UNIVERSITÀ DEGLI STUDI DI BERGAMO

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Corso di Laurea Magistrale in Ingegneria Informatica

Classe n. LM-32 - Classe delle lauree magistrali in Ingegneria Informatica

Anonimizzazione distribuita di dati: porting del framework Mondrian in ambiente Kubernetes

Relatore

Chiar.mo Prof. Stefano Paraboschi

Tesi di Laurea Magistrale

Gianluca MAFFIOLETTI

Matricola n. 1059062

ANNO ACCADEMICO 2022/2023

Sommario

L'anonimizzazione dei dati è un processo fondamentale per tutelare la privacy degli utenti e garantire la conformità alle normative sulla protezione dei dati. Il crescente bisogno di gestire grandi volumi di dati rende necessaria l'esplorazione di nuove soluzioni adatte ad affrontare il problema in maniera efficiente.

Questo studio si concentra sul porting del framework di anonimizzazione distribuita Mondrian, realizzato dal laboratorio di sicurezza informatica dell'Università degli Studi di Bergamo nel contesto del progetto europeo MOSAICrOWN, da un'architettura locale incentrata sull'utilizzo di Docker Compose a un ambiente distribuito basato su Kubernetes.

L'utilizzo di Kubernetes come orchestratore di container consente di ottenere un'applicazione altamente scalabile, in grado di suddividere dinamicamente carichi di lavoro su più nodi appartenenti ad un cluster, garantendo una maggiore flessibilità e affidabilità del sistema tramite il continuo monitoraggio dei pod e la rischedulazione in seguito ad errori o malfunzionamenti.

L'obiettivo principale è quello di valutare le prestazioni del sistema durante il processo di migrazione, confrontando le due architetture in termini di gestione delle risorse e tempi di esecuzione, al fine di valutare benefici ed eventuali criticità del porting in ambiente distribuito.

I risultati della fase sperimentale, ottenuti variando una serie di parametri di test quali il numero di workers su cui suddividere il lavoro e i parametri di anonimizzazione, evidenziano che la trasposizione del framework su Kubernetes permette di integrare funzionalità di orchestrazione avanzate, determinando un impatto limitato sulle performance del sistema.

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Introduzione

Nell'era digitale moderna, il tema della sicurezza dei dati è diventato di interesse primario per individui, aziende ed istituzioni. Ciò è legato principalmente al rapido e continuo aumento della quantità di informazioni digitali che circolano quotidianamente attraverso dispositivi, reti e sistemi. Uno studio del giugno 2021 condotto da International Data Corporation e Statista [1] mostra, infatti, come nella decade 2010-2020 il volume di dati globale sia incrementato da 2 a 64,2 zettabytes, e che le proiezioni prevedono un valore triplicato al 2025 (Figura 1.1).

In mezzo a questa mole di dati, assumono una particolare rilevanza i dati personali, definiti anche dati sensibili, ossia quelle informazioni che hanno un carattere altamente riservato e che permettono di caratterizzare un individuo. Alcuni esempi comuni sono le credenziali di accesso ad una piattaforma, dati sanitari, informazioni finanziarie, numeri di carte di credito. La protezione di questi dati sensibili è fondamentale per tutelare la privacy e la sicurezza degli individui, oltre che per conformarsi alle normative sulla protezione dei dati.

Diverse tecniche e metodologie possono essere applicate a questo scopo. Tra di esse, l'anonimizzazione è una procedura che permette di trasformare i dati personali, attraverso una serie di operazioni, in maniera irreversibile, al fine di renderli del tutto anonimi. Questo processo permette di garantire la privacy del soggetto a cui appartengono i dati, in quanto la sua identità non può essere ricavata direttamente dalle informazioni contenute nel dataset anonimizzato.

Negli ultimi anni, l'Unione Europea ha partecipato attivamente alla ricerca nel-

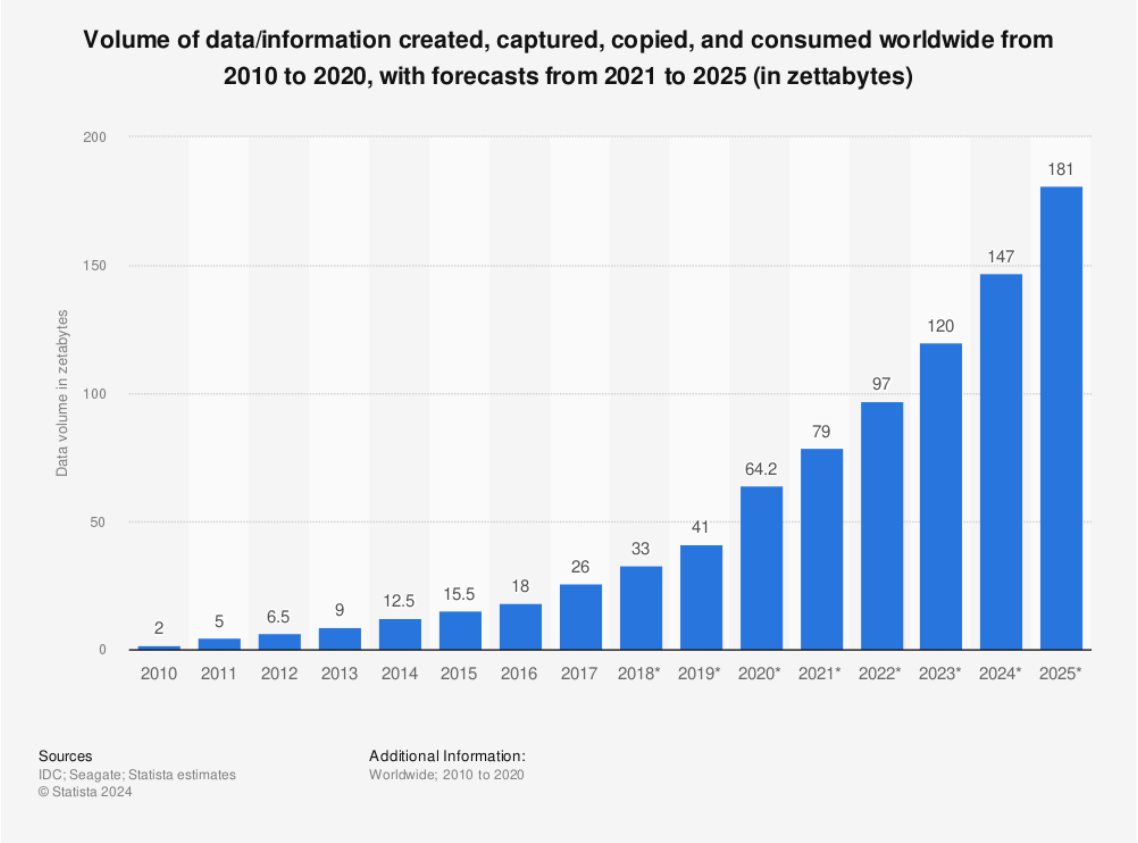


Figura 1.1: Volume globale di dati 2010-2025

l'ambito della gestione e protezione dei dati promuovendo alcuni progetti europei, dei quali l'Università degli Studi di Bergamo è stata ed è tuttora membro. In particolare, nel corso del progetto MOSAICrOWN avviato nel 2019 e concluso nel 2021, il team dell'UniBG Security Lab (Seclab) ha realizzato un framework di anonimizzazione distribuita di dati, denominato Mondrian. Questo sistema ha lo scopo di applicare la procedura di anonimizzazione su grandi dataset in maniera distribuita, ossia suddividendo il carico di lavoro e assegnandolo a diversi esecutori, che svolgono le operazioni indipendentemente, superando così i limiti di un processo centralizzato. Per la realizzazione, sono state utilizzate tecnologie quali Apache Spark, Docker Compose e HDFS, che operano in locale su un dispositivo.

L'obiettivo di questo progetto di tesi è quello di esplorare possibili evoluzioni del framework Mondrian. L'idea principale è di effettuare una migrazione da un'architettura locale ad un ambiente distribuito, sfruttando Kubernetes come orchestratore di container, la cui peculiarità è quella di poter gestire dei cluster di nodi dislocati

in rete, e sfruttarli per eseguire separatamente dei carichi di lavoro. Ciò consentirebbe teoricamente di migliorare l'efficienza del sistema, avendo a disposizione una maggiore quantità di risorse.

Il contenuto di questo studio è strutturato come segue. Nel Capitolo 2 verrà effettuato un ampio approfondimento sul tema della sicurezza dei dati e specialmente sull'anonimizzazione, trattando la normativa di riferimento e le tecniche applicabili. Nel Capitolo 3 verrà presentato in dettaglio il framework Mondrian realizzato dall'UniBG Seclab, con particolare riferimento alle tecnologie utilizzate, alle varie fasi di esecuzione e ai risultati sperimentali. Nel Capitolo 4 si discuterà dell'analisi e progettazione del porting, esponendo le limitazioni del framework attuale, la scelta delle tecnologie e la pianificazione dell'architettura distribuita. Il Capitolo 5 si concentrerà sull'implementazione del porting e sui dettagli tecnici di ciascuna fase operativa del nuovo sistema. Nel Capitolo 6 verranno valutate le prestazioni del framework realizzato, effettuando un confronto con l'architettura precedente. Infine, nel Capitolo 7 verranno esposte le conclusioni finali riguardanti il progetto di tesi.

Capitolo 2

Fondamenti dell'anonimizzazione di dati

La sicurezza informatica nasce con l'obiettivo di tutelare le informazioni e le tecnologie dall'azione di soggetti terzi malintenzionati o da rischi di qualsiasi altra natura, come difetti e malfunzionamenti. L'oggetto della protezione è sia la componente hardware, come computer, server o smartphone, sia la componente software.

Questo progetto di tesi è incentrato sul concetto di sicurezza delle informazioni, il quale si riferisce alla protezione dei dati, delle proprietà intellettuali e delle infrastrutture, dall'azione di attacchi informatici condotti da degli avversari, ossia soggetti intelligenti che intenzionalmente mirano a danneggiare o violare l'integrità di un sistema informatico. A differenza di guasti e malfunzionamenti, che possono essere considerati eventi randomici e involontari, il nemico è sconosciuto e può approfittare della minima vulnerabilità del sistema per causare gravi danni. Per rispondere quindi all'imprevedibilità di questi attacchi, occorre adottare un approccio responsivo, che unisce l'applicazione di misure tradizionali al continuo monitoraggio dell'evoluzione delle minacce.

2.1 Sicurezza delle informazioni

La sicurezza delle informazioni è, come già detto, un processo continuo e in crescita che comporta l'adozione di misure adeguate ad affrontare le nuove insidie che si

presentano nel mondo digitale.

Il National Institute of Standards and Technology (NIST) degli Stati Uniti definisce la sicurezza delle informazioni come «La protezione delle informazioni e dei sistemi informativi da accessi, usi, divulgazioni, interruzioni, modifiche o distruzioni non autorizzate, al fine di garantire riservatezza, integrità e disponibilità» [2].

Questa definizione mette in evidenza i tre principi cardine dell'information security, anche conosciuti in letteratura come triade CIA:

- **Confidentiality** (Riservatezza): le informazioni devono essere accessibili solo agli utenti autorizzati da chi detiene la proprietà dei dati. In questo ambito ricade il tema della privacy, ossia la protezione dei dati personali di un soggetto e il diritto di quest'ultimo di decidere chi e come può accedere ai propri dati. Alcuni strumenti per garantire la confidenzialità sono la crittografia dei dati o l'autenticazione a più fattori;

- **Integrity** (Integrità): le informazioni devono essere protette da modifiche non autorizzate che ne possano compromettere la validità, l'accuratezza o la completezza. La gestione delle autorizzazioni, delle identità e il controllo degli accessi sono alcuni dei meccanismi utilizzati per garantire l'integrità dei dati;

- **Availability** (Disponibilità): le informazioni devono essere disponibili agli utenti autorizzati quando ne hanno bisogno, secondo i requisiti di servizio stabiliti. La continuità dei servizi è cruciale in molti ambiti e deve essere tutelata da attacchi DoS (Denial of Service) o da guasti imprevisti tramite adeguate procedure di disaster recovery e strumenti di backup.

I sistemi di sicurezza moderni devono tener conto di questi tre concetti primari e realizzare una politica di protezione e gestione dei rischi che sia una combinazione di misure in ciascuno di questi ambiti. Risulta cruciale, quindi, avere consapevolezza delle minacce che possono pervenire nei sistemi informatici e degli strumenti e tecniche volte a contrastare, eliminare o quantomeno minimizzare, i rischi e i danni che ne possono derivare.

2.1.1 Minacce e strumenti di sicurezza

Negli ultimi anni, la trasformazione digitale accompagnata da altri fattori, tra cui la pandemia di coronavirus, che ha attribuito un ruolo sempre più significativo agli strumenti informatici per la comunicazione e il lavoro da remoto, hanno avuto come effetto inevitabile lo scatenamento di nuove minacce in termini di cybersicurezza e fornito ai criminali informatici maggiori opportunità d'attacco. Secondo la relazione sul panorama delle minacce nel 2022 [3] elaborata dall'Agenzia dell'Unione Europea per la sicurezza informatica (ENISA), sono otto le principali tipologie di minacce: malware, ramsomware, phishing, violazioni e fughe di dati, negazione di servizio, minacce alla disponibilità di Internet, disinformazione e malinformazione, minacce ala catena di fornitura.

Tutti questi esempi di minacce evidenziano quanto attualmente le informazioni abbiano un valore economico rilevante tanto per i proprietari dei dati quanto per i cybercriminali, e mostra quanto sia altrettanto fondamentale adottare misure appropriate di protezione delle informazioni.

Le misure di sicurezza informatica maggiormente diffuse consistono nell'utilizzo di antivirus, firewall, strumenti di backup e sistemi di log e monitoraggio. Queste misure offrono una protezione preventiva e possono essere considerate come una base della sicurezza informatica, a cui integrare misure più specifiche ed evolute, come l'anonimizzazione dei dati che verrà trattata nella Sezione 2.3.

Nonostante la disponibilità di una serie di strumenti e tecniche per prevenire gli attacchi e limitare i danni, occorre sottolineare che la prima e più importante misura di protezione è la conoscenza in tema di cybersicurezza e l'uso consapevole dei sistemi informatici da parte degli utenti. Uso di password deboli, negligenza nella gestione dei dati sensibili e utilizzo di software obsoleti o non autorizzati sono solo alcune delle vulnerabilità che possono essere sfruttare dai cybercriminali. Un sondaggio effettuato nel 2017 da Kaspersky Lab e B2B International su un campione di oltre 5000 aziende in tutto il mondo [4] evidenzia, in particolare, che il 52% delle aziende ritiene che il personale sia la principale debolezza in ambito sicurezza informatica, e che il 39% degli attacchi informatici subiti dalle aziende su un periodo di dodici mesi sia effettivamente causato dall'uso improprio delle risorse IT

da parte dei dipendenti. Per evitare errori umani, le aziende devono assumersi la responsabilità di educare gli impiegati in materia di sicurezza informatica e prendere delle precauzioni, come rafforzare il controllo degli accessi e la gestione dei privilegi degli utenti. Il fattore umano è, quindi, un elemento determinante per perseguire la sicurezza delle informazioni.

2.2 Dati sensibili

I dati sensibili costituiscono un elemento cruciale dell'ambiente informativo contemporaneo. Essi sono caratterizzati da una natura altamente riservata e comprendono informazioni personali, finanziarie e mediche che, se compromesse, possono ledere gravemente la privacy e la sicurezza degli individui. La gestione adeguata dei dati sensibili richiede, dunque, non solo robuste pratiche di sicurezza informatica, ma anche una profonda comprensione delle normative sulla privacy e dei principi etici. La protezione di tali dati è divenuta una priorità incontestabile in tutte le organizzazioni, richiedendo politiche rigorose per garantirne la riservatezza e l'integrità.

2.2.1 Normativa

In materia di dati sensibili, la normativa attuale consiste nel Regolamento UE 2016/679 sulla protezione dei dati personali e privacy, meglio noto come GDPR (General Data Protection Regulation) [5], entrato in vigore il 24 maggio 2016 e applicato dal 25 maggio 2018.

L'articolo 4 del GDPR specifica che per "dato personale" si intende «qualsiasi informazione riguardante una persona fisica identificata o identificabile», e precisa che «si considera identificabile la persona fisica che può essere identificata, direttamente o indirettamente». In questo articolo viene evidenziato il concetto di identificabilità, ossia la capacità di poter ricavare l'identità di un individuo. Con il termine identità si fa riferimento ad attributi come il nome, il codice fiscale, dati relativi all'ubicazione geografica, o elementi che caratterizzano l'identità fisica, economica, culturale o sociale.

L'articolo 9 del GDPR, inoltre, stabilisce una categoria di dati, definiti "particolari", che sono considerati sensibili e soggetti a specifiche condizioni di trattamento, della quale fanno parte:

- Dati personali che rivelino l'origine razziale o etnica, le opinioni politiche, le convinzioni religiose o filosofiche;
- L'appartenenza ad un sindacato;
- Dati genetici, dati biometrici trattati solo per identificare un essere umano;
- Dati relativi alla salute;
- Dati relativi alla vita sessuale o all'orientamento sessuale di una persona.

Sempre l'articolo 9 del GDPR, sancisce che il trattamento dei dati particolari è permesso solo se il soggetto ha prestato il proprio consenso esplicito oppure se ha reso manifestamente pubblici i propri dati sensibili. Ma sono presenti alcune eccezioni per cui non è necessario il consenso dell'interessato. In queste fattispecie, il trattamento dei dati può essere necessario:

- Per il calcolo della retribuzione e a fini pensionistici;
- Per tutelare un interesse vitale dell'interessato o di un'altra persona fisica;
- Per accertare, esercitare o difendere un diritto in sede giudiziaria, se l'interessato è coinvolto in un procedimento giudiziario;
- Per motivi di interesse pubblico rilevante;
- Per finalità di medicina preventiva o di medicina del lavoro;
- Per fini di archiviazione nel pubblico interesse, di ricerca scientifica o storica o a fini statistici.

Queste sono le principali indicazioni della normativa sulla protezione dei dati personali. In linea generale, quando si gestiscono dei dati è necessario comprendere la tipologia di dato che si sta trattando e chiedere il consenso esplicito dell'interessato, se non previsto diversamente dal GDPR.

2.2.2 Trattamento dei dati personali

Dopo aver chiarito quali dati siano considerati personali e quando questi possano essere trattati, bisogna capire quale sia la modalità di trattamento conforme alla regolamentazione.

L'articolo 32 del GDPR enuncia che il responsabile del trattamento è tenuto ad impiegare misure tecniche e organizzative adatte a garantire un livello di sicurezza adeguato al rischio. Nello specifico, indica la pseudonimizzazione e la cifratura come possibili procedure da utilizzare.

La pseudonimizzazione è una tecnica che prevede che il trattamento dei dati sia svolto in maniera da non permettere che questi possano essere attribuiti a un interessato specifico senza l'utilizzo di informazioni aggiuntive. La sua applicazione richiede, innanzitutto, l'individuazione degli attributi che possono identificare direttamente o indirettamente un soggetto e, successivamente, il ricorso a tecniche di sostituzione o modifica dei dati.

La cifratura, invece, è una tecnica che permette di rendere i dati illeggibili per chi non dispone della chiave di decifratura. Essa si basa fondamentalmente sulla complessità dell'algoritmo di cifratura e sulla segretezza delle chiavi utilizzate.

Oltre a questo, l'articolo 32 stabilisce che il titolare del trattamento debba anche assicurare la riservatezza, l'integrità, la disponibilità, la resilienza dei sistemi di trattamento e la capacità di ripristinare tempestivamente la disponibilità e l'accesso ai dati personali in caso di incidente fisico o tecnico. Infine, suggerisce di pianificare delle procedure per testare, valutare e verificare l'efficacia delle misure attuate al fine di garantire la sicurezza del trattamento, tenendo conto in special modo dei rischi che possono portare a distruzione, perdita, modifica, divulgazione o accesso non autorizzato.

2.2.3 Rilascio di dati sensibili e attacchi alla privacy

Le organizzazioni raccolgono ogni giorno dati sensibili dai consumatori, che possono spaziare dal nome e cognome di una persona, all'indirizzo di residenza, alle condizioni di salute e finanziarie. Queste informazioni vengono memorizzate in grandi set di

dati aggregati, strutturati tipicamente in tabelle. Gli attributi di queste tabelle possono essere distinti in:

- **Identificatori**: attributi che identificano univocamente un soggetto (ad esempio il codice fiscale);
- **Quasi-identificatori**: attributi che, combinati tra di loro o con altre informazioni esterne, possono identificare un soggetto o quantomeno ridurre l'incertezza sull'identità;
- **Confidenziali**: attributi che contengono informazioni sensibili;
- **Non confidenziali**: attributi non considerati sensibili e la cui conoscenza non comporta violazione della privacy.

Per ragioni principalmente statistiche o economiche, questi dataset possono essere diffusi o trasferiti ad altri soggetti. Tali operazioni risultano molto pericolose per la privacy dei soggetti titolari dei dati rilasciati, in quanto se ne perde il controllo e non si ha certezza sugli scopi per cui vengono utilizzati.

Pertanto, prima del rilascio, i dati sensibili devono essere propriamente de-identificati, ossia occorre rimuovere qualsiasi identificatore. Spesso, però, la de-identificazione non è sufficiente a proteggere l'identità degli individui, in quanto è possibile che tramite il controllo incrociato con altre fonti esterne si riesca a collegare gli attributi comuni e ricostruire le informazioni rimosse.

Tra gli attacchi alla privacy si possono distinguere:

- **Attacchi di re-identificazione**: consistono nel tentativo di associare l'identità di un soggetto ad un insieme di dati sensibili disponibili;
- **Attacchi di inferenza**: tramite i dati rilasciati, è possibile determinare, con un certo livello di confidenza, l'identità oppure i valori di altre caratteristiche di un certo soggetto;
- **Linking attacks**: il confronto tra i dati rilasciati e altre fonti esterne consente di collegare diversi attributi appartenenti allo stesso individuo, anche senza rilevare la sua identità.

Le minacce sopra elencate stanno diventando sempre più comuni e più semplici da realizzare a causa della disponibilità e della facilità di accesso a grandi quantità di informazioni. La presenza di fonti esterne, spesso sconosciute, consiste infatti in uno dei principali fattori che alimentano le minacce alla privacy degli individui.

Alcune contromisure che si possono attuare per limitare i rischi di divulgazione sono le seguenti:

- Campionamento dei dati, che limita la quantità di informazioni rilasciate;
- Inserimento di rumore nei dati, che limita la qualità delle informazioni rilasciate;
- Dati non aggiornati o espressi in forme differenti, che rendono più difficoltosa l'attività di collegamento con fonti esterne;

La de-identificazione e le contromisure presentate in questa sezione possono essere considerati dei punti di partenza, che presi singolarmente non offrono grandi garanzie per la protezione delle informazioni. Altre tecniche, più complesse e strutturate, vengono applicate nei contesti reali e con migliori risultati. L'anonimizzazione dei dati è una di queste.

2.3 Anonimizzazione dei dati

L'anonimizzazione dei dati è un'operazione di de-identificazione mirata a trasformare irreversibilmente i dati personali in dati anonimi. Consiste nel processo di rimozione o alterazione delle informazioni identificative da un insieme di dati, al fine di rendere le singole entità non identificabili.

La differenza sostanziale con la pseudonimizzazione, indicata nel GDPR come misura di protezione dei dati personali, sta nel fatto che un dato pseudonimizzato non elimina il rischio che lo stesso possa essere ricostruito e riassociato all'utente. Di conseguenza, non fa venire meno il presupposto di identificabilità per il quale si applica la disciplina GDPR.

Al contrario, l'anonimizzazione rende impossibile l'identificazione dell'utente e, pertanto, i dati anonimizzati non sono soggetti alle restrizioni sul trattamento dei

dati personali previste dalla normativa. Questo determina un grande vantaggio per le organizzazioni che gestiscono dati sensibili, in quanto elimina una fonte di rischio e consente l'utilizzo e la condivisione dei dati per scopi legittimi come la ricerca scientifica, l'analisi dei dati e lo sviluppo di prodotti e servizi, garantendo allo stesso tempo la tutela della privacy degli individui.

Inoltre, ciò che distingue l'anonimizzazione con la pseudonimizzazione e la crittografia è la reversibilità. L'anonimizzazione, per definizione, trasforma i dati irreversibilmente, mentre la pseudonimizzazione e la crittografia, per come sono costruiti, sono processi reversibili, in cui il dato iniziale può esserre ricostruito grazie ad informazioni aggiuntive, che consistono nella conoscenza di dati esterni o delle chiavi di cifratura. Questa caratteristica rende i due metodi meno sicuri a livello di protezione della privacy.

Il processo di anonimizzazione viene realizzato tramite l'applicazione di quattro principali tecniche:

- **Mascheramento**: consiste nell'eliminazione degli identificatori diretti, come ad esempio nome, cognome e codice fiscale;

- **Randomizzazione o Perturbazione**: famiglia di tecniche che prevedono l'alterazione dei dati con l'obiettivo di spezzare il legame tra questi e l'individuo di riferimento, e che comprendono l'aggiunta di rumore statistico oppure la permutazione;

- **Generalizzazione**: tecniche che modificano la scala o l'ordine di grandezza di un attributo, in modo che più individui condividano gli stessi valori. Comportano una perdita di dettaglio dei dati, ma riducono il rischio di corrispondenza univoca con fonti esterne;

- **Anonimizzazione stratificata**: detta anche ri-anonimizzazione, consiste in una seconda anonimizzazione di dati già resi anonimi in precedenza.

Nonostante si consideri che i dati anonimizzati rendano, teoricamente, impossibile la re-identificazione, va notato che l'anonimizzazione completa può essere difficile

da ottenere, oltre che onerosa in termini di tempo e denaro, e che gli sviluppi tecnologici hanno reso più complessa la protezione dei dati. Pertanto, è essenziale combinare l'anonimizzazione con altre misure di sicurezza dei dati per mitigare il rischio di de-anonimizzazione e proteggere la riservatezza delle informazioni sensibili.

2.3.1 k-Anonymity

Quando si tratta di anonimizzazione dei dati, uno dei concetti a cui si fa sempre riferimento è quello di k-Anonymity o, in italiano, k-Anonimato.

