METODI E TECNICHE PER IL MIDDLEWARE

Attività 2.1 - d

Sunto

Analisi e progettazione dei contenuti con cui alimentare il Multimedia Manager per supportare un generico processo di Manutenzione Predittiva

Attività 2.1- Metodi e tecniche per il Middleware

Deliverable (Technical Report): Progettazione del componente Multimedia Manager nel rispetto dei requisiti di integrabilità con gli altri moduli del Middleware.

Sommario

Le Funzionalità del Multimedia Manager 3
Pipeline di caricamento e tagging semantico dell'informazione 5
Data Flow 6
Analisi del contenuto video 8
Analisi del contenuto audio 9
Entità e relazioni della Pipeline 11
Interazione con la componente Ontologica 14
Stack tecnologici del Multimedia Manager 15
Mongo DB 15
Datamodel 16
Sharding Architecture 17
Memory Management 18
Journaling 20
Indici 21
Replicazione 23
Logstash 25
Kibana 27
ElasticSearch 27
Elasticsearch architecture 29
Jena 31
Allocazione delle entità agli stack tecnologici 32
Pipeline di ricerca delle informazioni attraverso query intelligenti 33

API RESTful 35
Load RDF Graph 35
Update RDF Graph 35
Delete RDF Graph 35
Load Multimedia File 36
Monitoring Processing 36
Query 36
BIBLIOGRAFIA 37

Le Funzionalità del Multimedia Manager

Il Multimedia Manager ha il compito di fornire un servizio di ricerca indicizzata rispetto ai **contenuti multimediali** (video, audio e documenti), per potervi accedere in maniera puntuale e precisa.

Supponendo di avere a disposizione una collezione di file multimediali di tipo video e audio, il Multimedia Manager è in grado di estrarre i dettagli da questi contenuti e di proporli a chi li carica.

Nello specifico, la componente Multimedia Manager permette di caricare un insieme di file multimediali su uno store integrato tramite l'utilizzo di apposite funzioni API RESTful, durante quella che sarà descritta come fase di **upload**. All'avvio di tale procedura, il Multimedia Manager eseguirà una serie di processi conseguenti, esemplificati successivamente, con i quali sarà in grado di estrarre tutte le informazioni più significative per gli scopi previsti, siano esse trascrizioni, entità riconosciute, metadati ed entità linkate.

Nella sua implementazione attuale, le entità riconosciute, così come le trascrizioni del file audio in testo e i cambi di scena, sono individuati grazie ai servizi messi a disposizione dalle Google Video Intelligence API [1]. Diversamente, i metadati saranno tutte quelle informazioni semantiche che un amministratore di sistema considererà necessarie da aggiungere durante la fase di caricamento del relativo file multimediale a cui si corredano. Come si vedrà nella sezione apposita, la fase di upload prevede la possibilità di arricchire il file in questione di metadati, costituenti veri e propri tag semantici, opportunamente scelti da un esperto di dominio e ritenuti fondamentali per una più rapida e significativa ricerca intelligente.

Infine, il sistema prevede la possibilità di immagazzinare un qualsiasi dataset RDF, sia esso coincidente con un Grafo di Conoscenza che con una Ontologia, caricato opportunamente nella sua fase di configurazione. In questo modo è possibile sia riutilizzare grafi di conoscenza già presenti nello scenario tecnologico odierno, selezionati e caricati da un amministratore seguendo le procedure previste dal Multimedia Manager, che identificarne di nuovi, costruiti a seconda delle necessità di dominio e caricati utilizzando la stessa procedura

[1] https://cloud.google.com/video-intelligence

precedentemente accennata. Ne consegue la possibilità di definire anche una innovativa Tassonomia, o una più generale Ontologia, che vada ad arricchire la semantica conosciuta dal Multimedia Manager.

In questo modo, esso potrà riferirsi direttamente al dataset RDF così caricato per realizzare l'annotazione semantica delle informazioni estraibili dalle componenti multimediali del file in input alla pipeline, rendendoli disponibili aggiuntivamente alle informazioni rilevate dalle Video Intelligence API e ai metadati corredanti lo stesso file d'interesse.

Sfruttando un particolare mapping su Elasticsearch[2], tutte queste informazioni collezionate consentiranno all'utente/software, che intende interagire con tale componente, di recepire quanto più correttamente possibile il file multimediale collegato al caso d'uso per cui lo stesso viene richiesto.

[2] https://www.elastic.co/

Pipeline di caricamento e tagging semantico dell'informazione

L'architettura del framework Multimedia Manager conta di una serie di moduli ben integrati e dalle funzionalità diversificate in modo da garantire un servizio scalabile e facile da manutenere. Il fulcro di tale componente coincide con la pipeline di processing orientata al flusso dei dati, per cui ogni modulo avrà una sua routine specifica intersecata sapientemente con i risultati e il processing effettuati dai restanti moduli sul dato multimediale centralizzante l'attenzione del framework.

Interfacciandosi con l'utente, sia esso il fruitore finale del servizio che un amministratore di sistema, per la fase di configurazione o di eventuale manutenzione del prodotto, attraverso delle API RESTful, il Multimedia Manager mette a disposizione le seguenti funzionalità:

- **Upload del file multimediale**: caricamento nello storage del servizio di un file multimediale, che sarà processato e conservato dalla componente includendo tutte le informazioni semanticamente valide individuate dal servizio stesso, o dichiarate manualmente da un amministratore durante questa fase;
- **Upload dataset RDF**: caricamento di un Grafo di Conoscenza, Tassonomia o Ontologia all'interno del Triple DataBase previsto per il Multimedia Manager (esso può coincidere sia con una Ontologia specifica di dominio che con un grafo di conoscenza più General Purpose come DBpedia, ed entrambe possono coesistere). Esso sarà interrogabile dalla componente stessa per poter realizzare delle annotazioni semantiche più sofisticate, che arricchiscono quelle individuate dall'annotazione manuale proposta dall'amministratore e quella del servizio Google Video Intelligence;
- **Ricerca intelligente**: ricerca dei contenuti multimediali tramite un testo esemplificante il caso d'uso del fruitore finale, in modo tale da restituire l'oggetto multimediale più appropriato alla risoluzione della problematica per cui si è richiesto l'intervento della componente.

Come intuibile, il tagging semantico può avvenire in tre fasi distinte, e a seconda del momento in cui viene realizzata, tale annotazione risulta essere di diverso tipo.

1. Una prima annotazione, infatti, può avvenire contemporaneamente alla fase di caricamento del file multimediale sulla piattaforma. Attraverso una POST, infatti, un amministratore o un utente esperto può corredare il file da caricare nella componente Multimedia Manager con un insieme di metadati ritenuti semanticamente validi dall'utente stesso, in modo tale che essi costituiscano una prima informazione semantica da associare, internamente al sistema, al video con cui si presentano. Il file multimediale e tali metadati vengono caricati contemporaneamente prevedendo, inoltre, la possibilità che al file in questione non venga corredata alcuna informazione aggiuntiva. In tal caso sarà il sistema ad individuare delle annotazioni valide in maniera automatica.
2. Una seconda annotazione (o prima nel caso in cui il video sia stato caricato privo di metadati aggiuntivi) viene realizzata in parallelo al processing del file multimediale dalla piattaforma progettata. In questo senso, è il servizio Google Video Intelligence che propone, oltre alla trascrizione della componente audio in testo, e alla identificazione dei cambi di scena, anche un proprio tagging semantico in base alle entità riconosciute dai propri algoritmi all'interno del file.
3. Un ultimo tagging semantico si realizza, invece, al termine della fase di processing del file multimediale, dove si sfrutta il testo risultante dalla trascrizione e l'ontologia, o grafo di conoscenza, presenti nel Triple Data Base (TDB) della componente. Qui l'annotazione automatica va alla ricerca di entità note nel TDB all'interno della trascrizione, arricchendo il file in questione di tutte quelle informazioni che tale processo riesce ad acquisire. Per rendere possibile ciò, ovviamente, deve essere stata precedentemente caricata un'ontologia, o grafo di conoscenza, all'interno del TDB.

Tali informazioni sono successivamente mappate opportunamente su una collezione Elasticsearch, che ne rende performante una successiva loro ricerca.

Data Flow

L'ecosistema del Multimedia Manager prevede uno specifico **flusso di dati** progettato secondo i principi di ottimizzazione dei processi, correttezza e coerenza delle informazioni oltre che scalabilità e integrabilità dell'applicativo.

Al fine di esemplificare completamente il percorso delle informazioni tracciato dal framework, si riporta il processing di un file video in quanto contenente la totalità delle componenti multimediali, video, audio e testo, che il sistema dovrà gestire.

