GESTIONE DI DATI SCIENTIFICI TRAMITE TRACCIAMENTO DEGLI EVENTI IN FILE SYSTEM DISTRIBUITI

Candidato **Tiziano De Cristofaro**
Relatore **Anna Ciampolini**
Correlatori **Antonio Falabella, Enrico Fattibene**

Laurea Magistrale in Ingegneria Informatica
Alma Mater Studiorum - Università di Bologna

Indice

1 Introduzione

1.1 Tema del lavoro di tesi

Il mio lavoro riguarda la gestione delle risorse di calcolo in un grande data center per collaborazioni scientifiche internazionali.
Le grandi collaborazioni scientifiche fanno uso di ingenti risorse informatiche per l'analisi dei dati prodotti dai loro apparati sperimentali. Tali risorse sono sia di calcolo che di archiviazione, e sono generalmente distribuite geograficamente su scala nazionale ed internazionale e condivise tra varie comunità scientifiche.
Considerata la complessità indotta dagli aspetti appena descritti, si delinea immediatamente la problematica della gestione efficiente di una grande quantità di risorse. In particolare l'attenzione di questo lavoro è rivolta allo storage, che deve garantire archiviazione e processamento di grandi quantità di dati in modo efficiente ed affidabile. La dimensione dello spazio di archiviazione installato nel centro di calcolo richiede che questo sia basato su dei file system distribuiti.
Un file system in generale può essere gestito e monitorato in maniera efficiente intercettando quelli che vengono chiamati eventi, ovvero le operazioni che avvengono all'interno di esso, come ad esempio lettura/scrittura di file o di una cartella.
Questi eventi corrispondono in genere a chiamate fatte al sistema operativo e non sono visibili né all'utente, né all'amministratore del file system. Un metodo per verificare a posteriori lo stato di un file system in un determinato momento consiste nella scansione dell'albero delle directory alla ricerca delle modifiche avvenute. Questo approccio impegna periodicamente delle risorse per eseguire il controllo delle modifiche al file system, inoltre non ci permette di ottenere nessun aggiornamento in tempo reale ma solo in seguito a una scansione.
Questo meccanismo può essere superato grazie all'impiego di strumenti specifici per la cattura di questi eventi. Gli strumenti possono essere offerti dal file system stesso oppure da componenti software esterni.
Il lavoro di tesi punta ad analizzare e confrontare implementazioni ed approcci diversi per la cattura di eventi in un file system.

1.2 Il centro di calcolo Tier-1 del CNAF

Il CNAF (Centro Nazionale Analisi Fotogrammi) è il centro per la ricerca e lo sviluppo di tecnologie informatiche e telematiche dell'INFN (Istituto Nazionale di Fisica Nucleare).
La sua attività è quella di fornire le risorse informatiche alle collaborazioni scientifiche in cui l'INFN è impegnato. Il CNAF ospita ed amministra il più grande centro di calcolo dell'INFN sul suolo nazionale.
Una buona parte delle risorse (circa l'80%) vengono impiegate per soddisfare i requisiti di calcolo e di archiviazione/gestione dati dei quattro principali esperimenti (Atlas, Alice, CMS, LHCb) installati presso l'acceleratore di particelle Large Hadron Collider (LHC) al CERN di Ginevra.
L'attività di calcolo degli esperimenti svolti al CERN è distribuita su scala mondiale in una griglia di centri di calcolo denominata WLCG (Worldwide LHC Computing Grid). All'interno di questa griglia di 170 siti in 42 nazioni diverse, i centri di calcolo sono organizzati in un'architettura cosiddetta a "Tier" di livello 0, 1, 2 e 3. Questa classificazione viene fatta in base al tipo di risorsa e alla qualità di servizio fornita. Il Tier-0 (unico e localizzato a Ginevra) è il centro che offre risorse di calcolo ed ospita la prima copia dei dati prodotti dagli apparati sperimentali. I centri di tipo Tier-1 (13 siti) offrono ulteriori risorse di calcolo e ospitano una seconda copia dei dati. In genere i siti Tier-1 devono fornire una certa qualità di servizio: sono online tutte le 24 ore tutti i 7 giorni della settimana, e si impegnano a mantenere la loro copia dei dati.
I centri Tier-2 e Tier-3 tendono ad essere più piccoli e non sono impegnati nella custodia dei dati ma solo nella parte computazionale.
Il CNAF è un centro di tipo Tier-1 e, per quel che riguarda l'attività scientifica, i principali servizi sono: calcolo, archiviazione e rete.

Calcolo Questo servizio viene messo a disposizione attraverso una farm di nodi di calcolo di circa 900 server. Grazie a una connessione veloce (400 Gbit/s), una parte dei job di calcolo vengono eseguiti su risorse installate presso il centro di Calcolo del Cineca, situato a pochi chilometri di distanza dal CNAF.

Archiviazione Lo storage è relativo invece all'archiviazione dei dati e alla loro gestione. Questa attività è svolta impiegando come supporti sia dischi magnetici che nastri (tape). Tipicamente i dischi magnetici forniscono delle

prestazioni più elevate e hanno un costo maggiore, mentre i tape hanno performance inferiori e un costo più basso. Queste proprietà rendono i dischi più adatti all'archiviazione di dati recenti e usati frequentemente, e i tape al contenimento di dati storici acceduti solo di rado. Il passaggio dei dati da disco a tape e viceversa è un'attività molto importante per il reparto storage, ed è fondamentale che sia svolta in modo efficiente.

Rete Per un centro di calcolo moderno inserito in un contesto di collaborazioni internazionali, la capacità di Input/Output costituisce un fattore fondamentale. La necessità di elaborare grandi quantità di dati con un grande numero di CPU, ha portato il CNAF a disegnare una rete locale ad alte prestazioni in grado di consentire una banda di accesso ai dati dell'ordine delle centinaia di Gigabit per secondo. I collegamenti verso gli altri centri di calcolo (Tier-0 e Tier-2) sono dell'ordine dei 100 Gbit/s.

1.3 File system distribuiti

Un file system distribuito è un particolare tipo di file system pensato per essere utilizzato su nodi distinti distribuiti all'interno di una rete. Ha lo scopo di dare all'utente finale una visione uniforme e coerente delle risorse, in modo da rendere trasparente la dislocazione fisica dei dati e dare l'impressione di operare in locale.
Uno degli aspetti critici per una tecnologia di questo tipo è la gestione degli accessi concorrenti ai dati. Ci si aspetta infatti che molteplici nodi accedano ai dati concorrentemente sia in lettura che in scrittura, ed è necessario regolare queste operazioni in modo da mantenere la coerenza e l'integrità dei dati. Per questo motivo spesso i file system distribuiti aderiscono allo standard POSIX.
I due principali esempi presi in considerazione da questo lavoro di tesi sono GPFS e Ceph.

GPFS (General Parallel File System - anche noto come Spectrum Scale) è un file system distribuito proprietario, introdotto da IBM a partire dal 1998[1][2]. È usato in produzione al CNAF.

Ceph nasce come object storage distribuito, ma l'introduzione di CephFS ha permesso di utilizzarlo anche come file system POSIX. È una tecnologia

gratuita e open source, che ha recentemente riscosso un grande successo, e anche per questo è supportata da una community molto estesa ed attiva[3]. Al CNAF Ceph è utilizzato come sistema storage di backend per infrastrutture Cloud basate su OpenStack. Inoltre è in corso una attività di valutazione di questo strumento come file system per ospitare i dati degli esperimenti scientifici.

1.4 Concetto di storage event

Gli storage events sono eventi che si verificano all'interno di un file system, e riguardano le principali operazioni che possono essere svolte su di esso. Al verificarsi di uno storage event questo viene immediatamente notificato, per cui è possibile avere tutti gli aggiornamenti in tempo reale ed eventualmente compiere delle azioni per reagire a determinati eventi[4].
Attualmente gli storage events stanno destando interesse in ambito scientifico. Un buon esempio è la realizzazione di dCache, un sistema per l'archiviazione distribuita di grandi quantità di dati, utilizzato in molti centri di livello Tier-1[5].
Lo standard POSIX definisce delle system call (o chiamate di sistema) relative a vari eventi che si verificano su file system. Le tecnologie che andremo ad analizzare sono in grado di intercettare le system call POSIX, e di generare delle notifiche per ognuna di esse.
Di seguito sono riportati alcuni esempi di eventi che possono verificarsi su un file system, indicando per ognuno le corrispondenti system call POSIX:

- creazione di un file - create(), open()
- modifica a un file - write()
- lettura di un file - read()
- eliminazione di un file - remove()
- accesso a una directory - open()
- creazione di una directory - create(), open()
- rimozione di una directory - remove()

Un primo esempio di storage events può essere trovato su Inotify, programma incluso nel kernel di Linux, che definisce degli eventi simili a quelli elencati proprio basandosi sullo standard POSIX. Inotify permette di osservare cambiamenti all'interno di un determinato punto del file system, ed eventualmente notificarli ad altre applicazioni. È disponibile sia come API in linguaggio

C che come comando nel terminale. Inotify è comunque uno strumento esterno non integrato nel file system, il rilevamento degli eventi richiede quindi che questi siano noti al kernel. Nel caso di file system distribuiti potrebbero esserci problemi dovuti al fatto che gli eventi che si verificano su un certo nodo sono visibili a livello kernel solo localmente[6].
GPFS fornisce un completo supporto agli storage events dalla versione 5.0.3. Tramite la funzionalità Clustered Watch Folder è possibile osservare un file system distribuito e ottenere notifiche per eventi costruiti sul modello di Inotify.
Ceph non ha una funzionalità che gestisce gli storage events esplicitamente, ma la sua architettura permette di catturarli in modo semplice. Ceph infatti è basato su un object storage, che normalmente non prevede l'uso di metadati. Per questo motivo il file system di Ceph (CephFS) utilizza dei Metadata Server dedicati, che tengono traccia degli eventi che si verificano. È quindi sufficiente guardare nei log di questi server per vedere gli eventi che vengono registrati.

