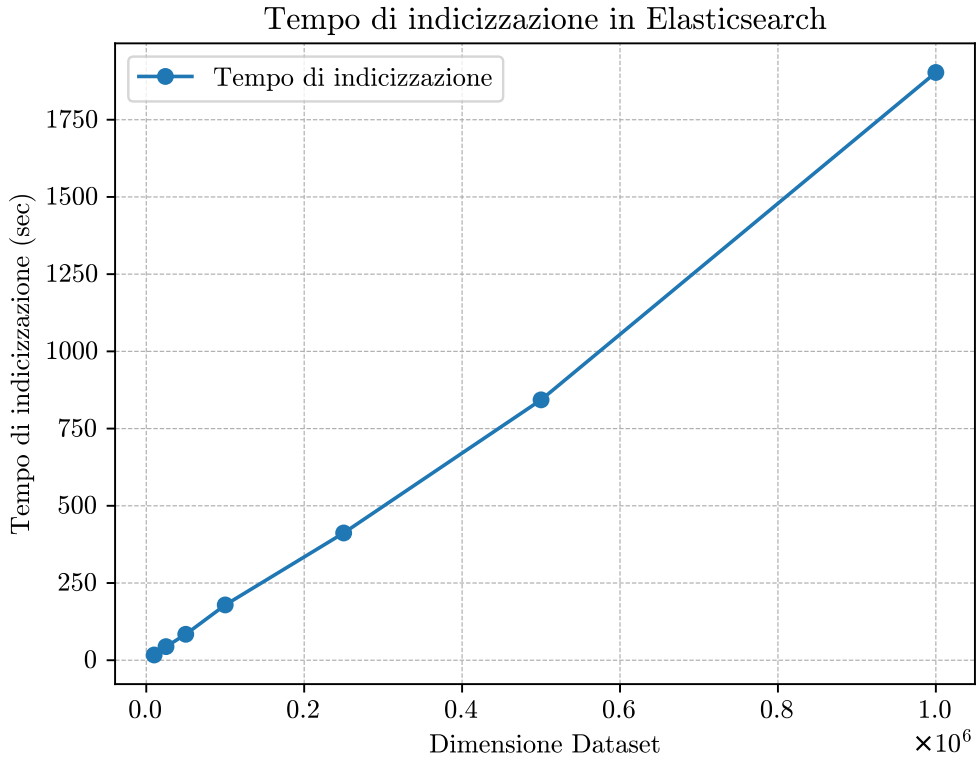


Figura 4.2: Tempo di esecuzione del Job `InitializeElasticsearchIndex`

variare della dimensione del dataset, fino ad arrivare a 32 minuti per indicizzare 1.000.000 documenti, un tempo più che accettabile.
Tutti i dati emersi dalle analisi effettuate sono consultabili nell'Appendice B.

4.2 Limitazioni riscontrate

Nonostante i vantaggi significativi di Elasticsearch, ci sono alcune limitazioni da evidenziare.
Elasticsearch, essendo un database non relazionale, non gestisce le join tra diversi set di dati a differenza di MySQL. È quindi necessario denormalizzare i dati per poter ottenere tutte le informazioni e, di conseguenza, aumenta lo spazio archiviazione utilizzato. La tabella `docdel_requests`, infatti, con 1.000.000 di documenti pesa su MySQL 101.7 MiB, mentre l'indice di Elasticsearch pesa cinque volte tan-