La k-Anonymity è una proprietà fondamentale nell'anonimizzazione dei dati che si basa sull'idea di rendere ogni record di dati indistinguibile da almeno altri $k-1$ record all'interno del dataset. Nello specifico, i dati rilasciati devono essere tali per cui qualsiasi combinazione di valori di quasi-identificatori possa essere indistintamente attribuita ad almeno k individui. In questo modo, viene concessa la possibilità di rilasciare dati veritieri, pur mantenendo protetta l'identità degli individui.

Si intuisce che un valore elevato del parametro k garantisce un alto livello di protezione, in quanto un set di dati sensibili sarà riconducibile a un maggior numero di individui, ma allo stesso tempo comporta delle problematiche in termini di perdita di utilità dei dati.

Le tecniche principali tramite le quali si può ottenere la k-Anonymity sono due: la generalizzazione e la soppressione.

La generalizzazione è una procedura che comporta la sostituzione dei valori specifici di un attributo con dei valori più generici. Essa si basa sulla definizione di una gerarchia di generalizzazione. Alcuni degli esempi più comuni sono:

- Per l'età, il reddito e in generale qualsiasi attributo numerico, è possibile sostituire il valore specifico con un intervallo di valori entro cui ricade il valore. Se il caso lo richiede, è possibile poi combinare più intervalli in uno unico più largo. Ad esempio, si può sostituire il valore 25 con il range $[20-30]$;

- I Paesi possono essere sostituiti dal continente di cui fanno parte. Ad esempio Italia, Francia, Germania vengono sostituiti con Europa;

Generalizzazione	**Soppressione**			
	Tupla	*Attributo*	*Cella*	*Nessuno*
Attributo	**AG_TS**	**AG_AS** ≡ AG_	**AG_CS**	**AG_** ≡ AG_AS
Cella	**CG_TS** non applicabile	**CG_AS** non applicabile	**CG_CS** ≡ CG_	**CG_** ≡ CG_CS
Nessuno	**_TS**	**_AS**	**_CS**	-

Tabella 2.1: Classificazione delle tecniche di k-Anonymity

- I codici di avviamento postale possono essere generalizzati rimuovendo una o più cifre e lasciando solo un prefisso. Ad esempio, 20129 diventa 2012* o anche 201**.

La generalizzazione può essere applicata a livello di attributo (colonna) o a livello di cella, per cui una specifica colonna potrebbe contenere valori con differenti livelli di generalizzazione.

La soppressione, invece, consiste nella semplice rimozione di informazioni sensibili. Viene utilizzata spesso per ridurre l'ammontare di generalizzazione necessaria a soddisfare i vincoli di k-Anonimato. Ad esempio, nei casi in cui sono presenti dei record univoci per cui la generalizzazione non è applicabile o renderebbe i dati poco significativi, è possibile rimuovere l'intero record. La soppressione può essere applicata a livello di record (rimozione di una tupla), di attributo (rimozione di una colonna) o di cella (rimozione di un singolo valore).

I due metodi sono spesso utilizzati insieme, e la combinazione dei livelli a cui vengono applicati determinano una classificazione delle tecniche di k-Anonymity, mostrate in Tabella 2.1. Ciascuna di esse può richiedere uno specifico algoritmo per essere implementata, e i risultati che vengono prodotti possono essere più o meno differenti. In Tabella 2.2 viene riportato un esempio di dataset composto da quattro attributi quasi-identificatori, mentre in Tabella 2.3 vengono riportate le sue versioni 2-anonime ottenute applicando diversi livelli di generalizzazione e soppressione. La

Nazionalità	Data di nascita	Sesso	CAP
italiana	12/04/1995	F	20132
italiana	13/09/1995	F	20131
italiana	15/04/1995	F	20129
italiana	13/03/1997	M	20129
italiana	18/03/1997	M	20129
francese	27/09/1995	F	20128
francese	27/09/1995	F	20129
spagnola	27/09/1995	F	20129
spagnola	27/09/1995	F	20131

Tabella 2.2: Tabella d'esempio con quattro quasi-identificatori per k-Anonymity

trasformazione di un dataset nella sua versione k-Anonima ottimale (ossia quella che riduce al minimo la perdita di informazione) è noto per essere un problema computazionalmente complesso e rientra nella classe dei problemi NP-hard. Ciò significa che non esiste un algoritmo polinomiale che possa risolverlo in tempo polinomiale. Nel dettaglio, la complessità computazionale del problema di k-Anonymity dipende dai seguenti fattori:

- **Numero di attributi sensibili e da generalizzare**: aumentando il numero di entrambe le tipologie di attributi, cresce sia il numero di combinazioni di quasi-identificatori da considerare sia la generalizzazione da applicare;
- **Dimensione del dataset**: maggiore è il numero di record nel dataset, più complessa diventa la ricerca di una generalizzazione che soddisfi il requisito di k-Anonymity;
- **Requisito di k-Anonymity**: aumentare il valore di k può rendere più difficile trovare una soluzione che soddisfi il requisito;
- **Tecnica di k-Anonymity**: le combinazioni dei livelli di generalizzazione e soppressione determinano algoritmi differenti, e di conseguenza complessità diverse.

Naz	Data	Sesso	CAP
italiana	04/1995	F	201**
italiana	04/1995	F	201**
italiana	03/1997	M	201**
italiana	03/1997	M	201**
francese	09/1995	F	201**
francese	09/1995	F	201**
spagnola	09/1995	F	201**
spagnola	09/1995	F	201**

(a) **AG_TS**

Naz	Data	Sesso	CAP
italiana	*	F	*
italiana	*	F	*
italiana	*	F	*
italiana	03/1997	M	2012*
italiana	03/1997	M	2012*
francese	09/1995	F	2012*
francese	09/1995	F	2012*
spagnola	09/1995	F	*
spagnola	09/1995	F	*

(b) **AG_CS**

Naz	Data	Sesso	CAP
italiana	1995	F	201**
italiana	1995	F	201**
italiana	1995	F	201**
italiana	1997	M	201**
italiana	1997	M	201**
francese	1995	F	201**
francese	1995	F	201**
spagnola	1995	F	201**
spagnola	1995	F	201**

(c) **AG_** ≡ **AG_AS**

Naz	Data	Sesso	CAP
italiana	1995	F	201**
italiana	1995	F	201**
italiana	1995	F	201**
italiana	03/1997	M	20129
italiana	03/1997	M	20129
francese	27/09/1995	F	2012*
francese	27/09/1995	F	2012*
spagnola	27/09/1995	F	201**
spagnola	27/09/1995	F	201**

(d) **CG_** ≡ **CG_CS**

Naz	Data	Sesso	CAP
italiana	*	F	*
italiana	*	F	*
italiana	*	F	*
italiana	*	M	*
italiana	*	M	*
francese	*	F	*
francese	*	F	*
spagnola	*	F	*
spagnola	*	F	*

(e) **_AS**

Naz	Data	Sesso	CAP
italiana	*	F	*
italiana	*	F	*
italiana	*	F	*
italiana	*	M	20129
italiana	*	M	20129
*	27/09/1995	F	*
*	27/09/1995	F	20129
*	27/09/1995	F	20129
*	27/09/1995	F	*

(f) **_CS**

Tabella 2.3: Esempi di applicazione delle tecniche di k-Anonymity

La maggioranza degli algoritmi esatti proposti in letteratura ha complessità temporale esponenziale nel numero degli attributi che compongono i quasi-identificatori. Quando questo numero è piccolo rispetto al numero di tuple nella tabella, allora questi algoritmi sono praticabili.

Alcuni algoritmi noti sono:

- **Samarati** [6]: si basa sulla costruzione di una matrice di distanze tra i record del dataset rispetto agli attributi quasi-identificatori. Queste distanze rappresentano il livello di similitudine tra i record. L'obiettivo dell'algoritmo è di ottenere una soluzione k-minimal, in cui le distanze tra i record sono minimizzate e la soluzione ottenuta garantisce perdita di informazioni minima;

- k**-Optimize** [7]: prevede l'ordinamento dei valori relativi agli attributi quasi-identificatori e l'assegnazione di un indice a ciascuno. Una generalizzazione è ottenuta come unione di valori di indici individuali. Viene costruito un albero in cui la radice è un set vuoto e i nodi intermedi sono ottenuti aggiungendo al nodo padre un indice a lui successivo. Ad ogni nodo è associato un costo che riflette l'ammontare di generalizzazione e di soppressione. L'algoritmo effettua una visita in profondità (depth-first-search) per trovare l'anonimizzazione con il minor costo, utilizzando tecniche di pruning per ridurre il costo della visita.

- **Incognito** [8]: si basa sul concetto che un dataset è k-Anonimo se anche i suoi sottoinsiemi propri sono k-Anonimi. Utilizzando degli alberi decisionali e una strategia bottom-up, alla prima iterazione calcola le generalizzazioni dei singoli attributi ed elimina quelle che non soddisfano la k-Anonymity, alla seconda iterazione considera tute le possibili coppie di generalizzazioni dall'iterazione precedente e ne verifica la k-Anonymity, alla terza considera le terne, e così via fino a considerare la combinazione di tutti gli attributi;

- **Mondrian**: algoritmo multidimensionale basato su una rappresentazione spaziale dei dati, verrà trattato in maniera approfondita nel Capitolo 3.

Nazionalità	Data di nascita	Sesso	CAP	Patologia
italiana	1995	F	201**	asma
italiana	1995	F	201**	ipertensione
italiana	1995	F	201**	epilessia
italiana	1997	M	201**	ipertensione
italiana	1997	M	201**	ipertensione
francese	1995	F	201**	diabete
francese	1995	F	201**	diabete
spagnola	1995	F	201**	epilessia
spagnola	1995	F	201**	diabete

Tabella 2.4: Problema dell'omogeneità dei dati sensibili

2.3.2 l-Diversity

La k-Anonymity, nonostante offra un grado di protezione elevato, non è immune da vulnerabilità. Si osservi l'esempio riportato in Tabella 2.4. I record appartenenti al secondo e al terzo gruppo presentano la stessa combinazione di valori degli attributi quasi-identificatori, il che soddisfa il requisito di 2-anonimato. Il problema, però, è che possiedono anche lo stesso valore dell'attributo sensibile. Questa situazione comporta che, se un soggetto malintenzionato sa che uno specifico individuo è all'interno della tabella e ne riconosce i quasi-identificatori, può dedurre con assoluta certezza la patologia di cui è affetto.

Per fronteggiare questa criticità, definita come problema di omogeneità dei dati sensibili, è possibile potenziare la k-Anonymity con un'altra tecnica di anonimizzazione: la l-Diversity. Tale proprietà impone che all'interno di ogni gruppo di dati anonimizzato, quindi con la stessa combinazione di quasi-identificatori, siano rappresentati almeno l diversi valori di attributi sensibili. In questo modo, un soggetto malintenzionato deve eliminare almeno $l - 1$ possibili valori per essere in grado di dedurre l'informazione sensibile di un determinato individuo.

L'aggiunta del requisito di l-Diversity incrementa il grado di protezione della privacy, ma allo stesso tempo comporta un maggior costo di implementazione e non sempre risulta efficace. Ad esempio, gli l valori presenti in un set di dati anonimizzato potrebbero essere sintatticamente differenti ma semanticamente simili.

Questa problematica viene chiamata attacco di similarità, e consente ai soggetti malintenzionati di ottenere un'informazione non precisa ma con un alto livello di dettaglio.

2.3.3 Altre tecniche avanzate

Per affrontare le sfide emergenti e le minacce alla privacy dei dati in contesti sempre più complessi, nel tempo sono state proposte una serie di tecniche avanzate di anonimizzazione:

- **T-Closeness**: mira a garantire che la distribuzione delle informazioni sensibili all'interno di un gruppo di dati anonimizzato sia quanto più simile alla distribuzione nel dataset originale;
- **δ-Presence**: impone che ci sia una presenza minima (δ) di valori sensibili poco comuni all'interno di ciascun gruppo, al fine di rendere più difficile l'identificazione individuale;
- **Differential Privacy**: tecnica che introduce rumore casuale nei dati per proteggere la privacy degli individui. L'idea di base è che l'aggiunta o la rimozione dei dati di un singolo individuo nel dataset non dovrebbe avere un impatto significativo sull'output di una query. Per fare questo, il rumore aggiunto ai dati è controllato da un parametro di privacy che determina quanto la risposta della query può variare in presenza o assenza di un singolo individuo nel dataset.

Le tecniche presentate in questa sezione hanno l'obiettivo di migliorare o integrare le proposte classiche di k-Anonymity e l-Diversity. La scelta di quali implementare varia in base al contesto, alla tipologia di dati che si sta trattando e alle possibili minacce a cui risultano vulnerabili.

2.4 Limiti e sfide dell'anonimizzazione

Per definizione, l'anonimizzazione è una procedura che rende impossibile la re-identificazione degli individui dal set di dati anonimizzato. Questo è vero sul piano

teorico, ma in contesti reali l'anonimizzazione al 100% non è sempre possibile, a causa di limiti computazionali ma anche di particolari caratteristiche del dataset di partenza. Perciò bisogna tenere sempre in considerazione che il rischio di re-identificazione, per quanto basso, non è mai nullo. Nella pratica, quindi, si tende a stabilire una certa soglia che rappresenta il rischio massimo che si è disposti a tollerare.

Gli attacchi di re-identificazione, come già espresso in precedenza, si basano fondamentalmente sulla disponibilità di fonti esterne, le quali permettono di effettuare dei confronti e di collegare i dati comuni. Con l'anonimizzazione questa attività viene ostacolata poiché gli attributi sensibili vengono alterati o rimossi, riducendo la possibilità di matching tra diverse fonti. Tuttavia, esiste un altro elemento critico che negli esempi proposti è stato implicitamente considerato ma non definito, che è la background knowledge (conoscenze pregresse, a priori). Questa tipologia di informazione è totalmente imprevedibile da parte del titolare dei dati e può essere sfruttata per effettuare attacchi di inferenza. Ad esempio, se un soggetto malintenzionato conosce alcune caratteristiche di un individuo e sa che questo individuo è presente nel dataset anonimizzato, allora potrebbe riconoscere i quasi-identificatori che si riferiscono al soggetto e ricavare i suoi dati sensibili o quantomeno ridurne l'incertezza.

Un altro fattore da tenere in considerazione nella pianificazione dell'anonimizzazione è l'utilità dei dati che devono essere rilasciati. Il processo di anonimizzazione, infatti, comporta una riduzione nella significatività delle informazioni, dovuta alle operazioni di modifica, generalizzazione e rimozione dei valori. Di conseguenza, oltre al rischio di re-identificazione, è necessario tenere in considerazione la capacità espressiva richiesta dai dati rilasciati nello stabilire il grado di anonimizzazione ricercato.

Per concludere, si è visto che l'anonimizzazione non è priva di rischi e non garantisce una protezione assoluta dei dati pubblicati. Perciò, prima dell'anonimizzazione, è fondamentale valutare la necessità di includere determinati attributi sensibili nel set di dati. Rimuovere o ridurre al minimo l'inclusione di dati sensibili non necessari è una buona norma che può incrementare ulteriormente la protezione della privacy.

Capitolo 3

Descrizione del framework Mondrian

In questo capitolo viene presentato il framework Mondrian, realizzato dal laboratorio di sicurezza informatica dell'Università degli Studi di Bergamo (UniBG Seclab), in collaborazione con l'Università degli Studi di Milano, nel contesto del progetto europeo MOSAICrOWN [9]. Questo progetto ha avuto durata triennale, dall'inizio del 2019 al termine del 2021, e ha avuto come finalità principale la realizzazione di soluzioni per la condivisione e l'analisi dei dati appartenenti a differenti proprietari, preservandone il controllo e la riservatezza.

3.1 Introduzione

k-Anonymity e l-Diversity sono due tecniche di protezione dei dati semplici da comprendere, ma la cui implementazione richiede particolare attenzione a diversi aspetti. In particolare, un fattore cruciale riguarda il bilanciamento tra tutela della privacy e utilità dei dati. Infatti, valori grandi dei parametri k e l consentono di ottenere un alto grado di protezione, dovuta alla elevata difficoltà nella re-identificazione degli individui, ma allo stesso tempo implica una consistente perdita di informazione, causata dall'eccessiva generalizzazione dei dati.

Oltre a questo, per ottenere delle soluzioni ottimali, bisogna stabilire dei criteri precisi per la scelta di quali attributi quasi-identificatori devono essere generalizzati e in quale maniera. Ciò richiede completa visibilità del dataset da anonimizzare. Soluzioni esistenti, che operano in ambienti centralizzati, risultano quindi non ido-

nei all'anonimizzazione di grandi volumi di dati, a causa dei limiti di memoria dei dispositivi.

Il ricorso ad architetture distribuite e scalabili rappresenta una valida alternativa per la risoluzione di questo problema, ma necessita di meticolosità nella progettazione. Infatti, una semplice distribuzione del carico di lavoro tra diversi esecutori potrebbe portare ad un peggioramento della qualità delle soluzioni e ad un aumento dei costi di computazione, dovuti ad esempio a sincronizzazioni e scambi di dati tra esecutori.

Il framework realizzato dall'UniBG Seclab ha lo scopo di risolvere in maniera ottimale il problema dell'anonimizzazione distribuita di grandi set di dati. La soluzione proposta è un'estensione di Mondrian [10], un approccio efficiente per il conseguimento della k-Anonymity in uno scenario centralizzato. Nello specifico, l'obiettivo è quello di rafforzare la k-Anonymity con la l-Diversity, agendo in ambiente distribuito.

3.1.1 Algoritmo Mondrian

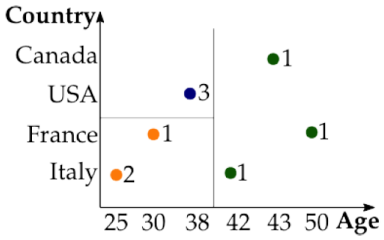
Prima di entrare nei dettagli del framework, occorre conoscere il funzionamento dell'algoritmo di anonimizzazione.

Mondrian è un algoritmo multidimensionale per la risoluzione dei problemi di k-Anonymity. Questo algoritmo si basa su una rappresentazione spaziale dei dati, che viene realizzata mappando ciascun attributo quasi-identificatore a una dimensione e ciascuna combinazione di valori dei quasi-identificatori a un punto in tale spazio.

Mondrian opera un processo ricorsivo con l'obiettivo di partizionare lo spazio in regioni contenenti un certo numero di punti, che equivale a suddividere il set di dati in frammenti con un certo numero di record. In particolare, ad ogni iterazione viene scelto un attributo quasi-identificatore ed effettuato un taglio di ogni frammento ricavato dall'iterazione precedente (l'intero dataset alla prima iterazione) in base al valore mediano di tale attributo. L'algoritmo termina quando qualsiasi possibile taglio ulteriore genererebbe solo frammenti con meno di k tuple. A quel punto i valori degli attributi quasi-identificatori vengono sostituiti con le loro generalizzazioni.

Age	Country	TopSpeed
25	Italy	132
25	Italy	132
30	France	128
42	Italy	110
50	France	115
43	Canada	115
38	USA	126
38	USA	127
38	USA	140

(a)



(b)

Age	Country	TopSpeed
[25,30]	Europe	132
[25,30]	Europe	132
[25,30]	Europe	128
[42,50]	World	110
[42,50]	World	115
[42,50]	World	115
38	USA	126
38	USA	127
38	USA	140

(c)

Figura 3.1: Un esempio di dataset (a), la sua rappresentazione spaziale (b) e una sua versione 3-anonymous e 2-diverse (c) [11]

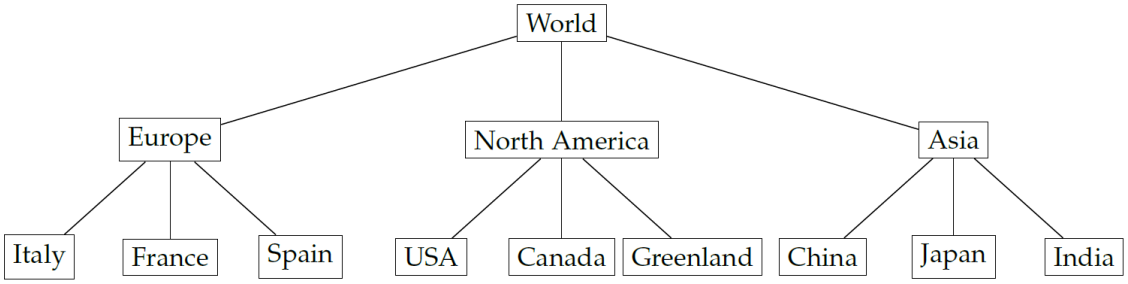


Figura 3.2: Gerarchia di generalizzazione dell'attributo `Country` [11]

In Figura 3.1(b) viene mostrato un esempio di rappresentazione spaziale, ottenuta dal dataset in Figura 3.1(a), dove i valori associati a ciascun punto rappresentano il numero di tuple che hanno quella specifica combinazione di quasi-identificatori (`Age` e `Country`). La versione 3-anonima del dataset riportata in Figura 3.1(c) è stata ottenuta tramite il seguente procedimento: il dataset di partenza viene inizialmente partizionato in relazione al valore mediano 38 dell'attributo `Age`, ottenendo i frammenti $F_{\texttt{Age}\leq 38}$ e $F_{\texttt{Age}>38}$; in seguito, il frammento $F_{\texttt{Age}\leq 38}$ viene ulteriormente partizionato in base all'attributo `Country`, ottenendo i sotto-frammenti $F_{\texttt{Age}\leq 38,\texttt{Country}\in\{Canada,Usa\}}$ e $F_{\texttt{Age}\leq 38,\texttt{Country}\in\{France,Italy\}}$; nessuna ulteriore frammentazione è possibile in quanto i frammenti contengono esattamente $k = 3$ tuple, e quindi si procede con la generalizzazione dei quasi-identificatori, utilizzando gli intervalli numerici per l'attributo `Age` e la gerarchia di generalizzazione in Figura 3.2 per l'attributo `Country`.

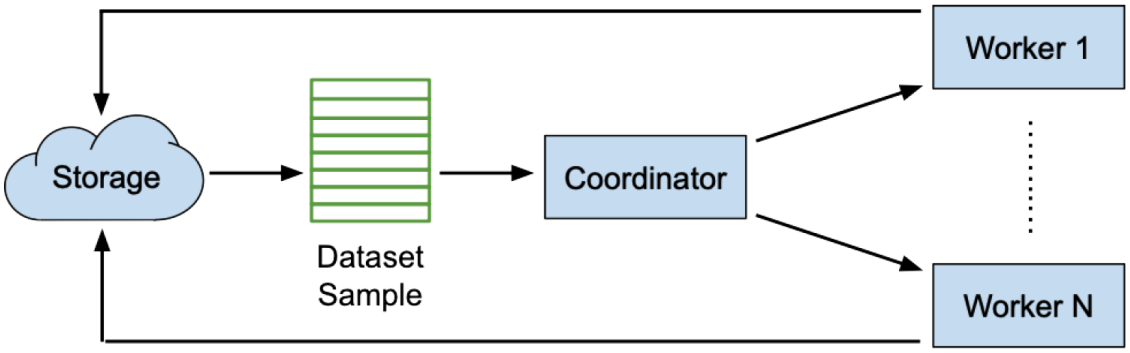


Figura 3.3: Struttura del processo di anonimizzazione distribuita [11]

3.2 Fasi di esecuzione

L'esecuzione del framework Mondrian si struttura in tre fasi principali:

- **Pre-processamento dei dati**: fase iniziale che prevede la suddivisione dei dati in frammenti e l'assegnazione degli stessi ai workers;
- **Anonimizzazione distribuita dei dati**: fase centrale in cui i dati vengono anonimizzati in parallelo da un numero stabilito di workers;
- **Conclusione e valutazione della perdita di informazione**: fase finale in cui i frammenti anonimizzati vengono combinati e viene valutata la qualità della soluzione ottenuta.

Tutte le fasi sono dirette e supervisionate da un Coordinatore, che ha la responsabilità di eseguire il partizionamento, distribuire i carichi di lavoro, raccogliere i risultati e comunicare gli esiti all'utente finale.

In Figura 3.3 viene rappresentato lo scenario distribuito del framework, caratterizzato dal Coordinatore, dal set di workers e da una piattaforma di storage (possibilmente distribuito), accessibile sia dal Coordinatore sia dai workers, che contiene il dataset da anonimizzare e, al termine del processo, il dataset anonimizzato.

3.2.1 Pre-processamento dei dati

La fase di pre-processamento dei dati è fondamentale per l'anonimizzazione distribuita.

La prima questione da affrontare riguarda la definizione di una strategia di frammentazione che stabilisca quali tuple appartengono a quali frammenti. La strategia adottata, la cui efficacia verrà dimostrata dai risultati sperimentali, prevede la suddivisione del set di dati in base ai valori degli attributi quasi-identificatori delle singole tuple. Questa metodologia richiederebbe completa visibilità del dataset da parte del Coordinatore per stabilire la composizione di ciascun frammento. Tuttavia, trattando grandi volumi di dati, la memoria centrale del Coordinatore potrebbe non essere in grado di contenere l'intero dataset. Perciò viene proposta una nuova strategia che permette di definire le condizioni di frammentazione a partire da un campionamento del dataset, la cui dimensione è stabilita dinamicamente in relazione alla capacità del Coordinatore. Questa tecnica rappresenta una caratteristica distintiva del framework, perché consente di ottenere lo stesso risultato in termini di frammentazione, risolvendo al tempo stesso il problema del volume dei dati.

Successivamente, il Coordinatore comunica le condizioni di frammentazione ai vari workers, i quali procedono con la lettura dei dati direttamente dalla piattaforma di storage. In questo modo, viene ridotto il carico di lavoro del Coordinatore, il quale non è più responsabile del trasferimento dei singoli frammenti, e vengono ottimizzati i tempi di esecuzione dell'intero processo di anonimizzazione.

Strategie di partizionamento

La selezione di una strategia di partizionamento appropriata è cruciale in questo scenario, poiché una suddivisione casuale implicherebbe una significativa perdita di informazione. Infatti, se un frammento contiene tuple con valori degli attributi quasi-identificatori molto eterogenei, ciascun worker deve applicare un'ammontare elevato di generalizzazione per soddisfare il requisito di k-Anonymity. Al contrario, se il partizionamento è in grado di creare frammenti con valori simili di tali attributi, la perdita di informazione viene sensibilmente limitata.