1.5 Situazione attuale presso il CNAF

Allo stato attuale il CNAF non utilizza in produzione alcuna tecnologia che utilizzi gli storage events.
Il tracciamento viene svolto a posteriori attraverso scansioni, con cui si analizzano eventuali modifiche avvenute sui file o alla struttura delle directory. Questa operazione per file system piccoli genera un carico trascurabile, ma per file system di diversi Petabyte con milioni di file e un annidamento di directory di diversi livelli diventa computazionalmente intensiva.
Il CNAF prevede inoltre di iniziare nel 2022 la migrazione in una nuova struttura presso il nuovo Tecnopolo di Bologna, che sarà completata probabilmente nel 2023. Questo spostamento coinciderà con un aumento delle dimensioni del centro per far fronte alle future esigenze computazionali degli esperimenti scientifici, primi fra tutti quelli dell'LHC al CERN. Si stima infatti che nell'arco di 4/5 anni il carico di lavoro per il CNAF potrebbe risultare doppio rispetto a quello attuale, in termini di quantità di spazio, rate di accesso ai dati e risorse di calcolo.
Tra le operazioni che coinvolgono i servizi di storage del CNAF, quelle che potrebbero trarre più giovamento da una evoluzione del metodo di rilevazione di eventi occorsi sul file system sono la migrazione dei dati da disco a tape e la copia automatica di file su altri siti. Gli eventi relativi ad entrambe ven-

gono attualmente rilevati con delle scansioni dell'intero file system, e la loro durata è influenzata negativamente dalle grandi dimensioni del file system GPFS (decine di milioni di file).
Nel contesto appena descritto l'utilizzo degli storage events rappresenterebbe un vantaggio consistente.

2 Ambiente di lavoro

In questa sezione sarà descritto l'approccio utilizzato per studiare le varie tecnologie e per misurarne le performance.

2.1 Metodologia di test

Le tecnologie in analisi sono state esaminate svolgendo concretamente delle operazioni sui file system distribuiti, e controllando che il numero di notifiche corrispondesse a quello atteso.
L'esecuzione delle operazioni su file system è stata automatizzata grazie ad uno script in linguaggio Python che permette di eseguire vari tipi di comandi in sequenza, come ad esempio creazione di file, scritture su file, creazioni di directory, letture di file, ecc...
Lo script prende in input il mountpoint del file system distribuito, il numero di operazioni da eseguire e il tipo (lettura, scrittura, altro). Durante l'esecuzione vengono stampati a video per ogni operazione l'esito e lo stato di avanzamento dello script.
Di seguito è possibile vedere i dettagli rilevanti dello script descritto in questo paragrafo:

```
OPERATIONS = [ "cp", "mv", "rm", "mkdir", "find",
    "finddir", "findfile",  "append", "read", "write" ]
```

Operazioni di cui è possibile richiedere l'esecuzione

```
def get_random_size():
    return random.randint(0,102400)
```

Quando si creano dei file viene scelta una dimensione casuale per ognuno di essi. Modificando i parametri passati a questa funzione è possibile scegliere rispettivamente il limite minimo e massimo alla dimensione casuale.

```
def op_cp():
    if random.randint(0,1):
    src_file=op_randfile()
    if not src_file:
        return 1
    else:
        src_file=op_randdir()
    dst_file=op_randdir() + '/' + get_random_name()
    try:
```

```
        shutil.copytree(src_file,dst_file)
    except shutil.Error, e:
        return 2
    except OSError, e:
        return 3
    finally:
        pass
```

Esempio di operazione effettuata dallo script. In questo caso si tratta di una copia. La funzione shutil.copytree(src_file,dst_file) prende una directory root ed esegue la copia ricorsiva (quindi includendo tutte le sottocartelle) nel path specificato come destinazione.

2.2 Descrizione del testbed

Lo studio e i test concreti sulle tecnologie sono stati resi possibili grazie a un testbed messo a disposizione presso il CNAF. Il testbed consta di due cluster differenti, destinati rispettivamente a GPFS e Ceph.
Nel cluster GPFS sono presenti quattro nodi, il cluster Ceph invece si compone di otto nodi. Si tratta in tutti i casi di macchine Linux su cui è stato installato CentOS 7[7]. Su tutte è presente un account inserito tra i sudoers, visto che tutte le operazioni richiedono l'accesso come super utente.
In un secondo momento sono stati aggiunti al cluster GPFS anche tutti i nodi inizialmente dedicati a Ceph. Ciò ha permesso lo svolgimento di test su una scala maggiore per le tecnologie, in modo da simulare situazioni molto più realistiche rispetto alle prime prove.

3 Soluzioni in ambiente Unix/Linux

3.1 Introduzione a Inotify

Un primo test sugli storage events può essere sperimentato con Inotify. Si tratta di un programma incluso di default all'interno del kernel Linux, la cui funzione è proprio quella di rilevare eventi che si verificano nel file system e generare le relative notifiche.
Come già anticipato nel capitolo 1, Inotify è uno strumento esterno al file system, per la cattura degli eventi. In questo lavoro sono state preferite facility offerte nativamente dai file system. Lo studio di Inotify è comunque utile per comprendere e verificare l'approccio basato sugli eventi.

3.1.1 Eventi in Inotify

Di seguito sono elencati e descritti gli eventi rilevati da Inotify, che in questo contesto sono detti "filesystem events".
La loro comprensione è utile anche nell'analisi di altre tecnologie, che propongono eventi simili (come accade in GPFS).
Per poter rilevare gli eventi su file system è necessario specificare una risorsa da osservare, può trattarsi di un singolo file o di una directory. In questo secondo caso si rilevano gli eventi per tutti i file (o altre directory) al suo interno.

- IN_ACCESS: un file è stato acceduto, ad esempio per eseguire un'operazione di lettura.
- IN_ATTRIB: i metadati di un file sono stati modificati. Questo evento potrebbe essere scatenato da un cambio di permessi con il comando chmod.
- IN_CLOSE_WRITE: un file aperto in scrittura è stato chiuso.
- IN_CLOSE_NOWRITE: un file aperto non in scrittura è stato chiuso.
- IN_CREATE: un file o una directory è stato creato. Tipicamente notificato in seguito a operazioni touch o mkdir.
- IN_DELETE: un file o una directory è stato rimosso, ad esempio in seguito ad un'operazione rm.
- IN_DELETE_SELF: la risorsa osservata stessa è stata eliminata.
- IN_MODIFY: un file è stato modificato.
- IN_MOVE_SELF: la risorsa osservata stessa è stata spostata.

- IN_MOVED_FROM: un file è stato rinominato in seguito a un comando mv. Questa notifica viene generata per il file con il nome originario.
- IN_MOVED_TO: un file è stato rinominato. Questa notifica viene generata per il file con il nome di destinazione.
- IN_OPEN: un file o una directory è stato aperto. Viene lanciata ad esempio quando si esegue cd su una directory.
- IN_ISDIR: associato ad un altro evento. Viene generata se l'oggetto interessato dall'evento associato è una cartella.

3.2 Software installato

Inotify viene fornito da Linux come una libreria in linguaggio C, da utilizzare per lo sviluppo di applicazioni.
In questo lavoro si è optato per un approccio più flessibile, in modo da riuscire ad utilizzare Inotify come comando da lanciare interattivamente. Per questo motivo è stato installato un software aggiuntivo chiamato inotify-tools[8], che permette di ricevere in tempo reale le notifiche per gli storage events che si verificano.

3.3 Test effettuati

I test più rilevanti per l'attività del CNAF sono sicuramente quelli che riguardano le operazioni di scrittura. Per questo motivo si è deciso di procedere con dei test sulla creazione di file.
Per richiedere a inotify-tools di mettersi in attesa di notifiche bisogna utilizzare il comando inotifywait.
Il primo test effettuato è incentrato esclusivamente sulla ricezione di notifiche di tipo IN_CREATE. Si è cominciato con delle prove minimali, creando pochissimi file e ottenendo l'esatto numero di notifiche atteso. Il test più significativo è stato però quello lanciato con la creazione di 10.000 file, e anche questo ha dato esito positivo in quanto sono state ricevute esattamente 10.000 notifiche IN_CREATE.

Il seguente comando utilizza lo script per generare 10.000 operazioni di scrittura sul file system:

```
./fstester -t 10000 -f /gpfs/test_inotify/ --write
```

- -t: numero di operazioni da svolgere

- -f: percorso in cui scrivere
- --write: tipo di operazioni da effettuare, in questo caso solo scritture

Questo il comando utilizzato per la ricezione delle notifiche in questo test:

```
inotifywait -r -m -e CREATE /gpfs/test_inotify/
```

- -r: modalità ricorsiva, saranno osservate anche tutte le sottocartelle del percorso indicato.
- -m: monitor, forza il programma a rimanere in esecuzione per un lasso di tempo illimitato. Se questa opzione non fosse specificata il programma terminerebbe la propria esecuzione dopo la ricezione di una sola notifica.
- -e: evento, permette di specificare i tipi di eventi che si vogliono rilevare e ricevere. In questo caso è stato specificato solo CREATE.
- /gpfs/test_inotify: percorso da osservare. Si tratta di una posizione creata appositamente per contenere i file da rilevare con inotify. Si trova all'interno di /gpfs, che è il mountpoint del file system distribuito sul cluster GPFS.