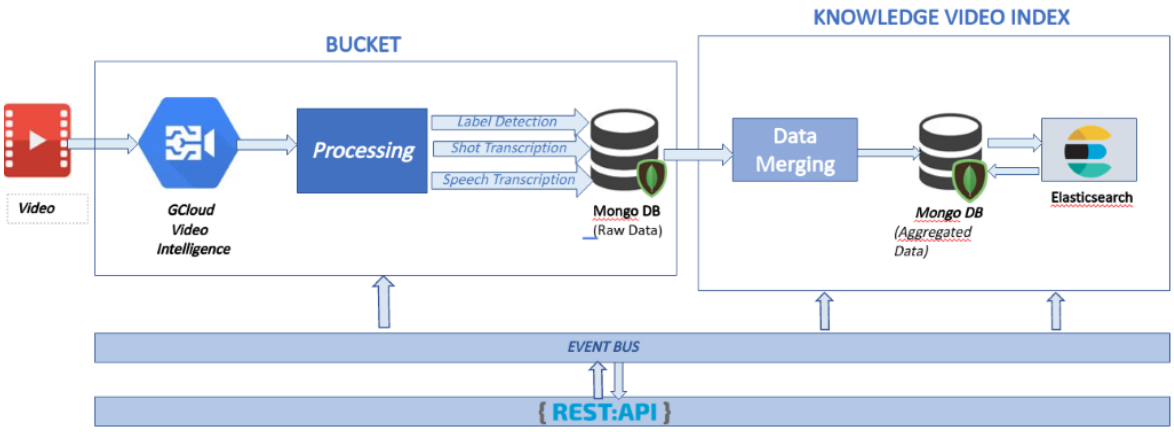

Figura 1: Architettura Data Flow del Sistema

<u>Upload.</u> Come raffigurato nella Figura 1, il file in questione viene caricato nel sistema attraverso una funzione di upload messa a disposizione dalle API corredanti il framework, insieme ad eventuali annotazioni manuali prodotte dall'amministratore/utente esperto. Al termine dell'upload, essa chiamerà in cascata un processo avente il compito di popolare un bucket della Google Cloud Platform, configurato in fase di installazione dell'applicativo, con lo stesso file in questione rinominato secondo una nomenclatura univoca, e la collezione "global" della Base di Dati Mongo per i dati aggregati con le informazioni utili al successivo processing e le prime annotazioni semantiche fornite in upload con il video.

<u>Annotazione.</u> Terminata la fase di upload, un secondo processo verrà avviato per effettuare l'analisi e il processing del video sfruttando le potenzialità della piattaforma GCloud Video Intelligence. Il servizio verrà avviato sul video, precedentemente caricato, selezionandolo attraverso l'identificativo univoco assegnatogli dal primo processo.

Le annotazioni grezze risultanti sono pertanto immagazzinate all'interno della collezione "video" del Raw Data MongoDB, congiuntamente all'alias univoco del file in questione, per il successivo processing attuato dalla componente Multimedia Manager finalizzante l'informazione strutturata, e identificante altre possibili annotazioni riscontrabili dal modello RDF precedentemente caricato all'interno della componente.

Ogni qual volta il database Aggregated Data viene aggiornato, grazie ad un esecutore di pipeline implementato tramite logstahs, viene finalizzato il mapping su Elasticsearch in modo tale da disporre tutte le informazioni finora riscontrate per la realizzazione di successive query intelligenti, che consentono quindi di restituire in output ad una generica richiesta effettuata al Multimedia Manager il video soddisfacente le condizioni richieste dall'utente e la sua precisa porzione in cui le entità volute possono essere ritrovate.

Analisi del contenuto video

La gestione di un contenuto video per il Multimedia Manager prevede una prima fase nella quale il file (con estensione .avi o .mp4) viene preso in carico dall'operatore e caricato sulla piattaforma. In questo momento, il video potrebbe essere taggato dall'operatore, il quale potrebbe classificarlo come appartenente ad una specifica categoria rispetto al dominio di riferimento [1].

Completata la fase di upload, il video viene processato in modo da estrarre quelle che sono le features salienti e utili da indicizzare. Parte a questo punto un processo che definiamo come processo di annotazione. In particolare, possiamo identificare tre fasi del processo che sono distinte tra loro ma che operano in parallelo sul video stesso.

1. **Video Labeling:** in questa fase il video viene annotato con delle etichette che identificano quelle che sono le entità riconosciute all'interno del flusso video stesso. Quello che otteniamo come risultato è una lista di annotazioni dei segmenti video in cui le entità sono state identificate, una lista di annotazioni dei frame in cui le entità sono state riconosciute ed infine una lista di inquadrature in cui le entità sono state trovate dal modello. La gestione di questa fase avviene tramite un modello di rete neurale che è pre-addestrato per compiere questo task di annotazione.
2. Obiettivo di questa prima fase è ottenere un insieme di quelle che sono le entità presenti globalmente nel video e identificare anche in quale porzione del video stesso queste sono presenti. Per questa ragione si è deciso di dividere le informazioni del video in due strutture dati separate. Una che riguardasse il video nella sua interezza ed un'altra, collegata alla prima, che ne tracciasse i singoli slice con tutte le informazioni annesse e relative solo a quella porzione.

3. **Shot Change Detection:** la fase in oggetto viene avviata quando si fa richiesta di un'analisi del video a livello di frame. In questo modo, il modello pre-addestrato di rete neurale lavora su tutte le immagini messe in sequenza che compongono l'intero video e sfrutta alcune funzionalità del Label Detection al fine di poter realizzare un cambio di scena più sensato, dunque non esclusivamente basato sul cambio repentino di scena da frame a frame, poiché questo potrebbe risultare anche semplicemente da un taglio del video stesso in fase di editing. Nello specifico, Shot Change Detection sfrutta delle annotazioni di entità realizzata al livello di frame in modo tale da individuare un cambio di scena quando queste entità si modificano durante alcune transizioni, combinato alla valutazione del cambio repentino del tipo di immagine caratterizzante frame conseguenti.

   Il risultato, pertanto, oltre a collezionare tutte le entità riconosciute prima e dopo il cambio di inquadratura considerato come tale, presenta al suo interno anche tutti i possibili cambi repentini dell'inquadratura analizzata solo sui frame.
   In questo caso le informazioni vengono solo memorizzate solo nella struttura dati che tiene traccia dei dettagli a livello di slice del video.
4. **Speech to text:** questa fase inizia parallelamente alle altre e si occupa di estrarre il contenuto audio dal flusso video. Questo processo viene realizzato mediante un altro modello di rete neurale pre-addestrata per compiere questo task. Il modulo dello speech to text fornisce come output la trascrizione del parlato per un determinato intervallo temporale che potremmo far coincidere con una scena, e per ognuna di queste scene fornisce una lista delle parole presenti nel frammento con l'indicazione temporale nel quale questa è pronunciata. Combinando insieme il risultato del modulo Speech to Text e il risultato del modulo Shot Change Detection possiamo stimare al meglio il cambio di scena nel video: non solo cambio di inquadratura ma cambio completo del contesto analizzato dal punto di vista semantico.

Analisi del contenuto audio

La gestione di un contenuto audio, come per il tipo di formato video, prevede una prima fase di caricamento del file (con estensione .mp3 o .wav o .acc) all'interno della piattaforma. Appena completato il caricamento inizia la fase di processing del contenuto audio presente nel

file. È importante sottolineare che la fase di analisi audio parte automaticamente quando si avvia anche l'analisi di un contenuto video, e quindi la fase di Speech to Text è in comune tra le due.

La peculiarità dell'analisi del contenuto riguardano le seguenti caratteristiche:

1. **Identificazione dello o degli speaker:** questa fase dell'analisi del contenuto audio si occupa di identificare, all'interno di finestre temporali che coincidono con le scene identificate come indicato nel modulo Speech to Text, quali sono le voci dei protagonisti coinvolti nel video e quando questi intervengono nella finestra temporale in analisi. Inoltre, fornisce una serie di statistiche, opzionali, rispetto a quest'ultima analisi
2. **Estrazione delle parole chiave e Named entitiy:** questo modulo si occupa di identificare ed estrarre tutte le parole chiave identificate nelle diverse finestre temporali. Quando possibile, questa fase restituisce anche un entity linking di queste keyword con una specifica base di conoscenza. In output avremo sempre le keyword relative alle possibili scene identificate ed eventualmente un linking di queste.
3. **Topic Inference:** il ruolo di questo modulo, all'interno dell'analisi del contenuto audio, è quello di intensificare uno o più possibili topics centrali per il contenuto audio in analisi. Saremo quindi in grado di identificare il contesto globale del nostro contenuto multimediale.
4. **Speech to text:** il ruolo di questo modulo è speculare rispetto a quanto già dettagliato per l'analisi del contenuto video.