In considerazione di questo obiettivo, sono state proposte due strategie alternative di partizionamento, entrambe implementate nel framework:

- **Approccio monodimensionale basato sui quantili**: seleziona un attributo quasi-identificatore e partiziona il dataset campionato in n frammenti

corrispondenti agli n-quantili dell'attributo nel dataset campionato;

- **Approccio multidimensionale**: simile all'algoritmo Mondrian, partiziona il dataset campionato in maniera ricorsiva. Dato un frammento (inizialmente l'intero dataset campionato), seleziona un attributo quasi-identificatore e lo suddivide in due sotto-frammenti con riferimento al valore mediano dell'attributo all'interno del frammento iniziale. In seguito, entrambi i sotto-frammenti vengono nuovamente partizionati, fino a quando si ottengono n frammenti in totale.

Entrambi i metodi si basano sull'ordinamento dei valori degli attributi quasi-identificatori scelti per il partizionamento, rispettivamente per il calcolo dei quantili e delle mediane. Perciò, dopo la selezione dell'attributo, viene effettuato un ordinamento in relazione alla tipologia di dato: se l'attributo è numerico, l'ordine è naturalmente definito; se l'attributo è categorico, l'ordine è dato da una gerarchia di generalizzazione e, più specificamente, dall'ordine in cui appaiono i nodi foglia, con l'obiettivo di mantenere nello stesso frammento valori più vicini, i quali generalizzano ad un valore più specifico (ad un livello più basso nella gerarchia), permettendo di ridurre la perdita di informazione. Ad esempio, con riferimento alla gerarchia dell'attributo `Country` in Figura 3.2, l'ordine delle foglie è $\langle Italy, France, Spain, USA, Canada, Greenland, China, Japan, India \rangle$. Questo ordinamento consente di combinare nello stesso frammento i valori $Italy$ e $France$, che generalizzano ad un valore più specifico ($Europe$) rispetto ad un frammento contenente i valori $Italy$ e $Japan$ ($World$).

I due approcci presentano delle differenze nella definizione dei frammenti. Il primo metodo basato sui quantili assicura una frammentazione bilanciata, risultando in frammenti contenenti lo stesso numero di tuple, ma la sua applicazione può essere limitata dal dominio dell'attributo quasi-identificatore. Ad esempio, non può essere utilizzato se il numero di frammenti è più grande del dominio dell'attributo scelto.

Al contrario, la strategia multidimensionale, pur essendo sempre applicabile, potrebbe determinare un carico di lavoro doppio per alcuni workers. L'algoritmo ricorsivo, infatti, raddoppia il numero di frammenti ad ogni iterazione, generando in

Q_PARTITION(D, W)
1: **let** a be the attribute used to partition D
2: $R := \{rank(t[a]) \mid t \in D\}$ /* rank of a's values in the ordering */
3: **let** q_i be the i^{th} $|W|$-quantile for $R, \forall i = 1, \ldots, |W|$
4: $F_1 := \{t \in D \mid rank(t[a]) \leq q_1\}$
5: **for each** $i = 2, \ldots, |W|$ **do**
6: $\quad F_i := \{t \in D \mid q_{i-1} < rank(t[a]) \leq q_i\}$

Figura 3.4: Partizionamento basato sui quantili [11]

totale 2^i frammenti alla i-esima iterazione. Se il numero n di workers è una potenza di 2, allora ciascun worker avrà assegnato un frammento. In caso contrario, allo scopo di utilizzare tutti i workers stabiliti, la procedura ricorsiva terminerà quando $n \leq 2^i$, e perciò si avranno $2^i - n$ workers con due frammenti assegnati, con la diretta conseguenza di un carico di lavoro sbilanciato. Ad esempio, se $n = 7$, il numero di frammenti generati sarà $2^3 = 8$ alla terza iterazione, e uno dei workers avrà in carico due frammenti da anonimizzare.

Anche in termini di complessità computazionale si evidenziano delle differenze, con l'approccio basato sui quantili che risulta più efficiente rispetto al multidimensionale. Rappresentando con D il dataset campionato e con W il set di workers, si ha che il primo metodo ha costo $O(|D|)$, dovuto ai calcoli della mediana e dei quantili, entrambi con costo lineare, mentre il secondo metodo ha costo $O(|D| \log |W|)$, dovuto al fatto che la ricorsione viene eseguita $\lceil \log |W| \rceil$ volte, e la dimensione dei frammenti in input alle diverse ricorsioni è di $|D|$.

Gli pseudocodici dei due algoritmi di partizionamento sono riportati rispettivamente in Figura 3.4 e in Figura 3.5.

Scelta degli attributi per il partizionamento

Il primo passo da compiere nell'operazione di partizionamento, indipendentemente dalla strategia adottata, è la selezione dell'attributo quasi-identificatore da utilizzare per la separazione dei frammenti.

Per l'approccio basato sui quantili, l'attributo scelto è quello con il maggior numero di valori distinti all'interno del dataset campionato. Questa strategia consente

M_PARTITION(D, W, i)
1: **let** a be the attribute used to partition D
2: $R := \{rank(t[a]) \mid t \in D\}$ /* rank of a's values in the ordering */
3: **let** m be the median of R
4: $F_1 := \{t \in D \mid rank(t[a]) \leq m\}$
5: $F_2 := \{t \in D \mid rank(t[a]) > m\}$
6: **if** $i < \lceil \log_2 |W| \rceil$ **then**
7: **M_Partition**$(F_1, W, i+1)$
8: **M_Partition**$(F_2, W, i+1)$

Figura 3.5: Partizionamento multidimensionale [11]

di ottenere frammenti con valori simili tra loro (dovuto all'ordinamento effettuato), in modo che l'ammontare di generalizzazione applicato in ogni frammento sia limitato. Ma allo stesso tempo, distribuisce valori distanti in frammenti differenti, facendo sì che i valori generalizzati siano diversi tra frammenti. Al contrario, se il dominio dell'attributo fosse molto ristretto, i valori sarebbero generalizzati eccessivamente e la perdita di informazione sarebbe elevata.

Per l'approccio multidimensionale, invece, la scelta ricade sull'attributo che, all'interno del frammento da partizionare, ha la maggiore rappresentatività dei valori assunti nel dataset campionato. Se l'attributo è numerico, la sua rappresentatività è definita come rapporto tra l'ampiezza del range di valori nel frammento e l'ampiezza del range di valori nel dataset campionato. Se l'attributo è categorico, la sua rappresentatività è definita come rapporto tra il numero di valori distinti nel frammento e il numero di valori distinti nel dataset campionato. Formalmente, la rappresentatività $rep(a)$ di un attributo quasi-identificatore a è definita come segue:

$$rep(a) = \begin{cases} \frac{\max_F\{t[a]\} - \min_F\{t[a]\}}{\max_D\{t[a]\} - \min_D\{t[a]\}} & \text{se } a \text{ è numerico} \\ \frac{\text{count}_F(\text{distinct } t[a])}{\text{count}_D(\text{distinct } t[a])} & \text{se } a \text{ è categorico} \end{cases} \tag{3.1}$$

dove $\max_F\{t[a]\}$ e $\min_F\{t[a]\}$ sono i valori massimo e minimo dell'attributo a all'interno del frammento, $\max_D\{t[a]\}$ e $\min_D\{t[a]\}$ sono i valori massimo e minimo dell'attributo a all'interno del dataset campionato e $\text{count}_F(\text{distinct } t[a])$ e $\text{count}_D(\text{distinct } t[a])$ sono il numero di valori distinti dell'attributo a rispettivamente nel frammento e nel dataset campionato. Si noti che la rappresentatività di un attributo assume valori compresi nell'intervallo $(0, 1]$, e che alla prima iterazione, in

cui $F = D$, tutti gli attributi hanno rappresentatività 1 e perciò si sceglie l'attributo con il maggior numero di valori distinti, come nel partizionamento basato sui quantili.

Ad esempio, si consideri un frammento F con attributi quasi-identificatori $QI = \{a_1, a_2, a_3\}$, dove a_1 è categorico e a_2, a_3 sono numerici. Supponendo che:

- I valori distinti di a_1 sono 1000 in D e 100 in F
- L'intervallo di valori di a_2 è di 1000 in D e 500 in F
- L'intervallo di valori di a_3 è di 1000 in D e 700 in F

si ha che $rep(a_1) = 100/1000 = 0.1$, $rep(a_2) = 500/1000 = 0.5$ e $rep(a_3) = 700/1000 = 0.7$. L'attributo scelto per il partizionamento sarà a_3.

Recupero dei frammenti

Al termine del processo di frammentazione, i frammenti devono essere comunicati ai workers. Per minimizzare il costo della comunicazione, il Coordinatore genera le condizioni di frammentazione e le trasferisce ai workers. Queste condizioni sono composte da confronti logici tra attributi e valori utilizzati per il partizionamento.

Nell'approccio basato sui quantili, la suddivisione avviene considerando un solo attributo quasi-identificatore. Perciò le condizioni c_i descrivono i valori dell'attributo a che sono inclusi nell'i-esimo n-quantile. Ad esempio:

- $c_1 =$ "(`Age` ≤ 30)";
- $c_2 =$ "(`Age` > 30) AND (`Age` ≤ 38)";
- $c_3 =$ "(`Age` > 38) AND (`Age` ≤ 42)";
- ...

Nell'approccio multidimensionale, invece, la frammentazione richiede più iterazioni, ciascuna delle quali può considerare un attributo differente. Perciò le condizioni c_i sono congiunzioni che mostrano il partizionamento ricorsivo dei frammenti. Ad esempio:

- c_1 = "(Age ≤ 38) AND (Country $\in \{Italy, France\}$)";
- c_2 = "(Age ≤ 38) AND (Country $\in \{USA, Canada\}$)";
- c_3 = "(Age > 38)";

Una volta ricevute tali condizioni, ciascun worker può recuperare le tuple appartenenti al frammento assegnato in maniera diretta, senza la necessità che sia il Coordinatore a trasferirle.

3.2.2 Anonimizzazione dei dati

La fase di anonimizzazione, che segue quella di pre-processamento dei dati, prevede l'esecuzione in parallelo dell'algoritmo di anonimizzazione da parte degli workers. Nello specifico, ciascun worker si occupa, in maniera totalmente indipendente dagli altri workers, di anonimizzare il frammento che gli è stato assegnato, applicando una versione distribuita dell'algoritmo Mondrian per il conseguimento della k-Anonymity e della l-Diversity.

La differenza con l'algoritmo Mondrian descritto nel Paragrafo 3.1.1 sta appunto nell'aggiunta del requisito di l-Diversity per rinforzare la k-Anonymity. Di conseguenza, l'esecuzione ricorsiva termina quando ogni possibile frammentazione ulteriore genererebbe sotto-frammenti con meno di k occorrenze nelle combinazioni di valori degli attributi quasi-identificatori (che violerebbe la k-Anonymity), oppure meno di l valori degli attributi sensibili (che violerebbe la l-Diversity). Il resto dell'algoritmo, invece, non subisce variazioni: ad ogni iterazione, viene inizialmente scelto un attributo, secondo le logiche descritte nel Paragrafo 3.2.1; vengono poi generati due sotto-frammenti, separando il frammento di partenza in base al valore mediano dell'attributo scelto; infine viene invocata la ricorsione passando in input i sotto-frammenti generati.

Lo pseudocodice dell'algoritmo di anonimizzazione è riportato in Figura 3.6. La sua complessità computazionale è $O(|F| \log |F|)$, identificando con F il frammento di partenza.

ANONYMIZE(F)
1: **if** no partitioning can be done without violating
k-anonymity or ℓ-diversity **then**
2: **generalize** F[QI]
3: **else**
4: **let** a be the attribute for partitioning
5: $R := \{rank(t[a]) \mid t \in F\}$ /* rank of a's values in the ordering */
6: **let** m be the median of R
7: $F_1 := \{t \in F \mid rank(t[a]) \leq m\}$
8: $F_2 := \{t \in F \mid rank(t[a]) > m\}$
9: **Anonymize**(F_1)
10: **Anonymize**(F_2)

Figura 3.6: Algoritmo di anonimizzazione di un frammento [11]

L'ultimo step della fase di anonimizzazione corrisponde alla generalizzazione dei valori degli attributi quasi-identificatori contenuti nei frammenti generati. Le strategie di generalizzazione supportate dal framework Mondrian considerano:

- **Gerarchie di generalizzazione**: applicabile solamente per gli attributi categorici, i valori degli attributi quasi-identificatori vengono sostituiti con i loro antenati comuni di livello più basso (lowest common ancestor) nella relativa gerarchia di generalizzazione (esempio in Figura 3.2);

- **Prefissi comuni**: applicabile sia per attributi categorici sia per attributi numerici interpretati come stringhe, prevede la sostituzione degli attributi con una stringa che include il prefisso comune e un carattere jolly (tipicamente l'asterisco) al posto dei caratteri che differiscono. Ad esempio, i valori 10010, 10020 e 10030 dell'attributo `ZIP` possono essere generalizzati con il valore 100**;

- **Definizione di insiemi**: applicabile sia per attributi categorici sia per attributi numerici, la sostituzione avviene con un insieme che contiene tutti i valori dell'attributo quasi-identificatore. Ad esempio, i valori *Italy*, *Italy* e *France* dell'attributo `Country` possono essere generalizzati con l'insieme $\{Italy, France\}$;

- **Definizione di intervalli**: applicabile solamente per gli attributi numerici definiti in un dominio totalmente ordinato, la sostituzione avviene con un intervallo che contiene tutti i valori dell'attributo quasi-identificatore. Ad esempio, i valori 42, 50 e 43 dell'attributo `Age` possono essere generalizzati con l'intervallo $[42, 50]$.

3.2.3 Conclusione e valutazione della perdita di informazione

L'ultima fase ha l'obiettivo di riunire i frammenti anonimizzati e valutare la perdita di informazione. A tale scopo, ogni worker memorizza il proprio frammento di dati anonimizzato nella piattaforma di storage distribuito e procede con il calcolo della perdita di informazione relativa ad esso.

Per la valutazione della perdita di informazione sono state adottate due metriche:

- **Discernability Penalty**: assegna una penalità ad ogni tupla in base alla dimensione della classe di equivalenza E a cui la tupla appartiene (maggiore è il numero di tuple generalizzate allo stesso valore, maggiore è la penalità). Definito con $\hat{D}$ il dataset anonimizzato, la formula della Discernability Penalty è la seguente:

$$\mathrm{DP}(\hat{D}) = \sum_{E \in \hat{D}} |E|^2 \tag{3.2}$$

- **Normalized Certainty Penalty**: assegna una penalità in base all'ammontare di generalizzazione applicato ai valori degli attributi quasi-identificatori (maggiore la generalizzazione, maggiore la penalità). È applicabile per tutte le tipologie di generalizzazione descritte nella Sezione 3.2.2, ma con formule diverse.

  Nel caso di attributi numerici generalizzati in intervalli, la penalità è pari al rapporto tra l'ampiezza dell'intervallo della tupla generalizzata e il range di valori dell'attributo nel dataset.

  Nel caso di attributi categorici per cui esiste una gerarchia di generalizzazione, è pari al rapporto fra il numero di valori che possono essere generalizzati a tale

valore (numero di nodi foglia discendenti da tale valore) e il numero totale di valori dell'attributo.

Formalmente, la NCP di una tupla generalizzata $\hat{t}$ per l'attributo a in questi due casi è:

$$\mathrm{NCP}_a(\hat{t}) = \begin{cases} \frac{v_{max} - v_{min}}{Range(a)} & \text{se } a \text{ è numerico} \\ \frac{|Ind(\hat{v})|}{|Dom(a)|} & \text{se } a \text{ è categorico} \end{cases} \tag{3.3}$$

Ad esempio, considerando l'attributo numerico `Age` e l'intervallo $[25, 30]$ si ha

$$\mathrm{NCP}_{\texttt{Age}}(\hat{t}) = \frac{30 - 25}{50 - 25} = \frac{5}{25} = 0.2$$

Invece, considerando l'attributo categorico `Country` e il valore generalizzato *Europe* si ha

$$\mathrm{NCP}_{\texttt{Country}}(\hat{t}) = \frac{|\{Italy, France, Spain\}|}{|Dom(\texttt{Country})|} = \frac{3}{9} = 0.33$$

Nel caso di attributi generalizzati con prefissi comuni, la penalità è pari al rapporto tra la larghezza del dominio rappresentato dal valore generalizzato e il numero totale di valori. Questo valore dipende molto dalla tipologia di simbolo che viene sostituito con un asterisco. Ad esempio, il valore 120** rappresenta potenzialmente 100 valori, perché ogni asterisco corrisponde ad un numero da 0 a 9, quindi con dominio di un singolo carattere di larghezza 10. D'altra parte, se l'asterisco sostituisce una lettera, allora il dominio del carattere è di grandezza 26.

Nell'ultimo caso di attributi generalizzati in un insieme, la penalità ha una forma molto simile a quella degli attributi categorici con gerarchia di generalizzazione, in quanto si fa il rapporto tra il numero di valori contenuti nell'insieme generalizzato e il numero di valori totali.

Definito con $\hat{D}$ il dataset anonimizzato e con QI l'insieme degli attributi quasi-identificatori, la Normalized Certainty Penalty del dataset $\hat{D}$ viene calcolata sommando le penalità di tutti gli attributi quasi-identificatori per tutte le tuple appartenenti a $\hat{D}$:

$$\mathrm{NCP}(\hat{D}) = \sum_{\hat{t} \in \hat{D}} \sum_{a \in \mathrm{QI}} \mathrm{NCP}_a(\hat{t}) \tag{3.4}$$

I valori delle metriche calcolate dai singoli worker vengono ricevuti dal Coordinatore, che determina la perdita di informazione complessiva semplicemente sommando i valori ottenuti.

La scelta di queste due metriche sta nel fatto che offrono un giudizio completo sulla perdita di informazione: la Discernability Penalty considera solamente la dimensione della classe di equivalenza, indipendentemente dall'ammontare di generalizzazione applicato; viceversa, la Normalized Certainty Penalty quantifica l'ammontare di generalizzazione, a prescindere dal numero di tuple nella classe di equivalenza.

3.3 Implementazione

In questa sezione vengono illustrati le tecnologie utilizzate, il design architetturale e le modalità di deployment dell'applicazione. L'implementazione del framework Mondrian è disponibile al link: `https://github.com/mosaicrown/mondrian`.

3.3.1 Tecnologie utilizzate

HDFS

HDFS [12], acronimo di Hadoop Distributed File System, è un file system distribuito progettato per l'archiviazione e l'elaborazione di grandi volumi di dati su cluster di computer. È uno dei componenti fondamentali del framework Apache Hadoop e offre una piattaforma affidabile e scalabile per la memorizzazione e l'accesso efficiente ai dati distribuiti.

I dati sono suddivisi in blocchi di dimensioni fisse e replicati su più nodi per garantire affidabilità e alta disponibilità. HDFS utilizza un'architettura master-slave, dove il NameNode è il responsabile della gestione del namespace e del tracciamento dei blocchi di dati, mentre i DataNode sono responsabili dello storage e della gestione dei blocchi di dati stessi.

Una delle caratteristiche principali di HDFS è la scalabilità orizzontale, che consente di aumentare la capacità di archiviazione e la velocità di elaborazione dei dati semplicemente aggiungendo nuovi nodi al cluster.

Apache Spark

Apache Spark [13] è un motore open-source progettato per l'elaborazione distribuita e ad alte prestazioni di grandi volumi di dati. È stato concepito per offrire la velocità di elaborazione, la scalabilità e la programmabilità richieste per i Big Data.

Una delle caratteristiche distintive di Apache Spark è la sua architettura in memoria, che consente di mantenere i dati in memoria centrale per l'elaborazione, riducendo così i tempi di accesso e migliorando le prestazioni complessive delle applicazioni. Il motore di analisi distribuita su vasta scala di Spark può eseguire carichi di lavoro da 10 fino a 100 volte più rapidamente rispetto alle altre alternative, come ad esempio Apache Hadoop che utilizza il paradigma MapReduce.

Spark gestisce i dati in particolari strutture chiamate Resilient Distributed Dataset (RDD). Gli RDD sono raccolte di elementi con tolleranza agli errori che possono essere distribuiti tra più nodi in un cluster e su cui è possibile lavorare in parallelo. Ciò significa che se un nodo del cluster subisce un guasto, le attività di Spark vengono rielaborate in modo che le operazioni possano continuare senza alcun intervento. Spark supporta due tipi principali di operazioni sugli RDD: le trasformazioni, che modificano RDD esistenti per creare nuovi RDD, e le azioni, che eseguono calcoli sugli RDD e restituiscono dei risultati.

Per ottimizzare l'esecuzione delle operazioni distribuite sugli RDD, Spark adotta una strategia di lazy evaluation per cui le trasformazioni non vengono eseguite immediatamente quando vengono chiamate, ma vengono memorizzate in dei grafi aciclici diretti chiamati DAG (Directed Acyclic Graph). In questi grafi, i vertici corrispondono agli RDD mentre gli archi rappresentano le trasformazioni. L'esecuzione avviene solo quando un'azione che richiede l'elaborazione dei dati viene richiamata, come ad esempio collect(), count(), show(), ecc. In quell'istante, il DAG Scheduler suddivide il grafo in stage, ossia in gruppi di trasformazioni ordinate, che poi ven-

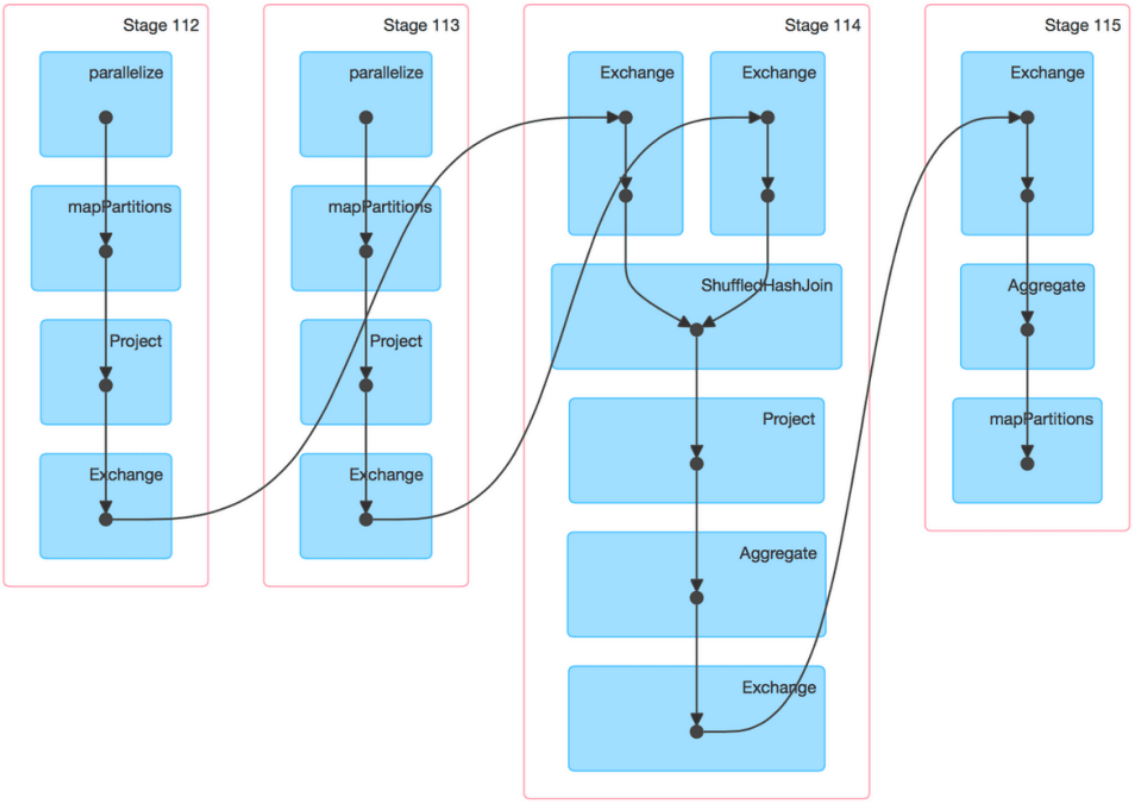


Figura 3.7: Esempio di DAG di trasformazioni in Apache Spark [14]

gono distribuite ed eseguite in parallelo. In Figura 3.7 è mostrato un esempio di trasformazioni pianificate da Apache Spark rappresentate in un DAG.

La lazy evaluation consente ad Apache Spark di pianificare in modo efficiente l'esecuzione delle trasformazioni, riducendo il numero di operazioni di I/O e di shuffle necessarie per completare il lavoro, l'utilizzo di memoria e il tempo complessivo di esecuzione. Inoltre, facilita l'integrazione con sistemi di storage esterni, come HDFS, ottimizzando il trasferimento dei dati.

Apache Spark fornisce un'interfaccia unificata per la programmazione di applicazioni attraverso un set di API in diversi linguaggi, tra cui Scala, Java, Python e R.

Tra i principali componenti di Spark vi sono: Spark Core, che fornisce le funzionalità di base per l'elaborazione distribuita; Spark SQL, per l'elaborazione di dati strutturati; Spark Streaming, per l'elaborazione di dati in tempo reale; MLlib, per il machine learning distribuito; GraphX, per l'elaborazione di grafi.

Grazie alla sua velocità, versatilità e facilità d'uso, Spark viene ampiamente

adottato in una vasta gamma di settori e applicazioni, dalla data science all'analisi dei big data e molto altro ancora.

Python

Per il codice del framework Mondrian, si è scelto di utilizzare Python, che è risaputo essere un linguaggio adatto per l'elaborazione dati. Inoltre, consente di poter sfruttare Pandas [15], una libreria molto utile per la manipolazione di grandi volumi di dati.

Le API di integrazione tra Python e Apache Spark prendono il nome di PySpark. PySpark combina l'apprendimento e la facilità d'uso di Python con la potenza di Apache Spark.

Con PySpark è possibile utilizzare una struttura dati chiamata DataFrame, definita da Apache Spark in alternativa ai classici RDD, la cui implementazione è basata sempre sugli RDD, ma che può processare dati strutturati o semi-strutturati e gestire lo schema dei dati. Concretamente, i DataFrame sono collezioni di dati distribuiti organizzati in formato tabulare con righe e colonne, quindi molto più simili ad un database relazionale.

Docker

Docker [16] è una piattaforma open-source che consente di creare, distribuire e gestire applicazioni in contenitori leggeri e portabili, chiamati container. I container sono componenti eseguibili standardizzati che combinano il codice sorgente delle applicazioni con le librerie e le dipendenze del sistema operativo necessarie per l'esecuzione.