Il secondo test è stato svolto su 10.000 operazioni di scrittura come il primo, ma in questo caso non sono state poste limitazioni sul tipo di notifiche generate. Sono stati quindi intercettati tutti i possibili eventi. Anche questo test ha dato esito positivo: sono state generate 48.748 notifiche, di cui 10.000 IN_CREATE, 10.000 IN_OPEN, 18.748 IN_MODIFY e 10.000 IN_CLOSE_WRITE.
Il comando per la ricezione delle notifiche risulta leggermente diverso per questo test, in quanto manca il flag per il filtraggio delle notifiche:

```
inotifywait -r -m /gpfs/test_inotify/
```

La durata dei due test è stata rispettivamente di 23 e 22 minuti, possiamo quindi dedurre che, in queste condizioni, l'intercettazione di un maggior numero di notifiche non impatta sulle performance del sistema.

Tipo test	N° operazioni	Tipo notifiche	N° notifiche ricevute	Dimensione max file	Durata test
scrittura	10.000	IN_CREATE	10.000	90 MB	23 min
scrittura	10.000	tutte	48.748	90 MB	22 min

4 IBM Spectrum Scale (GPFS)

4.1 Introduzione

IBM Spectrum Scale (in precedenza noto come GPFS - General Parallel File System) è attualmente il principale file system distribuito adottato dal CNAF[9].
Si tratta di una soluzione proprietaria consolidata ormai da diversi anni, in grado di estendersi su migliaia di nodi e di garantire una grande scalabilità e affidabilità.
Oltre ad essere un file system distribuito contiene anche una suite di tool utili per la sua gestione.
Essendo multipiattaforma, non è vincolante dal punto di vista del sistema operativo: può lavorare su server Linux, Microsoft Server o AIX (sistema operativo proprietario di IBM basato su Unix).
Come anticipato nel primo capitolo, aderisce all'interfaccia standard POSIX. Questo gli permette di interagire con applicazioni che rispettano lo stesso standard, per cui l'adozione di GPFS non richiede che le applicazioni POSIX precedentemente sviluppate vengano riscritte.
Dal punto di vista architetturale Spectrum Scale non è considerabile come un sistema client/server puro, essendo prevista la presenza di più nodi server contemporaneamente, che porta a un superamento del problema del single point of failure. Si tratta di un punto debole tipico delle architetture centralizzate: le operazioni principali sono assegnate a un singolo elemento solitamente più potente degli altri, ma nel caso di problemi con questo il sistema rimane sprovvisto di funzionalità fondamentali e potrebbe anche dover interrompere il proprio servizio.
Allo stesso tempo bisogna tenere presente che al momento del deployment i vari nodi vengono etichettati come "client" o "server". Questo porta a dei risvolti dal punto di vista operativo: i nodi client tipicamente consumano dati, mentre i server sono visti come produttori. I server possono inoltre svolgere funzioni da client, ma non viceversa.
GPFS è in grado di fornire ottime prestazioni di I/O grazie ai diversi meccanismi dedicati: multithreading, ottimizzazione per accessi sequenziali ai dati, data striping, pagepool (memoria cache utilizzata esclusivamente da GPFS).
Molte funzioni di gestione del file system sono affidate a dei nodi ben precisi:

- Metanode: mantiene l'integrità dei metadati.

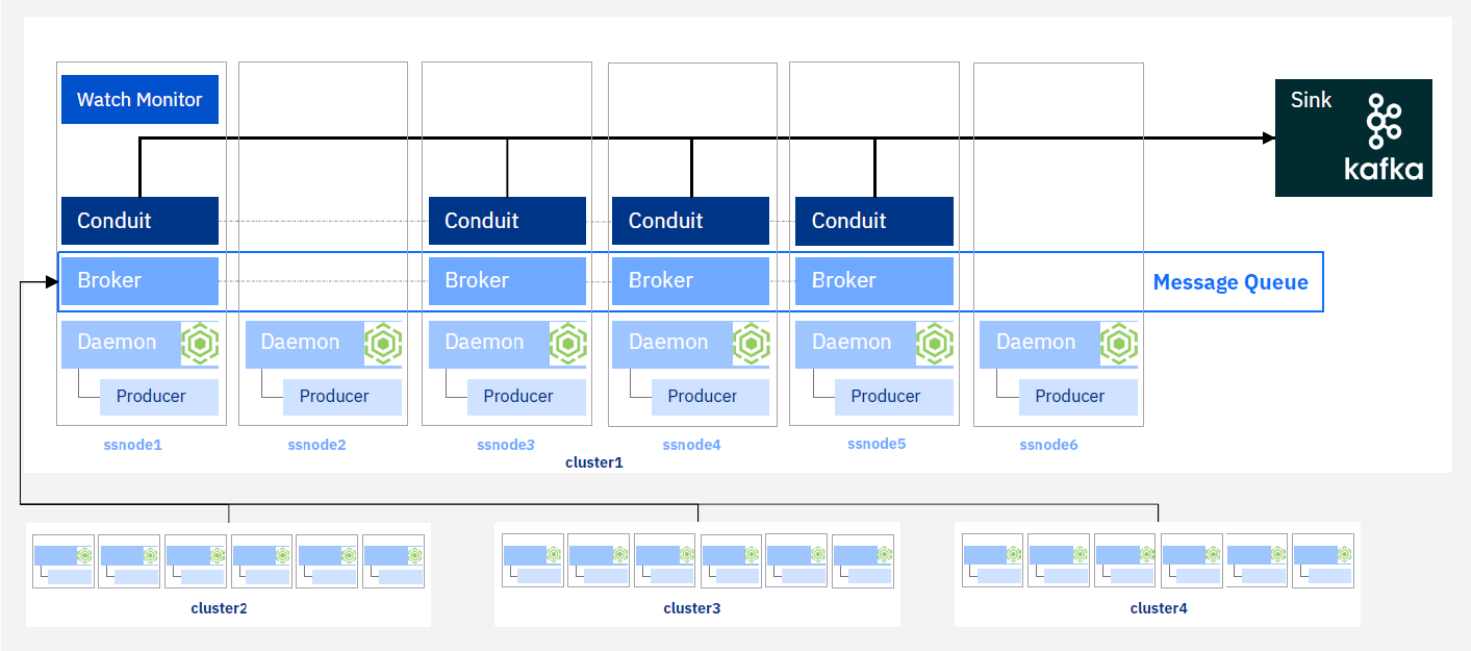
- Configuration Manager: gestisce il file di configurazione sui vari nodi. È presente anche un secondo Configuration Manager di backup, che deve essere attivo se si vuole modificare la configurazione.
- Cluster Manager: copre diverse funzioni di gestione del cluster, come ad esempio il controllo dello stato di salute tramite heartbeat o il coordinamento del ripristino in caso di errori.
- File System Managers: gestiscono vari aspetti del file system, come ad esempio l'aggiunta/rimozione di dischi o l'allocazione dello spazio su ognuno di essi.
- Token Managers: garantiscono l'accesso alle risorse su disco tramite un meccanismo di token.
- Quorum Nodes: sono preposti a definire il quorum per il cluster. GPFS usa il meccanismo di quorum per mantenere la consistenza del cluster in caso di fallimento di uno o più nodi. La maggioranza dei nodi quorum deve rimanere attiva.

4.2 Funzionamento degli storage events in GPFS

A partire dalla versione 5.0.3 è stata introdotta in GPFS la possibilità di lavorare con gli storage events, grazie a una funzionalità chiamata Clustered Watch Folder.
Clustered Watch Folder permette di osservare dei mountpoint specificati dall'utente e rilevare da essi degli eventi di interesse. Per fare ciò vengono creati dei "watch", che restano in osservazione di un determinato mountpoint e considerano un set di eventi indicato dall'utente al momento di creazione. I watch sono da considerare come entità distribuite, che lavorano contemporaneamente su tutti i nodi del cluster indicato.
A fronte di ognuno degli eventi intercettati viene generata una notifica, che viene poi inviata ad un "sink", ovvero un'applicazione esterna designata per la ricezione e la gestione delle notifiche. L'unico sink attualmente supportato è Apache Kafka, una piattaforma open source per la comunicazione di eventi in tempo reale[10]. Può essere visto come un broker di messaggi, che vengono immessi da dei producer e consegnati a dei consumer. Il bus per i messaggi è diviso in canali detti topic. Per la produzione o il consumo di un messaggio bisogna quindi specificare il topic di interesse.
I tipi di eventi disponibili in Clustered Watch Folder sono gli stessi già mostrati per Inotify.

4.2.1 Architettura di Clustered Watch Folder

Di seguito viene mostrato uno schema architetturale di Clustered Watch Folder[11], corredato da una descrizione dei componenti principali[12].



- Producer: quando si verifica un evento sul file system i producer generano la relativa notifica e la pubblicano su una coda di messaggi. Clustered Watch Folder genera un producer per ogni nodo che fa parte del file system distribuito.
- Coda di messaggi (Message Queue): contiene tutti i messaggi di notifica inviati dai vari producer. Ogni watch esistente pubblica le proprie notifiche su un topic dedicato all'interno della coda.
- Conduits: leggono le notifiche dalla coda e le pubblicano inviandole a un sink esterno. Le notifiche sono rese disponibili in formato JSON.

