Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Analisi e ottimizzazione delle prestazioni di servizi di data management per calcolo distribuito

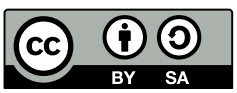
Relatore:
Prof. Renzo Davoli

Presentata da:
Luca Bassi

Correlatori:
Prof. Francesco Giacomini
Dott. Enrico Vianello

II Sessione
Anno Accademico 2024/2025

If you spend your whole life waiting for the storm,
you'll never enjoy the sunshine.
Morris West



Introduzione

Il Large Hadron Collider (LHC) del CERN è il più grande e potente acceleratore di particelle al mondo. Per soddisfare le esigenze di salvataggio e analisi dei dati prodotti dagli esperimenti di LHC è nata la Worldwide LHC Computing Grid (WLCG), una collaborazione internazionale di circa 160 data center in più di 40 paesi. L'INFN-CNAF ospita il centro di calcolo che costituisce il Tier 1 italiano di WLCG.

StoRM è lo storage resource manager utilizzato e sviluppato all'INFN-CNAF. In preparazione dell'aumento dei dati raccolti dagli esperimenti del CERN previsto con il Run 4 di LHC sono necessari sviluppi a StoRM per aumentarne l'efficienza.

Inoltre, visto i grandi trasferimenti di dati, il monitor di rete è una parte fondamentale. L'iniziativa SciTags promuove l'identificazione dei domini scientifici e della tipologia di attività a livello di rete. È quindi necessario aggiungere il supporto agli SciTags a StoRM.

eBPF è una tecnologia che permette di eseguire programmi all'interno di un sandbox del kernel Linux. È possibile collegare programmi eBPF a specifici *hook* di sistema come l'invocazione e l'uscita da funzioni. Si può quindi sfruttare questa tecnologia per fare *tracing* delle performance senza dover modificare il codice dei programmi.

Nel capitolo 1 viene presentato il contesto scientifico e tecnologico della fisica delle alte energie. Nel capitolo 2 viene illustrato StoRM, con le sue caratteristiche, il tipo di deployment e i risultati di alcuni test di carico effettuati nel 2024. Nel capitolo 3 vengono mostrati gli sviluppi effettuati su StoRM per migliorarne le prestazioni. Nel capitolo 4 viene presentata l'iniziativa SciTags e come è stato aggiunto il supporto al flow marking a StoRM WebDAV, uno dei componenti di StoRM. Nel capitolo 5 viene mostrato come è possibile utilizzare eBPF per fare tracing delle performance dei programmi.

Indice

Elenco delle figure

Elenco dei codici sorgente

Capitolo 1

Contesto

In questo capitolo viene descritto il contesto scientifico e tecnologico della fisica delle alte energie.

1.1 LHC e WLCG

Il Large Hadron Collider (LHC) [1] è il più grande e potente acceleratore di particelle al mondo. È stato costruito al CERN dal 1998 al 2008 e le prime collisioni sono avvenute nel 2010. Si tratta di un acceleratore circolare di circa 27 km di circonferenza che si trova a circa 100 m di profondità, composto da magneti superconduttori e da una serie di strutture acceleranti per aumentare l'energia delle particelle. I fasci di particelle, protoni o ioni, vengono fatti collidere in corrispondenza di quattro rivelatori: ATLAS, CMS, ALICE e LHCb.

LHC ha permesso di scoprire il bosone di Higgs, confermando il meccanismo di Brout-Englert-Higgs sull'origine della massa. Inoltre permette di investigare la teoria della supersimmetria, la materia oscura e l'energia oscura e perché nell'Universo è presente molta più materia che antimateria.

La Worldwide LHC Computing Grid (WLCG) [2] è una collaborazione globale di circa 160 data center in più di 40 nazioni. WLCG è nata per soddisfare le esigenze di salvataggio e analisi dei circa 200 Petabyte di dati prodotti da LHC. È ispirata al concetto di Grid Computing [3] introdotto da Ian Foster e Carl Kesselman nel 1998. Visto

l'avvento di reti più veloci e l'aumento di richiesta di potenza computazionale, i ricercatori del CERN hanno deciso di adottare questo modello computazionale per la condivisione di risorse eterogenee [4] che, pur essendo amministrate in modo indipendente, espongono un'interfaccia comune, con diverse implementazioni interoperabili.

1.2 Il data center dell'INFN-CNAF

Il CNAF è il centro nazionale dell'Istituto Nazionale di Fisica Nucleare (INFN) per la ricerca e lo sviluppo nelle tecnologie informatiche e telematiche. È stato fondato nel 1962 a Bologna per analizzare le pellicole fotografiche delle camere a bolle; infatti l'acronimo sta per Centro Nazionale Analisi Fotogrammi. Ospita dal 2003 un centro di calcolo che costituisce il Tier 1 italiano di WLCG [5]. Dal 2024, il centro di calcolo è ospitato all'interno del tecnopolo DAMA a Bologna [6].