Uno dei vantaggi di Docker è la sua capacità di isolare le applicazioni e le loro dipendenze in ambienti containerizzati, garantendo che le risorse siano completamente separate da altri contenitori sullo stesso host. Questo offre un'ottima flessibilità e portabilità, consentendo agli sviluppatori di creare, testare e distribuire le proprie applicazioni in modo rapido e affidabile.

Docker utilizza un approccio basato su immagini per la creazione di contenitori. Un'immagine Docker è un pacchetto leggero e autonomo che include tutto il neces-

sario per eseguire un'applicazione: il codice sorgente, le librerie, le dipendenze e le configurazioni dell'ambiente. La creazione di un'immagine Docker avviene a partire da un file di testo, chiamato Dockerfile e senza estensione, che contiene un elenco di istruzioni su come creare l'immagine. Gli sviluppatori possono creare e condividere facilmente le proprie immagini Docker tramite Docker Hub, un registro pubblico di immagini Docker, o utilizzando registri privati.

Inoltre, Docker offre un'ampia gamma di strumenti e funzionalità per gestire il ciclo di vita dei contenitori, tra cui Docker Compose per la gestione di applicazioni multi-container, Docker Swarm per l'orchestrazione di cluster di contenitori e Docker Desktop per lo sviluppo e il testing locali.

Il vantaggio di Docker nel contesto di questo progetto è quello di semplificare la configurazione di nodi isolati e indipendenti sui quali distribuire e scalare l'esecuzione dell'applicazione. Inoltre, l'utilizzo di container consente la realizzazione di un sistema altamente portabile ed eseguibile in diversi ambienti, fisici, virtuali e cloud.

3.3.2 Architettura

L'architettura del sistema consiste in un cluster Apache Spark, i cui nodi sono responsabili dell'esecuzione delle tre fasi del processo di anonimizzazione, e una piattaforma di storage, che può essere centralizzato o distribuito. In Figura 3.8 vengono rappresentati i componenti dell'architettura e l'esecuzione del framework, con i vari trasferimenti di dataset, frammenti e condizioni.

Il cluster Apache Spark è caratterizzato da uno *Spark Cluster Manager*, che rappresenta il responsabile del cluster e si occupa di gestire i nodi e assegnare le risorse, e un insieme di *Spark Workers*, che costituiscono i nodi del cluster che eseguono i task assegnati dal Cluster Manager.

Da un punto di vista applicativo, lo *Spark Driver* svolge il ruolo di Coordinatore. Tra i suoi compiti, vi è quello di invocare lo *Spark Context*, che rappresenta il collegamento tra l'applicazione e il cluster, e di tradurre il codice da eseguire in jobs, i quali sono divisi ulteriormente in più piccole unità di esecuzione, chiamati task, che verranno eseguite dai workers.

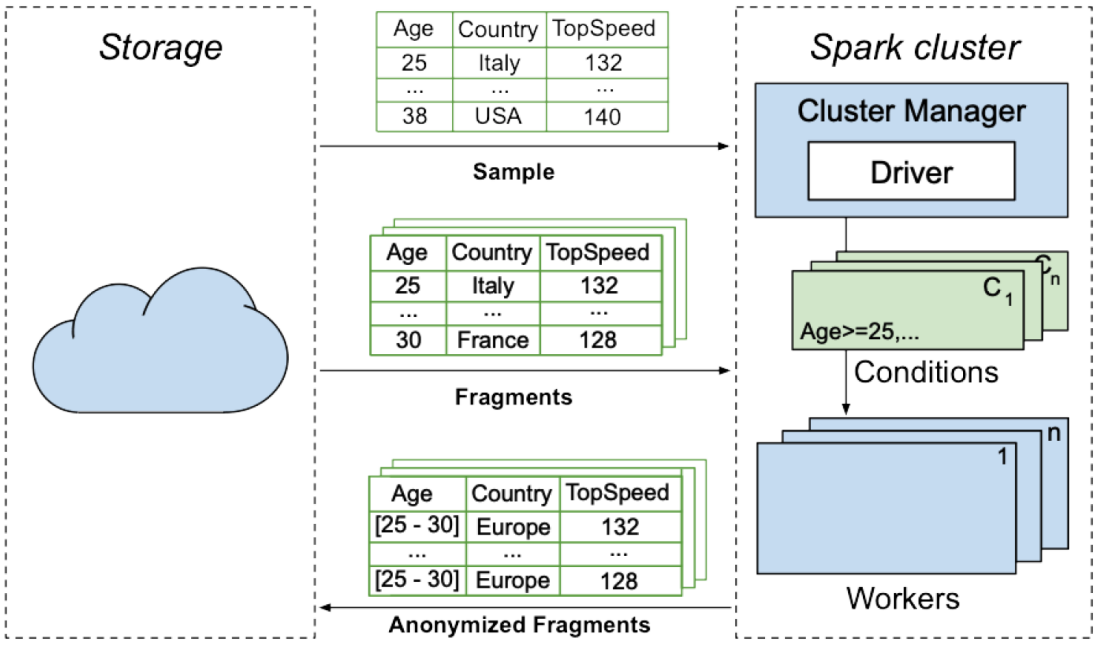


Figura 3.8: Architettura del sistema di anonimizzazione distribuita basato su Spark [11]

Nel contesto del framework Mondrian, lo *Spark Driver* inizialmente si occupa della fase di pre-processamento dei dati. Per cui memorizza un campione del dataset, esegue il partizionamento dei frammenti secondo la strategia stabilita (monodimensionale basata sui quantili o multidimensionale), e genera le condizioni di frammentazione. Successivamente, definisce una serie di *Spark Task*, ognuno dei quali corrisponde all'anonimizzazione di un singolo frammento e include le condizioni che descrivono tale frammento, e le comunica ai workers disponibili. Ciascun *Spark Worker*, poi, procede con l'esecuzione del proprio task: recupera il frammento direttamente dalla piattaforma di storage, applica l'algoritmo distribuito di anonimizzazione, calcola la perdita di informazione dovuta alla generalizzazione e memorizza i risultati tramite i servizi offerti dallo *Spark Driver*. Infine, lo *Spark Driver* combina le perdite di informazione parziali determinando la perdita di informazione globale dell'intero dataset anonimizzato.

3.3.3 Deployment

Con l'ottica di ottenere una soluzione facilmente implementabile in un'infrastruttura cloud, è stato scelto di realizzare un'applicazione multi-container, utilizzando Docker come strumento di gestione dei container.

In particolare, l'architettura in Figura 3.8 è stata sviluppata con:

- Un Docker container per lo *Spark Driver*;
- Un Docker container per lo *Spark Cluster Manager*;
- Un numero variabile di Docker container per gli *Spark Workers*;
- Un Docker container per l'esposizione dello *Spark History Server*, un componente che tiene traccia di informazioni addizionali riguardanti la schedulazione e l'assegnamento dei task.

Per la creazione dei containers è stato utilizzato Docker Compose, mentre per la distribuzione dei container nei nodi dello Spark cluster è stato adottato Docker Swarm, una soluzione di orchestrazione integrata in Docker di semplice utilizzo.

In Figura 3.9, viene mostrato un esempio di distribuzione dei Docker container all'interno di uno Spark cluster operata da Docker Swarm, dove i riquadri bianchi rappresentano i nodi dello Spark cluster mentre i riquadri blu rappresentano i Docker container. Si noti che ogni nodo dello Spark cluster può ospitare più di un container Docker.

Un nodo appartenente al cluster assume il ruolo di *Docker Swarm Manager* (in Figura 3.9 il riquadro bianco tratteggiato), il quale coordina e distribuisce il carico di lavoro ai *Docker Swarm Workers*. Uno dei *Docker Swarm Workers* è dedicato ad ospitare un container per lo *Spark Driver* e uno per lo *Spark Cluster Managers*, mentre i restanti sono disponibili per la creazione di container per gli *Spark Workers*.

3.4 Risultati sperimentali

Per valutare l'applicabilità e la scalabilità del framework sono stati eseguiti una serie di test esaustivi ed effettuati confronti con la versione centralizzata dell'algoritmo

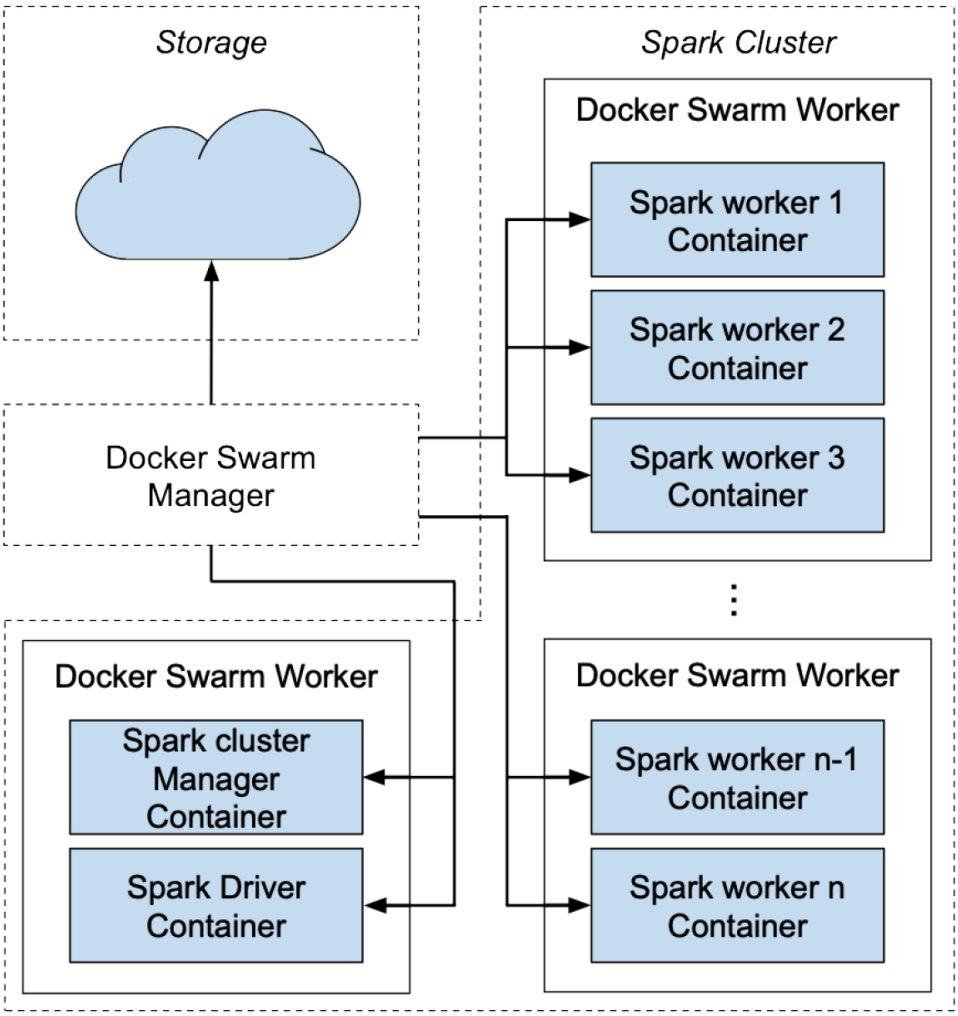


Figura 3.9: Distribuzione di Docker container in un cluster Spark [11]

Mondrian.

3.4.1 Specifiche di test

Per ottenere risultati affidabili, è stato simulato un ambiente cloud tramite l'utilizzo di Docker Compose. La macchina utilizzata è dotata di un processore AMD Ryzen 3900X (12 cores fisici, 24 cores logici), 64GB RAM e 2TB SSD, con sistema operativo Ubuntu 20.04 LTS, Apache Spark 3.0.1, Hadoop 3.2.1 e Pandas 1.1.3. A ciascun worker è stato assegnato 2GB di RAM e 1 CPU core. Per la versione centralizzata, invece, è stato assegnato 1 CPU core e nessun limite di utilizzo di RAM.

Come piattaforma di storage distribuito è stato scelto di utilizzare *Hadoop Distri-*

buted File System (HDFS). Il deployment del cluster HDFS prevede l'utilizzo di un Docker container per il Namenode Hadoop (responsabile della gestione del cluster) e molteplici container per i Datanodes Hadoop (responsabili della memorizzazione dei dati e delle operazioni di lettura e scrittura).

Per quanto riguarda la scelta del dataset di test, avendo la necessità di testare il framework su un volume elevato di dati, è stato considerato il dataset Poker Hand [17], composto da 1.000.000 di tuple. Ciascuna tupla rappresenta una mano di poker, in cui le carte sono descritte da due attributi: il seme (intero compreso tra 1 e 4) e il numero (intero compreso tra 1 e 13). Le carte rappresentano gli attributi quasi-identificatori, mentre l'attributo sensibile è un attributo addizionale che caratterizza la classe della mano con un numero intero compreso tra 0 e 9.

3.4.2 Risultati

Lo scopo dei test è quello di analizzare i tempi di esecuzione e le perdite di informazione per la valutazione della scalabilità e della qualità della soluzione proposta. I valori ottenuti corrispondono alla media di cinque ripetizioni, e vengono confrontati con i risultati dell'algoritmo Mondrian centralizzato.

Tempi di esecuzione In Figura 3.10 sono mostrati i tempi di esecuzione risultati dai test effettuati. Per il framework Mondrian, in particolare, sono stati considerati per il partizionamento sia il metodo basato sui quantili sia il metodo multidimensionale. Inoltre, il numero di worker è stato fatto variare da 2 a 12, il parametro k nell'insieme $\{5, 10, 20\}$ e il parametro l nell'insieme $\{2, 3, 4\}$.

Come previsto, i tempi di esecuzione diminuiscono all'aumentare del numero di workers, con percentuali di miglioramento rispetto all'algoritmo centralizzato che vanno dal 28% al 98%. Ciò conferma la scalabilità del sistema.

Dai grafici si nota che i due metodi di partizionamento non comportano differenze significative. Nello specifico, quando il numero di workers è pari a 2, i tempi di esecuzione si equivalgono, poiché i frammenti generati sono identici. All'aumentare del numero di workers, invece, si osservano comportamenti diversi: mentre l'approccio basato sui quantili determina una diminuzione dei tempi ad ogni worker

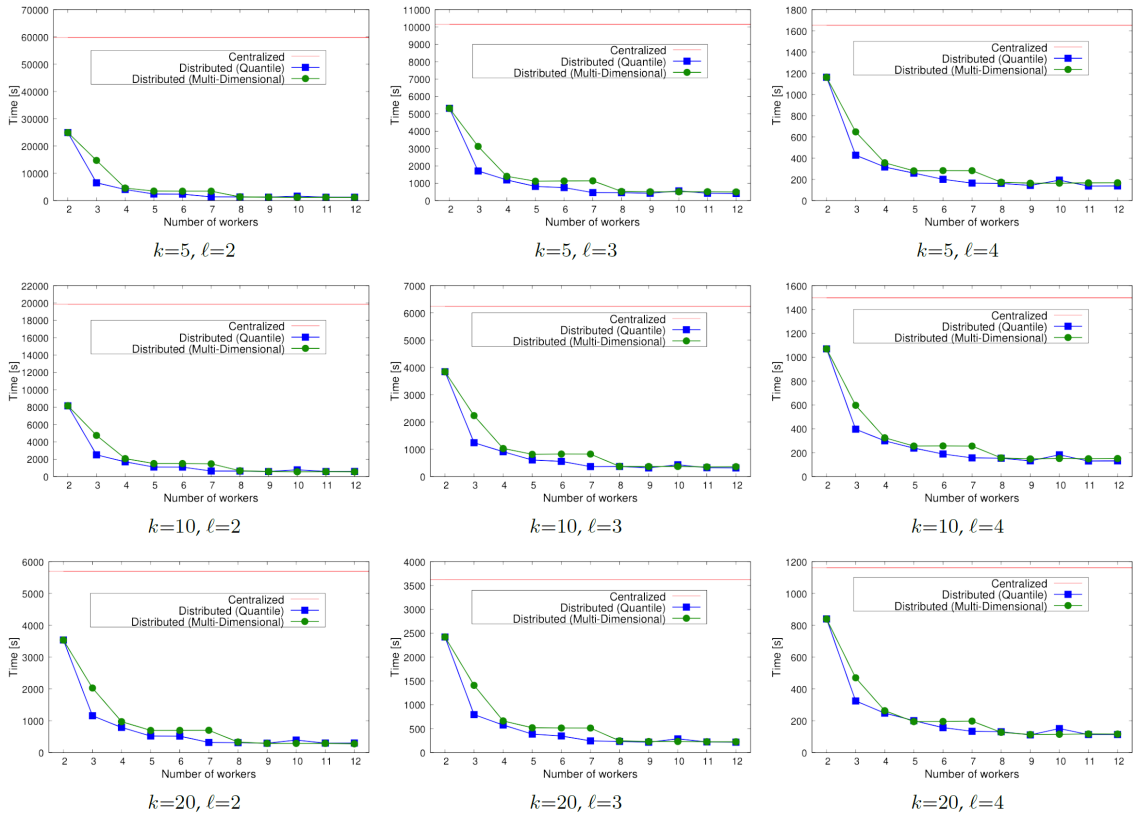


Figura 3.10: Tempi di esecuzione dell'algoritmo Mondrian centralizzato e distribuito al variare del numero di workers e dei parametri di anonimizzazione [11]

aggiuntivo, l'approccio multidimensionale comporta un significativo miglioramento quando il numero di worker diventa una potenza di 2. Infatti, utilizzando da 4 a 7 workers, i tempi rimangono quasi costanti, in quanto alcuni workers hanno in carico due frammenti, risultando dei colli di bottiglia. Al contrario, il passaggio da 7 a 8 workers evidenza una maggiore riduzione dei tempi.

Perdita di informazione In Figura 3.11 viene mostrata la perdita di informazione causata dall'anonimizzazione, in termini di Discernability Penalty e Normalized Certainty Penalty. I parametri che impattano sulla qualità dei dati sono il numero di workers e la dimensione del dataset campionato utilizzato per la fase di frammentazione. In particolare, sono stati valutati un numero di workers pari a 5 e 10, e percentuali di campionamento pari a $0,1\%$, $0,01\%$ e $0,001\%$.

I risultati evidenziano che, per tutti i valori di k $(5, 10, 20)$ e di l (2) considerati, sia il numero di workers che la percentuale di campionamento hanno un effetto tra-

Sampling	Partitioning	Information Loss (DP)		Information Loss (NCP)	
		5 workers	10 workers	5 workers	10 workers
0.1%	Quantile	$7.14e06 \pm 5.13e02$	$7.10e06 \pm 8.31e03$	$1.83e06 \pm 3.18e02$	$1.97e06 \pm 3.99e02$
	Multi-dimensional	$7.22e06 \pm 1.63e04$	$7.22e06 \pm 9.25e03$	$1.80e06 \pm 2.46e03$	$1.80e06 \pm 2.15e03$
0.01%	Quantile	$7.14e06 \pm 2.78e04$	$7.10e06 \pm 9.35e03$	$1.83e06 \pm 3.41e03$	$1.96e06 \pm 1.01e04$
	Multi-dimensional	$7.17e06 \pm 1.93e04$	$7.15e06 \pm 1.36e04$	$1.79e06 \pm 4.02e03$	$1.80e06 \pm 7.44e03$
0.001%	Quantile	$7.14e06 \pm 2.78e04$	$7.13e06 \pm 1.27e04$	$1.83e06 \pm 3.41e03$	$1.91e06 \pm 7.92e02$
	Multi-dimensional	$7.20e06 \pm 5.02e04$	$7.20e06 \pm 5.02e04$	$1.80e06 \pm 3.82e03$	$1.80e06 \pm 3.82e03$
Centralized		$7.23e06$		$1.50e06$	

(a) k=5, ℓ=2

Sampling	Partitioning	Information Loss (DP)		Information Loss (NCP)	
		5 workers	10 workers	5 workers	10 workers
0.1%	Quantile	$1.42e07 \pm 7.44e03$	$1.41e07 \pm 9.94e03$	$2.20e06 \pm 1.38e03$	$2.34e06 \pm 1.14e03$
	Multi-dimensional	$1.42e07 \pm 2.39e04$	$1.43e07 \pm 1.47e04$	$2.17e06 \pm 6.50e02$	$2.17e06 \pm 8.62e02$
0.01%	Quantile	$1.42e07 \pm 2.16e04$	$1.41e07 \pm 7.92e03$	$2.20e06 \pm 1.67e03$	$2.34e06 \pm 1.10e04$
	Multi-dimensional	$1.42e07 \pm 2.35e04$	$1.42e07 \pm 1.71e04$	$2.17e06 \pm 4.90e03$	$2.17e06 \pm 4.29e03$
0.001%	Quantile	$1.42e07 \pm 2.16e04$	$1.41e07 \pm 2.95e04$	$2.20e06 \pm 1.67e03$	$2.24e06 \pm 8.00e04$
	Multi-dimensional	$1.43e07 \pm 2.36e04$	$1.47e07 \pm 2.36e04$	$2.17e06 \pm 4.67e03$	$2.17e06 \pm 4.51e03$
Centralized		$1.43e07$		$1.82e06$	

(b) k=10, ℓ=2

Sampling	Partitioning	Information Loss (DP)		Information Loss (NCP)	
		5 workers	10 workers	5 workers	10 workers
0.1%	Quantile	$2.88e07 \pm 2.82e04$	$2.86e07 \pm 2.06e04$	$2.50e06 \pm 3.30e01$	$2.66e06 \pm 1.55e03$
	Multi-dimensional	$2.88e07 \pm 4.44e04$	$2.88e07 \pm 5.77e04$	$2.47e06 \pm 1.56e03$	$2.46e06 \pm 4.32e03$
0.01%	Quantile	$2.88e07 \pm 6.73e04$	$2.85e07 \pm 4.21e04$	$2.50e06 \pm 1.32e03$	$2.65e06 \pm 1.76e04$
	Multi-dimensional	$2.88e07 \pm 5.03e04$	$2.88e07 \pm 1.86e04$	$2.47e06 \pm 5.75e03$	$2.46e06 \pm 4.00e03$
0.001%	Quantile	$2.88e07 \pm 6.73e04$	$2.86e07 \pm 2.86e04$	$2.50e06 \pm 1.32e03$	$2.63e06 \pm 3.43e03$
	Multi-dimensional	$2.88e07 \pm 7.23e04$	$2.88e07 \pm 6.83e04$	$2.47e06 \pm 8.32e03$	$2.46e06 \pm 6.17e03$
Centralized		$2.88e07$		$2.07e06$	

(c) k=20, ℓ=2

Figura 3.11: Perdita di informazione al variare della percentuale di campionamento, del numero di workers e dei parametri di anonimizzazione [11]

scurabile sulla perdita di informazione. Più dettagliatamente, all'aumentare del numero di workers, l'approccio multidimensionale mantiene performance simili, mentre l'approccio basato sui quantili produce una maggiore perdita di informazione. Inoltre, i valori della Discernability Penalty mostrano che l'approccio multidimensionale tende a generare classi di equivalenza simili a quelle del sistema centralizzato, mentre l'approccio monodimensionale basato sui quantili determina classi di equivalenza leggermente più piccole, specialmente quando il campione usato per il partizionamento è piccolo. Ciò significa che il campionamento del dataset ha un impatto sulla dimensione delle classi di equivalenza. Tuttavia, classi di equivalenza più piccole non implicano minore generalizzazione. Infatti, i valori della Normalized Certainty

Penalty risultano simili per tutti e tre i metodi.

In conclusione, i dati sperimentali confermano che la versione distribuita dell'algoritmo Mondrian garantisce scalabilità, limitando l'impatto sulla qualità dei dati anonimizzati. Quando il numero di workers non è una potenza di 2, la frammentazione basata sui quantili consente di ottenere tempi di esecuzione leggermente migliori rispetto al metodo multidimensionale. La perdita di informazione risulta simile nei due scenari, con l'approccio multidimensionale che ottiene valori più alti per la DP (classi di equivalenza più grandi) e più bassi per la NCP (minore generalizzazione) rispetto all'approccio basato sui quantili.

Capitolo 4

Analisi e progettazione del porting

Questo capitolo è incentrato sulla valutazione degli strumenti e delle tecnologie scelti per l'implementazione del porting del framework Mondrian in ambiente distribuito, nell'ottica di superare i limiti della precedente architettura e introdurre nuove funzionalità. I criteri principali considerati per la selezione consistono nella capacità di fornire funzionalità avanzate, combinando l'efficienza nella gestione delle risorse e la semplicità di utilizzo nella fase di sviluppo, e la possibilità di realizzare un sistema altamente portabile e scalabile, in grado di gestire carichi di lavoro di entità diversa e adattarsi a differenti ambienti.

4.1 Limitazioni del framework Mondrian

Il framework Mondrian realizzato dall'UniBG Seclab propone un approccio efficiente ed efficace per l'esecuzione dell'anonimizzazione distribuita di grandi volumi di dati. I risultati sperimentali presentati nella Sezione 3.4.2 mostrano, infatti, performance significativamente superiori rispetto alla versione centralizzata dell'algoritmo Mondrian, mantenendo la qualità dei dati anonimizzati pressoché inalterata.

Tuttavia, si possono osservare alcuni spunti di miglioramento. L'architettura attuale basata sull'utilizzo di Docker Compose permette la gestione di servizi container su un singolo host, limitando l'utilità della distribuzione dell'applicazione. Inoltre, non fornisce funzionalità avanzate per la scalabilità, l'alta disponibilità, il bilanciamento dei carichi tra i nodi, e la gestione delle risorse. L'utilizzo di Docker

Swarm come orchestratore di container cerca di superare questi limiti, consentendo la distribuzione su un cluster di dispositivi. Ciononostante, Swarm offre funzionalità di orchestrazione di base, risultando non adatto per la gestione di cluster di grandi dimensioni o con requisiti particolarmente complessi. Inoltre, ha opzioni più limitate per l'integrazione con ambienti cloud pubblici.

4.2 Scelta delle tecnologie

In fase di progettazione del porting, sono state analizzate diverse alternative per superare i limiti e migliorare il framework Mondrian.

Per sostituire Docker Swarm nel ruolo di orchestratore di container è stato scelto Kubernetes, che offre maggiore flessibilità nell'orchestrazione, maggiore scalabilità orizzontale e una serie di funzionalità avanzate per la gestione delle risorse, il bilanciamento dinamico dei carichi di lavoro, il monitoraggio delle prestazioni, la gestione degli aggiornamenti e dei failover. Inoltre, vi è la possibilità di integrare i servizi di Apache Spark in Kubernetes ricorrendo all'utilizzo dello Spark operator, il quale semplifica e automatizza la gestione delle applicazioni Spark.