4.3 Configurazione

Per poter attivare Clustered Watch Folder è necessario soddisfare alcune dipendenze ed attivare dei servizi di GPFS.
Prima di tutto è richiesta l'installazione di GPFS-Kafka, ovvero un broker Kafka incluso in GPFS e utilizzato internamente come coda di messaggi a

disposizione dei watch. Kafka è scritto in Java e Scala, bisogna dunque installare Java Runtime Environment. A questo punto si procede con l'installazione di GPFS-Kafka, che va ripetuta su ognuno dei nodi che compongono il cluster. Da questo momento possiamo riferirci all'istanza di Kafka appena installata anche come "broker Kafka interno".
Una volta attivato il broker Kafka interno è necessario attivare un servizio di GPFS chiamato mmmsgqueue[13], che si occupa di gestire la coda di messaggi. Per farlo si utilizza il seguente comando:

```
mmmsgqueue enable -N <host list>
```

Il flag -N serve a specificare un elenco di nodi su cui si vuole attivare la coda di messaggi.
Per poter concretamente ricevere le notifiche prodotte da Clustered Watch Folder è necessario munirsi di un sink. Il broker interno infatti è già impiegato nelle comunicazioni tra i watch, inoltre la gestione della sua configurazione è riservata a GPFS e non può essere modificata manualmente. Per questo motivo viene installato un altro broker Kafka, diverso da quello integrato in GPFS, che da qui in avanti potremo chiamare "sink Kafka esterno". L'installazione può avvenire su un nodo qualsiasi, sia esso appartenente al cluster monitorato o meno. Si noti come la scelta di Kafka come sink esterno sia al momento obbligata, visto che è l'unico supportato da Clustered Watch Folder, ma in futuro potrebbero essere rese disponibili alternative diverse.
Prima di poter utilizzare Kafka è richiesto l'avvio di Zookeeper, fornito all'interno del pacchetto di Kafka. Apache Zookeeper è un servizio per la gestione di ambienti distribuiti, che si occupa di aspetti come il mantenimento della configurazione, la sincronizzazione tra le varie entità e l'uniformità del namespace nel cluster. Per farlo si pone come elemento centralizzato che fa da raccordo tra i vari componenti del sistema distribuito[14].
Avvio di Zookeeper:

```
bin/zookeeper-server-start.sh
    config/zookeeper.properties
```

Avvio di Kafka:

```
bin/kafka-server-start.sh config/server.properties
```

Entrambi i comandi fanno riferimento a dei file di configurazione, che possono essere modificati per adattare meglio Kafka al contesto. In particolare è possibile impostare gli indirizzi e le porte degli host su cui si vuole lavorare, e delle politiche di autenticazione per garantire una maggiore sicurezza. Di

default Zookeeper e Kafka comunicano rispettivamente sulle porte 2181 e 9092, ma il sink utilizzato per questo lavoro è stato configurato con 2182 e 9093, essendo quelle di default già utilizzate dal broker Kafka interno.
In zookeeper.properties:

```
clientPort=2182
```

In server.properties:

```
listeners=PLAINTEXT://<hostname>:<port>
```

PLAINTEXT sta ad indicare il fatto che non utilizziamo nessun protocollo di sicurezza nelle comunicazioni. In questo caso infatti stiamo lavorando con nodi ben conosciuti e all'interno di un ambiente protetto da un firewall sulla frontiera, per cui non c'è bisogno di utilizzare particolari protocolli.
Il sink Kafka esterno riceverà le notifiche su topic creati dall'utente, per cui è possibile ad esempio raggruppare gli eventi segnalati da determinati watch o certi tipi di eventi su dei topic dedicati.
Comando per la creazione di un topic:

```
bin/kafka-topics.sh --create --topic StorageEvents
    --bootstrap-server <host>:<port>
```

- --topic: consente di specificare il nome del topic su cui si vuole comunicare, sia in trasmissione che in ricezione. In questo caso è stato assegnato il nome "StorageEvents" per poter ricevere le notifiche, lo stesso topic sarà utilizzato in trasmissione dai conduits di Clustered Watch Folder.
- --bootstrap-server: server di bootstrap a cui vengono richiesti tutti i metadati necessari per poter comunicare.

Una volta creato un topic sarà possibile ricevere notifiche semplicemente mettendosi in ascolto su di esso con il seguente comando:

```
bin/kafka-console-consumer.sh --topic StorageEvents
    --bootstrap-server <host>:<port>
```

È possibile a questo punto definire un watch che intercetti gli eventi sul file system e li mandi al sink esterno. Per farlo si utilizza il comando mmwatch[15], specificando un device e dei parametri di configurazione:

```
mmwatch gpfs enable -F myconf.conf
```

- gpfs: nome del device, ovvero il cluster dedicato a GPFS utilizzato per lavorare con Clustered Watch Folder.
- enable: specifica a mmwatch che è richiesta la creazione di un watch. Sono presenti anche altre funzionalità come "disable" per eliminare un watch o "list" per mostrare i watch esistenti.
- -F: permette di specificare i paramentri di configurazione tramite un file, che in questo caso si chiama "myconf.conf". Bisogna far presente che questo non è l'unico modo, ma è possibile anche indicare manualmente i parametri di configurazione direttamente nel comando.

Questi sono i possibili parametri di configurazione per la creazione del watch:

- WATCH_ID: eventuale identificativo desiderato per il watch, se non specificato ne sarà generato uno in automatico.
- FILESET: permette di specificare un certo fileset da osservare all'interno del device indicato. Se non specificato sarà osservato l'intero device.
- EVENTS: tipi di eventi da rilevare. Se non inserito il watch osserverà eventi di tutti i tipi.
- EVENT_HANDLER: Campo obbligatorio. Tipo di sink di destinazione per le notifiche. Al momento "kafkasink" è l'unica scelta possibile.
- SINK_BROKERS: Campo obbligatorio. Elenco degli indirizzi e porte dei sink esterni che riceveranno le notifiche.
- SINK_TOPIC: Campo obbligatorio. Nome del topic creato sul sink esterno su cui inviare le notifiche.
- SINK_AUTH_CONFIG: consente di indicare un file che contiene i dettagli sull'autenticazione presso il sink esterno.
- DEGRADED: lavora con meno partizioni, in modo da risparmiare spazio sul disco.

Esempio di file di configurazione myconf.conf utilizzato per creare uno dei watch di questo lavoro:

```
EVENTS:IN_CREATE, IN_MODIFY
EVENT_HANDLER:kafkasink
SINK_BROKERS:<host>:<port>
SINK_TOPIC:StorageEvents
```

In questo caso siamo interessati solo a eventi di scrittura come la creazione di file o la loro modifica. Notare anche come il topic sia lo stesso precedentemente definito.

Si può controllare lo stato del watch appena creato usando il comando mmwatch status:

```
mmwatch gpfs status
```

Esempio di status di un watch:

```
Device      Watch Path        Watch ID        Watch State
gpfs        /gpfs             CLW1619633334   Active
Node Name                                         Status
ds-008.cr.cnaf.infn.it                            HEALTHY
ds-009.cr.cnaf.infn.it                            HEALTHY
ds-010.cr.cnaf.infn.it                            HEALTHY
ds-011.cr.cnaf.infn.it                            HEALTHY
```

Una volta attivato un watch sarà possibile svolgere operazioni sul cluster indicato, e vedere come le notifiche vengono ricevute dal sink esterno in tempo reale. Le notifiche prodotte dai watch sono emesse in formato JSON e contengono diversi campi[16]. Di seguito ne vengono mostrati alcuni:

- event: tipo di evento
- path: percorso del file interessato dall'evento
- nodeName: nome del nodo su cui si è verificato l'evento
- eventTime: momento in cui l'evento si è verificato, espresso in formato UTC

Per garantire una migliore visualizzazione dei JSON in arrivo, è stato utilizzato il comando jq per la formattazione e la selezione dei campi rilevanti. L'output risultante somiglia a quello mostrato negli esempi di seguito:

```
[
"IN_CREATE",
"ds-008",
"2021-03-20_10:51:15+0100",
"/gpfs/test_fs/N_67M4ykdbhQIOuw9Gr5BpERVPoX2W"
]

[
"IN_MODIFY",
"ds-008",
"2021-03-20_10:51:21+0100",
"/gpfs/test_fs/i4Lv91gO_PDlF65RKfIzCrajB"
]
```

4.4 Test effettuati

Dopo aver configurato Clustered Watch Folder sono state svolte delle prime prove minimali con un numero ridotto di operazioni, per verificare che effettivamente venissero ricevute le notifiche in tempo reale.