to, arrivando a 555.9 MiB.
Inoltre, Elasticsearch limita la dimensione dei dataset in output a 10.000 per le query di ricerca, per ragioni tecniche e di performance. Ad esempio, si ipotizzi una query che ricerca la parola "Covid" in tutti i riferimenti bibliografici oggetto di richiesta e che restituisce più di 10.000 elementi. Elasticsearch mostrerà solo i 10.000 elementi più pertinenti in base al rank calcolato dall'Equazione (1.1) presente nella Sezione 1.3, notificando l'utente nel campo `hits.total.relation` che ne sono disponibili degli altri, con `relation: gte`, ovvero che la relazione tra il numero di documenti che sono restituiti dalla query è maggiore o uguale al numero di documenti mostrato. La gestione delle query su grandi dataset, quindi, richiede l'implementazione di strategie per superare il limite imposto e questo significa che per ottenere tutti i dati necessari per analisi approfondite si renderà necessario implementare un sistema di "scrolling" o "search after" che permetta di suddividere la ricerca in più "blocchi" e scorrere l'intero dataset senza perdere documenti. Questa soluzione aggiunge una certa complessità all'architettura del sistema. Va però precisato che questo limite di 10.000 documenti si applica solo alle query di ricerca, mentre le aggregazioni, utilizzate per il calcolo delle statistiche globali operano senza limiti di dimensione.

5 Conclusioni e sviluppi futuri

In questa tesi è stato proposto l'utilizzo di Elasticsearch per l'elaborazione di dati statistici nel Resource sharing (RS) tra biblioteche, in particolare in Talaria, un software open source per la gestione di richieste di RS.
Il Capitolo 1 introduce gli scenari del caso di studio: la nascita e l'evoluzione del Resource sharing (RS) in Italia e nel mondo, i sistemi per il Resource sharing (RS) quali NILDE e la sua evoluzione in Talaria, un sistema di nuova generazione basato su Laravel, React e MySQL. Nonostante NILDE sia un sistema di Resource sharing popolare, presenta alcuni limiti nell'elaborazione di statistiche. Il calcolo delle statistiche è effettuato mediante l'uso di tabelle ausiliarie nel database create appositamente per velocizzare le query. Tuttavia l'indicizzazione e le ricerche si basano unicamente sugli strumenti messi a disposizione da MySQL, inoltre il ricorso a tabelle ausiliarie aumenta la complessità del codice. Questo approccio comporta sfide maggiori nel trattamento di grandi volumi di dati, rendendo il sistema meno scalabile e performante. Elasticsearch possiede alcune caratteristiche chiave come la capacità di gestire grandi quantità di dati in tempo reale e avanzate opzioni di ricerca, illustrate nella Sezione 1.3, che permettono di superare i limiti evidenziati.
Nel Capitolo 2 viene descritto il processo di integrazione di Elasticsearch con il backend di Talaria, sviluppato in Laravel, e con il frontend basato su React con l'ausilio di Redux e Redux-Saga, e come l'insieme di queste tecnologie renda possibile una visualizzazione interattiva delle statistiche. Viene inoltre portato alla luce il problema di sincronizzazione tra il database di MySQL e l'indice di Elasticsearch e proposto un semplice meccanismo per la sua risoluzione.
Il Capitolo 3 affronta i dettagli tecnici essenziali dell'integrazione di Elasticsearch