Per quanto riguarda la piattaforma di storage distribuito, Kubernetes non supporta nativamente HDFS. Perciò, è stato deciso di ricorrere a Rook, un orchestratore di storage distribuito che consente di integrare in Kubernetes un sistema di storage altamente affidabile e scalabile come Ceph, in maniera semplice e intuitiva.

Nelle prossime sezioni verranno approfonditi singolarmente gli strumenti selezionati per implementare la migrazione del framework Mondrian.

4.2.1 Kubernetes

Kubernetes [18] è un sistema open source per l'orchestrazione di container, originariamente sviluppato da Google e ora gestito dalla Cloud Native Computing Foundation (CNCF) [19]. Kubernetes si occupa di automatizzare le operazioni di gestione dei container, assicurando la continuità dei servizi e l'allocazione efficiente delle risorse.

I principali vantaggi dell'utilizzo di Kubernetes sono i seguenti:

- **Bilanciamento del carico (scalabilità)**: consente di scalare orizzontalmente le applicazioni in modo dinamico in base al carico di lavoro, aumentando o riducendo il numero di istanze dei container in esecuzione per soddisfare le esigenze di traffico dell'applicazione;

- **Elevata disponibilità**: garantisce che l'applicazione sia sempre accessibile dagli utenti, azzerando il downtime. Per fare questo, monitora costantemente i container in esecuzione e provvede automaticamente al ripristino in caso di fallimenti;

- **Portabilità**: offre una piattaforma robusta per la distribuzione di applicazioni containerizzate in diversi ambienti (locali, cloud e ibridi). Ciò consente di migrare facilmente tra ambienti di sviluppo, test e produzione;

- **Allocazione dinamica delle risorse**: fornisce automaticamente ad ogni container un livello adeguato e predefinito di risorse computazionali e memoria;

- **Rollout e rollback automatizzati**: può creare e rimuovere i container riequilibrando le risorse assegnate per raggiungere lo stato desiderato;

- **Servizi aggiuntivi**: dispone di un vasto ecosistema di strumenti, servizi e plug-in che estendono le funzionalità di base. Ad esempio, strumenti per il monitoraggio, la registrazione dei log, la sicurezza, la gestione delle reti, e altro ancora.

Tutti questi vantaggi consentono di ridurre il carico operativo per gli amministratori di sistema e gli sviluppatori.

L'architettura di Kubernetes consiste in un cluster di nodi, fisici o virtuali, suddivisi in due categorie:

1. **Master Node**: chiamato anche *control plane*, è il nodo responsabile della gestione del cluster Kubernetes. Ogni cluster deve averne almeno uno. Include diversi componenti chiave:

    - **API Server**: espone le Kubernetes API che permettono agli utenti di interagire con il cluster e ai componenti di comunicare tra loro;

- **Scheduler**: decide su quale nodo eseguire le nuove istanze dei container, in relazione a diversi fattori (richiesta di risorse, vincoli hardware e software, regole di affinità e anti-affinità, disponibilità di dati/volumi, interferenze nei carichi di lavoro, scadenze);
- **Controller Manager**: gestisce i controller che regolano lo stato del cluster. Ad esempio, il Node Controller, responsabile del monitoraggio dei nodi, e il Replication Controller, responsabile per il mantenimento del corretto numero di istanze di esecuzione presenti rispetto a quelle richieste;
- **etcd**: un database distribuito utilizzato per memorizzare lo stato del cluster.

2. **Worker Node**: nodi su cui vengono eseguiti i container. Ogni worker node include i seguenti componenti:
   - **Kubelet**: un agente che comunica con il master node e gestisce i container sul nodo;
   - **Kube-proxy**: un proxy di rete che gestisce il traffico ai servizi all'interno del cluster. Implementa le regole di instradamento e fornisce bilanciamento del carico;
   - **Container Runtime**: software responsabile dell'esecuzione dei container (ad esempio Docker).

In Figura 4.1 viene mostrata l'architettura di Kubernetes in un cluster d'esempio composto da un control plane e tre nodi workers.

Entrando nel dettaglio applicativo, i componenti principali utilizzati per la configurazione del cluster e l'esecuzione delle applicazioni sono i seguenti:

- **Pod**: è l'unità di base di Kubernetes e rappresenta un gruppo di uno o più container che condividono lo stesso contesto di rete e di archiviazione;
- **Deployment**: definisce lo stato desiderato di un'applicazione e gestisce il processo di distribuzione e aggiornamento dei pod;

Figura 4.1: Architettura di Kubernetes [20]

- **Service**: definisce un set di pod e una politica di accesso a essi. Fornisce un'interfaccia stabile per accedere ai servizi all'interno del cluster, indipendentemente dalle istanze dei pod sottostanti;
- **Namespace**: fornisce un'area virtuale all'interno di un cluster Kubernetes. È utile per organizzare e isolare le risorse di più utenti o applicazioni all'interno di un unico cluster;
- **Ingress**: gestisce l'accesso esterno alle applicazioni all'interno del cluster;
- **ConfigMap e Secret**: contengono la configurazione e le informazioni sensibili utilizzate dalle applicazioni;

- **PersistentVolume e PersistentVolumeClaim**: componenti per la gestione dello storage persistente per le applicazioni;
- **StatefulSet**: gestisce le applicazioni con stato, garantendo la persistenza e l'identità dei dati.

Tutti questi componenti vengono definiti all'interno di file in formato YAML o JSON, chiamati manifesti Kubernetes.

Strumenti per l'utilizzo di Kubernetes

Per interagire con i cluster Kubernetes si utilizza la linea di comando *kubectl*. Le sue funzionalità principali sono:

- **Gestione delle risorse**: consente agli utenti di creare, visualizzare, aggiornare e eliminare risorse Kubernetes come pod, deployment, service, ecc;
- **Monitoraggio dello stato del cluster**: fornisce comandi per monitorare lo stato del cluster Kubernetes, inclusi i nodi, i pod, i servizi e le risorse;
- **Debugging e diagnostica**: fornisce strumenti per il debugging e la diagnostica delle applicazioni in esecuzione su Kubernetes, inclusa la visualizzazione dei logs dei container e l'esecuzione di comandi all'interno dei container;
- **Gestione degli utenti e delle autorizzazioni**: consente agli amministratori di gestire gli utenti, i ruoli e le autorizzazioni all'interno del cluster Kubernetes utilizzando risorse come Role, RoleBinding, ClusterRole e ClusterRoleBinding.

Un altro strumento per semplificare la distribuzione e la gestione delle applicazioni su Kubernetes è *Helm* [21]. Esso si basa sulla creazione di pacchetti, chiamati Charts, che definiscono le applicazioni e le loro dipendenze. Un Helm Chart contiene una serie di risorse Kubernetes preconfigurate insieme a un file di configurazione YAML.

Helm include anche una funzionalità di template, che consente di configurare dinamicamente i manifesti Kubernetes e facilitarne l'installazione in base a delle esigenze specifiche.

Inoltre, Helm supporta la distribuzione dei pacchetti attraverso dei registri chiamati Helm repositories. Ciò permette agli utenti di poter installare e usufruire di applicazioni rese disponibili da altri utenti, senza conoscerne i dettagli implementativi e senza dover preoccuparsi della loro gestione.

Kubernetes operator

Kubernetes consente di sviluppare applicazioni containerizzate e distribuite, facendo leva sulla capacità di automatizzarne la gestione. Tuttavia, quando si tratta di applicazioni complesse, le funzionalità base di Kubernetes non sono in grado di sostituire un operatore umano, il quale possiede conoscenze profonde su come l'applicazione è strutturata, come deve comportarsi e come reagire a determinati problemi. Per superare questo limite, è possibile ricorrere all'utilizzo dei Kubernetes operator.

Un operatore Kubernetes è un'applicazione Kubernetes nativa che estende le funzionalità della piattaforma per automatizzare e semplificare il ciclo di vita delle applicazioni complesse. Si basa sull'utilizzo di Custom Resource Definitions (CRD) per ampliare lo schema Kubernetes e definire nuovi tipi di risorse personalizzate. Queste risorse personalizzate rappresentano le applicazioni o i componenti dell'applicazione gestiti dall'operatore.

Gli operatori Kubernetes sono implementati come Controller personalizzati, che monitorano costantemente lo stato delle risorse personalizzate all'interno del cluster e reagiscono agli eventi eseguendo azioni specifiche. In particolare, gestiscono il ciclo di vita delle risorse, garantendo che lo stato desiderato sia mantenuto. Se viene rilevata una discrepanza, l'operatore esegue azioni correttive per riportare le risorse nello stato desiderato.

Gli operatori, quindi, automatizzano molte operazioni di gestione delle applicazioni, tra cui l'installazione, la configurazione, l'aggiornamento e la manutenzione, limitando l'intervento manuale da parte degli amministratori di sistema e degli sviluppatori.

In Figura 4.2 viene rappresentato il ruolo di un operatore Kubernetes.

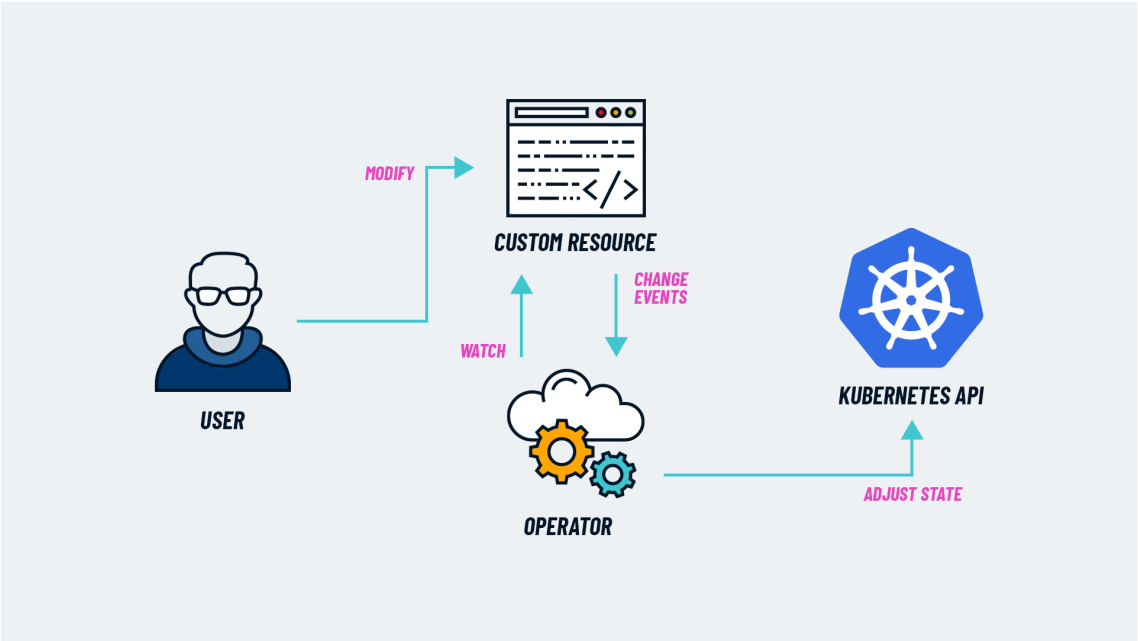


Figura 4.2: Funzionalità di un operatore Kubernetes [22]

4.2.2 Spark operator

Lo Spark operator [23] è uno strumento open-source che estende le funzionalità di Kubernetes per la gestione e l'esecuzione di applicazioni Apache Spark in un cluster Kubernetes. È stato progettato per semplificare il processo di distribuzione, gestione e scalabilità delle applicazioni Spark su Kubernetes, consentendo agli utenti di sfruttare appieno il potenziale di entrambe le tecnologie.

Essendo un operatore Kubernetes, si basa sulla definizione di risorse personalizzate CRD per rappresentare e gestire le componenti Spark, come SparkApplication e SparkCluster. Queste risorse permettono agli utenti di definire le specifiche delle applicazioni Spark e dei cluster Spark all'interno del cluster Kubernetes, come le configurazioni, le dipendenze e le risorse necessarie.

Lo Spark operator gestisce il ciclo di vita completo delle applicazioni Spark, inclusa la creazione, l'aggiornamento e l'eliminazione. Inoltre, consente di automatizzare la scalabilità dinamica delle applicazioni in base al carico di lavoro e la gestione dei guasti, rimuovendo i componenti in errore e sostituendoli con dei nuovi a cui viene assegnato lo stesso lavoro, in modo da garantire la continuità del servizio.

Kubernetes è supportato da Spark in qualità di schedulatore di applicazioni

distribuite a partire dalla versione 2.3 di Spark, in aggiunta ai già esistenti Apache Mesos, Hadoop Yarn e alla modalità standalone di Apache Spark. Tipicamente, le applicazioni Spark vengono eseguite in Kubernetes eseguendo il comando `spark-submit` dal dispositivo client, passando i parametri di configurazione dell'applicazione. Il suo funzionamento è il seguente:

- Spark crea lo Spark driver all'interno di un pod Kubernetes;
- Il driver crea gli esecutori all'interno di altri pod Kubernetes e distribuisce l'esecuzione dei task su di essi;
- Quando l'applicazione è completata, gli Executor pods vengono terminati e rimossi, mentre il Driver pod rimane nello stato "Completed" fino a quando viene manualmente rimosso oppure viene ripulito dal garbage collector di Kubernetes. In questo stato, il Driver pod non consuma risorse computazionali o di memoria, e può essere utilizzato per visualizzare i logs dell'esecuzione.

Figura 4.3 mostra il processo di esecuzione di un'applicazione Spark in Kubernetes.

Sfruttando lo Spark operator, l'esecuzione di un'applicazione Spark in Kubernetes viene semplificata e vengono introdotti diversi componenti per la gestione automatica del suo ciclo di vita. Le specifiche di configurazione di un'applicazione Spark vengono definite all'interno di un oggetto SparkApplication in un manifesto Kubernetes con formato YAML. Quando l'oggetto SparkApplication viene creato tramite la linea di comando kubectl, lo Spark oerator si incarica di inizializzare l'applicazione Spark e le attività di monitoraggio connesse.

L'architettura dello Spark operator è formata da quattro principali elementi:

- **SparkApplication Controller**: è il responsabile degli eventi relativi alla creazione, all'aggiornamento e all'eliminazione delle applicazioni Spark in Kubernetes. Quando l'oggetto SparkApplication viene creato, il Controller invoca il Submission Runner per iniziare l'esecuzione dell'applicazione Spark. Quando l'oggetto SparkApplication viene aggiornato, il Controller verifica se le specifiche dell'applicazione sono cambiate e, in tal caso, ferma l'esecuzione corrente

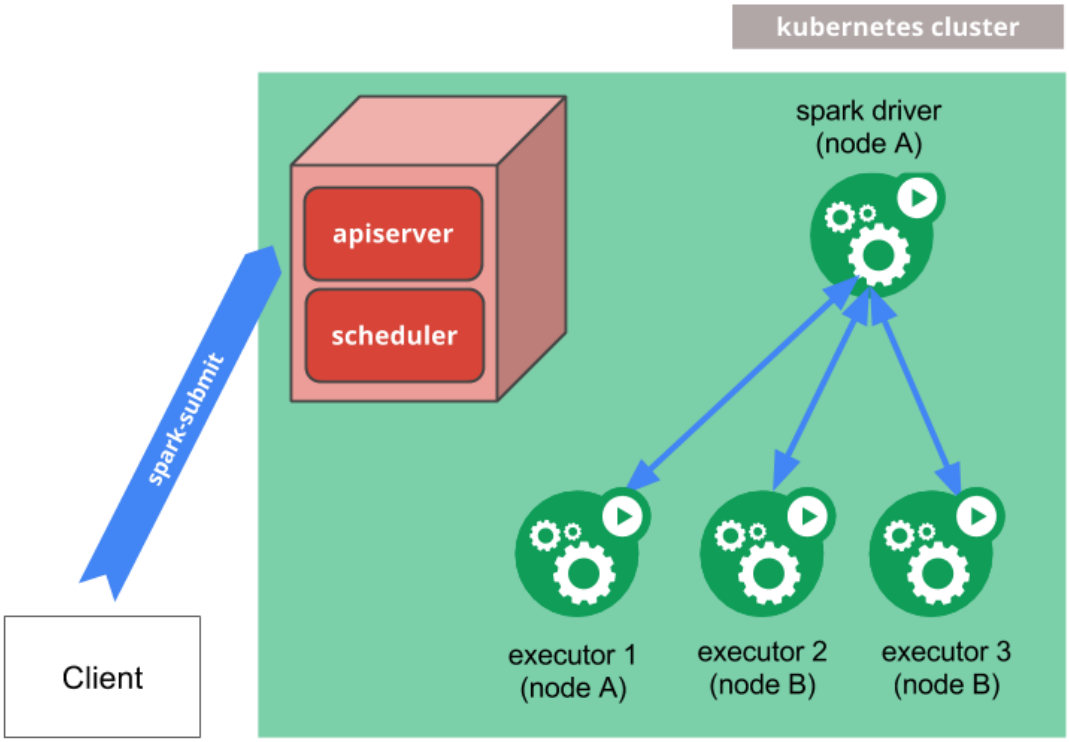


Figura 4.3: Esecuzione di Spark in un cluster Kubernetes [24]

e avvia la nuova versione. Inoltre, il Controller si occupa del monitoraggio delle risorse Spark in collaborazione con lo Spark Pod Monitor, assicurando il corretto funzionamento dell'applicazione e garantendo che lo stato attuale delle risorse corrisponda allo stato desiderato;

- **Submission Runner**: ha il compito di eseguire il comando `spark-submit` con i parametri specificati nell'oggetto SparkApplication. La schedulazione dei Driver pod e degli Executor pods avviene come descritto precedentemente;

- **Spark Pod Monitor**: si occupa del monitoraggio degli Spark pods e comunica gli aggiornamento dello stato dei pods al Controller;

- **Mutating Admission Webhook**: componente opzionale che, quando abilitato, consente di personalizzare gli Spark pods in base alle annotazioni fornite dal Controller. Queste annotazioni sono definite nelle specifiche della SparkApplication, e permettono di sfruttare delle funzionalità di Kubernetes quali

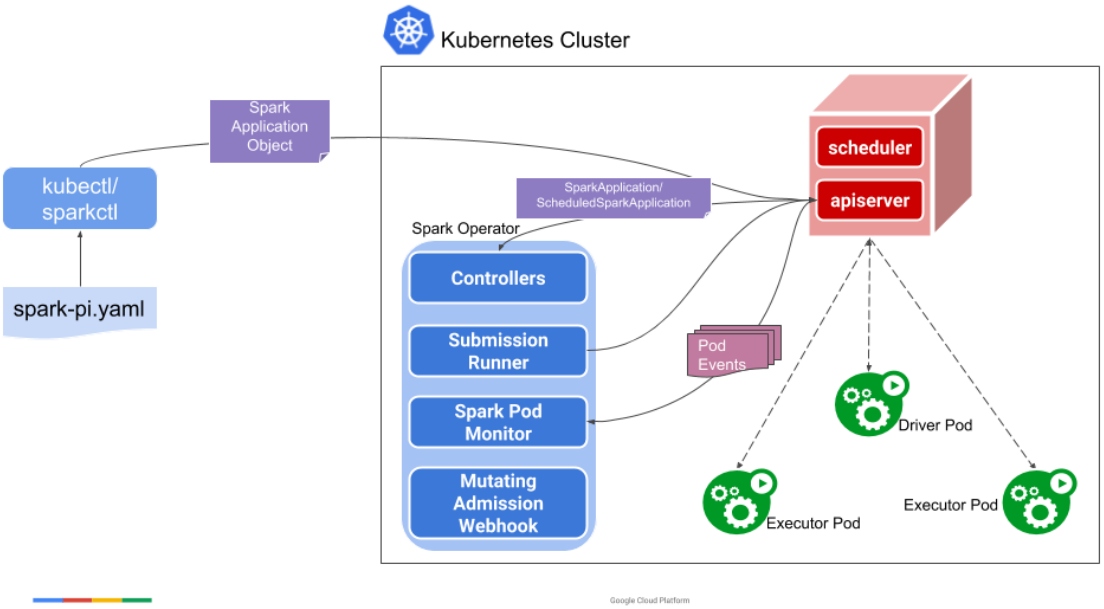


Figura 4.4: Architettura dello Spark operator [25]

il montaggio di volumi, ConfigMaps e Secrets, o la definizione di politiche di affinità e anti-affinità.

L'architettura dello Spark operator e la procedura di esecuzione di una SparkApplication sono rappresentati in Figura 4.4.

Lo Spark operator fornisce la possibilità di stabilire la policy di riavvio di un'applicazione all'interno delle specifiche dell'oggetto SparkApplication. L'operatore determina se l'applicazione deve essere riavviata in base allo stato di terminazione del Driver pod. Le policy di restart disponibili sono:

- **Never**: indipendentemente dallo stato di terminazione, l'applicazione non viene mai riavviata;
- **Always**: l'applicazione viene riavviata in qualsiasi caso;
- **OnFailure**: l'applicazione viene riavviata solo se la sua esecuzione fallisce e il limite di riavvii stabilito non è stato raggiunto.

Quando l'operatore decide di riavviare un'applicazione, le risorse Kubernetes associate alla precedente esecuzione vengono pulite e viene nuovamente invocato lo `spark-submit`, senza che il Driver pod sia riavviato.

4.2.3 Rook

Rook [26] è un progetto open-source per l'orchestrazione di storage distribuito in Kubernetes. Fondamentalmente, Rook trasforma lo storage tradizionale, come block, file, e object storage, in servizi nativi di Kubernetes, consentendo agli utenti di gestire lo storage distribuito in modo semplice e affidabile all'interno di un cluster Kubernetes.

Rook utilizza Ceph [27] come piattaforma di storage sottostante, il quale fornisce un sistema distribuito altamente scalabile e affidabile progettato per funzionare su cluster di nodi.

L'architettura di Ceph si basa su quattro componenti principali:

- **Monitor** (MON): mantiene la mappatura del cluster, permettendo così la coordinazione e comunicazione tra i suoi elementi. Inoltre, è responsabile della gestione dell'autenticazione tra i client e i demoni;
- **Manager** (MGR): ha la responsabilità di tenere traccia dello stato del cluster Ceph, incluso l'utilizzo dello storage, il carico del sistema e i valori delle metriche di performance. Inoltre, ospita dei moduli Python per la gestione e l'esposizione di informazioni sul cluster Ceph, tra cui una Ceph Dashboard e delle REST API;
- **Object Storage Deamon** (OSD): gli OSDs sono i componenti di archiviazione primari in un cluster Ceph. Ogni OSD è responsabile dell'archiviazione dei dati, inclusa la replicazione, la ridondanza e il recovery. Inoltre, forniscono informazioni di monitoraggio a monitor e manager controllando altri OSDs;
- **Metadata Server** (MDS): memorizza metadati per conto dei Ceph File Systems e permette agli utenti di eseguire comandi base (ls, find, ecc.) senza gravare sull'intero cluster Ceph.
- **Reliable Autonomic Distributed Object Stores** (RADOS): object storage che sta alla base del cluster Ceph. Si occupa di garantire la consistenza dei dati ed è responsabile della replicazione dei dati, del rilevamento di guasti e delle procedure di recupero.

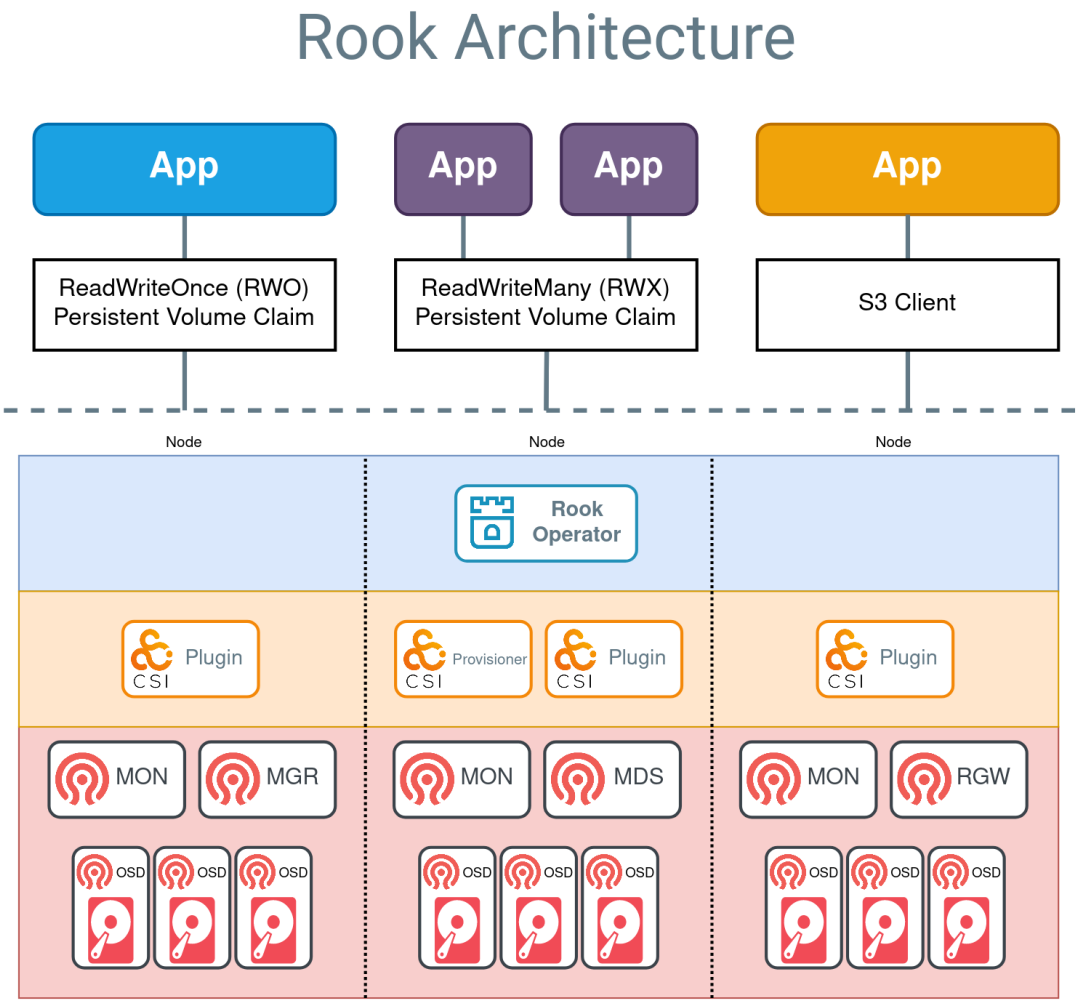


Figura 4.5: Architettura di Rook [28]

Tipicamente, per garantire ridondanza e alta disponibilità in un cluster di piccole dimensioni, sono richiesti almeno tre Ceph monitors, due Ceph managers e tre Ceph OSDs.