4.4.1 Studio preliminare delle notifiche

Come riferimento per i test successivi si è cercato prima di tutto di ottenere una mappatura tra gli eventi su file system e le notifiche che vengono generate al verificarsi di questi. Di seguito vengono riportate le notifiche prodotte da alcune operazioni di riferimento su un singolo elemento, eventualmente distinguendo il caso in cui si tratti di un file o di una cartella.

cp Nel caso di copia su file: IN_OPEN, IN_ACCESS, IN_CLOSE_NOWRITE sul file di partenza, IN_CREATE, IN_OPEN, IN_CLOSE_WRITE sul nuovo file.
Nel caso di copia su cartella: IN_OPEN e IN_CLOSE_NOWRITE sulla prima, IN_CREATE sulla nuova, ognuna con IN_ISDIR.

mv Nel caso in cui venga spostato un file: IN_MOVED_FROM sul file di partenza, IN_MOVED_TO sul file di destinazione.
Nel caso in cui venga spostata una cartella: IN_MOVED_FROM sulla prima cartella, IN_MOVED_TO sulla seconda, ognuna con IN_ISDIR.

rm Nel caso di rimozione di un file: IN_DELETE sul file rimosso.
Nel caso di rimozione di una cartella: IN_OPEN, IN_CLOSE_NOWRITE, IN_DELETE, ognuna con IN_ISDIR.

mkdir IN_CREATE, IN_ISDIR sulla cartella creata.

find Nel caso di ricerca di un file: IN_OPEN, IN_CLOSE_NOWRITE, entrambi con IN_ISDIR sulla directory che contiene il file cercato.
Nel caso di ricerca di una cartella: stesse notifiche della ricerca su file, oltre a IN_OPEN e IN_CLOSE_NOWRITE (sempre con IN_ISDIR) della cartella cercata.

append IN_OPEN, IN_MODIFY, IN_CLOSE_WRITE sul file di destinazione.

read IN_OPEN, IN_ACCESS oltre a una notifica IN_ACCESS per ogni riga letta, IN_CLOSE_NOWRITE sul file letto.

write IN_OPEN, IN_MODIFY, IN_CLOSE_WRITE sul file da scrivere.

touch IN_CREATE, IN_OPEN, IN_CLOSE_WRITE sul file creato.

4.4.2 Test su operazioni di lettura

Qui viene riportato il comando per svolgere in automatico le letture da file system utilizzando lo script fstester:

```
./fstester -t 10000 -f /gpfs/lettura/ --read
```

- -t 10000: vengono richieste 10.000 operazioni di lettura.
- -f /gpfs/lettura/: percorso nel file system distribuito dedicato a contenere file con cui svolgere i test in lettura.
- --read: flag che richiede a fstester di eseguire solo operazioni di lettura.

Comando utilizzato per la ricezione di notifiche tramite il sink esterno:

```
bin/kafka-console-consumer.sh --topic StorageEvents
    --timeout-ms 120000
    --bootstrap-server ds-009.cr.cnaf.infn.it:9093 |
    jq '[.event, .nodeName, .eventTime, .path]' |
    grep '"IN_OPEN"' | wc -l
```

- --timeout-ms: permette di specificare una soglia massima di attesa se non si ricevono più notifiche, allo scadere di questa la ricezione termina. Può accadere infatti che le notifiche continuino ad arrivare dopo il termine delle operazioni vere e proprie. In questo caso sono stati considerati 2 minuti come margine per permettere a tutte le notifiche di arrivare, visto l'alto numero di operazioni nel test.

- jq: viene utilizzato per rendere più leggibili i messaggi emessi in formato JSON e selezionare solo i campi di interesse. In questo caso vengono presi in considerazione il tipo di evento, il nodo su cui l'evento si è verificato, il timestamp dell'evento e il file (con percorso completo) interessato dall'evento.
- grep: filtra le righe dell'output in modo da tenere conto solo di quelle relative alle notifiche IN_OPEN, ovvero quelle che riguardano strettamente la lettura di file.
- wc -l: conta il numero di righe prodotte dal comando precedente nella pipeline, ovvero il numero notifiche IN_OPEN ricevute. Questo permette di verificare che ci sia una corrispondenza esatta tra le operazioni richieste e le notifiche prodotte da GPFS.

Il test ha dato esito positivo, con la ricezione di 10.000 notifiche IN_OPEN.

4.4.3 Test su operazioni di scrittura

Questo tipo di operazioni rappresenta il principale punto di interesse per il reparto storage, perché ad esse sono legati i casi d'uso di storage events individuati.
Esecuzione delle operazioni di scrittura su file system:

```
    ./fstester -t 10000 -f /gpfs/scrittura/ --write
```

Ricezione delle notifiche sul Kafka sink esterno:

```
bin/kafka-console-consumer.sh --topic StorageEvents
    --timeout-ms 120000
    --bootstrap-server ds-009.cr.cnaf.infn.it:9093 |
    jq '[.event, .nodeName, .eventTime, .path]' |
    grep 'IN_CREATE' | wc -l
```

Valgono le stesse considerazioni viste per le operazioni di lettura, ma in questo caso le righe di interesse filtrate da grep contengono "IN_CREATE".
Anche in questo caso il test ha dato esito positivo, con la ricezione correttamente avvenuta di 10.000 notifiche.

È stato svolto anche un altro test con le stesse operazioni, ma questa volta con un watch che intercetta tutti i tipi di notifiche.

```
bin/kafka-console-consumer.sh --topic StorageEvents
    --timeout-ms 120000
```

```
    --bootstrap-server ds-009.cr.cnaf.infn.it:9093 |
    jq '[.event, .nodeName, .eventTime, .path]' |
    grep 'ds-0' | wc -l
```

Si può notare come il conteggio non avvenga più su "IN_CREATE" ma su "ds-0", in quanto ogni notifica a prescindere dal tipo contiene un singolo riferimento al nome del nodo su cui l'evento si è verificato.
Nell'esecuzione di questo test sono state generate 58.707 notifiche, di cui 10.000 IN_CREATE, 10.000 IN_OPEN, 28.707 IN_MODIFY e 10.000 IN_CLOSE, per cui si può considerare concluso con successo.

Le tempistiche per lo svolgimento dei due test sulle scritture sono state rispettivamente 41 e 42 minuti. Si può dedurre quindi che anche in questo caso, come già accadeva su Inotify, l'intercettazione di un numero più alto di notifiche non peggiora le performance del sistema.

4.4.4 Estensione dei test

Dopo aver concluso con successo i test appena descritti, si è deciso di effettuare dei test più intensivi per il sistema. Questo permette di avere una prospettiva più realistica sul carico di lavoro che sarà necessario gestire in un contesto di produzione.
Sono state aggiunte 8 macchine al cluster GPFS come client, in modo da poter lavorare sul mountpoint /gpfs/.
I nuovi test puntano ad eseguire da ognuna delle macchine 10.000 operazioni sul file system distribuito.
Per poter svolgere il test è stato necessario prima di tutto installare il broker Kafka interno. Le nuove macchine sono state poi aggiunte alla coda di messaggi utilizzando mmmsgqueue, che dà la possibilità di modificare la configurazione per aggiungere nodi:

```
mmmsgqueue config --add-nodes -N cs-001.cr.cnaf.infn.it
    cs-002.cr.cnaf.infn.it cs-003.cr.cnaf.infn.it
    cs-004.cr.cnaf.infn.it cs-005.cr.cnaf.infn.it
    cs-006.cr.cnaf.infn.it cs-007.cr.cnaf.infn.it
    cs-008.cr.cnaf.infn.it
```

Lo script fstester è stato aggiunto a tutte le macchine in modo da poter essere lanciato da tutte contemporaneamente.

Lettura Sono state richieste 10.000 letture su un percorso contenente 1000 file, considerando solo le notifiche di tipo IN_OPEN.
Il test ha dato un riscontro positivo: sono state correttamente ricevute 240.000 notifiche, di cui 80.000 contenenti solo IN_OPEN e 160.000 comprendenti anche IN_ISDIR.

Scrittura Con un comando del tutto simile a quello già visto per il test di scrittura in precedenza, sono state richieste 10.000 scritture. Il test è stato portato a termine con successo anche in questo caso: sono stati scritti 80.000 file e ricevute 80.000 notifiche.
La stessa operazione è stata poi ripetuta tenendo conto di tutte le notifiche generate, portando ad un esito positivo: sono stati scritti 80.000 file e ricevute 469.909 notifiche, di cui 80.000 IN_CREATE, 80.000 IN_OPEN, 80.000 IN_CLOSE e 229.909 IN_MODIFY.
Le durate dei due test sono state rispettivamente di 30 e 32 minuti, mostrando come anche questa volta il maggior carico di notifiche non abbia creato problemi di prestazioni per il sistema.

Di seguito è riportata una tabella riassuntiva con i risultati di tutti i test svolti su GPFS:

Tipo test	N° operazioni	Tipo notifiche	N° notifiche ricevute	Dimensione max file	Durata test
lettura	10.000	IN_OPEN	10.000	90 MB	44 min
lettura	80.000	IN_OPEN	80.000	90 MB	104 min
scrittura	10.000	IN_CREATE	10.000	90 MB	41 min
scrittura	10.000	tutte	58.707	90 MB	42 min
scrittura	80.000	IN_CREATE	80.000	10 MB	30 min
scrittura	80.000	tutte	469.909	10 MB	32 min

5 Ceph

5.1 Introduzione

Ceph[17] è una piattaforma open-source per lo storage distribuito. Si compone di più moduli che permettono di gestire l'archiviazione dei dati in tre forme diverse: object storage, block storage e file system.

Object storage Un object storage è uno spazio di archiviazione in cui non sono presenti gerarchie di directory. Gli oggetti vengono inseriti al suo interno corredati di metadati e di un identificatore univoco. Tipicamente questa soluzione è utilizzata in contesti che hanno bisogno di lavorare con dati non strutturati e di poter contare su una grande scalabilità.