Il CNAF offre servizi sia di calcolo che di archiviazione dati. I dati sono archiviati sia su disco sia su nastro. In particolare a inizio ottobre 2025 sono archiviati 78.5 PB di dati su disco con una capacità totale di 97.8 PB e 185 PB di dati su nastro con una capacità totale di 293 PB. Vengono utilizzate le *tape library* sia per i costi più bassi sia perché contrariamente agli hard disk incrementano la capacità più velocemente (del 30-40% ogni anno) [7].

Oltre a ospitare un centro di calcolo, l'INFN-CNAF si occupa anche della gestione e dello sviluppo di prodotti middleware utilizzati in WLCG.

1.3 La rete GARR

Il Consortium GARR progetta e gestisce la rete nazionale dedicata alla comunità dell'istruzione, della ricerca e della cultura. Nato come commissione ministeriale nel 1988, il Gruppo per l'Armonizzazione delle Reti della Ricerca (GARR) è stato fondato da 6 enti: CNR, INFN, ENEA, CILEA, CINECA, Tecnopolis CSATA [8]. All'epoca il protocollo TCP/IP non era diventato lo standard de facto; ogni istituto di ricerca usava una rete differente e venivano usate molte soluzioni tecnologiche incompatibili tra loro, provocando un grande dispendio di risorse economiche e di energie.

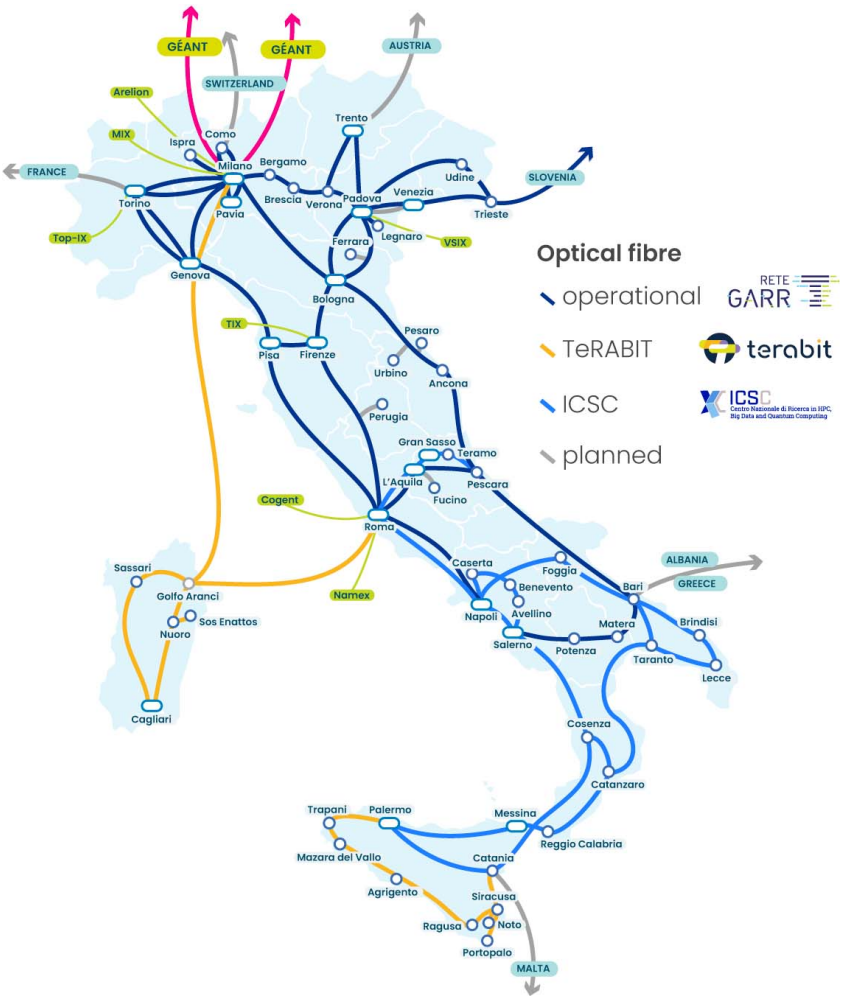


Figura 1.1: Mappa della rete GARR aggiornata a luglio 2024 (Immagine di GARR distribuita con licenza Creative Commons BY-NC-SA 4.0)