in Talaria. L'implementazione delle API per le statistiche e la sincronizzazione continua dei dati tra Elasticsearch e MySQL sono spiegate dopo aver mostrato la configurazione iniziale di Elasticsearch. Uno dei risultati più importanti è stata la possibilità di lavorare efficacemente su vasti dataset in modo rapido e scalabile. Utilizzando le funzionalità di aggregazione di Elasticsearch (descritte nella Sezione 3.1) è stato possibile rendere disponibili le statistiche avanzate molto più rapidamente di quanto possa essere fatto utilizzando MySQL. In particolare, le aggregazioni di Elasticsearch hanno permesso di eseguire calcoli complessi sui dati, come la media dei tempi di lavorazione delle richieste e la distribuzione delle transazioni tra le biblioteche.
Nel Capitolo 4 viene illustrata l'analisi delle performance di Elasticsearch, mettendola a confronto con MySQL. L'introduzione di Elasticsearch in Talaria ha comportato considerevoli migliorie sia in termini di efficienza del sistema che di flessibilità nel trattamento delle statistiche. I test sono stati effettuati sul dataset reale di un'istanza in produzione di Talaria: l'iniziativa di Resource sharing internazionale RSCVD, che conta, ad oggi, 250 biblioteche da tutto il mondo. I test hanno permesso di valutare l'accuratezza dei dati, così come l'affidabilità delle statistiche calcolate. Inoltre, i test condotti su dataset simulati hanno dimostrato che Elasticsearch è in grado di mantenere prestazioni eccezionali anche con grandi quantità di dati (Sezione 4.1). All'aumentare della dimensione del dataset la curva di Elasticsearch assume un andamento asintotico. Elasticsearch consente di elaborare le stesse statistiche fino a cinque volte più velocemente di MySQL, il quale, al contrario, mostra un aumento lineare dei tempi di elaborazione con l'aumento della dimensione del dataset (Figura 4.1). Tuttavia, prima di memorizzare i dati, è necessario denormalizzarli affinché Elasticsearch possa elaborarli in modo efficace. Sebbene questo metodo migliori le prestazioni, richiede più spazio su disco per l'indice di Elasticsearch, che può occupare fino a cinque volte lo spazio delle tabelle di MySQL (Sezione 4.2).
Un altro vantaggio è la semplicità dell'integrazione di Elasticsearch in Talaria derivante dalla sua facilità di configurazione, infatti bastano poche righe di codice per ottenere un'istanza di Elasticsearch funzionante. Tuttavia è anche vero che

Elasticsearch deve rimanere in esecuzione su una macchina e sincronizzare continuamente i dati con il database MySQL denormalizzando più tabelle e questo richiede una maggiore attenzione allo sviluppatore del software, come ampiamente discusso nelle Sezioni 4.1 e 4.2.
Sebbene questa tesi si sia concentrata sull'implementazione di Elasticsearch in un ambito applicativo specifico, quello dell'elaborazione di dati statistici in Talaria, le metodologie e le soluzioni suggerite possono essere un punto di partenza per qualsiasi progetto che utilizzi Elasticsearch in un ambiente Laravel, fornendo un approccio flessibile e replicabile in contesti applicativi completamente diversi.
Dati i risultati estremamente soddisfacenti, il nuovo modulo per l'elaborazione delle statistiche sviluppato nel presente lavoro di tesi sarà integrato nella prossima release software di Talaria.
Il lavoro svolto si presta a ulteriori ottimizzazioni che possono essere oggetto di interessanti sviluppi futuri. Sarà utile mettere a punto la dimensione della memoria allocata per avere un'indicizzazione ancora più rapida e studiare un meccanismo di sincronizzazione più sofisticato che si limiti ad indicizzare solamente i documenti che hanno avuto dei problemi durante la sincronizzazione tra Elasticsearch e MySQL. Con l'aumento del numero di biblioteche partecipanti e quindi anche di richieste di RS, sarà cruciale monitorare le prestazioni del sistema e garantire che resti scalabile e reattivo anche con dataset ancora più ampi. La natura distribuita di Elasticsearch rende estremamente facile la scalarità del sistema grazie all'aggiunta di nodi che distribuiscono il carico di lavoro (Sezione 1.3).
Attualmente le statistiche implementate si concentrano principalmente sulle richieste di RS, ma in un'ottica di miglioramento, è desiderabile ottenere statistiche che permettano l'analisi della distribuzione dei titoli delle riviste scientifiche più richieste. Tuttavia l'identificazione in maniera univoca dei titoli è un problema noto in letteratura ([MB16]) che non può essere risolto con la semplice estrazione dei dati presenti nel database, ma necessita di strumenti avanzati di analisi del testo. Per il futuro, sarà quindi importante approfondire lo studio per poter sfruttare le funzionalità offerte da Elasticsearch, come il machine learning per le analisi predittive.

Bibliografia

[AAİ16] Akca Mustafa Ali, Aydoğan Tuncay e İlkuçar Muhammer. «An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and Elasticsearch». In: *International Journal of Intelligent Systems and Applications in Engineering* 4 (Special Issue-1 26 dic. 2016). Number: Special Issue-1 Publisher: İsmail SARITAŞ, pp. 8–12. ISSN: 2147-6799. DOI: 10.18201/ijisae.271328. URL: https://dergipark.org.tr/en/pub/ijisae/issue/25999/271328 (visitato il 17/06/2024).