Block storage Questa tecnica di archiviazione prevede la divisione dei dati in blocchi di uguali dimensioni per consentirne una gestione ottimizzata. Ognuno dei blocchi è identificato da un indirizzo, e quando è richiesta una risorsa si parte da un certo blocco e si recuperano quelli mancanti. Normalmente questo tipo di archiviazione garantisce ottime performance, ma ha allo stesso tempo un costo elevato e non prevede l'utilizzo di metadati. Bisogna quindi valutare con cautela i casi in cui invece i metadati potrebbero essere necessari.

File system Approccio tradizionale: i dati vengono salvati all'interno di una gerarchia di directory. I metadati gestiti in questo caso sono ridotti, tipicamente costituiti dal percorso che identifica ognuno dei file. Anche le performance sono tendenzialmente buone, ma potrebbero nascere delle limitazioni quando si vuole ottenere un alto grado di scalabilità.

Quest'ultimo è il caso tenuto in considerazione in questo lavoro. La parte di Ceph dedicata all'archiviazione tramite file system è detta CephFS, e come le altre tecnologie trattate supporta lo standard POSIX.

5.1.1 Ceph Storage Cluster

Uno Storage Cluster Ceph si compone di alcuni demoni che possono essere di quattro tipi fondamentali: Monitor, Manager, OSD, MDS.

Monitor Sono incaricati di tenere traccia dello stato del cluster, quindi lo stato degli altri demoni e di ognuno dei nodi. Si occupano anche di garantire la corretta autenticazione tra demoni e client. I monitor mantengono anche informazioni sul posizionamento dei dati all'interno del cluster. Perché Ceph possa funzionare è richiesto che almeno un Monitor sia presente, ma tipicamente se ne usano almeno tre per assicurare una buona ridondanza.

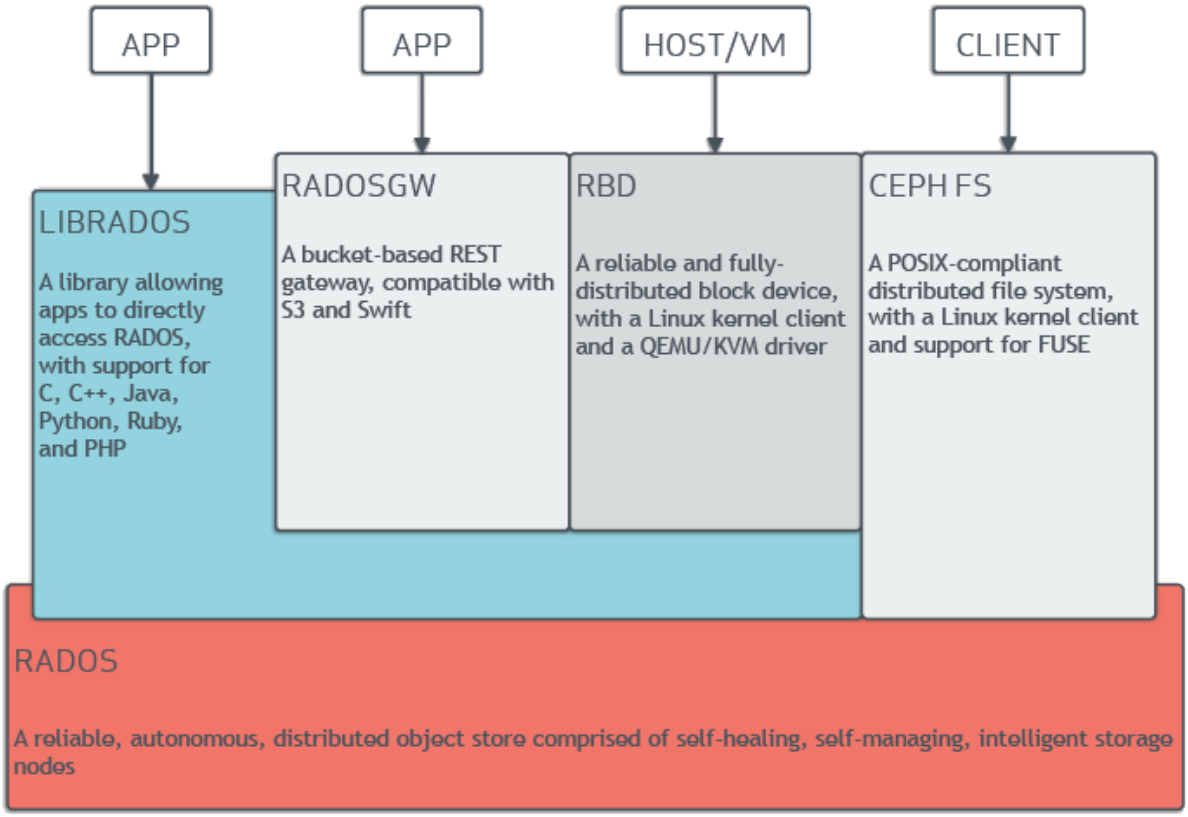
Manager Sono dedicati alla gestione di metriche che riguardano il sistema, come il grado di utilizzo dello storage o il carico di lavoro in un dato momento. È affidato ai Manager anche il compito di rendere disponibili all'esterno informazioni sul sistema e sul suo stato. Perché Ceph possa funzionare è richiesto che sia presente almeno un Manager, ma per avere un'alta disponibilità se ne usano solitamente almeno due.

OSD (Object Storage Demon) Lavorano a contatto diretto con i dati veri e propri, occupandosi ad esempio della loro archiviazione e replicazione. Oltre a questo possono controllare lo stato di altri OSD, e comunicare le informazioni ai Monitor e i Manager. Ne è richiesto solo uno per il corretto funzionamento di Ceph, ma normalmente se ne tengono 3 per ridondanza.

MDS (Metadata Server) Gestiscono i metadati per il file system di Ceph. Object storage e block storage non fanno uso di metadati, e quindi non necessitano di demoni di questo tipo.

5.1.2 Architettura

Lo schema seguente[18] mostra i principali componenti architetturali di Ceph:



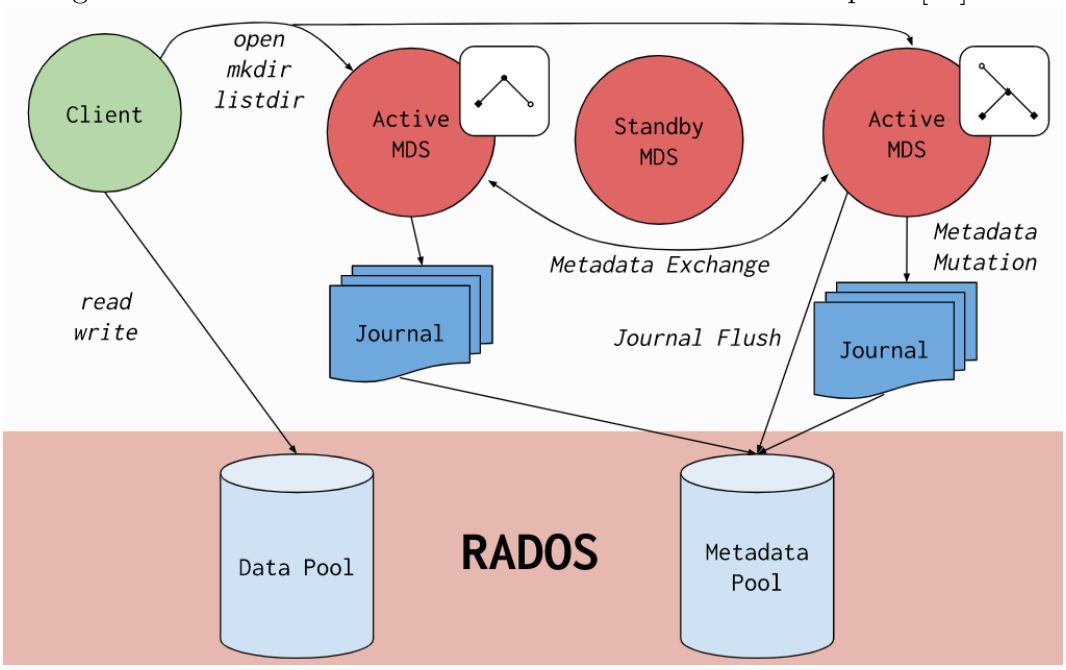
Tutto il sistema poggia su RADOS (Reliable Autonomic Distributed Object Store), un object storage distribuito su cui lo Storage Cluster Ceph archivia i dati e svolge le proprie operazioni. LIBRADOS è una libreria che fa da interfaccia a RADOS grazie al supporto a diversi linguaggi di programmazione. Può essere utilizzata sia da altri componenti di Ceph che da applicazioni esterne. RADOSGW è un gateway che permette ad applicazioni esterne di ottenere il servizio di object storage, mentre gli RBD (RADOS Block Device) coprono la funzionalità di block storage. Sia RADOSGW che RBD poggiano su LIBRADOS per interfacciarsi con l'object storage. CephFS, come visto precedentemente, permette di usare Ceph come file system. È costruito su RADOS, per cui concretamente Ceph svolge le proprie operazioni sempre facendo riferimento a un modello object storage.
Ceph presenta quindi un'architettura distribuita e affidata a diversi nodi.

Questo porta non solo al superamento del single point of failure, ma anche a una grande scalabilità. Il sistema è anche fault tolerant, visto che i demoni OSD creano delle repliche degli oggetti con cui lavorano.