La rete è diventata operativa nel 1991 connettendo alla velocità di 2 Mbps (molto alta per l'epoca) 7 nodi: Milano (CILEA), Bologna (CINECA e il polo ENEA e INFN-CNAF), Pisa (CNR-CNUCE), Roma (INFN), Frascati (ENEA e INFN) e Bari (CSATA). Nel 1994 è diventata attiva la rete GARR-2 che arriverà a toccare nel 1996 i 34 Mpbs. Passando per GARR-B nel 1998 che raggiunge i 155 Mbps, si arriva nei primi anni del 2000 alla rete GARR-G con collegamenti fino a 10 Gbps. Nel 2011 è partita GARR-X che porterà la velocità fino a 100 Gbps. La nuova evoluzione è la rete GARR-T [9]

che ha portato alla realizzazione di 750 km di fibra ottica, 42 Point of Presence (PoP) ottici distruiti su 6200 km di fibra e 9 nuovi PoP metropolitani. Inoltre ha migliorato i collegamenti con la Sardegna, regione candidata a ospitare l'Einstein Telescope [10], e fornirà una connettività fino a 400 Gbps. Una volta completata l'espansione della rete GARR-T si saranno aggiunti 5000 km di fibra ottica e una capacità complessiva di circa 40 Tbps.

La rete GARR, mostrata in Fig. 1.1, è presente su tutto il territorio nazionale grazie a oltre 24000 km in fibra ottica.

La rete GARR fa parte della rete della ricerca europea GÉANT [11]. GÉANT è la dorsale europea ad altissima capacità che interconnette tutte le reti della ricerca e dell'istruzione europee con collegamenti multipli fino a 100 Gbps e in grado di arrivare fino a 8 Tbps.

1.4 Run 4 di LHC

Il progetto High Luminosity LHC (HL-LHC) aumenterà di un fattore 10 la luminosità attuale di LHC. La luminosità è un importante indicatore delle prestazioni di un acceleratore, proporzionale al numero di collisioni che si verificano in un dato intervallo di tempo. Nel 2030 è prevista la quarta campagna di presa dati di LHC ed è previsto un incremento del volume di dati di circa 9 volte rispetto al Run 3. In particolare per HL-LHC la richiesta minima per il CNAF è di 690 Gbps, ma viste le esperienze passate è necessaria una banda doppia, cioè di 1380 Gpbs, per evitare la saturazione della rete e i problemi derivanti da ciò [12].

Nel 2023, la rete GARR e la rete europea GÉANT hanno collegato il CNAF e il centro di calcolo del CERN con una capacità di 1,6 Tbps e una latenza di 9,5 ms grazie allo spettro condiviso multidominio [13]. La Data Centre Interconnection (DCI) permetterà al Tier 1 del CNAF di partecipare, oltre che all'elaborazione dei dati offline, anche alla selezione degli eventi effettuata dalle trigger farm situate in prossimità degli esperimenti.

In preparazione del Run 4 è stato deciso di fare una serie di Data Challenge per verificare che l'infrastruttura di rete e i data center siano pronti al traffico generato dalla campagna di presa dati [14]. Due Data Challenge sono già state svolte: una nel 2021

con il 10% del traffico previsto e una nel 2024 con il 25%. Altre due Data Challenge porteranno queste percentuali al 50% nel 2027 e infine al 100% nel 2029.

In aggiunta alle Data Challenge previste, i singoli Tier 1 e Tier 2 possono richiedere delle Mini Data Challenge intermedie, mirate all'investigazione di problemi locali e alla verifica della loro risoluzione.

Capitolo 2

StoRM

In questo capitolo viene presentato StoRM e il protocollo WebDAV e la sua estensione Third-Party Copy. Viene inoltre illustrato il tipico deployment e i risultati di alcuni test di carico effettuati nel 2024.

2.1 StoRM: Storage Resource Manager

StoRM (Storage Resource Manager) è un servizio software sviluppato per la gestione di sistemi di archiviazione costituiti da file system POSIX, eventualmente associati a un sistema di storage su nastro. Nei centri di calcolo, l'esigenza di gestire un sistema distribuito di archiviazione dei dati porta all'impiego di file system distribuiti come IBM GPFS o Lustre.

StoRM nasce storicamente come implementazione della specifica SRM [15], che descrive uno standard middleware per la gestione di dati in sistemi di storage eterogenei. Utilizzata in combinazione con un protocollo di trasporto come GridFTP [16], è stato utilizzato a lungo per il trasferimento dei dati in WLCG. Nel tempo la suite di prodotti StoRM si è evoluta e ampliata, in particolare per supportare una migrazione a HTTP come protocollo di trasporto e usando il protocollo WebDAV [17], un'estensione di HTTP, per il management dei dati. Per la gestione dei dati residenti su nastro, StoRM ha contribuito alla definizione e all'implementazione della WLCG Tape REST API [18].