[BA15] Baich Tina. «Open access: help or hindrance to resource sharing?» In: *Interlending & Document Supply* 43.2 (1 gen. 2015). Publisher: Emerald Group Publishing Limited, pp. 68–75. ISSN: 0264-1615. DOI: 10.1108/ILDS-01-2015-0003. URL: https://doi.org/10.1108/ILDS-01-2015-0003 (visitato il 11/09/2024).

[BM11] Bernardini Elena e Mangiaracina Silvana. «The relationship between ILL/document supply and journal subscriptions». In: *Interlending & Document Supply* 39.1 (1 gen. 2011). Publisher: Emerald Group Publishing Limited, pp. 9–25. ISSN: 0264-1615. DOI: 10.1108/02641611111112101. URL: https://doi.org/10.1108/02641611111112101 (visitato il 24/09/2024).

[BR12] Brewer Eric. «CAP twelve years later: How the "rules" have changed». In: *Computer* 45.2 (feb. 2012). Conference Name: Computer, pp. 23–29. ISSN: 1558-0814. DOI: 10.1109/MC.2012.37. URL: https://ieeexplore.ieee.org/document/6133253 (visitato il 27/06/2024).

[CO24] Committee STARS International Interlibrary Loan. «2023 ALA RUSA STARS International Interlibrary Loan Survey Executive Report». In: (19 apr. 2024). Publisher: SharingTransforming Access to Resources Section (STARS) International ILL Committee, Reference & User Services Association (RUSA). URL: https://hdl.handle.net/11213/22471 (visitato il 24/09/2024).

[GA20] Gachet Jeremy. *Denormalization for Elasticsearch index*. Medium. 14 Ago. 2020. URL: https://medium.com/@jeremy.gachet/denormalization-for-elasticsearch-index-984ce7cfa50a (visitato il 12/07/2024).

[GGS15] Gyorödi Cornelia, Gyorödi Robert e Sotoc Roxana. «A Comparative Study of Relational and Non-Relational Database Models in a Web-Based Application». In: *International Journal of Advanced Computer Science and Applications (ijacsa)* 6.11 (15 gen. 2015). Number: 11 Publisher: The Science and Information (SAI) Organization Limited. ISSN: 2156-5570. DOI: 10.14569/IJACSA.2015.061111. URL: https://thesai.org/Publications/ViewPaper?Volume=6&Issue=11&Code=ijacsa&SerialNo=11 (visitato il 13/06/2024).

[KA23] Karunanithi Anandhan et al. «Data Synchronization Between MongoDB and Elasticsearch Using Monstache in Real-Time Data». In: *2023 4th International Conference on Communication, Computing and Industry 6.0 (C216)*. 2023 4th International Conference on Communication, Computing and Industry 6.0 (C216). Dic. 2023, pp. 1–6. DOI: 10.1109/C2I659362.2023.10430502. URL: https://ieeexplore-ieee-org.ezproxy.unibo.it/document/10430502 (visitato il 13/06/2024).

[KL17] Klepmann Martin. *Designing Data-Intensive Applications*. First. ISBN: 9781491903100. O'Reilly Media, Inc., mar. 2017, pp. 321–329. 611 pp. ISBN: 978-1-4919-0310-0. URL: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/ (visitato il 08/10/2024).

[LMM23] Lomba Carmen, Marzocchi Stefania e Mazza Debora. «HERMES, an international project on free digital resource sharing». In: (1 apr. 2023). Book Title: Beyond the Library Collections Publisher: ULiège Library. DOI: 10.25518/978-2-87019-313-6.15. URL: https://e-publish.uliege.be/beyond-the-library-collections/chapter/hermes-an-international-project-on-free-digital-resource-sharing/ (visitato il 01/08/2024).