Entità e relazioni della Pipeline

L'intera Pipeline del Multimedia Manager può essere scomposta in cinque macro-moduli, ognuno implementato sfruttando vari servizi esemplificati successivamente:

- **Loading Module**: atto al caricamento dei file multimediali e dei dataset RDF per ontologie/grafi di conoscenza;
- **Processing Module**: per l'analisi dei contenuti dei file e l'estrazione di informazioni raw utili agli scopi previsti per il Multimedia Manager;
- **Composer Module**: per la gestione e organizzazione strutturata delle informazioni raccolte per i file processati;
- **Mapping Module**: per l'indicizzazione dei contenuti informativi per ogni file multimediale allo scopo di renderli facilmente esplorabili/interrogabili;
- **Searching Module**: per effettuare ricerche mirate ed intelligenti al fine di ottenere uno specifico file multimediale soddisfacente determinate condizioni.

Essendo una pipeline, i moduli sono relazionati tra di loro in cascata, seguendo esattamente il precedente ordine di presentazione degli stessi, pertanto ogni entità avvierà la propria esecuzione al termine della precedente, se non diversamente specificato.

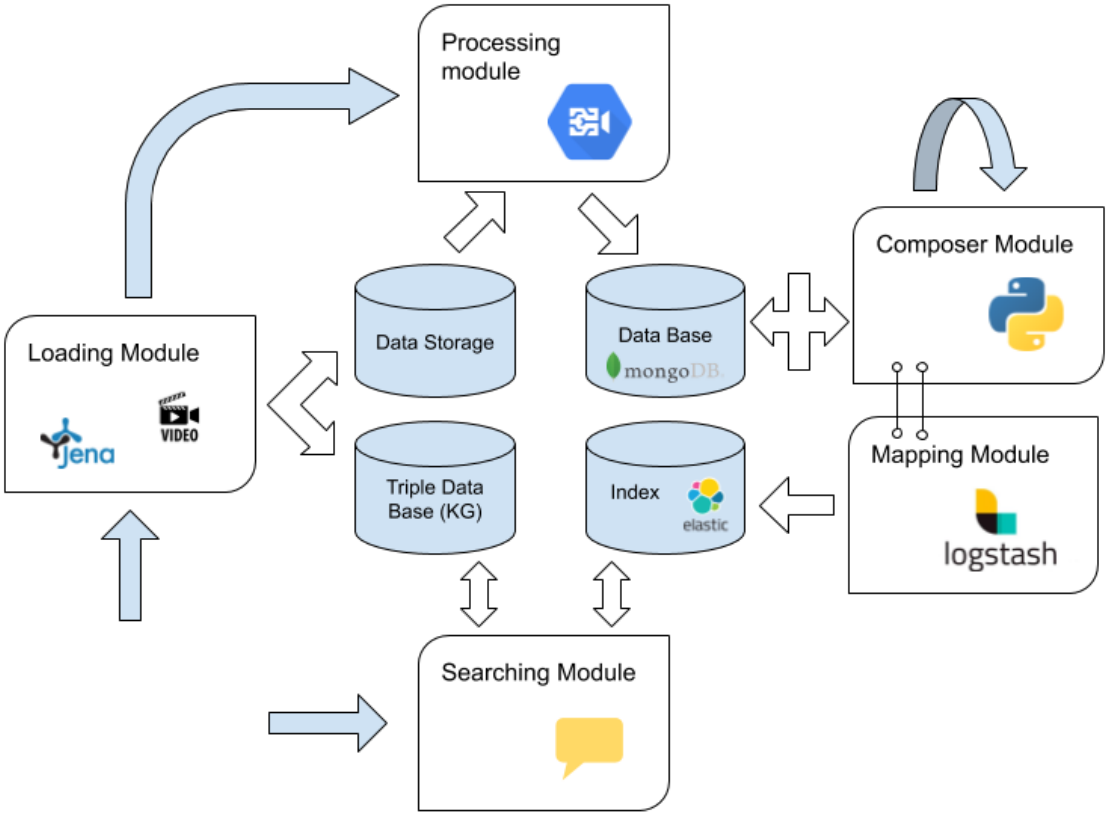

Figura 2: Entità del Multimedia Manager

Nel dettaglio, alcuni di essi opteranno per un controllo di jobs da effettuare ad intervalli regolari analizzando i dati raccolti fino al momento di esecuzione del controllo stesso, mentre altri si avvieranno non appena sarà verificato un evento scaturito dal termine del modulo precedente all'interno della pipeline.

Si può, quindi, desumere che la relazione orchestrante tutti i moduli progettati internamente al Multimedia Manager sia data dal flusso dati della piattaforma, rendendolo il suo cuore principale.

Praticamente, il Loading Module viene avviato successivamente all'esecuzione di una chiamata REST alle API previste, diversificando le sue operazioni in caricamento del file multimediale all'interno della piattaforma e caricamento del modello RDF per la componente ontologica. Esso provvederà al caricamento all'interno della piattaforma dello specifico file in ingresso, potendo gestire anche le informazioni aggiuntive previste dall'utente per un generico file multimediale.

Una volta terminata l'esecuzione del Loading Module, si avvierà successivamente l'esecuzione del Processing Module, che provvederà all'avvio dei servizi GCloud sul file multimediale caricato allo scopo di ottenere un'annotazione, informazioni relative al cambio di scena e trascrizioni della componente audio. Esso si occuperà, inoltre, di provvedere al tagging semantico del testo trascritto, risultante dalle operazioni di GCloud, nel caso in cui la componente ontologica del Multimedia Manager risulti essere già definita, sfruttando quest'ultima per l'identificazione di entità, relazioni e categorie utili.

Il risultato finale del Processing Module sarà il popolamento di *document* all'interno di MongoDB Raw Data con tutte le informazioni utili ricavate, immagazzinate in forma grezza.

Il Composer Module, invece, avvia i suoi processi controllando periodicamente i document immagazzinati all'interno del DB Aggregated Data, ispezionando opportuni flag segnalanti la necessità di processare i dati grezzi collezionati per uno o più file. Qualora tale condizione sia soddisfatta, il Composer prosegue con la strutturazione e riorganizzazione dei dati grezzi in dati aggregati, preparando le informazioni a garantire la ricerca minuziosa di porzioni di video in base alla loro semantica sottostante.

Ogni volta che il DB Aggregated Data viene aggiornato di nuovi elementi, logstash identifica un evento che consente l'avvio del modulo successivo, ossia il Mapping Module, che permette di indicizzare tutte le nuove informazioni presenti sulla precedente collezione all'interno di Elasticsearch, che permetterà di effettuare ricerche performanti ed intelligenti.

Il Searching Module, infine, verrà avviato tramite chiamata REST alle API previste ammettendo in ingresso una generica stringa contenente termini ritenuti fondamentali per la ricerca da parte dell'utente. Il modulo produrrà, pertanto, una query su Elasticsearch in modo tale che possa restituire il video che meglio soddisfa la ricerca dell'utente, la porzione di testo in cui i termini ricercati dall'utente sono identificati e il minutaggio video in cui si identifica la scena di conteso alla ricerca dell'utente.

Interazione con la componente Ontologica

La componente ontologica prevista all'interno del Multimedia Manager è realizzata tramite l'uso di Apache Jena Fuseki, esemplificata nella sezione seguente. Essa prevede, sempre interfacciandosi tramite delle RESTful API, la possibilità di caricare, aggiornare o cancellare dei modelli RDF costituenti grafi di conoscenza, tassonomie e ontologie. In questo senso, il framework così ideato avrà al suo interno una propria base di conoscenza a cui fare affidamento per qualsiasi task semantico, sia esso affine al Web Semantico che a ricerche intelligenti che vanno oltre la classica ricerca di contenuti mediante keyword.

In questo modo, è possibile interagire con la componente ontologia del Multimedia Manager sia attraverso delle chiamate REST alle API previste, che effettuando delle query SPARQL, supportate dalla stessa piattaforma Jena Fuseki integrata, dando anche la possibilità di realizzare un tool completamente automatizzato per il tagging semantico degli stessi contenuti collezionati all'interno dello storage del Multimedia Manager.

Stack tecnologici del Multimedia Manager

Il Componente del Multimedia Manager, si serve di diverse componenti software tra loro interconnesse. Di seguito le caratteristiche di ognuna.

Mongo DB

MongoDB è un database orientato ai documenti che fa parte della categoria dei database non relazionali o NoSQL [2]. Si tratta di una soluzione che sfrutta il formato JSON per la memorizzazione e la rappresentazione dei dati.

Un database orientato ai documenti memorizza i dati in documenti, ossia in oggetti complessi codificati senza uno schema rigido.

I Database NoSQL hanno diverse caratteristiche, tra le tante citiamo:

1) non utilizzare il modello relazionale;

2) utilizzare tabelle e campi ma senza schemi fissi;

3) non permettono vincoli di integrità referenziale;

Di seguito un esempio di possibile *document* collezionato all'interno di MongoDB:

```
{   nato: 1940,
    morto: 2020,
    lingue: ["italiano", "cinese", “francese”, “arabo”],
    libro: {  titolo: "Homo Deus",
              lingua: "italiano",
              capitoli: ["Prefazione", "Introduzione","Bibliografia"]
    }
}
```

Non esiste uno schema fisso per il raggruppamento dei documenti di MongoDB, questi vengono raggruppati in collezioni che possono essere anche eterogenee tra di loro. Tra le collezioni non esistono relazioni o legami garantiti da MongoDB.