StoRM è una suite di componenti software; i principali componenti sono:

- **StoRM Frontend** e **StoRM Backend** che implementano ed espongono un'interfaccia di tipo SRM a client e framework. Implementano i comandi SRM in modo sincrono e asincrono utilizzando un database come mezzo di comunicazione.
- **StoRM GridFTP plugin** che, installato su un server GridFTP (un'estensione del protocollo FTP), viene usato per accedere direttamente al file system dietro un deployment di StoRM.
- **StoRM WebDAV** che fornisce le funzionalità di trasferimenti dati e gestione utilizzando il protocollo WebDAV (un'estensione del protocollo HTTP).
- **StoRM Tape** implementa la WLCG Tape REST API che permette la recall di file da nastro. Questo componente espone un endpoint interno usato dal sistema di gestione del tape.

StoRM supporta diversi meccanismi di autenticazione e autorizzazione, dai certificati X.509, estesi con con un Attribute Certificate di tipo VOMS (Virtual Organization Membership Service) [19], a OAuth JSON Web Token. Nativo è anche il supporto dei token rilasciati da provider basati su INDIGO IAM [20], un altro software sviluppato dall'INFN-CNAF che implementa un OIDC provider e scelto da WLCG per la migrazione della sua infrastruttura ai token. StoRM Tape gestisce l'autorizzazione delegandola a Open Policy Agent (OPA).

StoRM supporta i sistemi tape grazie all'integrazione di GEMSS [21], un sistema di Hierarchical Storage Management (HSM) che integra IBM GPFS, IBM Tivoli Storage Manager (TSM) con StoRM Backend e StoRM Tape.

2.2 Procollo WebDAV

Il protocollo WebDAV è un'estensione del protocollo HTTP. Permette agli utenti di creare, modificare e spostare documenti su un server. Oltre a definire i comportamenti e i requisiti dei metodi HTTP `GET`, `HEAD`, `PUT`, `DELETE` per risorse e collezioni (che nel caso di StoRM WebDAV corrispondono a file e cartelle), aggiunge dei metodi tra cui:

COPY per copiare una risorsa da un URI a un altro;

MKCOL per creare collezioni;

MOVE per spostare risorse;

PROPFIND per ottenere le proprietà di una risorsa;

PROPPATCH per impostare o eliminare proprietà di una risorsa;

LOCK per bloccare una risorsa;

UNLOCK per sbloccare una risorsa.

2.3 Third-Party Copy

La HTTP Third-Party Copy (HTTP-TPC) [22] è un'estensione del metodo del protocollo WebDAV `COPY` che permette di spostare grandi quantità di dati direttamente tra i data center WLCG, senza quindi passare per il client richiedente.

Gli esperimenti di LHC, infatti, hanno la necessità di trasferire grandi quantità di dati per aumentare la replicazione degli stessi e per trasferirli ai data center in cui verranno poi elaborati. Nel 2017 Globus ha annunciato che avrebbe deprecato il Globus Toolkit [23] che forniva l'implementazione di riferimento del protocollo GridFTP. La comunità WLCG ha quindi approfittato di questa deprecazione per passare a utilizzare HTTP evitando così di dipendere da protocolli specializzati quali SRM e GridFTP.

Per copiare i dati da un data center a un altro con una TPC, il client invia una richiesta HTTP `COPY` all'endpoint attivo, inserendo l'URL del secondo server tra gli header HTTP.

Esistono due modalità di trasferimento conseguenti alla richiesta `COPY` ricevuta dall'endpoint attivo:

Pull mode la richiesta contiene l'header `Source`, l'endpoint attivo effettua una richiesta `GET` verso l'endpoint passivo;

Push mode la richiesta contiene l'header `Destination`, l'endpoint attivo effettua una richiesta `PUT` verso l'endpoint passivo.

La modalità pull viene preferita poiché utilizza richieste `GET` che sono idempotenti e l'endpoint può bilanciare molteplici richieste effettuate in pipeline dividendole in diversi stream TCP paralleli in modo da aumentare il *throughput*.

Nella richiesta `COPY` possono essere inseriti ulteriori header con prefisso `TransferHeader` che sono copiati nella richiesta `GET` o `PUT` senza prefisso [24].

Dopo che una TPC è stata accettata dall'endpoint attivo, questo deve inviare al client dei *performance marker* (Perf Marker) che includono informazioni sul progresso del trasferimento dati, in particolare il numero di byte trasferiti.

Prima degli sviluppi effettuati per questa tesi, StoRM WebDAV supporta le TPC, ma non includeva nei Perf Marker le informazioni `RemoteConnections` e `Connection`.

`RemoteConnections` è un elenco delle connessioni remote attive per la TPC, nella forma di lista separata da virgole di elementi nella forma `<transport>:<ip>:<port>`. `Connection` è il nuovo header che sostituisce il precedente e che include anche le informazioni sull'indirizzo IP locale della TPC. Questo dato è interessante per vedere direttamente dai log quale server sta effettuando la TPC, in particolare nei casi in cui più server stanno dietro lo stesso alias DNS.