[MA22] Mangiaracina Silvana et al. «Designing TALARIA - A New Software to Support Resource Sharing of International Communities». In: (2 dic. 2022). Accepted: 2022-12-02T12:34:01Z Publisher: International Federation of Library Associations and Institutions (IFLA). URL: https://repository.ifla.org/handle/123456789/2379 (visitato il 13/06/2024).

[MA23a] Madhusudhan Konda. *Elasticsearch in Action, Second Edition*. Second. Manning, set. 2023, pp. 60–99, 67–71. 368 pp. ISBN: 978-1-61729-985-8. URL: https://www.manning.com/books/elasticsearch-in-action-second-edition (visitato il 13/06/2024).

[MA23b] Mangiaracina Silvana et al. *TALARIA Software User Manual*. 24 Mar. 2023. (Visitato il 25/09/2024).

[MB16] Mangiaracina Silvana e Bernardini Elena. «Trends in interlibrary lending: a longitudinal data analysis on article sharing in Italy». In: 2016. URL: https://library.ifla.org/id/eprint/1425/ (visitato il 31/07/2024).

[MI07] Miguel-Stearns Teresa M. *Exchanging Books in Western Europe: A Brief History of International Interlibrary Loan*. Rochester, NY, 1 feb. 2007. URL: https://papers.ssrn.com/abstract=1345542 (visitato il 06/09/2024).

[MT12] Mangiaracina Silvana e Tugnoli Alessandro. «NILDE reloaded: a new system open to international interlibrary loan». In: *Interlending & Document Supply* 40.2 (1 gen. 2012). Publisher: Emerald Group Publi-

shing Limited, pp. 88–92. ISSN: 0264-1615. DOI: 10.1108/02641611211239551. URL: https://doi.org/10.1108/02641611211239551 (visitato il 08/09/2024).

[NA05] National Information Standards Organization. *ANSI/NISO Z39.88-2004 (R2010), The OpenURL Framework for Context-Sensitive Services*. 15 Apr. 2005. DOI: 10.3789/ansi.niso.z39.88-2004R2010. URL: https://www.niso.org/publications/z3988-2004-r2010 (visitato il 24/09/2024).

[SE18] Seda Pavel et al. «Performance testing of NoSQL and RDBMS for storing big data in e-applications». In: *2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG)*. 2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG). Apr. 2018, pp. 1–4. DOI: 10.1109/IGBSG.2018.8393559. URL: https://ieeexplore-ieee-org.ezproxy.unibo.it/document/8393559 (visitato il 13/06/2024).

[TAM17] Tugnoli Alessandro, Anderlini Jacopo e Mangiaracina Silvana. «NILDE Web Services: API e l'integrazione con altri sistemi». In: *Dead or alive? Le frontiere dei servizi bibliotecari nell'era della condivisione: 15 anni della comunità NILDE*. A cura di Elena De Carolis et al. RomaTrE-Press, ott. 2017. DOI: 10.13134/978-88-94885-41-5/19.

[VUL16] Vokorokos Liberios, Uchnár Matúš e Leščišin Lubor. «Performance optimization of applications based on non-relational databases». In: *2016 International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 2016 International Conference on Emerging eLearning Technologies and Applications (ICETA). Nov. 2016, pp. 371–376. DOI: 10.1109/ICETA.2016.7802079. URL: https://ieeexplore.ieee.org/document/7802079 (visitato il 17/06/2024).

[WE20] Wei Bizhong et al. «An Optimization Method for Elasticsearch Index Shard Number». In: *2020 16th International Conference on Computational Intelligence and Security (CIS)*. 2020 16th International Con-

ference on Computational Intelligence and Security (CIS). Nov. 2020, pp. 191–195. DOI: 10.1109/CIS52066.2020.00048. URL: https://ieeexplore-ieee-org.ezproxy.unibo.it/document/9407368 (visitato il 13/06/2024).