Di seguito due delle caratteristiche più importanti di MongoDB:

- Sharding o scalabilità automatica: garantiscono la possibilità di distribuire le collezioni in cluster di nodi, in maniera tale da poter supportare grosse quantità di dati senza influire negativamente sulle performance;
- Replica Set o alta disponibilità dei servizi: la replicazione dei servizi può avvenire in modo semplice.

MongoDB è una soluzione che si adatta molto bene quando si manipolano grandi quantità di dati eterogenei e senza uno schema, mentre non è consigliato quando si devono gestire molte relazioni tra oggetti e si vuole garantire l'integrità referenziale tra essi.

Datamodel

In MongoDB il document è l'unità di base del database ed equivale ad una riga dei database relazionali [2].

È possibile riassumere le equivalenze di termini fra gli RDBMS e MongoDB con la seguente Tabella:

MongoDB	RDBMS
Database	Database
Collection	Tabella
Document	Record
Field	Colonna

Tabella 1: equivalenza di termini fra MongoDB e RDBMS

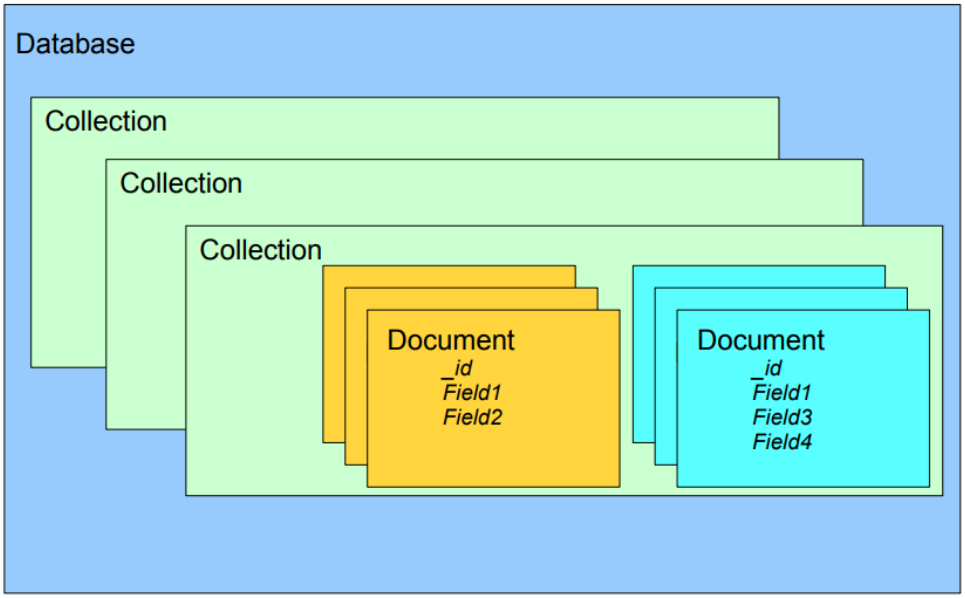

Figura 3: MongoDB Datamodel

Sharding Architecture

MongoDB può scalare orizzontalmente in maniera lineare. L'architettura Sharding è costituita da tre componenti:

1) **Shards:** immagazzinano i dati, per garantire la consistenza e la disponibilità dei dati. Sono spesso costituiti da dei Replica Sets;

2) **Query Routers o istanze mongos:** interfacce che redirezionano le richieste verso gli shards corretti e restituiscono il risultato ai vari client.

3) **Config servers:** contengono i metadati che descrivono il mapping tra dati e shards.

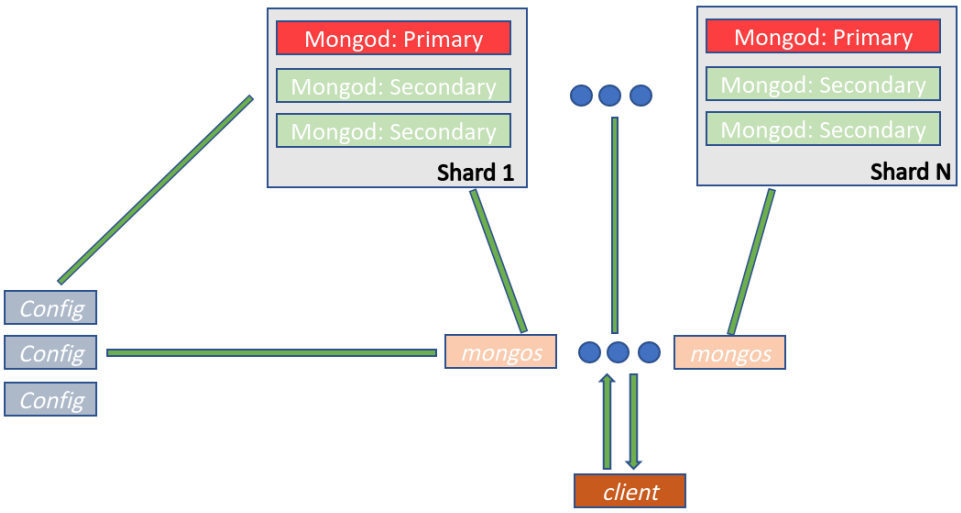

Figura 4: MongoDB Sharding Architecture

Il partizionamento ed il bilanciamento vengono fatti in maniera automatica in base ad una Field del document (Shard Key) che si decide di utilizzare. Successivamente è possibile decidere, per la suddivisione dei dati in chunks, se farlo in modalità Range Based Sharding o Based Sharding:

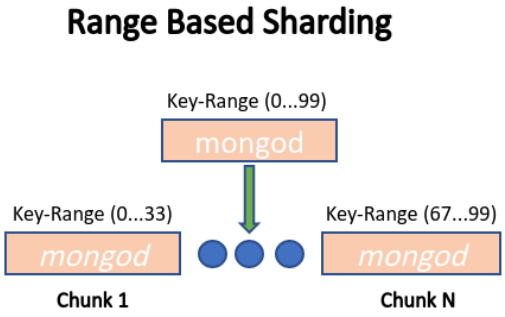

Figura 5: Range Based Sharding in Mongo DB

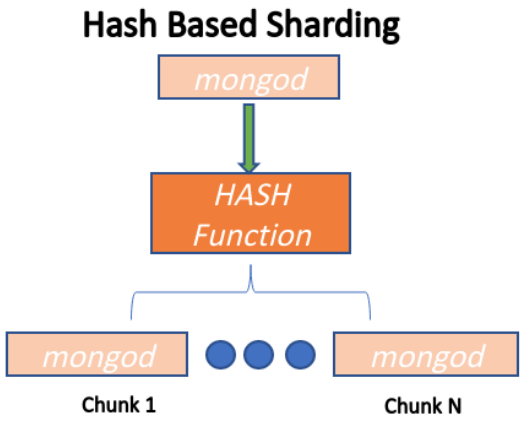

Figura 6: Hash Based Sharding in Mongo DB

Memory Management

La gestione della memoria in MongoDB è mirata alla massima semplificazione e sfrutta il concetto di file mappato in memoria in modo persistente, dove ai segmenti della memoria virtuale viene assegnata una correlazione byte a byte con una porzione del file fisico in cui risiedono i dati.

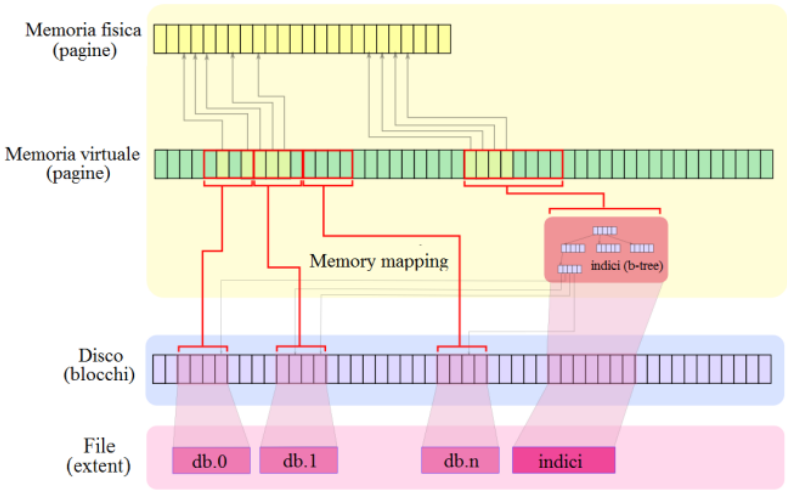

Figura 7: Mappatura di un file in memoria.