Come richiesto dal WLCG Data Organization Management Access evolution project (DOMA) sono stati aggiunti questi due dati ai Perf Marker inviati da StoRM WebDAV. Questa integrazione è stata semplificata dal fatto che questi dati venivano già ottenuti per essere inseriti nei pacchetti firefly degli SciTags (Sezione 4.2).

2.4 Deployment di StoRM

Nella Fig. 2.1 è illustrato il tipico deployment di StoRM con tutti i componenti. Gli endpoint SRM e GridFTP supportano l'autenticazione soltanto tramite VOMS proxy, mentre gli endpoint HTTP e WebDAV supportano anche gli access token OAuth.

I futuri deployment si stanno muovendo verso una configurazione senza i componenti SRM e gsiFTP, che è deprecato.

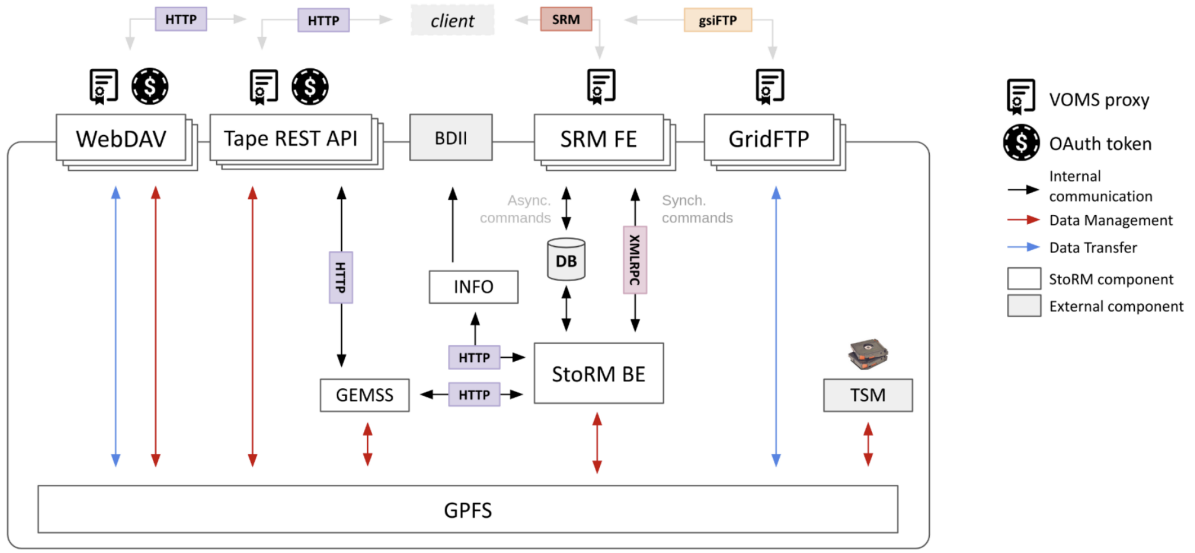


Figura 2.1: Il deployment di StoRM al momento dell'introduzione dei componenti HTTP. Le frecce rosse rappresentano le operazioni di gestione dei dati, quelle blu i trasferimenti dati e quelle nere le comunicazione interne tra i vari componenti.

2.5 Data Challenge 2024

Dal 12 al 23 febbraio 2024 si è svolta una Data Challenge che ha coinvolto tutti i centri di calcolo WLCG. Al Tier 1 del CNAF gli esperimenti ATLAS, CMS, LHCb e Belle II hanno utilizzato gli endpoint esposti dai componenti StoRM, in particolare l'interfaccia WebDAV [25], con risultati altalenanti.

Nel caso di ATLAS non sono stati raggiunti i rate previsti nella seconda settimana, ma è stato probabilmente dovuto a problemi riconducibili a FTS (File Transfer Service), il servizio di alto livello responsabile per la distribuzione della maggior parte dei dati nell'infrastruttura WLCG. In generale, i rate hanno superato i valori anticipati e sono stati notati dei fallimenti e disparità tra il monitor interno e quello di FTS.

I trasferimenti di CMS, anche grazie ad aggiustamenti a parametri di FTS fatti prima della Data Challenge, hanno funzionato durante la prima settimana. Gli endpoint di StoRM WebDAV del CNAF invece si sono saturati durante la seconda settimana, portando a molti fallimenti.

LHCb ha avuto diversi fallimenti dovuti ai troppi trasferimenti contemporanei. Riducendo i trasferimenti da parte di FTS da 200 a 50, il rate di successo è aumentato significativamente.