Rook comprende un operatore Kubernetes per effettuare il deployment e gestire i servizi di storage forniti da Ceph. Il Rook operator automatizza la configurazione delle componenti dello storage e monitora lo stato del cluster per assicurare la disponibilità e il corretto funzionamento dello storage.

Un esempio di architettura di Rook per i tre tipi di storage supportati è mostrato in Figura 4.5.

- Il primo riquadro blu rappresenta un'applicazione dotata di Block Storage. Questo tipo di storage permette il montaggio di un volume su un solo pod, perciò viene utilizzato tramite un PersistentVolumeClaim con accesso ReadWriteOnce (RWO);

- Il secondo caso con due riquadri viola consiste in un'applicazione dotata di File System condiviso, in cui molteplici istanze possono leggere e scrivere simultaneamente. In questo caso, diversi pod possono accedere allo storage e montare i volumi relativi, perciò viene utilizzato un claim di tipo ReadWriteMany (RWX);

- Il terzo caso con riquadro arancione mostra un'applicazione dotata di Object Storage. Questo tipo di storage espone un API S3 che consente alle applicazioni di accedere al cluster di archiviazione;

- I componenti CSI Plugin e CSI provisioner (Container Storage Interface) distribuiti sui nodi si occupano della creazione e del montaggio dei volumi.

4.3 Architettura distribuita

In Figura 4.6 viene rappresentata l'architettura del framework Mondrian migrato in ambiente Kubernetes, integrato con i servizi dello Spark operator e con lo storage distribuito Ceph fornito dal Rook operator.

In questo esempio, il cluster Kubernetes è composto da quattro nodi. Sul nodo principale, che ospita anche i componenti del control plane, vengono schedulati i pod relativi a:

- **Spark operator**: responsabile per la gestione delle applicazioni Spark in Kubernetes;

- **Spark driver**: processo che esegue il codice principale dell'applicazione Spark e distribuisce il carico di lavoro agli Spark executors;

- **Rook operator**: responsabile per la gestione del cluster di archiviazione Ceph.

Quando lo Spark driver riceve un job da distribuire, vengono creati sui nodi worker del cluster un numero specificato di pod relativi agli Spark executors, i quali ricevono i task da svolgere e restituiscono i risultati dell'esecuzione. Si osservi che ciascun nodo worker può ospitare molteplici istanze di esecutori Spark, fin tanto che le richieste di risorse dei pod non superano la disponibilità sul nodo.

Per quanto riguarda gli elementi dello storage distribuito:

- Ogni nodo ha un numero variabile di Ceph OSDs, in base ai device disponibili sul nodo e alla configurazione del cluster stabilita nel relativo manifesto Kubernetes;
- Almeno un Ceph manager e un Ceph monitor devono essere presenti su un nodo, ma possono esserci delle repliche per entrambi i componenti per garantire ridondanza;
- Almeno un Ceph MDS deve essere presente su un nodo del cluster. Nel caso di più istanze, una parte stabilita di esse sarà attiva mentre le altre saranno in modalità standby.

Si noti che la presenza del Ceph MDS è necessaria in quanto la tipologia di storage che verrà utilizzato è il file system condiviso, che permette a diversi pod l'accesso simultaneo ad uno stesso volume.

Nella rappresentazione dell'architettura in Figura 4.6, ogni nodo del cluster è composto da riquadri separati. Questi riquadri corrispondono ai namespace di Kubernetes. In particolare, lo Spark driver pod e gli Spark executor pods sono schedulati nel namespace *default*, lo Spark operator pod è schedulato nel namespace *spark-operator*, mentre i pod relativi al Rook operator e ai componenti di Ceph sono schedulati nel namespace *rook-ceph*.

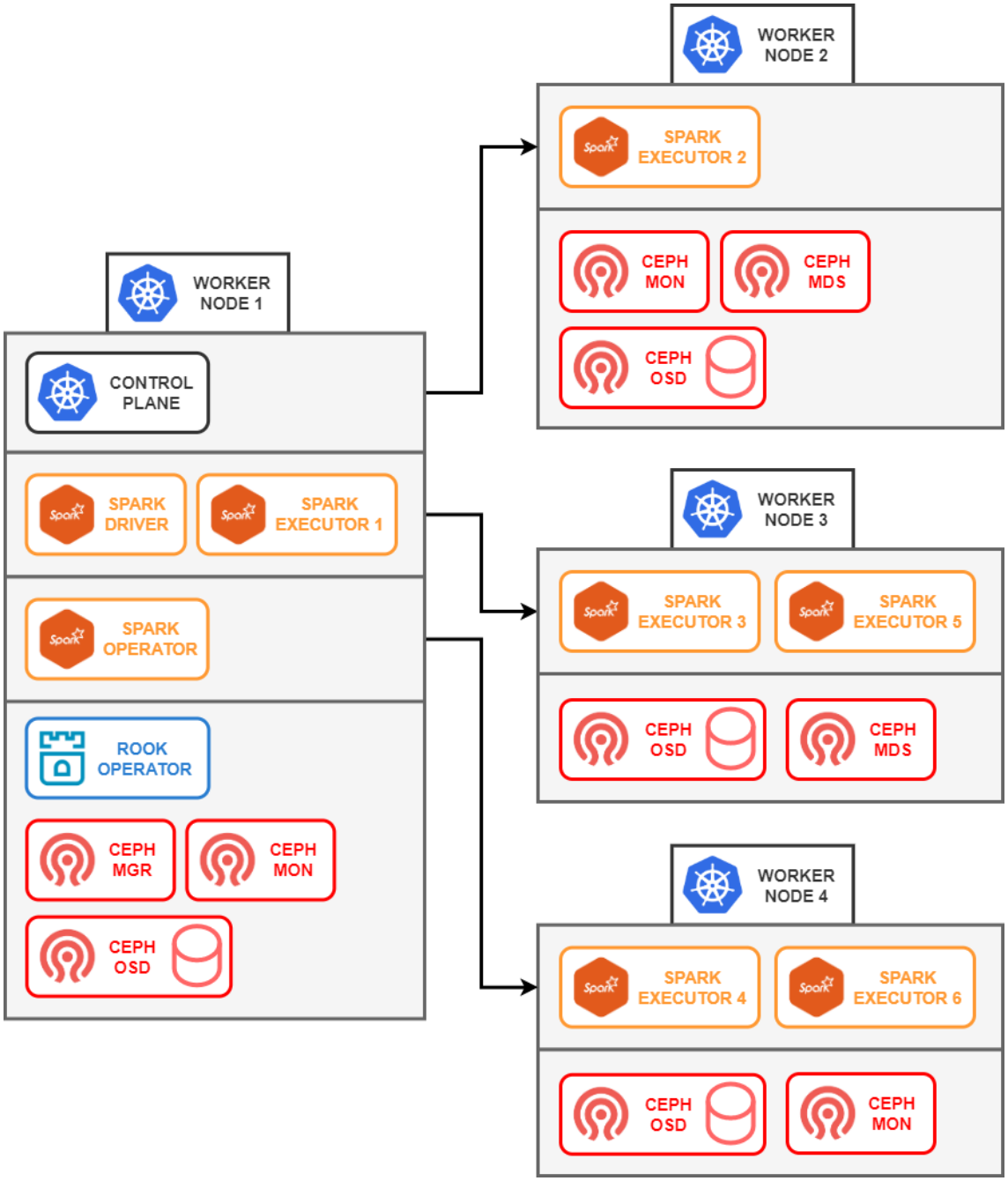


Figura 4.6: Architettura Kubernetes integrata con Spark operator e Rook storage

Capitolo 5

Implementazione del porting

In questo capitolo viene descritta l'implementazione del porting del framework Mondrian in ambiente Kubernetes. L'esecuzione del nuovo sistema viene suddivisa in cinque fasi principali:

1. **Creazione del cluster Kubernetes**: fase iniziale che consiste nella predisposizione del cluster di nodi Kubernetes, formato da un insieme di macchine virtuali, tramite il software Minikube;

2. **Generazione dell'immagine Docker**: fase di costruzione dell'immagine Docker con cui verranno generati i container e suo caricamento all'interno del cluster;

3. **Configurazione dello storage distribuito tramite Rook**: fase di configurazione dello storage distribuito, che prevede l'installazione dell'operatore Rook e dei vari componenti del cluster di archiviazione Ceph, nonché il caricamento dei file necessari all'applicazione;

4. **Esecuzione dell'applicazione Spark**: fase esecutiva che comporta l'installazione dell'operatore Spark e l'avvio dell'applicazione di anonimizzazione distribuita;

5. **Terminazione dell'applicazione**: fase finale in cui vengono esportati i risultati, terminata l'applicazione Spark, rimossi gli operatori di Spark e Rook, inclusi i pod e le risorse relative, e fermato ed eliminato il cluster Kubernetes.

5.1 Creazione del cluster Kubernetes

La creazione di un cluster Kubernetes consiste nel predisporre un insieme di risorse informatiche, inclusi nodi fisici o macchine virtuali, che collaborano per costituire un ambiente di esecuzione per applicazioni containerizzate. In primo luogo, è necessario pianificare e preparare l'infrastruttura, che può essere basata su ambienti cloud, on-premise oppure ibridi. Successivamente, si procede con l'installazione di Kubernetes su ciascun nodo e con la configurazione del control plane, dei nodi worker e della rete che consente la comunicazione tra nodi e container all'interno del cluster. Una volta che tutti i nodi sono stati aggiunti al cluster e i componenti principali sono in esecuzione, è possibile iniziare a distribuire le applicazioni containerizzate.

Per creare un cluster Kubernetes, quindi, la prima opzione è quella di utilizzare dei dispositivi fisici, situati in locale o in rete, oppure delle macchine virtuali. In questo contesto, l'installazione di Kubernetes e le varie operazioni di configurazione del cluster e dei nodi possono essere semplificate e automatizzate con Kubeadm [29]. Questa tipologia di cluster può essere sfruttata sia come ambiente di sviluppo che come ambiente di produzione.

In caso la disponibilità di risorse fisiche in locale non fosse sufficiente, diverse piattaforme di cloud computing forniscono la possibilità di creare e gestire cluster Kubernetes online. Tra le più famose ci sono: Amazon Web Services (AWS) con Amazon Elastic Kubernetes Service (EKS) [30]; Google Cloud Platform (GCP) con Google Kubernetes Engine (GKE) [31]; Microsoft Azure con Azure Kubernetes Service (AKS) [32]. Questi servizi hanno il vantaggio di semplificare notevolmente i processi di provisioning, configurazione e gestione dei cluster Kubernetes, consentendo agli sviluppatori di concentrarsi sul lato applicativo del sistema senza doversi preoccupare dell'infrastruttura sottostante.

Come ulteriore alternativa, è possibile ricorrere a dei software di simulazione di cluster Kubernetes, tra cui Kind [33] e K3d [34], che utilizzano i container Docker come nodi del cluster. Questi sistemi sono stati progettati per funzionare come ambienti di sviluppo e test per applicazioni in Kubernetes, in quanto permettono di riprodurre un numero elevato di nodi impiegando un volume ridotto di risorse.

5.1.1 Minikube

Per questo progetto, la scelta è ricaduta su Minikube [35], strumento open-source che consente di eseguire un cluster Kubernetes locale composto di macchine virtuali. Le qualità di Minikube sono la facilità di installazione e utilizzo, attraverso un'interfaccia semplice da linea di comando con cui avviare, fermare e gestire il cluster Kubernetes, l'integrazione con molti strumenti e framework di sviluppo, inclusi kubectl e helm, e la possibilità di lavorare in un ambiente isolato dalle risorse e dalla configurazione del sistema operativo host. Inoltre, risulta compatibile con tutti i sistemi operativi e supporta diversi driver per la creazione di macchine virtuali (KVM2, QEMU, VirtualBox, Podman, Hyper-V, e anche Docker, che integra container e macchine virtuali). Per ultimo, la creazione dello storage distribuito Ceph richiede la disponibilità di dischi liberi per poter creare gli OSDs e permettere ai pod di consumare lo storage, come verrà spiegato nella Sezione 5.3. Perciò il vantaggio di Minikube, a differenza delle soluzioni in container, è quello di fornire la possibilità di allocare dischi virtuali su ciascuna macchina in fase di creazione del cluster.

La creazione di un cluster Minikube avviene tramite il comando *minikube start*, come mostrato nel Listato 5.1.

```
1 minikube start        \
2     --driver kvm2     \
3     --cpus 4          \
4     --memory 16G      \
5     --extra-disks 1   \
6     --disk-size 16G   \
7     --nodes 3
```

Listato 5.1: Creazione del cluster Kubernetes con Minikube

I parametri passati al comando start sono:

- *driver*: il driver per la creazione delle macchine virtuali, in questo caso *kvm2* (Kernel based Virtual Machine), strumento di virtualizzazione integrato in Linux;
- *cpus*: il numero di cpu assegnate ad ogni macchina virtuale (4), il numero minimo è 2;

Figura 5.1: Stato dei nodi del cluster Minikube

- *memory*: la quantità di memoria RAM assegnata ad ogni macchina virtuale, in questo caso 16 GB;
- *extra-disks*: numero di dischi virtuali extra allocati su ogni nodo, per questo progetto è sufficiente un solo disco per ogni nodo;
- *disk-size*: dimensione dei dischi virtuali allocati su ogni nodo (16 GB);
- *nodes*: numero di nodi del cluster, che equivale al numero di macchine virtuali create.

In seguito alla creazione del cluster, è possibile verificare lo stato dei nodi tramite l'esecuzione del comando contenuto nel Listato 5.2:

```
1 minikube status
```

Listato 5.2: Visualizzazione dello stato dei nodi del cluster Kubernetes

L'output viene riportato in Figura 5.1. Nell'elenco dei nodi del cluster si può distinguere il nodo "minikube" etichettato come "Control Plane", il che significa che su quel nodo è stato installato il control plane e i suoi componenti. I restanti nodi invece sono etichettati come "Worker". Tutti i nodi risultano correttamente configurati e in stato "Running".

5.2 Generazione dell'immagine Docker

L'esecuzione di un'applicazione containerizzata e distribuita in un cluster Kubernetes ha come prerequisito la costruzione di un'immagine Docker specifica dell'applicazione. Questa immagine viene utilizzata come base per la creazione dei container schedulati sui nodi worker.

Un'immagine Docker viene configurata a partire da un file chiamato Dockerfile, senza estensione, contenente le istruzioni per la costruzione dell'immagine. Innanzitutto, viene definita l'immagine base da utilizzare, che può essere relativa ad un sistema operativo oppure ad un particolare servizio o applicazione. Dopo di che è possibile installare dipendenze, copiare file necessari all'interno del file system dell'immagine, eseguire comandi di setup, e altro ancora.

Successivamente, viene avviato il processo di costruzione utilizzando il comando *docker build*. Durante questo processo, Docker legge il Dockerfile ed esegue le istruzioni in modo sequenziale, creando strati dell'immagine che riflettono le modifiche apportata da ciascuna istruzione. Questa struttura a strati consente di ottimizzare la gestione delle modifiche e delle versioni dell'immagine, riducendone la dimensione effettiva e consentendo l'uso efficiente della cache durante il processo di costruzione.

Infine, l'immagine creata, identificata da un nome e una versione, può essere utilizzata localmente oppure caricata in un repository in rete.

Per questo progetto è necessario costruire un'immagine Docker per l'esecuzione di applicazioni PySpark in ambiente distribuito. Il Dockerfile utilizzato per la creazione dell'immagine è mostrato nel Listato 5.3.

```
1 FROM apache/spark-py:v3.4.0
2 USER root
3 COPY requirements.txt /tmp/
4 RUN ["pip", "install", "--upgrade", "pip"]
5 RUN ["pip", "install", "--requirement", "/tmp/requirements.txt"]
6 RUN mkdir "/mondrian"
7 COPY anonymize.py /mondrian/
8 COPY mondrian.zip /mondrian/
```

Listato 5.3: Dockerfile per la creazione dell'immagine Docker

Entrando nei dettagli delle singole istruzioni:

- La prima istruzione indica l'immagine di base utilizzata, ossia la versione 3.4.0 dell'immagine ufficiale "spark-py" contenuta nel repository "apache", specifica per applicazioni PySpark;

- L'istruzione "USER root" alla riga 2 ha lo scopo di cambiare l'attuale utente in "root", che corrisponde al superuser in ambiente Linux, e permette di ottenere l'autorizzazione ad accedere alle risorse dell'immagine, e quindi ad eseguire comandi al suo interno;

- Le istruzioni alle righe 3-4-5 servono per copiare un file di requisiti all'interno dell'immagine ed eseguire i comandi di aggiornamento del package installer di Python (pip) e di installazione delle librerie Python richieste;

- Le istruzioni alle righe 6-7-8 prevedono la creazione di una cartella nominata "mondrian", all'interno della quale vengono importati il file principale dell'applicazione "anonymize.py" e una cartella compressa "mondrian.zip" contenente le dipendenze Python, necessari per l'esecuzione distribuita sui vari container.

Le librerie di Python (con relativa versione) utilizzate nell'applicazione e installate nell'immagine Docker sono le seguenti:

- **numpy (1.26.2)**: consente di eseguire operazioni complesse e applicare funzioni matematiche avanzate su strutture dati multidimensionali, garantendo velocità e flessibilità nell'analisi dei dati;

- **pandas (2.1.4)**: offre strutture dati potenti e flessibili, in particolare la DataFrame, per la manipolazione e l'analisi di dati tabulari provenienti da diverse fonti (file CSV, Excel, database SQL, e molti altri);

- **pyarrow (14.0.2)**: permette di lavorare in maniera semplice ed efficiente con dati strutturati e array multidimensionali in Apache Arrow [36], una piattaforma che definisce un formato di dati colonnare in memoria progettato per l'analisi ad alte prestazioni;

- **scipy (1.11.4)**: fornisce funzionalità avanzate per la computazione scientifica e l'analisi dei dati, estende la libreria numpy per la risoluzione di problemi complessi di ottimizzazione, simulazione ed elaborazione dati;
- **treelib(1.7.0)**: fornisce strumenti per la rappresentazione e l'analisi di strutture dati ad albero, utilizzata in questo progetto per la gestione delle gerarchie di generalizzazione degli attributi categorici presenti nei dataset da anonimizzare;

Una volta definito il Dockerfile, si procede con la costruzione dell'immagine invocando il comando *docker build*, mostrato nel Listato 5.4.

```
1 docker build -t kube-mondrian:latest .
```

Listato 5.4: Generazione dell'immagine Docker

Il flag "-t" specifica il tag dell'immagine che si sta creando, composto da un nome ("kube-mondrian") e una versione ("latest"), separati dal simbolo due punti. Inoltre, viene indicato il Dockerfile o il percorso dove cercare il Dockerfile di riferimento per la creazione dell'immagine, che può essere in locale oppure online. In questo caso, il Dockerfile è collocato nella stessa cartella da cui viene invocato il comando, ossia la directory locale contenente i file del progetto tra cui il file di requisiti citato precedentemente, perciò viene utilizzato il simbolo per la cartella corrente ('.').

L'ultimo passaggio consiste nell'importare l'immagine creata all'interno del cluster Minikube. Il comando *docker build*, infatti, se non specificato diversamente, salva l'immagine nel repository locale, che di default non è accessibile direttamente da un cluster Minikube. Quindi, uno dei metodi per rendere disponibile l'immagine Docker consiste nell'eseguire il comando mostrato nel Listato 5.5.

```
1 minikube image load kube-mondrian:latest
```

Listato 5.5: Caricamento dell'immagine Docker nel cluster Minikube

In questo modo, l'immagine Docker viene resa accessibile ai nodi del cluster Minikube, che possono utilizzarla per configurare i container. Alternativamente, è possibile salvare l'immagine in un repository pubblico, ad esempio Docker Hub, da cui poi i nodi del cluster, in fase di creazione dei container, recuperano l'immagine richiesta.

5.3 Configurazione dello storage distribuito tramite Rook

La configurazione dello storage distribuito in ambiente Kubernetes consiste nell'installazione dell'operatore Rook e di una serie di componenti che permettono di gestire in maniera semplice ed automatica il cluster di archiviazione Ceph e le operazioni di lettura e scrittura.

Nonostante sia possibile semplificare la procedura di installazione utilizzando il repository pubblico di Rook e i relativi Helm Charts, la necessità di modificare alcuni parametri per adattare il cluster e le risorse allo specifico ambiente Kubernetes rende preferibile effettuare un'installazione manuale. Per questo motivo, sono stati scaricati in locale i manifesti Kubernetes necessari, disponibili nel repository Github di Rook al seguente link: `https://github.com/rook/rook`.

Innanzitutto, occorre predisporre all'interno del cluster Kubernetes i componenti base per poter gestire un cluster di archiviazione Ceph. Per fare questo, bisogna importare le definizioni delle risorse personalizzate CRD (Custom Resource Definition) e dei service account, ossia i ruoli e le autorizzazioni detenute da ciascun singolo componente all'interno del cluster, contenuti rispettivamente nei manifesti "crds.yaml" e "common.yaml". A questo punto, è possibile creare il Rook operator, configurato nel manifesto "operator.yaml", che si occuperà prima dell'installazione dei componenti necessari e poi della supervisione del cluster di archiviazione Ceph, svolgendo il ruolo di intermediario per le operazioni di richiesta di storage da parte dei pod, monitorando lo stato di salute del sistema di archiviazione e intervenendo nel caso di malfunzionamenti con la sostituzione dei componenti guasti.

In seguito, quando il pod relativo all'operatore Rook ha terminato la fase d creazione e si trova in stato "Running", si procede con la generazione del cluster di archiviazione Ceph, definito nel manifesto "cluster.yaml" riportato nel Listato 5.6. Con questa operazione vengono creati la maggior parte dei pod necessari, relativi ai Ceph manager, Ceph monitor, Ceph OSDs, e altro ancora.

```
1 apiVersion: ceph.rook.io/v1
2 kind: CephCluster
```

```yaml
metadata:
  name: my-cluster
  namespace: rook-ceph
spec:
  dataDirHostPath: /data/rook
  cephVersion:
    image: quay.io/ceph/ceph:v18
    allowUnsupported: true
  mon:
    count: 3
    allowMultiplePerNode: true
  mgr:
    count: 1
    allowMultiplePerNode: true
  dashboard:
    enabled: true
  crashCollector:
    disable: true
  storage:
    useAllNodes: false
    useAllDevices: false
    config:
      osdsPerDevice: "1"
    nodes:
      - name: "minikube"
        devices:
          - name: "vdb"
      - name: "minikube-m02"
        devices:
          - name: "vdb"
      - name: "minikube-m03"
        devices:
          - name: "vdb"
  monitoring:
    enabled: false
  healthCheck:
    daemonHealth:
```

```
40      mon:
41        interval: 45s
42        timeout: 600s
43    priorityClassNames:
44      all: system-node-critical
45      mgr: system-cluster-critical
46    disruptionManagement:
47      managePodBudgets: true
48    cephConfig:
49      global:
50        osd_pool_default_size: "1"
51        mon_warn_on_pool_no_redundancy: "false"
52        bdev_flock_retry: "20"
53        bluefs_buffered_io: "false"
54        mon_data_avail_warn: "10"
```

Listato 5.6: Manifesto Kubernetes per la configurazione del cluster di archiviazione

Le parti più importanti del manifesto relativo all'oggetto CephCluster sono:

- *dataDirHostPath* (riga 7): rappresenta il percorso sul nodo in cui vengono memorizzati i dati e i file di configurazione;
- *mon* (righe da 11 a 13): definisce il numero di Ceph monitor da schedulare e la possibilità di schedularne molteplici sullo stesso nodo;
- *mgr* (righe da 14 a 16): simile al campo *mon*, definisce il numero di Ceph manager e la schedulazione multipla;
- *storage* (righe da 21 a 38): vengono specificati i dispositivi (in questo caso i dischi virtuali) utilizzati per ospitare gli Object Storage Daemon (OSD) e i nodi sui quali si trovano. In alternativa, si può delegare la scelta all'operatore mettendo il valore *true* ai parametri *useAllNodes* e *useAllDevices*, avendo però meno controllo sulla selezione;

I restanti parametri sono riferiti al monitoraggio del cluster, la verifica dello stato di salute, la rischedulazione dei monitor e alcune configurazioni secondarie di Ceph.

Dopo aver creato il cluster Ceph, si prosegue con la configurazione del sistema di storage. In questo contesto, è stato scelto di utilizzare un file system distribuito che, a differenza del block storage, permette a diversi pod schedulati su diversi nodi di accedere allo stesso volume di dati. Questa tipologia di storage presenta diversi vantaggi nell'ottica di questo progetto. Da un punto di vista tecnico, facilita la gestione dei dati all'interno del cluster, eliminando la necessità di replicare i dati in più volumi e semplificando il provisioning e la gestione dei volumi stessi. In ambito applicativo, invece, migliora la condivisione e la collaborazione tra driver ed esecutori. Inoltre, consente una maggiore flessibilità nella distribuzione dell'applicazione, permettendo di scalare orizzontalmente l'esecuzione dei processi di anonimizzazione tra un numero variabile di esecutori senza doversi preoccupare della gestione dei dati e della creazione di nuovi volumi. Il relativo manifesto Kubernetes è rappresentato nel Listato 5.7.

```yaml
apiVersion: ceph.rook.io/v1
kind: CephFilesystem
metadata:
  name: myfs
  namespace: rook-ceph
spec:
  metadataPool:
    replicated:
      size: 2
      requireSafeReplicaSize: true
  dataPools:
    - name: replicated
      failureDomain: osd
      replicated:
        size: 2
        requireSafeReplicaSize: true
  preserveFilesystemOnDelete: false
  metadataServer:
    activeCount: 1
    activeStandby: true
```

Listato 5.7: Manifesto Kubernetes per la configurazione del file system distribuito

I parametri più rilevanti sono:

- *metadataPool* (righe da 7 a 10): informazioni sui metadata pool del file system (con il termine "pool" ci si riferisce ad un pool di dati, ossia ad una collezione di spazio di archiviazione sui dispositivi di storage distribuiti nel cluster), nello specifico il numero di repliche da creare e il campo *requireSafeReplicaSize*, che in caso sia *true* indica che prima di completare un'operazione le repliche dei dati devono essere distribuite correttamente;

- *dataPools* (righe da 11 a 16): informazioni sui datapool del file system, tra cui il numero di repliche e il *failureDomain*, ossia l'indicazione sul livello a cui vengono distribuite le repliche dei dati per garantire ridondanza e disponibilità in caso di guasti. Il valore *osd* indica che le repliche devono essere distribuite su 2 OSD differenti. In alternativa, è possibile impostare il valore *host*, che impone che le repliche siano distribuite su 2 OSD appartenenti a nodi diversi, oppure *rack*, in cui le repliche sono distribuite su gruppi di nodi differenti. Più si alza il livello, migliore è la garanzia di disponibilità dei dati. In questo caso, trattandosi di un piccolo cluster con un OSD per ogni nodo, il valore *osd* risulta equivalente al valore *host* e fornisce un grado di protezione più che sufficiente. In cluster di dimensioni maggiori, con molteplici istanze di OSD distribuite su un numero elevato di nodi, occorre alzare il livello di protezione poiché, ad esempio, se le repliche fossero distribuite su un unico nodo e quel nodo si guastasse, l'intero cluster perderebbe i dati immagazzinati;

- *metadataServer* (righe da 18 a 20): contiene le impostazioni relative ai demoni Metadata Server. In particolare, *activeCount* rappresenta il numero di istanze MDS attive (Rook crea il doppo delle istanze specificate, le altre restano in modalità standby), mentre *activeStandby* indica, se *true*, che le istanze extra rimangono in modalità standby attiva, ossia mantengono una cache fresca dei metadati per recuperi più veloci.