La memoria virtuale è una rappresentazione privata a livello di processo della memoria fisica e permette di astrarre l'indirizzamento su di essa.

Dal punto di vista del processo, un file mappato in memoria virtuale ha le stesse metodologie di accesso di un array di byte. L'indirizzamento della memoria virtuale su quella fisica è compiuto in modo trasparente dal sistema operativo, che si occupa di gestire la complessità di tutte le operazioni di accesso. Quando il processo richiede una pagina non ancora mappata in memoria, si genera un page fault, a cui il sistema operativo farà seguire la ricerca dell'informazione su disco. Alla conclusione di questa operazione si possono presentare due scenari: nel caso la memoria fisica non sia interamente occupata, la nuova pagina viene semplicemente caricata in RAM, mentre se non esiste spazio libero a disposizione, il sistema operativo si occuperà di scambiare la nuova pagina con una già presente in memoria (swap-out). Sia l'accesso su disco che lo scambio di pagine sono operazioni costose in termini di tempo d'esecuzione e, per minimizzarne l'occorrenza, MongoDB tende a caricare e mantenere in memoria l'intero database (o una sua porzione significativa, qualora la dimensione del database sia superiore al quantitativo di RAM disponibile).

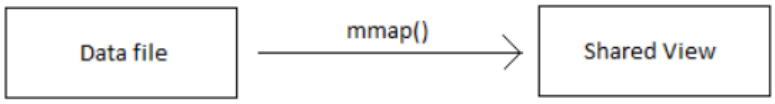

Figura 8: DB Memory Mapping.

All'avvio dell'istanza di MongoDB, i data file vengono mappati in memoria attraverso la system call mmap(). La funzione mmap() è una system call conforme agli standard POSIX, pertanto

viene garantita la piena compatibilità tra i principali sistemi operativi, quali Linux, Mac OS X e BSD, mentre per ambiente Windows sono disponibili funzionalità equivalenti. Una volta mappato il contenuto del file nella sezione di memoria virtuale denominata shared view, è possibile manipolare i dati lasciando al sistema operativo la complessità di salvataggio su disco delle modifiche effettuate al database (a sua volta suddiviso in extent, ovvero porzioni fisiche di dati di dimensione preallocata), operazione che viene eseguita in piena autonomia da quest'ultimo a intervalli regolari (per offrire garanzie di periodicità, MongoDB impone comunque al sistema operativo di aggiornare il data file ogni 60 secondi).

Journaling

MongoDB offre alcune alcune garanzie in merito all'atomicità delle scritture nello stesso documento e la durabilità single-server dei dati. Tale caratteristica viene garantita attraverso determinate politiche di Journaling. Il journal è un sistema di write-ahead logging che permette il recupero dei dati in caso di situazioni impreviste quali system failure. Più precisamente, le informazioni memorizzate nel journal permettono di recuperare operazioni quali modifiche ai documenti (insert/update/remove), modifiche agli indici e modifiche al namespace (file contenenti descrizioni di collection e indici).

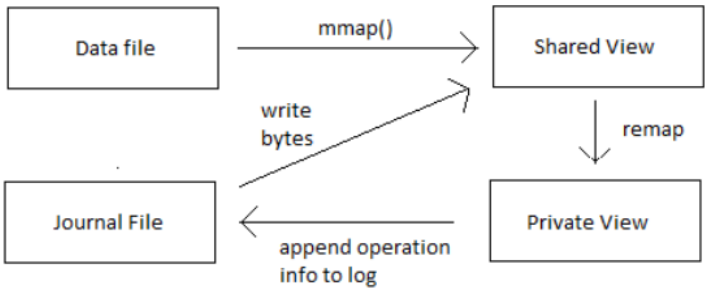

Figura 9: Gestione della Memoria con journaling attivato.

L'attivazione del Journaling introduce due nuovi elementi nella gestione della memoria, ovvero la private view e il file di journal. La private view è un'area di memoria virtuale che viene inizializzata con il contenuto della shared view ma senza supporto al datafile, dunque le modifiche ai dati presenti in quest'area non si rifletteranno su di esso. Il file di journal è invece un file binario di dimensione predefinita e preallocata (1GB di default) che agisce come file di log. Ogni volta che il server MongoDB dovrà effettuare modifiche ai dati, quest'ultime verranno

effettuate all'interno della private view e successivamente scritte nel file di journal entro un intervallo di tempo personalizzabile, il cui valore predefinito è 100ms. La scrittura delle modifiche eseguite non avviene per singola operazione, ma in batch di esse, in modo da rendere minore l'impatto prestazionale del journaling. Il file di Journal conterrà dunque la descrizione dei byte che dovranno essere modificati nel datafile; in caso di system failure, al successivo riavvio le informazioni presenti nel Journal permetteranno di riportare i dati a uno stato consistente. Una volta che le informazioni sono salvate nel Journal, il DBMS si occuperà di replicare queste operazioni nella shared view, per poi rimappare la private view in modo da riflettere il contenuto aggiornato. Periodicamente, verranno infine salvate le informazioni sul datafile fisico, con le stesse meccaniche e tempistiche che si hanno senza la presenza del journaling.

Indici

Un indice è una struttura dati che permette la localizzazione rapida delle informazioni relative al campo su cui è costruito e riveste una parte fondamentale in fase di ottimizzazione delle query. In MongoDB tutti gli indici sono basati su B-tree. Quando un indice "copre" la query, ovvero quando tutti i campi della query fanno parte dell'indice, sia come condizione sia come valore ritornato, MongoDB può eseguire l'interrogazione senza accedere alla collection, massimizzando così le prestazioni. In MongoDB è possibile creare un indice su qualsiasi campo di un documento, indipendentemente dal tipo di dato, rendendo così possibile l'indicizzazione di campi complessi quali array e documenti. Sono supportati indici composti, ossia indici costruiti su campi multipli, con la possibilità di specificare l'ordinamento ascendente o discendente per ogni campo.

Un esempio di creazione di indice composto è mostrato di seguito:

```
"mappings": {
        "properties" : {
                "@timestamp" : {
                        "enabled": false
                },
                "@version" : {
                        "enabled": false
                },
                "document" : {
                        "type": "nested",
```

```
                "properties" : {
                    "alias" : {
                        "enabled": false
                    },
                    "annotated" : {
                        "enabled": false
                    },
                    "annotations" : {
                        "type": "nested",
                        "properties" : {
                            "labels":{
                                "properties" : {
                                    "name" : {
                                        "type": "keyword"
                                    },
                                    "segments" : {
                                        "enabled": false
                                    }
                                }
                            },
                            "entities":{
                                "type": "keyword"
                            },
                            "raw_text" : {
                                "type" : "text",
                                "analyzer": "std_italian"
                            }
                        }
                    },
                    "categories":{
                        "type": "keyword"
                    },
                    "date" : {
                        "enabled": false
                    },
                    "filename" : {
                        "enabled": false
                    },
                    "last_modified" : {
                        "enabled": false
                    },
                    "path" : {
                        "enabled": false
                    }
                }
            },
            "fingerprint" : {
                "enabled": false
            },
            "tags": {
                "enabled": false
            }
        }
}
```

Replicazione

La possibilità di replicare un database su server differenti è un aspetto fondamentale in un ambiente di produzione. La replicazione crea copie multiple dei dati su nodi diversi e la ridondanza così ottenuta permette di avere dei server di backup o disaster recovery sempre aggiornati, in modo da fare fronte a eventi potenzialmente dannosi per l'integrità dei dati come l'interruzione dell'alimentazione. In MongoDB la replicazione permette di aumentare l'accessibilità dei dati e le performance di lettura, infatti per un client è possibile impostare il driver (le librerie che si occupano di interfacciarsi con il database) in modo che sia consentita la lettura dei dati da qualsiasi membro di un replica set; questo aspetto è particolarmente importante quando il nodo principale preferito per la comunicazione abbia un carico di lavoro eccessivo e l'utente possa tollerare la possibilità di ottenere dati non aggiornati.

MongoDB implementa i meccanismi di replicazione dei dati basandosi su un modello master/slave con failover automatico, chiamato replica set.

Un replica set prevede un nodo primario a cui vengono indirizzate tutte le operazioni sui dati, sia di lettura che di scrittura. Effettuata l'interrogazione, il server primario registra l'operazione sul proprio "oplog", una capped collection salvata nel database local che si occupa di memorizzate un set di informazioni minime necessarie per la riproduzione dell'operazione.

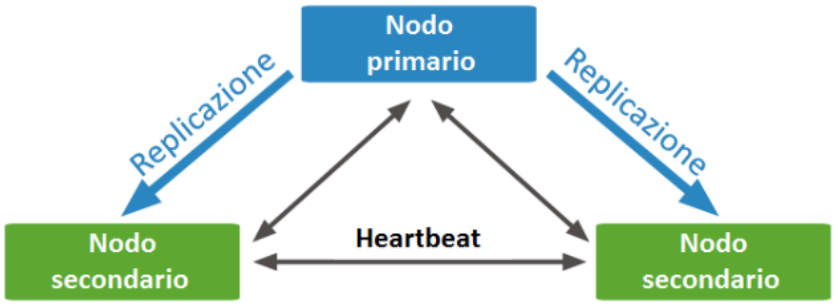

Figura 10: Replica set con 3 nodi.