5.1.3 Ceph File System

Il file system di Ceph lavora utilizzando RADOS come tutti gli altri componenti, fornisce cioè un'astrazione sul modo in cui i file sono concretamente salvati. CephFS permette di operare con gerarchie di file e directory come in tutti i file system, ma i file vengono poi comunque tradotti in oggetti e gestiti tramite RADOS. CephFS utilizza un Metadata Server (MDS) per tenere traccia della mappatura tra oggetti e gerarchia del filesystem. È possibile tenere diversi Metadata Server organizzati in un cluster, in modo da avere ridondanza in caso di fail su quello principale. Un'altra possibilità è quella di tenere attivi più MDS allo stesso momento, in modo da poter parallelizzare le operazioni.

Di seguito è mostrato uno schema architetturale relativo a CephFS[19]:



Dalla figura è possibile vedere come i metadati e i dati veri e propri siano tenuti da RADOS in due pool distinti. Il client esegue le proprie operazioni di lettura e scrittura direttamente sui dati contenuti in RADOS. Le opera-

zioni che invece riguardano l'esplorazione o la modifica della struttura del file system (open, mkdir, etc...) passano dai Metadata Server. Si noti che i MDS attivi si occupano di porzioni diverse del file system, ma possono scambiare metadati utili quando necessario.

5.2 Funzionamento degli storage events in Ceph

Ceph non è dotato di una funzionalità specifica per la gestione di storage events, come invece accade su Clustered Watch Folder. Grazie alla sua architettura però è comunque possibile renderli visibili nei log del metadata server.
Per fare in modo che gli eventi siano visibili nei log è necessario richiedere una modalità di logging molto verbosa. Il comando "ceph tell" permette di modificare la configurazione di vari demoni senza dover accedere direttamente ai nodi su cui sono ospitati. Viene contattato un nodo Monitor, che quindi si occupa di modificare la configurazione come richiesto dall'utente.

```
ceph tell mds.0 injectargs --debug-mds 5/5
```

- mds.0: demone di cui si vuole modificare la configurazione. In questo caso si tratta del primo Metadata Server.
- injectargs: comando da eseguire sul demone di destinazione. Qui viene eseguita l'iniezione di argomenti per la configurazione.
- --debug-mds 5/5: imposta al massimo la quantità di informazioni scritte nel log.

Il cluster Ceph presso il CNAF è composto da 8 macchine.
Ceph utilizza una denominazione degli eventi che è diversa da quella vista in precedenza per Inotify e Clustered Watch Folder. Di seguito sono riportati alcuni esempi di notifiche prodotte nel log a fronte di varie operazioni svolte sul file system distribuito:

- touch: openc, cap update, scatter_writebehind
- cp: openc, cap update
- rm: unlink_local, purge_stray
- rm -r: unlink_local, purge_stray, scatter_writebehind
- mv: rename, scatter_writebehind
- mkdir: mkdir

cap update, scatter_writebehind e purge_stray sono eventi interni a Ceph che non rispettano lo standard POSIX.

5.3 Test effettuati

Come per le tecnologie già viste, anche in questo caso sono stati svolti dei test minimali di partenza, che hanno dato subito esito positivo. Il testing su Ceph è stato impostato solo sulle operazioni di scrittura.
Le scritture su file system distribuito sono state svolte utilizzando lo script fstester sull'opportuno mountpoint, come in precedenza.

```
./fstester -t 10000 -f /mnt/ceph/scrittura/ --write
```

Il controllo delle notifiche è stato invece effettuato leggendo il log come uno stream e selezionando le righe di output rilevanti.
Come nei test precedenti, anche qui sono state considerate 10.000 operazioni.

Solo notifiche openc Ogni creazione di file genera una sola notifica di tipo openc.

```
tail -f /var/log/ceph/ceph-mds.cs-001.log |
    grep -m 10000 openc
```

- -f: con questo flag il comando tail continua a leggere le righe del log incrementalmente.
- grep -m 10000 openc: vengono tenute solo le notifiche per eventi "openc". Il flag -m permette di definire un limite massimo per il numero di matching da considerare, dopo 10.000 righe contenenti "openc" il comando termina. In questo caso non è necessario intercettare le righe contenenti "metablob", in quanto è sicuramente contenuto nelle righe che hanno già "openc".

Questo test ha dato esito positivo, rilevando 10.000 righe contenenti "openc" all'interno del log.

Tutte le notifiche In questo caso si è optato per una visione sul numero complessivo di notifiche prodotte, sempre verificando che il numero di scritture sia corretto.

```
timeout 7200 tail -f
    /var/log/ceph/ceph-mds.cs-001.log | grep metablob
```

- timeout 7200: la durata è espressa in secondi. Si sono considerate due ore di tempo massimo per l'esecuzione del test, in modo da avere ampio margine sull'arrivo di notifiche visto l'elevato numero di operazioni.
- grep metablob: in questo modo si riescono a reperire le righe del log che riguardano tutti i possibili tipi di eventi su file system.

Anche questo test ha avuto esito positivo. Complessivamente sono state generate 31838 notifiche, di cui 10.000 openc. Si noti come il numero di notifiche di eventi openc corrisponda esattamente al numero di scritture effettuate.

5.3.1 Estensione dei test

Come già era stato fatto con GPFS, anche su Ceph i test sono stati estesi al caso con 8 nodi che lavorano parallelamente.
Il test è incentrato sull'esecuzione di 10.000 operazioni di scrittura da ogni nodo, catturando le sole notifiche openc.
Il test ha avuto successo: sono stati scritti 80.000 file e ricevute 80.000 notifiche openc.
Lo stesso test è stato ripetuto tenendo conto di tutte le notifiche, anche questa volta con esito positivo: sono stati scritti correttamente 80.000 file e ricevute 229.447 notifiche, tra cui 80.000 openc.

Di seguito è riportata una tabella riassuntiva contenente i dati dei test svolti su Ceph:

Tipo test	N° operazioni	Tipo notifiche	N° notifiche ricevute	Dimensione max file	Durata test
scrittura	10.000	openc	10.000	90 MB	29 min
scrittura	10.000	tutte	31.838	90 MB	29 min
scrittura	80.000	openc	80.000	10 MB	30 min
scrittura	80.000	tutte	229.447	10 MB	30 min

6 Confronto tra le tecnologie analizzate

Vengono di seguito presentati dei confronti tra le tecnologie analizzate, prendendo in considerazione sia le somiglianze che le divergenze.
Inotify non viene realmente preso in considerazione nei confronti, dato che lavora solo in locale. Va comunque ricordato che esistono varianti che sono in grado di lavorare anche in distribuito.

Efficacia Tutte le tecnologie testate hanno dimostrato di essere efficaci. In nessun caso sono state perse notifiche.

Standard POSIX Tutte le tecnologie aderiscono allo standard POSIX, e sono quindi in grado di rilevare eventi su sistemi operativi che aderiscono allo stesso standard.

Watch distribuiti Solo GPFS prevede un vero e proprio meccanismo di watch distribuiti, mentre in Ceph il concetto di watch distribuito sul file system è assente.
In ogni caso l'osservazione dei Metadata Server di Ceph garantisce comunque la possibilità di vedere gli eventi nel momento in cui essi si verificano.
Ceph richiede un'esplorazione più profonda per trovare il meccanismo di rilevamento degli eventi, oltre che la costruzione di un comando per ottenere solo le informazioni necessarie dal log.

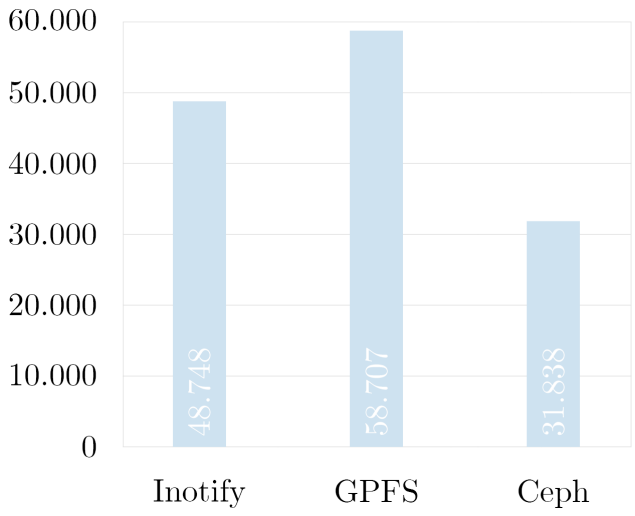
Sink GPFS prevede l'invio di notifiche verso un sink esterno. Va ribadito che attualmente l'unico sink supportato è Kafka, ma in futuro è prevista l'aggiunta di altri, aprendo quindi alla possibilità di gestire le notifiche diversamente in ricezione.
Su Ceph gli eventi vengono semplicemente riportati nel log, un eventuale invio verso l'esterno deve essere realizzato appositamente. Ad ogni modo questo aspetto è di semplice risoluzione e non rappresenta un ostacolo all'operatività di Ceph.

Filtraggio La possibilità di filtrare le notifiche presenta modalità diverse tra GPFS e Ceph.
GPFS permette di definire dei watch, e nella loro configurazione è presente un elenco con i tipi di eventi che si desidera rilevare. Di conseguenza il watch

si occuperà solo degli eventi specificati e genererà notifiche solo per questi, eseguendo un filtraggio a monte.
Nel caso di Ceph troviamo un approccio diverso: tutti gli eventi che si verificano su file system vengono scritti all'interno del log, di cui possiamo decidere quali righe considerare e quali scartare. Il numero di eventi notificati rimane quindi sempre lo stesso.