Il team di Storage riscontrava inoltre una saturazione dei thread di Jetty, il web server utilizzato da StoRM WebDAV, in caso di carico elevato. Si ritiene che questo fosse dovuto al fatto che Jetty crea almeno un thread per ogni richiesta e quando il numero di thread diventava molto superiore al numero di core del processore, il load dovuto al context switch fosse predominante, comportando il blocco completo di StoRM WebDAV. I riavvi del servizio erano giornalieri, tanto che veniva utilizzato un *remediator* di Sensu, cioè un *handler* per effettuare un'azione in automatico quando si verificano certe condizioni, per effettuare il *restart* del servizio in caso di problemi.

Capitolo 3

Evoluzione di StoRM

In questo capitolo vengono descritti i lavori di sviluppo svolti nel contesto di questa tesi per il miglioramento sia della codebase di StoRM WebDAV, sia del deployment sull'infrastruttura del Tier 1.

3.1 Nuovo deployment di StoRM

Nella Fig. 3.1 è mostrato il nuovo deployment di StoRM. Gli endpoint SRM e GridFTP sono stati rimossi poiché StoRM WebDAV e StoRM Tape permettono di eseguire tutte le operazioni di trasferimento dati e di management necessarie su sistemi di storage comprensivi di una tape library.

Per effettuare un completo passaggio dal deployment con i componenti SRM a quello basato solo su componenti HTTP, è necessario aggiungere al nuovo scenario il supporto allo Storage Resource Reporting (SSR) [26], nonché esporre le informazioni di reporting previste dal servizio BDII [27]. Tutte queste funzioni sono in capo al componente StoRM Info Provider che recupera queste informazioni da StoRM Backend. Nel futuro scenario basato solo su componenti HTTP, il componente StoRM Info Provider dovrà essere rivisto e disaccoppiato dal Backend SRM.

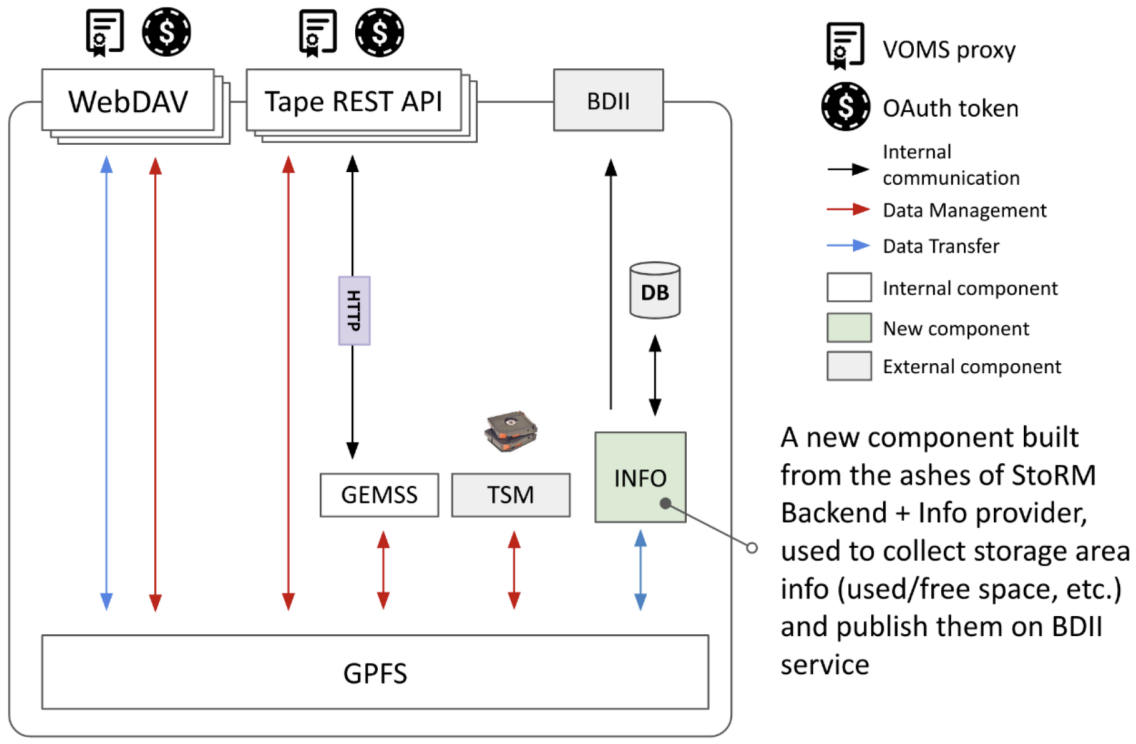


Figura 3.1: Il nuovo deployment di StoRM, soltanto con StoRM WebDAV e StoRM Tape. Per esporre le informazioni necessarie allo Storage Resource Reporting verrà creato un nuovo componente.