[WSZ23] Wei Peiyang, Shi Xiaoyu e Zhang Gang. «A Highly Accurate Data Synchronization and Full-text Search Algorithm for Canal and Elasticsearch». In: *2023 IEEE International Conference on Networking, Sensing and Control (ICNSC)*. 2023 IEEE International Conference on Networking, Sensing and Control (ICNSC). Vol. 1. ISSN: 2766-8665. Ott. 2023, pp. 1–6. DOI: 10.1109/ICNSC58704.2023.10318999. URL: https://ieeexplore-ieee-org.ezproxy.unibo.it/document/10318999 (visitato il 13/06/2024).

[ZM21] Zmaranda Doina R. et al. «An Analysis of the Performance and Configuration Features of MySQL Document Store and Elasticsearch as an Alternative Backend in a Data Replication Solution». In: *Applied Sciences* 11.24 (gen. 2021). Number: 24 Publisher: Multidisciplinary Digital Publishing Institute, p. 11590. ISSN: 2076-3417. DOI: 10.3390/app112411590. URL: https://www.mdpi.com/2076-3417/11/24/11590 (visitato il 13/06/2024).

A Risposta della chiamata API

Di seguito si illustra per intero la risposta di una chiamata `GET` all'endpoint `/borrowing-requests-stats`


```
{
  "took": 37,
  "timed_out": false,
  "_shards": {
      "total": 1,
      "successful": 1,
      "skipped": 0,
      "failed": 0
  },
  "hits": {
    "total": {
      "value": 2897,
      "relation": "eq"
    },
    "max_score": null,
    "hits": []
  },
  "aggregations": {
    "by_material_type": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
```

```
      "buckets": [
        {
          "key": 1,
          "doc_count": 2510
        },
        {
          "key": 2,
          "doc_count": 378
        },
        {
          "key": 3,
          "doc_count": 6
        },
        {
          "key": 5,
          "doc_count": 3
        }
      ]
    },
    "by_borrowing_status": {
      "buckets": {
        "canceled": {
          "doc_count": 19
        },
        "fulfilled": {
          "doc_count": 2141
        },
        "fulfillled_not_received": {
          "doc_count": 8
        },
        "fulfillled_not_received_forwarded": {
```

```
          "doc_count": 8
        },
        "in_progress": {
          "doc_count": 302
        },
        "not_fulfilled": {
          "doc_count": 199
        },
        "not_fulfilled_forwarded": {
          "doc_count": 220
        }
      }
    },
    "by_fulfill_type": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": 1,
          "doc_count": 2081
        },
        {
          "key": 4,
          "doc_count": 40
        },
        {
          "key": 2,
          "doc_count": 21
        },
        {
          "key": 6,
```

```
          "doc_count": 17
        },
        {
          "key": 5,
          "doc_count": 3
        },
        {
          "key": 3,
          "doc_count": 1
        }
      ]
    },
    "by_notfulfill_type": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": 2,
          "doc_count": 249
        },
        {
          "key": 1,
          "doc_count": 89
        },
        {
          "key": 7,
          "doc_count": 41
        },
        {
          "key": 5,
          "doc_count": 15
```

```
        },
        {
          "key": 4,
          "doc_count": 12
        },
        {
          "key": 3,
          "doc_count": 11
        },
        {
          "key": 6,
          "doc_count": 2
        }
      ]
    }
  }
}
```

B Tempi di esecuzione

Di seguito vengono elencati, al variare della dimensione del dataset, i tempi di esecuzione della query `avg-working-time` in MySQL ed Elasticsearch e il tempo di esecuzione del Job `InitializeElasticsearchIndex` che indicizza tutti i record della tabella `docdel_requests` di MySQL.

Dimensione dataset	**MySQL (ms)**	**Elasticsearch (ms)**	**Indicizzazione (sec)**
10.000	14,60	12,77	17
25.000	30,32	17,40	44
50.000	85,10	30,50	84
100.000	116,89	54,53	179
250.000	269,84	54,23	412
500.000	525,98	174,77	843
1.000.000	1055,96	213,07	1903

Tabella B.1: Tempi di esecuzione della query `avg-working-time` e del Job `InitializeElasticsearchIndex`