Periodicamente i server secondari analizzeranno il proprio oplog ottenendo il timestamp dell'ultima operazione; svolta questa fase si occuperanno di interrogare l'oplog del server primario alla ricerca di tutte le operazioni temporalmente successive al timestamp cercato precedentemente. Le operazioni individuate verranno copiate nell'oplog del nodo secondario,

per poi essere applicate sulla propria copia dei dati concludendo il ciclo di replicazione. Il procedimento descritto è asincrono e nonostante il periodo di tempo con cui i secondari interroghino i primari sia molto basso (nell'ordine dei millisecondi) è possibile che i server secondari presentino copie dei dati più vecchie dell'attuale. Al contrario, la scrittura delle informazioni nel journal (qualora sia attivato) e nell'oplog avviene attraverso una transazione atomica.

Una delle caratteristiche più diffuse nei database NoSQL è la capacità di autogestione del sistema in caso di malfunzionamento. MongoDB implementa una logica di automate failover basata su votazione dei server secondari. All'interno di un replica set, tutti i nodi sono in contatto tra loro attraverso lo scambio di piccoli messaggi di stato, denominati heartbeat. Quando il server primario non è più raggiungibile, i nodi secondari riconoscono l'interruzione dei messaggi e si preparano a eleggere un successore. Ogni server ha una priorità, attributo numerico impostato a priori in fase di configurazione. Quando i server secondari dovranno stabilire quale sarà il nuovo nodo primario, i membri con maggiore priorità confronteranno tra di loro il periodo di tempo trascorso dall'ultima sincronizzazione dei dati. Il nodo secondario con la maggior priorità e i dati più recenti verrà eletto a server primario.

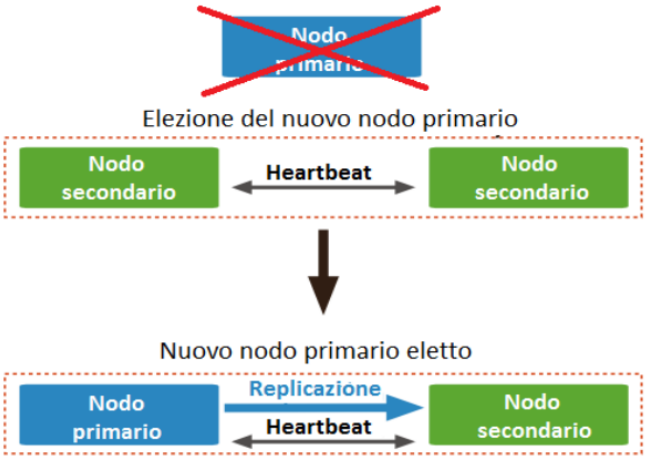

Figure 11: Elezione di un nuovo membro primario in un replica set

All'elezione del nuovo server primario è possibile che si verifichino scenari di spareggio, dove più server siano eleggibili. Per ovviare queste situazioni, è possibile istituire una macchina "arbitro" il cui scopo sia quello di stabilire il successore. Un server arbitro non richiede necessariamente hardware dedicato in quanto non contiene frammenti o copie dei dati,

dunque il consumo di risorse è minimo e può risiedere su macchine già in uso. Quando è in corso una votazione viene imposto un lock sul database e non sono permesse scritture per tutta la durata dell'elezione, operazione che generalmente impiega un tempo inferiore al minuto.

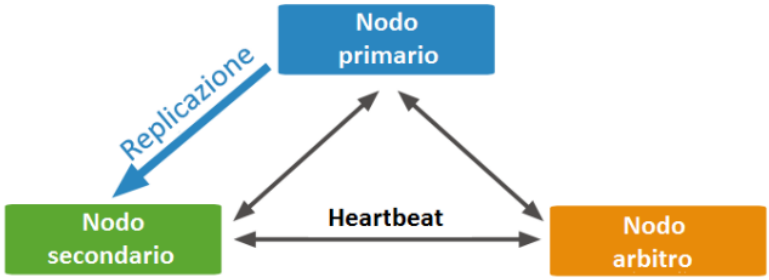

Figure 12: Struttura semplificata con nodo primario, secondario e arbitro.

Come si è visto, in MongoDB i criteri di replicazione sono automatizzati ovunque sia possibile e conveniente, ma questo comportamento non preclude la possibilità di configurare situazioni ad-hoc. Alcune operazioni non sono configurate lato server ma disponibili a livello di driver e, dunque, lato utente. Si è già accennato alla possibilità per l'utente di leggere i dati da un server secondario, piuttosto che da un primario, operazione attivabile da driver. Sempre lato client è anche possibile richiedere un Replica Acknowledgment, dove la corretta scrittura di un dato viene confermata solo quando replicata su un certo numero di nodi.

Logstash

LogStash è un tool open souce flessibile e altamente portabile [3]. È un software per la Log Analisi gratuito (L. Apache 2.0) sviluppato principalmente da Jordan Sissel della DreamHost. LogStash è scritto in JRuby, processa eventi generati da specifici input che vengono filtrati e rilasciati sotto forma di Log, attraverso determinati Output.

Una delle convenienze nel suo utilizzo, è la possibilità di instaurare un sistema centralizzato, ovvero prevede l'invio di messaggi da più macchine verso un elaboratore principale che si fa carico degli eventi. Dispone di una struttura precisa di configurazione che viene lanciata insieme al programma attraverso un file nel formato JSON dove si specificano input, output e filtri applicati all'analisi. Oltre ad avere un buon supporto dalla comunità, LogStash si dimostra

vincente nella sua rapidità di instaurare un sistema efficace fin dal principio, utilizzando un pacchetto Jar precompilato e dei tool integrativi quali: ElasticSearch, Kibana e Redis.

LogStash viene eseguito attraverso una Java Virtual Machine e proposto agli utilizzatori sotto forma di un pacchetto jar pronto per l'utilizzo immediato. La sua particolare progettazione favorisce una semplice configurazione attraverso un file .conf ed estendibilità in ambienti scalabili, poiché è in grado di processare eventi di svariato genere: Syslog, Microsoft EventLogs, STDIN etc. Tali eventi possono essere manipolati da particolari filtri all'interno di LogStash. Una volta modificati assumono la forma di veri e propri log, caratterizzati da campi che ne indentificano la formattazione. Terminato il processo di manipolazione i log vengono espulsi sugli output, anche questi di diverso genere: TCP/UDP, email, files, HTTP, Nagios, ecc... LogStash promuove la sua progettazione attraverso livelli di Information Hiding, i livelli architetturali sono separati e lavorano attraverso interfacce interconnesse.

Il settore più alto è quello dei plugin (input, filter e output), in questo modo uno sviluppatore alle prime armi può facilmente estendere le funzionalità di LogStash attraverso l'utilizzo di metodi predefiniti e semplici comandi Ruby. Il file di configurazione .conf, detto agente o evento di pipeline, contiene la definizione delle fasi di input, filter e output che si vogliono utilizzare all'avvio di LogStash. Gli eventi che incombono vengono passati ad ogni fase attraverso delle code interne, in particolare delle 00SizedQueue00 in Ruby. Queste code fra fasi, gestite da Thread, così come evince in Fig.13 hanno una capienza limitata e bloccano la scrittura di ulteriori eventi raggiunta la capacità limite.

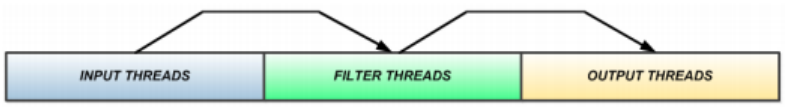

Figura 13: Comunicazione tra i thread di LogStash.

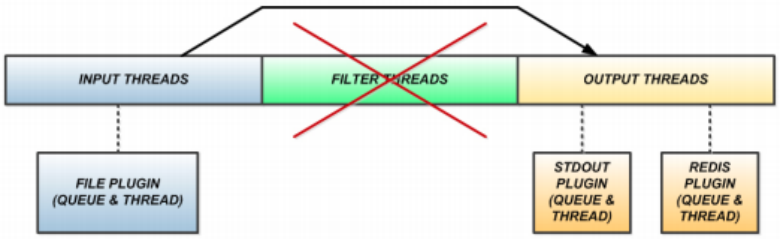

Figura 14: Comunicazione fra i thread sulla configurazione dei shipper.

Kibana

Kibana, sviluppato da Rashid Khan, è un tool open source per la visualizzazione di eventi manipolati dal LogStash e indicizzati da ElasticSearch [4]. Scritto in Ruby e Java Script, utilizza il framework Sinatra per svolgere le proprie funzionalità di classificazione grafica.