A questo punto nel cluster Kubernetes è stato correttamente configurato un file system distribuito. Per poter fornire storage ai pod, tuttavia, è necessario creare una risorsa StorageClass basata sul file system, necessario per interagire con i driver

CSI (Container Storage Interface) di Kubernetes nella creazione dei volumi. Nel Listato 5.8 viene riportato il relativo manifesto.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-cephfs
provisioner: rook-ceph.cephfs.csi.ceph.com
parameters:
  clusterID: rook-ceph
  fsName: myfs
  pool: myfs-replicated
  csi.storage.k8s.io/provisioner-secret-name: rook-csi-cephfs-
    provisioner
  csi.storage.k8s.io/provisioner-secret-namespace: rook-ceph
  csi.storage.k8s.io/controller-expand-secret-name: rook-csi-cephfs-
    provisioner
  csi.storage.k8s.io/controller-expand-secret-namespace: rook-ceph
  csi.storage.k8s.io/node-stage-secret-name: rook-csi-cephfs-node
  csi.storage.k8s.io/node-stage-secret-namespace: rook-ceph
reclaimPolicy: Delete
```

Listato 5.8: Manifesto Kubernetes per la configurazione dello storage class

Nel file sono contenuti i riferimenti alle risorse del cluster (nome del cluster, del file system e del pool di dati), il provisioner incaricato della gestione dei volumi e della comunicazione con i provider dello storage sottostante, e altre informazioni su nomi e namespace dei componenti di CSI (Container Storage Interface). CSI è uno standard di interfaccia che consente di integrare sistemi di storage in ambienti containerizzati come Kubernetes, separando la gestione dello storage dallo sviluppo del software e migliorando la flessibilità e la portabilità dell'applicazione. Il supporto di CSI, inoltre, permette di montare volumi di storage forniti da gestori terzi, come in questo caso Ceph e Rook.

Per ultimo viene creato il toolbox pod, un oggetto che permette di accedere a strumenti di diagnostica e comandi utili per gestire e monitorare lo storage distribuito Ceph. Ad esempio, è possibile monitorare lo stato del cluster di archiviazione eseguendo all'interno del toolbox pod il comando *ceph status*, come riportato nel

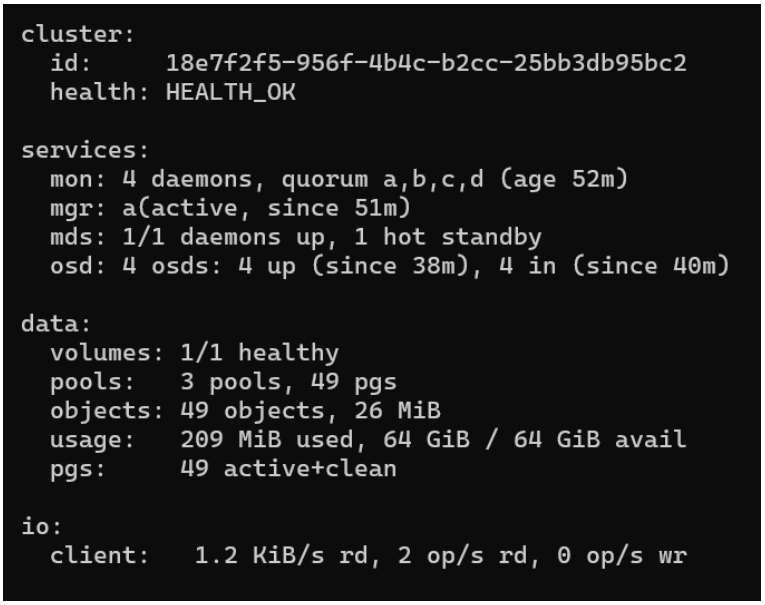


Figura 5.2: Stato del cluster di archiviazione Ceph

Listato 5.9. Il relativo output viene rappresentato in Figura 5.2, ed evidenzia che il cluster Ceph sta funzionando correttamente (*HEALTH_OK*), oltre a mostrare lo stato dei servizi attivi (*mon, mgr, mds, osd*) e dello storage disponibile.

```
kubectl -n rook-ceph exec -it deploy/rook-ceph-tools -- ceph status
```

Listato 5.9: Monitoraggio dello stato del cluster Ceph

La serie di istruzioni per la completa configurazione del sistema di storage distribuito è contenuta nel Listato 5.10.

```
kubectl create -f ./rook/crds.yaml -f ./rook/common.yaml
kubectl create -f ./rook/operator.yaml
kubectl create -f ./rook/cluster.yaml
kubectl create -f ./rook/filesystem.yaml
kubectl create -f ./rook/storageclass.yaml
kubectl create -f ./rook/toolbox.yaml
```

Listato 5.10: Installazione dell'operatore Rook e del sistema di storage Ceph

5.3.1 Caricamento di file all'interno dello storage distribuito

I passaggi mostrati finora hanno illustrato come installare l'orchestratore Rook e configurare lo storage distribuito Ceph all'interno del cluster Kubernetes. Lo

step successivo è quello di popolare l'archivio con i file necessari per l'esecuzione dell'applicazione distribuita. Nello specifico:

- Il dataset da anonimizzare;
- Il file di configurazione del job di anonimizzazione, contenente informazioni quali il percorso del dataset da anonimizzare, i parametri di anonimizzazione, le modalità di pre-processamento, le metriche di valutazione, e altro ancora. La sua struttura verrà descritta in dettaglio nella Sezione 6.1;
- I file, se necessari, contenenti le tassonomie degli attributi gerarchici all'interno del dataset.

Il trasferimento di questi file da un percorso locale al file system distribuito nel cluster Kubernetes avviene tramite un pod temporaneo, sul quale viene montato un volume collegato allo storage Ceph. Per fare ciò è necessario definire un oggetto PersistentVolumeClaim specifico (Listato 5.11).

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: cephfs-pvc
spec:
  resources:
    requests:
      storage: 10Gi
  accessModes:
    - ReadWriteMany
  storageClassName: rook-cephfs
  volumeMode: Filesystem
```

Listato 5.11: Manifesto Kubernetes per la creazione del PersistentVolumeClaim

Il claim viene configurato come segue:

- Il nome del claim è *cephfs-pvc*;
- L'ammontare di storage richiesto è di *10Gi*;
- La modalità di accesso è *ReadWriteMany*;

- Il nome dello StorageClass è *rook-cephfs* (definito in precedenza nel Listato 5.8);
- La modalità di montaggio del volume è *Filesystem*.

La modalità di accesso RWX è dettata dalla scelta del file system condiviso come tipologia di storage. Essendo un claim RWX, verrà riutilizzato anche in seguito durante l'esecuzione dell'applicazione Spark e servirà per montare lo stesso volume sul driver pod e sugli executor pods. Ciò risulta molto vantaggioso anche in ottica di sviluppo, poiché avendo un numero variabile di esecutori non vi è la necessità di generare diversi claim ma è possibile assegnare la medesima istanza a tutti i pod. Inoltre, montando lo stesso volume su tutti i pod, non vi è la necessità di distribuire delle copie ma l'accesso avviene sulla stessa versione dei file, che risultano sempre aggiornati e non richiedono sincronizzazioni aggiuntive.

Dopo di che, si procede con la definizione del pod utilizzato come intermediario per il caricamento dei file. Il relativo manifesto Kubernetes viene riportato nel Listato 5.12.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: data-transfer-pod
spec:
  containers:
  - name: data-transfer-pod
    image: nginx
    volumeMounts:
      - name: cephfs
        mountPath: /data
  volumes:
    - name: cephfs
      persistentVolumeClaim:
        claimName: cephfs-pvc
        readOnly: false
```

Listato 5.12: Manifesto Kubernetes per il trasferimento dei file all'interno del cluster di archiviazione Ceph

Le righe da 9 a 16 mostrano la modalità con cui viene montato un volume all'interno di un pod. Nel campo *volumes* (righe da 12 a 16) viene definito il nome del volume, il claim collegato e viene abilitata la scrittura, mettendo *false* al parametro *readOnly*. Nel campo *volumeMounts* (righe da 9 a 11), invece, si forniscono le indicazioni per il montaggio nel pod, ossia il nome del volume e il percorso di montaggio nel file system del pod.

Dopo aver creato il claim e il pod per il trasferimento, si procede con la copiatura dei file, come mostrato nel Listato 5.13.

```
1  kubectl exec -it data-transfer-pod -- mkdir /data/config \
2                                              /data/dataset \
3                                              /data/taxonomy \
4                                              /data/anonymized
5  kubectl cp -n default \
6             ./config/poker.json \
7             data-transfer-pod:/data/config/poker.json
8  kubectl cp -n default \
9             ./dataset/poker.csv \
10            data-transfer-pod:/data/dataset/poker.csv
```

Listato 5.13: Trasferimento dei file all'interno del cluster di archiviazione Ceph

Con il primo comando si entra nella shell del pod e si creano una serie di directory, utilizzate per separare i file di input (*config, dataset, taxonomy*) e per predisporre la cartella in cui verrano salvati i risultati finali (*anonymized*). Il comando *kubectl cp*, invece, permette di copiare un file da un percorso in locale ad un pod nel cluster Kubernetes e viceversa. I parametri del comando sono: il namespace in cui si trova il pod (*default* in questo caso), il percorso di origine del file da copiare e il percorso di destinazione, preceduto dal nome del pod.

Con questa operazione i file risultano correttamente memorizzati nell'archivio Ceph e si conclude la fase di preparazione dello storage distribuito nel cluster Kubernetes.

5.4 Esecuzione dell'applicazione Spark

Dopo aver predisposto il cluster Kubernetes, generato l'immagine Docker dell'applicazione Spark e configurato il sistema di storage distribuito tramite Rook, l'ambiente di sviluppo è pronto per l'esecuzione dell'applicazione Spark.

Il primo passo consiste nell'installazione dell'operatore Spark all'interno del cluster, tramite l'esecuzione della serie di istruzioni riportate nel Listato 5.14.

```
1 helm repo add spark-operator https://googlecloudplatform.github.io/
     spark-on-k8s-operator
2 helm repo update
3 helm install spark-operator spark-operator/spark-operator \
4     --namespace spark-operator --create-namespace \
5     --set sparkJobNamespace=default \
6     --set webhook.enable=true
```

Listato 5.14: Installazione dello Spark operator

Innanzitutto, si procede con l'aggiungere al cluster in locale il repository Helm all'interno del quale sono contenuti i manifesti e le definizioni delle risorse necessarie per l'installazione dello Spark operator (riga 1). In seguito, può essere utile aggiornare l'elenco dei repositories per ottenere le ultime versioni dei pacchetti (riga 2). Infine, si procede con l'installazione dell'operatore eseguendo il comando *helm install* (riga 3), i cui parametri sono:

- il nome attribuito alla release, ossia l'istanza di un Helm Chart in esecuzione (*spark-operator*);
- il nome dell'operatore da installare preceduto dal nome del repository in cui è contenuto (*spark-operator/spark-operator*);
- il namespace all'interno del quale vengono schedulati i pod relativi all'operatore e alle altre risorse necessarie (*spark-operator*, il flag *--create-namespace* indica di creare il namespace se non esiste);
- il namespace nel quale verrà schedulato l'oggetto SparkApplication, che può essere lo stesso dell'operatore oppure uno diverso (in questo caso *default*);

- l'abilitazione del "mutating admission webhook", un componente che consente di personalizzare gli Spark pod integrandoli con delle funzionalità native di Kubernetes (ad esempio le regole di affinità o il montaggio di mappe di configurazione).

Successivamente, si prosegue con la realizzazione del manifesto Kubernetes che contiene la definizione della SparkApplication, oggetto personalizzato appartenente allo Spark operator che permette di semplificare l'esecuzione dell'applicazione Spark specificando al suo interno i parametri di configurazione. Nel Listato 5.15 viene illustrato il contenuto del manifesto Kubernetes utilizzato per questo progetto.

```yaml
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: kube-mondrian
  namespace: default
spec:
  type: Python
  pythonVersion: "3"
  sparkVersion: "3.5.0"
  mode: cluster
  image: kube-mondrian:latest
  imagePullPolicy: IfNotPresent
  mainApplicationFile: local:///mondrian/anonymize.py
  deps:
    pyFiles:
      - local:///mondrian/mondrian.zip
  arguments:
    - /data/config/poker.json
    - "3"
    - "0"
    - "0"
  restartPolicy:
    type: OnFailure
    onFailureRetries: 3
    onFailureRetryInterval: 10
    onSubmissionFailureRetries: 5
```

```
27    onSubmissionFailureRetryInterval: 20
28  driver:
29    cores: 1
30    coreLimit: "1200m"
31    memory: 2g
32    serviceAccount: spark-operator-spark
33    volumeMounts:
34      - name: kube-mondrian-storage
35        mountPath: /data
36  executor:
37    cores: 1
38    instances: 3
39    memory: 2g
40    volumeMounts:
41      - name: kube-mondrian-storage
42        mountPath: /data
43  volumes:
44    - name: kube-mondrian-storage
45      persistentVolumeClaim:
46        claimName: cephfs-pvc
47  sparkConf:
48    spark.ui.port: "4045"
49    spark.eventLog.enabled: "true"
50    spark.eventLog.dir": "/data/spark-events"
51    spark.scheduler.mode: "FIFO"
52    spark.default.parallelism: "3"
53    spark.sql.shuffle.partitions: "3"
```

Listato 5.15: Manifesto Kubernetes per la creazione della SparkApplication Mondrian

Nella sezione *metadata* vengono riportati il nome attribuito all'applicazione (*kube-mondrian*) e il namespace all'interno del quale viene schedulato il pod. Nella sezione *spec* vengono invece elencate tutte le specifiche dell'oggetto SparkApplication. In particolare, vengono indicati:

- *type* e *pythonVersion* (righe 7-8): tipologia di applicazione e relativa versione (*Python 3*);

- *sparkVersion* (riga 9): versione di Spark utilizzata (*3.5.0*);
- *mode* (riga 10): modalità di esecuzione dell'applicazione Spark (modalità *cluster*, che si contrappone alla modalità *client*, la differenza tra le due sta nel nodo che ospita lo Spark driver, nella prima viene schedulato su un nodo appartenente al cluster mentre nella seconda viene schedulato in locale su un client esterno al cluster);
- *image* e *imagePullPolicy* (righe 11-12): immagine Docker utilizzata per configurare i container (*kube-mondrian:latest*) e politica di download dell'immagine (*ifNotPresent*, l'immagine viene scaricata solo se non già presente nel nodo del cluster);
- *mainApplicationFile* e *deps.pyFiles* (righe da 13 a 16): percorsi locali ai container in cui sono presenti il codice dell'applicazione principale (*anonymize.py*) e le dipendenze Python (*mondrian.zip*);
- *arguments* (righe da 17 a 21): argomenti passati al main dell'applicazione. In ordine si trovano il file di configurazione dello Spark job (contenente informazioni sul dataset da anonimizzare, sui parametri di anonimizzazione, sulle strategie e metriche da utilizzare, ecc), il numero di worker su cui suddividere l'esecuzione, e due flag che indicano le modalità demo e test;
- *restartPolicy* (righe da 22 a 27): informazioni sulla politica di riavvio dell'applicazione
  - *type*: il riavvio del container avviene solo quando si verifica un errore (*onFailure*)
  - *onFailureRetries* e *onFailureRetryInterval*: se si verifica un errore durante l'esecuzione del pod, il riavvio avviene dopo 10 secondi e per un massimo di 3 volte;
  - *onSubmissionFailureRetries* e *onSubmissionFailureRetryInterval*: se si verifica un errore durante la creazione del pod, il riavvio avviene dopo 20 secondi e per un massimo di 5 volte;

- *driver* (righe da 28 a 35): specifiche di configurazione dello Spark driver pod. In dettaglio:
  - *cores* e *coreLimit*: numero di CPU cores assegnati (1) e limite massimo utilizzabile (1200m);
  - *memory*: quantità di memoria RAM assegnata (2 GB);
  - *serviceAccount*: account di servizio Kubernetes autorizzato a schedulare gli Spark pod (creato nel processo di installazione dello Spark operator);
  - *volumeMounts*: informazioni sul montaggio di volumi (nome e percorso in locale del volume da montare).
- *executor* (righe da 36 a 42): specifiche di configurazione degli Spark executor pods (simili a quelle del driver pod)
- *volumes* (righe da 43 a 46): elenco delle informazioni sui volumi da montare nei pod (nome del volume e nome dell'oggetto PVC);
- *sparkConf* (righe da 47 a 53): altre specifiche di configurazione di Spark
  - *spark.ui.port*: porta sulla quale esporre la dashboard di Spark, che mostra informazioni riguardanti la gestione dei job e della memoria;
  - *spark.eventLog.enabled*: abilitazione alla registrazione dei log delle operazioni di Spark;
  - *spark.eventLog.dir*: cartella nella quale vengono memorizzati i file di log, solo se *spark.eventLog.enabled* è *true*;
  - *spark.scheduler.mode*: modalità di schedulazione degli Spark pod, *FIFO* equivale a "First-In-First-Out" e significa che i job vengono eseguiti in ordine di arrivo;
  - *spark.default.parallelism*: numero di default di partizioni degli RDD ritornati da trasformazioni come "parallelize()";
  - *spark.sql.shuffle.partitions*: numero di partizioni dei DataFrame ritornati da trasformazioni come "join()";

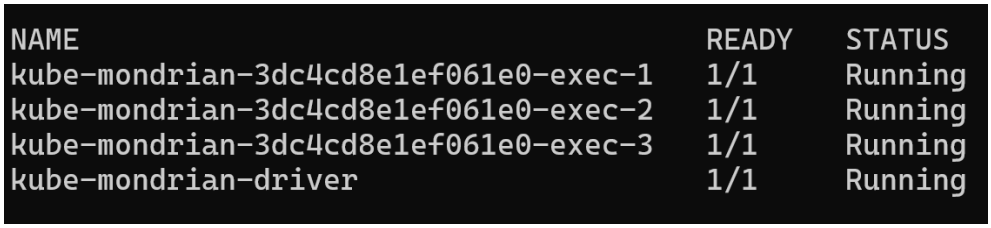


Figura 5.3: Stato di esecuzione degli Spark pods

A questo punto, dopo aver verificato che il pod relativo allo Spark operator sia in stato "Running", si può proseguire con l'avvio dell'applicazione Spark, invocando il comando di creazione del relativo manifesto Kubernetes (Listato 5.16).

```
1 kubectl create -f ./deploy/kube-mondrian.yaml
```

Listato 5.16: Esecuzione della SparkApplication Mondrian

Il primo pod creato è quello relativo allo Spark driver, il quale è incaricato di eseguire il codice dell'applicazione principale. Successivamente, vengono generati i pod relativi agli Spark executors, sui quali vengono distribuiti i task di esecuzione da parte dello Spark driver. Un esempio di stato di esecuzione degli Spark pods è mostrato in Figura 5.3. Il nome dei pods è costituito da un prefisso comune, indicante il nome dell'istanza SparkApplication (*kube-mondrian*, specificato a riga 4 del Listato 5.15), e dalla tipologia di pod (*driver* per lo Spark driver, *exec* per gli esecutori, preceduto da una sequenza di caratteri casuali assegnati automaticamente e seguito dal numero di esecutore).

L'esecuzione dell'applicazione Spark termina quando gli executor pods vengono eliminati e il driver pod entra nello stato "Completed". Eseguendo il comando mostrato nel Listato 5.17 è possibile visualizzare il log dello Spark driver pod, contenente le informazioni sulla gestione interna dell'applicazione Spark e i risultati dell'esecuzione.

```
1 kubectl logs kube-mondrian-driver
```

Listato 5.17: Visualizzazione dei logs dello Spark driver pod

5.5 Terminazione dell'applicazione

L'ultima fase inizia con l'esportazione del dataset anonimizzato dal file system Ceph. La procedura è analoga a quella di caricamento dei file illustrata nella sezione 5.3.1. Tramite lo stesso pod temporaneo, si esegue il comando *minikube cp*, specificando però come sorgente il percorso interno al pod e come destinazione il percorso in locale prestabilito. La procedura è riportata nel Listato 5.18.

```
1 kubectl create -f ./deploy/data-transfer.yaml
2 kubectl cp -n default \
3            data-transfer-pod:/data/anonymized/poker.csv \
4            ./anonymized/poker.csv
5 kubectl delete -f ./deploy/data-transfer.yaml
```

Listato 5.18: Esportazione del dataset anonimizzato

Infine, si procede con la rimozione di tutti i pod e le risorse istanziate e con la chiusura del cluster Minikube. L'elenco dei comandi necessari è mostrato nel Listato 5.19.

```
1 kubectl delete -f ./deploy/kube-mondrian.yaml
2 kubectl delete -f ./deploy/cephfs-pvc.yaml
3 kubectl delete -f ./rook/toolbox.yaml
4 kubectl delete -f ./rook/storageclass.yaml
5 kubectl delete -f ./rook/filesystem.yaml
6 kubectl delete -f ./rook/cluster.yaml
7 kubectl delete -f ./rook/crds.yaml -f ./rook/common.yaml
8 kubectl delete -f ./rook/operator.yaml
9 helm uninstall spark-operator -n spark-operator
10 helm repo remove spark-operator
11 minikube delete
```

Listato 5.19: Terminazione dell'applicazione

Nello specifico:

- La riga 1 termina l'applicazione Spark, eliminando lo Spark driver pod rimasto in stato "Completed";
- La riga 2 elimina il claim PVC utilizzato per montare i volumi del file system distribuito;

- Le righe da 3 a 8 rimuovono tutti gli elementi dello storage distribuito;
- Le riga 9 disinstalla lo Spark operator;
- La riga 10 rimuove il repository dello Spark operator;
- Infine, la riga 11 chiude il cluster Minikube, eliminando le macchine virtuali e rilasciando le risorse assegnate.

Si puntualizza che la fase di terminazione dell'applicazione risulta necessaria solo se l'intenzione è quella di eliminare il cluster Kubernetes e rimuovere tutte le risorse create. Alternativamente, è possibile fermare il cluster Minikube, tramite il comando *minikube stop*, in modo da poterlo riutilizzare successivamente senza dover reinstallare tutti i componenti. In questo modo, è possibile eseguire diverse applicazioni Spark con la stessa configurazione del cluster Kubernetes.

Con queste ultime operazioni si conclude l'esecuzione del framework Mondrian in ambiente Kubernetes.

Capitolo 6

Valutazione delle prestazioni

Questo capitolo è dedicato alla descrizione dei test effettuati per valutare le prestazioni del framework Mondrian in ambiente Kubernetes e confrontarne i risultati con la precedente versione con architettura locale e con la versione centralizzata. L'obiettivo è quello di evidenziare i punti di forza e i vantaggi nell'utilizzo dell'orchestratore di container Kubernetes per l'applicazione dell'anonimizzazione di dati in cluster di nodi distribuiti, nonché rilevare le eventuali criticità e i possibili miglioramenti.

6.1 Pianificazione dei test

La fase sperimentale di questo progetto prevede l'esecuzione di una serie di test esaustivi volti a misurare i tempi di esecuzione del framework in ambiente Kubernetes al variare di un insieme di parametri. Questi parametri sono specificati all'interno di un file di configurazione, che viene memorizzato nel cluster di archiviazione e passato come argomento al metodo main dell'applicazione. Un esempio viene riportato nel Listato 6.1.