3.2 Sviluppi

StoRM WebDAV è un'applicazione sviluppata in Java e che si basa sul framework Spring Boot [28]. All'inizio degli sviluppi per questa tesi, l'applicazione utilizzava Spring Boot 2.7.18, non più supportato ufficialmente poiché giunto a fine ciclo di vita (End-of-Life) già nel giugno 2023 [29]. Per garantire la sicurezza e la manutenibilità del software, è stato quindi pianificato e realizzato l'aggiornamento a Spring Boot 3. StoRM WebDAV utilizza Jetty [30] come web server e Milton [31] come framework per l'implementazione del protocollo WebDAV.

Durante questa migrazione, si è deciso di fare un primo passaggio intermedio con l'aggiornamento di Spring Security alla versione 5.8, requisito propedeutico alla successiva

adozione della versione 6, versione utilizzata da Spring Boot 3.

Il passaggio a Spring Boot 3 ha permesso anche l'aggiornamento di Jetty dalla versione 9 alla più recente versione 12. Questo permette di beneficiare delle nuove funzionalità disponibili e di migliorare le prestazioni complessive del web server integrato.

Per incrementare il throughput dei trasferimenti dati si è ritenuto interessante provare a delegare alcune operazioni a NGINX [32]. NGINX è un web server tra i più diffusi e offre moltissime opzioni di configurazione per adattarlo al tipo di traffico previsto. In particolare è stato scelto per il supporto all'header `X-Accel-Redirect` che permette di effettuare un *redirect* interno. Si è scelto di concentrarsi sulle richieste `GET` poiché sono le richieste HTTP più pesanti dal punto di vista del traffico generato. Queste avvengono infatti quando un server è parte passiva di una Third-Party Copy in pull mode, che è la modalità di default quindi quella maggiormente usata. Non si è scelto di ottimizzare la parte attiva delle Third-Party Copy poiché, essendo richieste utilizzate solo all'interno di WLCG, non sono supportate da software di terze parti e richiedono un'implementazione ad hoc.

Il web server NGINX viene usato come terminazione SSL/TLS; questo permette di semplificare il codice di StoRM WebDAV ed evitare gli overhead dovuti all'implementazione Java delle librerie di crittografia. È stato sviluppato inoltre il modulo per NGINX `ngx_http_voms_module` [33] che si occupa dell'autenticazione tramite certificati proxy X.509 estesi con gli attributi VOMS [34]. Le informazioni ottenute dal modulo vengono inserite in alcuni header HTTP e la richiesta viene inoltrata a StoRM WebDAV. In questo scenario di deployment con NGINX, StoRM WebDAV in pratica si occupa di verificare l'autorizzazione della richiesta. È previsto che in futuro tale verifica venga delegata, completamente o in parte, a un engine esterno basato su OPA, come avviene già nel caso di StoRM Tape.

Nel caso di una richiesta `GET` che abbia superato i controlli di autorizzazione, StoRM WebDAV invia una risposta a NGINX con l'header HTTP `X-Accel-Redirect` al fine di redirigere la richiesta a una location interna di NGINX. Il valore dell'header permette di identificare il percorso reale del file richiesto in modo da permettere a NGINX di identificare il file da inviare al client. Per tutte le altre richieste diverse da `GET`, StoRM WebDAV si occupa di gestire la richiesta.

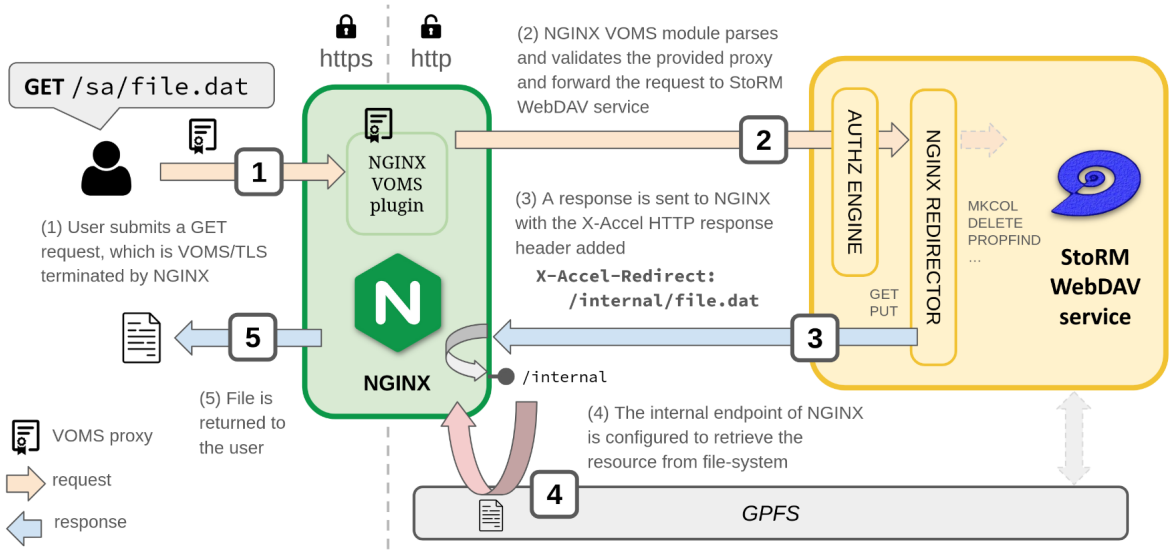


Figura 3.2: Esempio di una richiesta GET a StoRM WebDAV che utilizza NGINX.