Kibana ricerca, attraverso delle query, le informazioni all'interno del cluster di ElasticSearch, mettendosi in ascolto della sua porta 9200. I dati raccolti vengono visualizzati attraverso una pagina web sul proprio browser alla porta 5600 (http://127.0.0.1:5600). In tale pagina si possono svolgere delle query manualmente o effettuare ricerche in base al tempo o ai campi contemplati; inoltre si dispone di una sezione apposita per la visualizzazione in streaming di eventi raccolti da ElasticSearch. In LogStash, inizialmente, si utilizzava un output plugin per la visualizzazione grafica degli eventi, tuttavia la continua fama di Kibana riscontrata come applicativo di supporto alla analisi dei log ha indotto gli sviluppatori di LogStash ad operare sulla possibilità d'integrare Kibana all'interno della libreria di LogStash [5]. Kibana, per il progetto, ha rappresentato l'interfaccia in tempo reale per la visualizzazione dei Log, in questo modo i gestori aziendali possono essere al corrente di eventuali malfunzionamenti presenti o passati in cui è in corso DriveFarm.

ElasticSearch

ElasticSearch è un database server, scritto in Java, che si occupa di memorizzare grandi quantità di dati in un formato sofisticato ottimizzato per ricerche basate sul linguaggio. Si tratta di un motore di ricerca open source altamente scalabile, che consente di memorizzare e analizzare una grande quantità di informazioni praticamente in tempo reale, sfruttando l'indicizzazione dei documenti [6].

Esso lavora con documenti in formato JSON. Utilizzando una struttura interna, può analizzare i dati quasi in tempo reale per cercare le informazioni di cui si ha bisogno. Risulta essere quindi una tecnologia molto utile quando si lavora con i Big Data.

Elasticsearch nasce come una evoluzione di Compass, applicazione realizzata dal fondatore del progetto Shay Banon nel 2004, basata sulla libreria Apache Lucene.

Questo potente strumento è ottimizzato per ricerche di tipo full-text, e contiene la maggior parte degli algoritmi di ricerca e indicizzazione. Negli anni successivi Elasticsearch viene arricchito con funzionalità sempre più potenti, fino ad arrivare alla versione attuale 7.3 [6].

Di fatto, Elasticsearch è la prima tecnologia a proporre un sistema che permetta di eseguire ricerche full-text in modo parallelo, scalabile e real-time. Per questa caratteristica e per il suo primato nel settore, Elasticsearch è il secondo motore di ricerca più popolare nel mondo. Tra i suoi utilizzatori più famosi ricordiamo Mozilla, GitHub, CERN, StackExchange e Netflix.

Si descrivono ora alcune caratteristiche fondamentali che costituiscono il nucleo di Elasticsearch [7]:

- **Protocollo HTTP/JSON:** Elasticsearch comunica utilizzando un formato JSON particolare abbinato alle REST API. Sfrutta quindi il protocollo HTTP per il trasferimento di rete tramite chiamate REST, e il JSON, utilizzato per costruire il formato di comunicazione e memorizzare informazioni in maniera organizzata e facilmente accessibile, garantendo che la collezione di dati sia semplice da leggere e da accedere;
- **Scalabilità e affidabilità:** Elasticsearch nasce come sistema di ricerca parallelo, perciò la capacità del database può essere aumentata semplicemente aggiungendo nodi al cluster. Inoltre, Elasticsearch gestisce autonomamente il problema dell'affidabilità, migrando dati o promuovendo nodi replica a nodi primari al momento del malfunzionamento di un qualsiasi nodo;
- **Analisi del linguaggio:** Cuore delle funzionalità di Elasticsearch è senza dubbio il sistema automatizzato di analisi del linguaggio. Il database è in grado di analizzare porzioni di testo in qualsiasi lingua e formato, indicizzando in modo efficiente termini chiave e persino sinonimi, garantendo un sistema di ricerca dalle elevate potenzialità;
- **Query DSL:** Elasticsearch utilizza un particolare linguaggio basato su JSON per codificare la struttura di una ricerca. A differenza del linguaggio SQL, la query DSL è componibile a piacere in modo da garantire la massima libertà di ricerca.

Tuttavia, se da una parte questa libertà permette di effettuare ricerche a qualsiasi livello, dall'altra spesso il modo di interrogare il database risulta piuttosto complesso.

Elasticsearch architecture

Un indice è un meccanismo di organizzazione dei dati, che consente un loro partizionamento a piacere [8]. In ElasticSearch, si può pensare ad un indice come al suo equivalente di una base dati relazionale, ovvero un complesso contenitore di dati, come mostrato nella tabella seguente:

Elasticsearch	Indice	Tipo (Type)	Documento
Oracle	Database	Tabella	Tupla

Tabella 2: Elasticsearch vs. Oracle.

Ogni oggetto indicizzato in Elasticsearch appartiene ad una classe che ne definisce le proprietà e i dati ad esso associati. In un database relazionale solitamente si memorizzano oggetti della stessa classe nella stessa tabella, perché condividono la stessa struttura. Di fatto, in Elasticsearch il Type rappresenta la struttura di una classe di oggetti, dunque il Type corrisponde ad una tabella in accezione relazionale. In parallelo con una base dati relazionale, così come un database contiene delle tabelle, un indice Elasticsearch può contenere più Type. Le entità o oggetti nella maggior parte delle applicazioni possono essere serializzati in un oggetto JSON tramite delle coppie chiave/valore.

Spesso si utilizzano i termini oggetto e documento in modo intercambiabile. Esiste però una distinzione: un oggetto è semplicemente un oggetto JSON che può contenere altri oggetti. In Elasticsearch, il termine documento ha un significato specifico, ovvero si riferisce all'oggetto radice serializzato in JSON e memorizzato in Elasticsearch con un ID univoco. Come si può notare nella tabella precedente, facendo un parallelo con le basi di dati relazionali, un documento Elasticsearch corrisponde alla riga di una tabella in accezione relazionale.

In maniera del tutto generale, conosciamo gli indici come semplici contenitori di dati che in qualche modo sono legati tra di loro. In realtà un indice è soltanto un namespace logico che punta ad una o più shard. Una shard è una unità di lavoro di basso livello che contiene una porzione di dati dell'indice. Ogni shard è una singola istanza Lucene, dunque è un sistema di

ricerca autonomo. Le shard costituiscono lo strumento con cui Elasticsearch distribuisce i dati all'interno del cluster, ed è per questo che un indice può essere distribuito su più nodi.

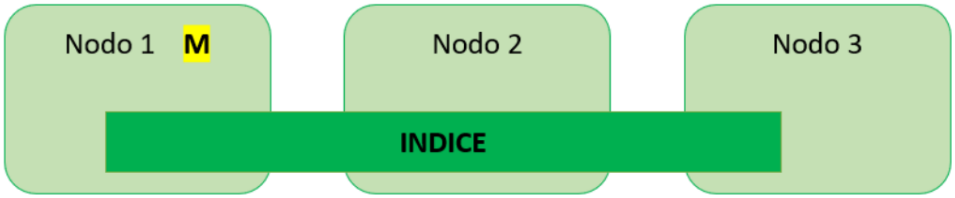

Figura 15: Struttura a nodi distribuiti.

È possibile distinguere due tipi di shard: **primarie** e **replica**; le replica sono delle semplici copie di shard primarie che permettono al cluster di continuare a funzionare anche in caso di fault o sostituzione di un nodo. Se un nodo qualsiasi smette di funzionare, le shard replica vengono promosse a primarie, e all'inserimento di un nuovo nodo verranno rigenerate le replica come nella situazione antecedente il fault. Attraverso un particolare algoritmo Elasticsearch è in grado di garantire grande affidabilità distribuendo i dati in modo da garantire la maggior fault tolerance possibile. In *Figura 15* viene mostrata la promozione di shard replica a primarie.

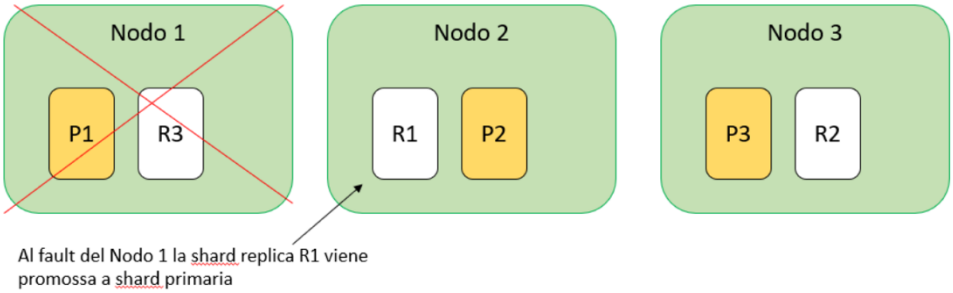

Figura 16: Promozione di shard replica a primarie.