```
{
    "input": "/data/dataset/poker.csv",
    "output": "/data/anonymized/poker.csv",
    "fraction": 1,
    "id_columns": [],
    "redact": false,
    "quasiid_columns": [
```

```
        "s1", "r1", "s2", "r2", "s3", "r3", "s4", "r4", "s5", "r5"
    ],
    "sensitive_columns": ["hand"],
    "column_score": "entropy",
    "fragmentation": "quantile",
    "K": 5,
    "L": 2,
    "measures": [
        "discernability_penalty",
        "normalized_certainty_penalty",
        "global_certainty_penalty"
    ]
}
```

Listato 6.1: File di configurazione del job di anonimizzazione

La struttura del file di configurazione è la seguente:

- *input*: percorso interno al container in cui si trova il dataset da anonimizzare ("data" corrisponde alla directory in cui è montato il volume di dati relativo al file system condiviso);
- *output*: percorso in cui viene salvato il dataset anonimizzato;
- *fraction*: frazione di campionamento del dataset utilizzato per la fase di partizionamento dei frammenti;
- *id_columns*: elenco degli attribuiti identificatori presenti nel dataset da anonimizzare;
- *redact*: opzione che se assume il valore *true* permette di oscurare i valori degli attributi identificatori nel dataset anonimizzato;
- *quasiid_columns*: elenco degli attributi quasi-identificatori presenti nel dataset da anonimizzare;
- *sensitive_columns*: elenco degli attributi sensibili presenti nel dataset da anonimizzare;

- *column_score*: misura considerata nella fase di partizionamento per la scelta dell'attributo quasi-identificatore da utilizzare per frammentare il dataset. I valori consentiti sono *entropy* e *span*;
- *fragmentation*: strategia di partizionamento dei dati, che può essere basata sui quantili (*quantile*) oppure multidimensionale (*mondrian*);
- K: parametro di k-Anonymity;
- L: parametro di l-Diversity;
- *measures*: metriche utilizzate per la valutazione della perdita di informazione causata dal processo di anonimizzazione. I valori ammissibili sono *discernability_penalty*, *normalized_certainty_penalty* e *global_certainty_penalty*.

Per l'esecuzione dei test si è scelto di utilizzare il dataset Poker Hand [17], il quale è composto da 1.000.000 di tuple, ciascuna rappresentante una mano di poker costituita da cinque carte. Le carte sono descritte da due attributi: il seme (intero compreso tra 1 e 4) e il numero (intero compreso tra 1 e 13). Queste rappresentano gli attributi quasi-identificatori (dieci in totale), mentre l'attributo sensibile è un attributo addizionale che descrive la classe della mano con un numero intero compreso tra 0 e 9.

Il valore dei parametri di anonimizzazione è stato scelto in modo tale da coprire un buon numero di casi di test e sottoporre il framework a carichi di lavoro di diversa entità. In particolare, il parametro di k-Anonimity è stato fatto variare nell'insieme $\{5, 10, 20\}$, mentre il parametro di l-Diversity ha assunto i valori contenuti nell'insieme $\{2, 3, 4\}$.

Entrambe le strategie di partizionamento del dataset sono state valutate, ossia la strategia basata sui quantili e la strategia multidimensionale. Data la dimensione non eccessiva del dataset da anonimizzare (22,4 MB), non è stato fatto ricorso al campionamento del dataset per la fase di creazione dei frammenti.

Il numero di worker su cui distribuire il carico di lavoro è stato fatto variare da 2 a 11. Lo Spark driver pod e gli Spark executor pods sono stati configurati con 1 CPU core e 2 GB di memoria ciascuno.

Il cluster Kubernetes di test gestito tramite Minikube è stato configurato con quattro nodi, corrispondenti a quattro diverse macchine virtuali KVM, ciascuno dotato di 4 CPU, 16 GB di memoria RAM e 1 disco di dimensione 16 GB.

Le caratteristiche del server utilizzato per ospitare il cluster Minikube sono elencate di seguito:

- **CPU**: Intel Xeon E5-2620 v4 (8 cores, 16 threads, frequenza 2.10 GHz);
- **RAM**: 8x 32GB 2400 MHz (0,4 ns);
- **Dischi**: 2x Samsung SSD 850 PRO 256GB;
- **Sistema operativo**: Ubuntu 20.04 LTS.

Lo stesso server è stato impiegato per l'esecuzione dei test sia del framework in ambiente Kubernetes sia della precedente versione con architettura locale, in modo da poter effettuare un confronto a parità di specifiche hardware.

6.2 Analisi e confronto dei risultati

La valutazione dei risultati si divide in due parti: in primo luogo vengono analizzate le prestazioni del framework Mondrian in ambiente Kubernetes; successivamente, si procede con il confronto dei tempi di esecuzione con la precedente versione locale e la versione centralizzata.

Figura 6.1 mostra i tempi di esecuzione ottenuti dal sistema realizzato in Kubernetes, applicando i due metodi di partizionamento con le diverse possibili combinazioni dei parametri di k-Anonymity e l-Diversity.

I grafici riportati evidenziano che i due metodi di partizionamento non presentano significative differenze in termini di tempistiche, confermando quanto espresso nella trattazione del framework Mondrian realizzato dall'UniBG Seclab [11].

Osservando il trend dei dati è possibile affermare che l'aumento del numero di workers oltre una certa soglia non comporta una diminuzione rilevante dei tempi di esecuzione, ma tende a raggiungere un valore abbastanza costante. Ciò significa

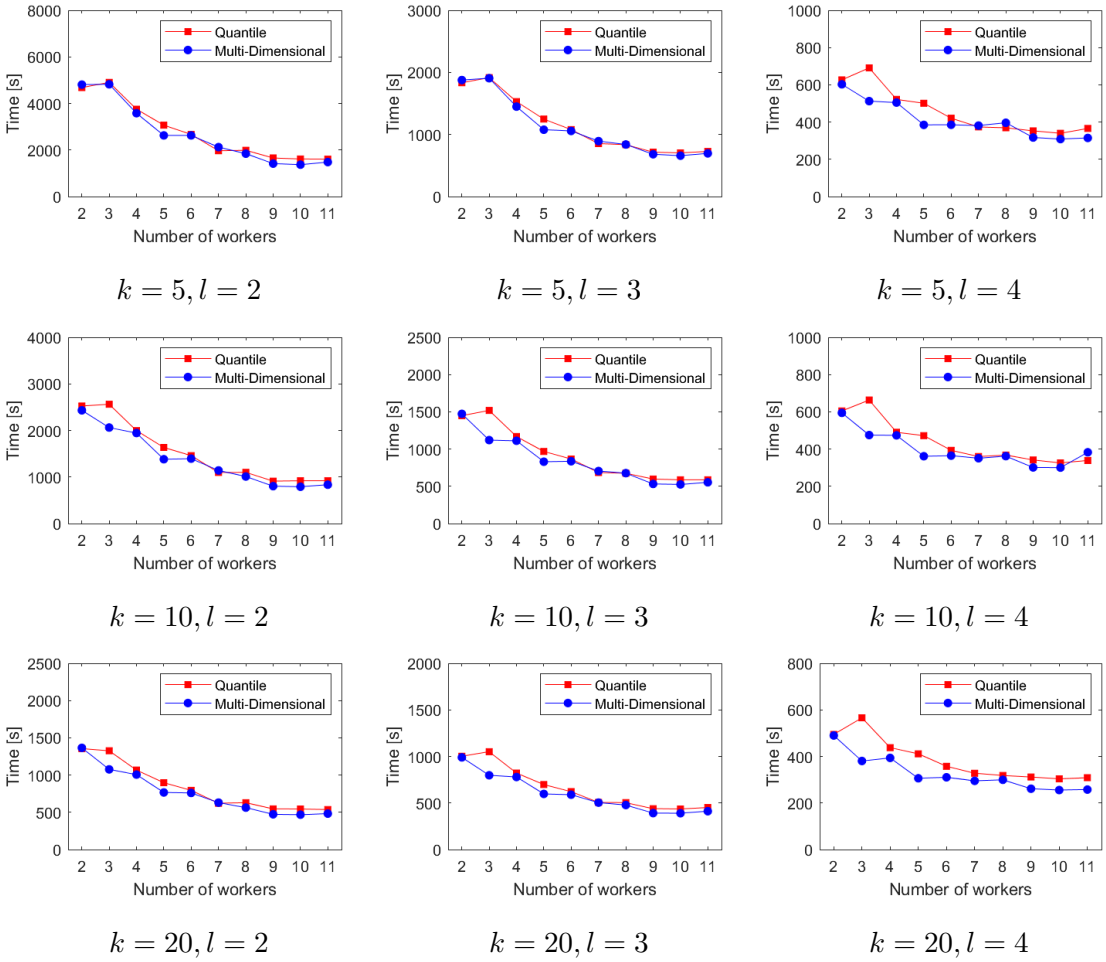


Figura 6.1: Tempi di esecuzione del framework Mondrian in Kubernetes

che, all'aggiunta di un ulteriore worker, la dimensione dei frammenti varia in misura limitata, mantenendo il carico di lavoro pressoché inalterato. In questo caso, essendo la dimensione del dataset di test piuttosto contenuta, superando all'incirca i 4 o 5 wokers, a seconda delle combinazioni dei parametri di anonimizzazione, non si registrano miglioramenti delle performance, evidenziando che l'impiego di risorse nelle attività di creazione e sincronizzazione dei pod, nelle operazioni di lettura e scrittura dei frammenti e in generale nella gestione del cluster, risulta alla pari, o addirittura superiore, rispetto a quello dei task di anonimizzazione dei dati. Variando la dimensione del dataset da anonimizzare, ci si può aspettare che volumi di dati maggiori possano tendere a spostare la soglia verso un numero più alto di workers, in quanto frammenti più grandi consentirebbero di sfruttare meglio un numero maggiore di nodi.

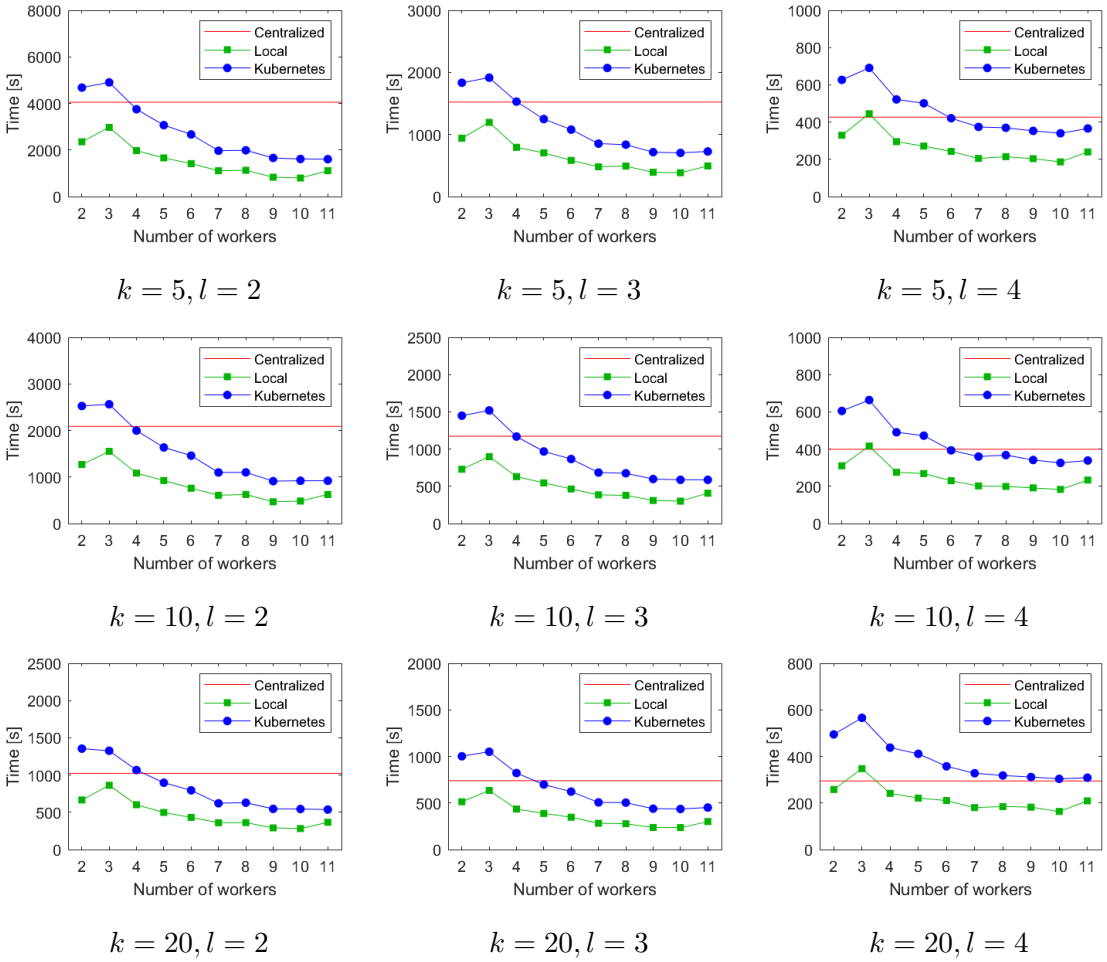


Figura 6.2: Confronto dei tempi di esecuzione tra Mondrian centralizzato, locale e in Kubernetes con partizionamento basato sui quantili

Il monitoraggio dei test in esecuzione ha permesso, inoltre, di mettere in rilievo un vantaggio nell'utilizzo dell'orchestratore Kubernetes. In alcuni casi, quando le risorse assegnate ai pod esecutori non risultano sufficienti per gestire il carico di lavoro o si verificano altre tipologie di guasti, i pod potrebbero andare in errore. In queste situazioni, Kubernetes, grazie alla sua funzionalità di monitoraggio continuo dello stato dei pod in esecuzione, è in grado di intervenire tempestivamente, terminando il pod in errore e sostituendolo con uno nuovo, determinando un impatto limitato sui tempi di esecuzione complessivi.

Passando poi al confronto dei tempi di esecuzione tra framework con architettura locale e framework in Kubernetes, in Figura 6.2 e in Figura 6.3 vengono riportati i dati ottenuti applicando, rispettivamente, il partizionamento basato sui quantili e il

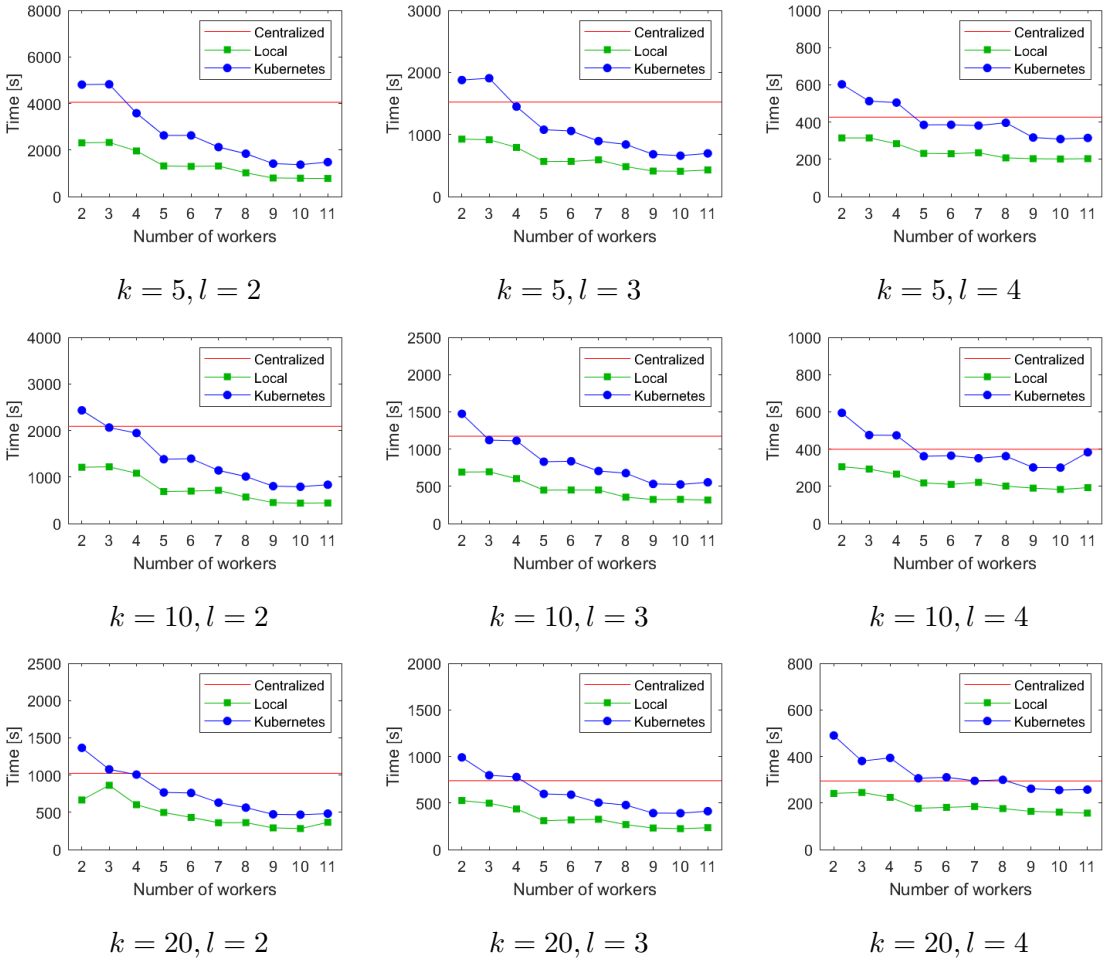


Figura 6.3: Confronto dei tempi di esecuzione tra Mondrian centralizzato, locale e in Kubernetes con partizionamento multidimensionale

partizionamento multidimensionale. Insieme ad essi, sono rappresentati i tempi di esecuzione della versione centralizzata dell'algoritmo Mondrian, che sfrutta un solo esecutore per l'anonimizzazione dell'intero dataset.

Con entrambi i metodi di frammentazione, i grafici mostrano un andamento del tutto analogo ma con il sistema realizzato in Kubernetes che ottiene tempi di esecuzione superiori rispetto alla versione locale e in alcuni casi, quando vengono impiegati pochi workers, anche alla versione centralizzata. Ciò non indica che Kubernetes comporta un peggioramento delle performance. Al contrario, risulta motivabile da una serie di considerazioni.

Innanzitutto, Kubernetes è un orchestratore avanzato che offre un numero elevato di funzionalità complesse, le quali impattano necessariamente sul computo totale

dell'utilizzazione delle risorse dell'host sottraendone, seppur limitatamente, la disponibilità alle applicazioni containerizzate. Tra i processi principali si trovano quelli del control plane (api server, etcd, scheduler e controller manager), che gestiscono la schedulazione dei pod, le richieste interne ed esterne al cluster e il monitoraggio delle risorse, e quelli dei nodi worker (kubelet e kube-proxy), che gestiscono il ciclo di vita dei pod e le comunicazioni tra i vari nodi. Oltre agli strumenti nativi di Kubernetes, sono presenti anche le altre componenti installate nel cluster per questa specifica applicazione, in particolare il cluster di archiviazione Ceph e gli operatori Kubernetes di Rook e Spark. L'insieme di tutti questi elementi determina un carico di lavoro rilevante, che si somma a quello dei task di anonimizzazione eseguiti dagli Spark pod.

Anche in questo discorso, la dimensione del dataset da anonimizzare assume un ruolo importante. Infatti, con grandi volumi di dati le operazioni di anonimizzazione prevalgono sul resto, rendendo trascurabile l'effetto dei processi di Kubernetes.

Infine, un'ulteriore fattore da considerare riguarda la configurazione dell'ambiente di test. Per questo progetto è stato utilizzato un cluster Minikube composto da macchine virtuali, scelta guidata dall'esigenza di disporre di dischi virtuali per l'installazione del sistema di storage distribuito Ceph. Nonostante il driver KVM2 sia piuttosto performante nella gestione delle macchine virtuali, la virtualizzazione in sé introduce notoriamente un overhead computazionale.

Per quanto riguarda, invece, la valutazione della perdita di informazione causata dalla generalizzazione degli attributi quasi-identificatori, i valori assunti dalle metriche considerate risultano pressoché identici nelle due versioni del framework, a riprova del fatto che la logica applicativa è rimasta invariata nell'implementazione del porting.

In sintesi, i test eseguiti non sono stati in grado di quantificare la capacità di scalabilità orizzontale offerta da Kubernetes, in quanto sono stati eseguiti su un singolo nodo e a parità di risorse con la soluzione locale. Tuttavia, il giudizio complessivo è molto positivo, poiché evidenzia che il nuovo sistema ottiene performance in linea con quelle della versione precedente, integrando una serie di funzionalità avanzate in un ambiente altamente affidabile e flessibile.

Capitolo 7

Considerazioni e conclusioni

La tutela dei dati sensibili costituisce al giorno d'oggi un tema di importanza cruciale per garantire la privacy degli individui e richiede l'adozione di adeguate misure di protezione. L'anonimizzazione dei dati rappresenta una tecnica avanzata che consente di rendere i dati non associabili ai soggetti a cui fanno riferimento, in modo da contrastare gli attacchi di reidentificazione e consentire la pubblicazione delle informazioni in conformità della normativa. Il framework Mondrian realizzato dall'UniBG Seclab si propone come soluzione efficiente per l'esecuzione del processo di anonimizzazione dei dati in maniera distribuita, applicando i concetti di k-Anonymity e l-Diversity.

L'obiettivo di questo progetto di tesi era la trasposizione del framework Mondrian in ambiente Kubernetes, per superare i limiti di un'architettura locale incentrata su Docker e HDFS e migliorare le proprietà di scalabilità e affidabilità del sistema. Allo scopo di automatizzare e semplificare la fase di sviluppo, è stato fatto ricorso agli operatori Kubernetes di Spark, per la gestione della distribuzione dei task di esecuzione, e Rook, per l'implementazione dello storage distribuito Ceph.

La valutazione dei risultati ottenuti nella fase sperimentale ha evidenziato un comportamento positivo del sistema, in linea con le aspettative. Il confronto con la versione precedente del framework ha mostrato un aumento dei tempi di esecuzione contenuto, riconducibile al maggior impiego di risorse dovuto ai processi di Kubernetes per la gestione del cluster di nodi e alla particolare configurazione dell'ambiente di test.

Questo studio rappresenta un punto di partenza per la realizzazione di un sistema avanzato di anonimizzazione distribuita di dati. L'estensione dei test a contesti più complessi, caratterizzati da ampi volumi di dati e grandi cluster di nodi distribuiti in rete, consentirebbe di approfondire l'analisi e indagare le performance del framework in ambienti reali.

In conclusione, si può affermare che il porting del framework di anonimizzazione distribuita Mondrian in ambiente Kubernetes ha consentito di ottenere un sistema affidabile, in grado di integrare diverse funzionalità di orchestrazione avanzate e servizi offerti da fornitori terzi e garantire scalabilità nell'esecuzione di algoritmi di anonimizzazione di dati.

Bibliografia

[1] IDC e Statista. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (in zettabytes)*. Giu. 2021. URL: https://www.statista.com/statistics/871513/worldwide-data-created (visitato il 01/02/2024).

[2] National Institute of Standards e Computer Security Resource Center (CSRS) Technology. *INFOSEC*. URL: https://csrc.nist.gov/glossary/term/INFOSEC (visitato il 04/02/2024).

[3] European Union Agency for Cybersecurity (ENISA). *ENISA Threat Landscape 2022*. Nov. 2022. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022 (visitato il 04/02/2024).

[4] Kaspersky Lab e B2B International. *The Human Factor in IT Security: How Employees are Making Businesses Vulnerable from Within*. 2017. URL: https://www.kaspersky.com/blog/the-human-factor-in-it-security (visitato il 04/02/2024).

[5] Parlamento Europeo e Consiglio Europeo. *REGOLAMENTO (UE) 2016/679 DEL PARLAMENTO EUROPEO E DEL CONSIGLIO del 27 aprile 2016*. Mag. 2016. URL: https://eur-lex.europa.eu/legal-content/IT/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ITA&toc=OJ:L:2016:119:TOC (visitato il 05/02/2024).

[6] P. Samarati. «Protecting respondents identities in microdata release». In: *IEEE Transactions on Knowledge and Data Engineering* (2001). DOI: 10.1109/69.971193.

[7] R.J. Bayardo e A. Rakesh. «Data privacy through optimal k-anonymization». In: *21st International Conference on Data Engineering (ICDE'05)*. 2005. DOI: 10.1109/ICDE.2005.42.

[8] Kristen LeFevre, David J. DeWitt e Raghu Ramakrishnan. «Incognito: efficient full-domain K-anonymity». In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2005. DOI: 10.1145/1066157.1066164.

[9] *Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and Owner control*. URL: https://cordis.europa.eu/project/id/825333.

[10] LeFevre K., DeWitt D.J. e Ramakrishnan R. «Mondrian Multidimensional K-Anonymity». In: *22nd International Conference on Data Engineering (ICDE'06)*. 2006.

[11] Sabrina De Capitani di Vimercati et al. «Scalable Distributed Data Anonymization». In: *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. 2021.

[12] *Hadoop File System (HDFS)*. URL: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html.

[13] *Apache Spark*. URL: https://spark.apache.org/.

[14] *Understanding your Apache Spark Application Through Visualization*. URL: https://www.databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html.

[15] *Pandas*. URL: https://pandas.pydata.org/.

[16] *Docker*. URL: https://www.docker.com/.

[17] Robert Cattral e Franz Oppacher. *Poker Hand*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5KW38. 2007.

[18] *Kubernetes*. URL: https://kubernetes.io/.

[19] *Cloud Native Computing Foundation*. URL: https://www.cncf.io/.

[20] *Kubernetes: cos'è, come funziona, pro e contro, quando usarlo*. URL: https://blog.sparkfabrik.com/it/guides/kubernetes-guida-completa.

[21] *Helm - The package manager for Kubernetes*. URL: https://helm.sh/.

[22] *Kubernetes Operators: what are they? Some examples*. URL: https://blog.sparkfabrik.com/en/what-are-kubernetes-operators.

[23] *Spark Operator*. URL: https://github.com/GoogleCloudPlatform/spark-on-k8s-operator.

[24] *Running Spark on Kubernetes*. URL: https://spark.apache.org/docs/latest/running-on-kubernetes.html.

[25] *Spark operator architecture diagram*. URL: https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/docs/architecture-diagram.png.

[26] *Rook - Open-Source, Cloud-Native Storage for Kubernetes*. URL: https://rook.io/.

[27] *Ceph*. URL: https://ceph.io/en/.

[28] *Storage Architecture - Rook Ceph Documentation*. URL: https://rook.io/docs/rook/latest-release/Getting-Started/storage-architecture/.

[29] *Kubeadm*. URL: https://kubernetes.io/docs/reference/setup-tools/kubeadm/.

[30] *Amazon Elastic Kubernetes Service*. URL: https://aws.amazon.com/it/eks.

[31] *Google Kubernetes Engine*. URL: https://cloud.google.com/kubernetes-engine?hl=it.

[32] *Azure Kubernetes Service*. URL: https://azure.microsoft.com/it-it/products/kubernetes-service.

[33] *kind - Kubernetes in Docker*. URL: https://kind.sigs.k8s.io/.

[34] *k3d*. URL: https://k3d.io/v5.6.0/.

[35] *Minikube*. URL: https://minikube.sigs.k8s.io/docs/.

[36] *Apache Arrow*. URL: https://arrow.apache.org/.

[37] Michele Beretta. *UniBG LaTeX Thesis Template*. URL: https://github.com/micheleberetta98/unibg-thesis-template.

Ringraziamenti

Desidero ringraziare l'UniBG Seclab per aver fornito il template LaTeX [37] utilizzato per la scrittura di questa tesi e per tutto il supporto dato nel corso di questi mesi, e il mio relatore, Prof. Stefano Paraboschi, per la stima e la considerazione mostrata nei miei confronti.