Notifiche prodotte Ognuna delle tecnologie analizzate ha un criterio diverso per la produzione di notifiche a fronte di determinati eventi.
Considerando il caso di creazione di file, è possibile confrontare il numero di notifiche generate da ognuna delle tecnologie considerando tutti i tipi di notifica. Di seguito sono considerati i valori ottenuti dai test su 10.000 scritture già visti nei capitoli precedenti:

- Inotify: 48.748
- GPFS: 58.707
- Ceph: 31.838



Performance Nei test effettuati su tutte le tecnologie non sono stati riscontrati rallentamenti a fronte di un numero più elevato di notifiche intercettate. Questo comunque non garantisce che in contesti più estesi (ad esempio con migliaia di nodi) la problematica non possa verificarsi.

7 Conclusioni

Il lavoro svolto mostra i possibili vantaggi apportati dall'approccio a eventi alla gestione dei file system distribuiti. Ad oggi sono ancora molti i grandi contesti in cui gli eventi importanti su file system vengono rilevati tramite scansioni periodiche, portando spesso a sprechi di risorse e inefficienze.
L'ambizione di questo lavoro è quella di portare un elemento di novità, esplorando delle strade concretamente percorribili. I test effettuati sulle tecnologie prese in esame hanno dimostrato la loro efficacia, anche in caso di carichi di lavoro più consistenti nessuna notifica è stata persa.
L'impiego di storage events potrebbe rivelarsi poco utile in situazioni operative che utilizzano file system piuttosto ridotti. In casi di quel tipo è normalmente possibile svolgere scansioni manuali senza conseguenze sulle performance del sistema, a patto di poter accettare informazioni sul file system periodicamente e non in tempo reale.

7.1 Sviluppi futuri

Sono di seguito proposti alcuni possibili spunti per estendere questo lavoro in futuro.

Utilizzo presso il CNAF Il CNAF potrebbe decidere di adottare in produzione l'approccio a eventi impiegando una delle tecnologie viste. Questo permetterebbe di evitare le scansioni cicliche di grandi quantità di file che vengono svolte anche più volte al giorno sul file system.
Due operazioni importanti in cui potrebbero essere impiegati sono il passaggio dei dati da dischi a tape[20] e il trasferimento di dati verso altri siti. Questo lavoro potrebbe essere esteso proprio svolgendo delle prove concrete sui casi d'uso appena menzionati.
In particolare nel caso di passaggio di dati da disco a tape sono attualmente impiegate delle scansioni periodiche dell'intero file system. Quando si lavora con decine di milioni di file questa operazione risulta particolarmente onerosa, il suo svolgimento infatti può richiedere anche diversi minuti. A ciò si aggiunge il fatto che durante le scansioni si riscontra un peggioramento sensibile delle prestazioni complessive di lettura/scrittura tra disco e tape. Un'inefficienza come questa è ancora meno accettabile se si considerano le prospettive di aumento dei dati e conseguentemente delle prestazioni che il CNAF prevede. Per questo è auspicabile trovare una soluzione che eviti del

tutto le scansioni del file system, o che comunque le riduca notevolmente. Il rilevamento dei file scritti su file system da copiare su tape potrebbe essere svolto proprio impiegando gli storage events, tenendo eventualmente una scansione sequenziale con frequenza molto più bassa (ad esempio giornaliera) per verificare che non siano state perse notifiche riguardanti file da copiare.
Queste considerazioni sono di particolare interesse considerando il trasferimento del CNAF presso il nuovo Tecnopolo di Bologna, previsto per il 2022. Il trasferimento coinciderà con una evoluzione del CNAF in termini sia di dati (si prevede di gestire il doppio dei dati nell'arco di 4-5 anni), sia di nuove soluzioni tecnologiche attualmente in fase di studio.

Scalabilità I test sono stati fino ad ora svolti su sistemi dedicati allo scopo, non impegnati in altre operazioni e in archi di tempo piuttosto limitati (circa un'ora per i test più lunghi). Le notifiche sono sempre state ricevute rapidamente, e la loro intercettazione non ha creato overhead sul sistema.
Il lavoro potrebbe proseguire estendendo i test a cluster in cui operano migliaia di nodi, per verificare quale sia il livello di scalabilità di queste tecnologie, ed eventualmente comprendere la soglia oltre la quale l'intercettazione di notifiche genera un overhead sul sistema.
In Ceph va considerato anche il potenziale impatto della modalità di logging molto verbosa utilizzata in un ambiente di produzione. Questa può infatti portare alla generazione di oltre 1 GB di log ogni ora, potenzialmente causando l'interruzione dell'attività del nodo se lo spazio disponibile sul suo disco dovesse esaurirsi[21].

Sink alternativi per GPFS Come già detto nei capitoli precedenti Clustered Watch Folder prevede solo Kafka come possibile sink, ma ci si aspetta che questa limitazione venga superata in futuro.
Questo permetterebbe di estendere il lavoro e fare in modo che le notifiche siano ricevute da altri sink, che potrebbero gestirle in modo diverso.

Reazione agli eventi Questo lavoro ha posto l'attenzione sul meccanismo di rilevamento degli eventi, e sul modo in cui questi vengono notificati.
Un'attività significativa potrebbe essere la realizzazione di un sistema che reagisca alle notifiche nel momento in cui queste vengono ricevute, magari distinguendole in base al tipo. Si aprirebbero così delle possibilità interessanti sulla gestione automatica degli eventi all'interno del file system distribuito.

8 Bibliografia

[1] GPFS: A Shared-Disk File System for Large Computing Clusters. Frank Schmuck and Roger Haskin (1998)

[2] GPFS: https://www.ibm.com/it-it/products/spectrum-scale (2021)

[3] Ceph: https://ceph.io/ceph-storage/file-system/ (2021)

[4] Storage events: distributed users, federation and beyond.
A. Paul Millar, Olufemi Adeyemi, Vincent Garonne, Dmitry Litvintsev, Tigran Mkrtchyan, Albert Rossi, Marina Sahakyan, and Jürgen Starek (2019)

[5] Building a scalable, interactive and event-driven computing platform in multi-cloud environments with dCache.
Michael Schuh, Patrick Fuhrmann, Paul Millar, Tigran Mkrtchyan (2019)

[6] Inotify: https://en.wikipedia.org/wiki/Inotify (2021)

[7] CentOS: https://www.centos.org/about/ (2021)

[8] inotify-tools: https://github.com/inotify-tools/inotify-tools/wiki (2020)

[9] Descrizione generale di GPFS:
https://agenda.infn.it/event/6922/sessions/10553/attachments/47155/55802/GPFS_tutorial_part_1_2013.pdf (2013)

[10] Introduzione a Kafka: https://kafka.apache.org/intro (2021)

[11] Clustered Watch Folder:
https://www.spectrumscaleug.org/wp-content/uploads/2019/05/SSUG19US-Day-2-B09-Spectrum-Scale-Watch-Folder.pdf
(2019)

[12] Componenti di Clustered Watch Folder:
https://www.ibm.com/docs/en/spectrum-scale/5.0.4?topic=overview-introduction-clustered-watch-folder (2020)

[13] mmmsgqueue:
https://www.ibm.com/docs/en/spectrum-scale/5.0.4?topic=reference-mmmsgqueue-command (2020)

[14] Zookeeper: https://zookeeper.apache.org/ (2020)

[15] mmwatch:
https://www.ibm.com/docs/en/spectrum-scale/5.0.5?topic=reference-mmwatch-command (2021)

[16] Notifiche in formato JSON:
https://www.ibm.com/docs/en/spectrum-scale/5.0.5?topic=folder-json-attributes-in-clustered-watch (2021)

[17] Introduzione a Ceph: https://docs.ceph.com/en/latest/ (2016)

[18] Architettura di Ceph: https://docs.ceph.com/en/latest/architecture/ (2016)

[19] Ceph File System: https://docs.ceph.com/en/latest/cephfs/ (2016)

[20] Passaggio di dati da disco a tape presso il CNAF:
HSM and backup services at INFN-CNAF. Alessandro Cavalli, Daniele Cesini, Enrico Fattibene, Andrea Prosperini, Vladimir Sapunenko (2019)

[21] Logging e debugging in Ceph:
https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/1.2.3/html/ceph_configuration_guide/logging-and-debugging (2021)

Ringraziamenti

Voglio esprimere un ringraziamento a tutte le persone che mi hanno accompagnato durante gli anni di studio, fino al raggiungimento di questo traguardo.
La mia relatrice, la prof.ssa Anna Ciampolini, che prontamente ha sostenuto l'idea di questa tesi.
I miei correlatori Antonio Falabella ed Enrico Fattibiene, che ancora una volta hanno condiviso con me la loro esperienza e mi hanno guidato al meglio durante il tirocinio e la stesura della tesi.
Vladimir Sapunenko per la sua attiva partecipazione al lavoro e i suoi ottimi consigli.
Daniele Cesini, per aver proposto l'argomento della tesi.
Il personale del CNAF, sempre disponibile e accogliente nei miei confronti.
Tutta la mia famiglia, per il sostegno costante durante il mio percorso di studi.
Ruiyu, che è stata al mio fianco ogni giorno e mi ha spinto a dare il meglio.
I miei compagni di studio che hanno condiviso con me l'impegno e le soddisfazioni sia a Bologna che a Cesena.
I miei amici, per l'affetto e il supporto che mi hanno dato durante questi anni.