Come effetto di questo meccanismo, all'interno di un cluster è possibile aggiungere e rimuovere nodi a piacimento senza alterarne in alcun modo il funzionamento. Questo rende di fatto Elasticsearch uno strumento che permette al sistema di scalare orizzontalmente in maniera quasi automatica abbattendo sia i costi di manutenzione sia i costi che sarebbero necessari per mantenere personale competente. Elasticsearch è inoltre asincrono e concorrente, per cui le modifiche vengono eseguite in parallelo, pertanto potrebbero arrivare a destinazione fuori sequenza. Per risolvere questo problema Elasticsearch utilizza un sistema

di versioning dei documenti, evitando che vecchie versioni di un documento sovrascrivano quelle nuove arrivate fuori sequenza. La diversità nel modo di operare delle due tecnologie trova le sue ragioni nel diverso scopo per cui sono state progettate: una base di dati relazionale nasce con l'obiettivo di fornire non solo un metodo sicuro per gestire la consistenza dei dati, ma anche di garantire che le operazioni generate da più sessioni in concorrenza equivalgano all'esecuzione seriale delle stesse. Elasticsearch, al contrario, nasce come sistema di ricerca ed analisi su una grande quantità di dati; è perciò importante mantenere elevate prestazioni a fronte di analisi complesse ed in tempo reale. Spesso non è necessario mantenere una stretta consistenza dei dati, poiché sarà molto raro incorrere in una modifica concorrente sulla stessa informazione.

Jena

È un Framework open source per il Semantic Web, sviluppato in Java. Jena mantiene in memoria tutto il modello RDF, fornendo un insieme di API per manipolarlo secondo il paradigma a grafo. Nelle ultime versioni sono stati integrati anche dei reasoner RDF e per OWL, permettendo così di manipolare la gerarchia delle classi, di creare e rimuovere istanza e concetti e di inferire nuove triple in base ai dai presenti nel modello. È inoltre capace di garantire la persistenza dei dati sia interpretando le varie sintassi XML/RDF sia interfacciandosi con diversi database relazionali. Jena supporta SPARQL grazie all'aggiunta di una libreria ARQ, ed inoltre esiste anche un'estensione chiamata Joseki che implementa un server capace di esporre un endpoint SPARQL con cui è possibile pubblicare e interrogare i dati all'interno del modello gestito da Jena. Le classi di interesse sono:

- Query: classe che rappresenta la query dell'applicazione;
- QueryExecution: classe che rappresenta l'esecuzione di una query;
- QueryExecutionFactory: classe usata per ottenere una istanza di QueryExecution a partire da un oggetto Query e un riferimento al servizio SPARQL.

Allocazione delle entità agli stack tecnologici

Le entità, di cui alla precedente sezione, sono così istanziati sullo stack tecnologico del Multimedia Manager:

- Loading Module: API REST, Multimedia Manager Storage, MongoDB per il caricamento dei file Multimediali, Jena Fuseki per il caricamento della componente ontologica;
- Processing Module: MongoDB, GCloud Video Intelligence, Jena Fuseki;
- Composer Module: MongoDB;
- Mapping Module: Logstash, Elasticsearch;
- Searching Module: API REST, Elasticsearch.

Pipeline di ricerca delle informazioni attraverso query intelligenti

La pipeline di ricerca viene avviata tramite una chiamata REST secondo l'API specifica. Essa provvede alla collezione della query espressa dall'utente in forma di stringa e la trasforma in una query DSL di elasticsearch per la ricerca dei contenuti multimediali relativi alle informazioni raccolte nella stringa proposta dall'utente. Viene di conseguenza avviato il Searching Module, che provvederà a restituire il file multimediale individuato, le menzioni del testo ricercato dall'utente all'interno delle entità globali e della trascrizione testuale della componente audio, più il minutaggio specifico del file multimediale in cui tali entità sono state identificate. Di seguito un esempio di query DSL eseguita su Elasticsearch:

```
{ "_source": ["document.alias","document.filename"],
        "query": {
                "nested": {
                        "path": "document",
                        "inner_hits": {
                                "_source":false,
                                "highlight": {
                                        "fields": {
                                                "document.annotations.raw_text": {},
                                                "document.annotations.entities": {},
                                                "document.annotations.labels.name": {},
                                                "document.categories": {}
                                        },
                                        "number_of_fragments": 0,
                                        "pre_tags": ["<span>"],
                                        "post_tags": ["</span>"]
                                }
                        },
                        "query": {
                                "nested": {
                                        "path": "document.annotations",
                                        "query": {
                                                "bool" : {
                                                        "should": [{"multi_match": {
                                                                "query": "cambio shimano",
                                                                "type": "most_fields",
                                                                "fields":
                                                        ["document.annotations.raw_text^3"]
                                                                }
                                                        },{"multi_match": {
                                                                "query": "bicycle",
                                                                "type": "most_fields",
                                                                "fields": ["document.categories",
                                                        "document.annotations.entities",
                                                        "document.annotations.labels.name"]
                                                                }
```

```
                                                    }]
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
```

API RESTful

Le API previste per il Multimedia Manager prevedono di risolvere le seguenti funzioni:

- Caricamento del modello RDF per la componente ontologica;
- Aggiornamento dell'Ontologia/Grafo di Conoscenza;
- Cancellazione dell'Ontologia/Grafo di Conoscenza;
- Caricamento file multimediale;
- Controllo processing file multimediale;
- Query intelligente;

Tali funzioni faranno riferimento ad un unico URL di dominio espletante il servizio, ossia:

https://ilab.exprivia.com/mmanager/api

Ognuna di esse si diversificherà nelle chiamate definite successivamente.

Load RDF Graph

Il metodo da adoperare e il metodo POST, utilizzando i campi esemplificati in tabella.

Parameter	Value	Type	Description
op	loadRdf	String	
name		String	RDF dataset name
rdf		Data	RDF format file

Il risultato sarà una notifica di caricamento del dataset RDF avvenuto con successo o l'eventuale codice di errore di mancato caricamento.

Update RDF Graph

Il metodo è ancora il metodo POST:

Parameter	Value	Type	Description
op	updateRdf	String	
name		String	Stored RDF dataset name
rdf		Data	RDF format file

Il risultato sarà una notifica di aggiornamento del dataset RDF avvenuto con successo o l'eventuale codice di errore di mancato aggiornamento.

Delete RDF Graph

Il metodo è, come intuibile, il metodo DELETE:

Parameter	Value	Type	Description
op	deleteRdf	String	
name		String	Stored RDF dataset name

Il risultato sarà una notifica di cancellazione del dataset RDF avvenuta con successo o l'eventuale codice di errore di mancata cancellazione.

Load Multimedia File

Il metodo coincide con il metodo POST:

Parameter	Value	Type	Description
op	loadMfile	String	
name		String	Multimedia filename
file		Data	Multimedia file

Il risultato sarà una notifica di avvenuto caricamento del file multimediale o l'eventuale codice di errore di mancato caricamento.

Monitoring Processing

Il metodo corrisponde con una GET:

Parameter	Value	Type	Description
op	infoProcessing	String	
name		String	Multimedia filename

Il risultato sarà un JSON contenente tutte le informazioni collezionabili sullo status del processing del file multimediale richiesto.

Query

Il metodo coincide con il metodo GET:

Parameter	Value	Type	Description
op	smartQuery	String	
query		String	String provided by user for searching purpose

Il risultato sarà un JSON popolato dalla stringa sottomessa in input al sistema da parte dell'utente, il video per cui sono state riconosciute le entità semanticamente simili alla richiesta effettuata dall'utente, la porzione di testo trascritto del file multimediale in cui sono state individuate le entità precedenti, opportunamente evidenziate tramite markup, il minutaggio del file multimediale in cui tali entità sono state individuate.

BIBLIOGRAFIA

[1] A.M. Ferman et al., "Content-Based Filtering and Personalization Using Structured Metadata," to appear in Joint Conf. Digital Libraries, 2002.

[2] The Little Mongo DB Schema Design Book Christian Kvalheim - 2015-10-19, Christian Kvalheim.

[3] The logstash Book. Log management made easy – 2016, James Turnbull.

[4] Learning Kibana 7. Build powerful Elastic dashboard with Kibana's data visualization capabilities – 2015, Anurag Srivastava and Bahaaldine Azarmi.

[5] Kibana Essentials. Use the functionalities of Kibana to reveal insights from the data and build attractive visualizations and dashboards for real-world scenarios – 2015, Yuvraj Gupta.

[6] ElasticSearch, LogStash, Kibana (the ELK Stack) - 2018, Manuj Agarwal.

[7] ElasticSearch: A complete guide. End-to-end Search and Analytics – 2017, Birmingham – Mumbai.

[8] ElasticSearch The Definitive Guide. A Distributed Real-Time Search and Analytics Engine – 2015, Clinton Gormley and Zachary Tong.