La Fig. 3.2 mostra un esempio di una richiesta `GET` a StoRM WebDAV che utilizza NGINX:

1. il client richiede il file `/sa/file.dat`, NGINX è la terminazione TLS e il modulo `ngx_http_voms_module` effettua il parsing e l'autenticazione del certificato VOMS;
2. la richiesta viene inoltrata a StoRM WebDAV che si occupa dell'autorizzazione;
3. se l'autorizzazione ha successo viene inviata a NGINX una risposta con l'header `X-Accel-Redirect` contenente il percorso reale del file, in questo caso `/internal/file.dat`;
4. l'endpoint interno di NGINX recupera il file dal file system;
5. il file viene inviato al client.

A metà giugno 2025 il primo server StoRM WebDAV è stato aggiornato per utilizzare come reverse proxy NGINX, delegando a quest'ultimo le richieste `GET`. Questo server era uno di quelli presenti nel cluster di produzione dedicato all'esperimento ATLAS. È emersa da subito una migliore efficienza della nuova versione rispetto alla precedente: è

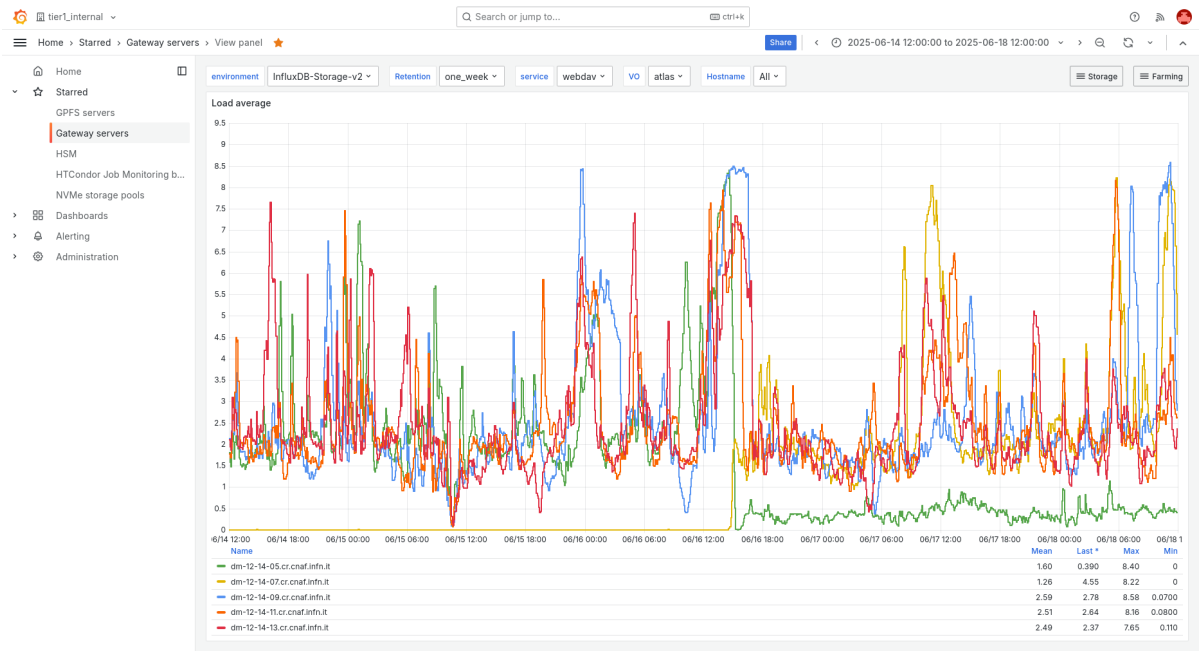


Figura 3.3: Load average sui server di StoRM WebDAV di ATLAS prima e dopo l'aggiornamento di uno dei server alla versione con delega delle GET a NGINX (linea verde).

possibile vedere il load average sui server prima e dopo l'aggiornamento di uno dei nodi nella Fig. 3.3.

Monitorando il load average per un paio di giorni è stato constatato un decremento del carico di 6 volte, come mostrato in Fig. 3.4.

Visti gli ottimi risultati ottenuti da questo singolo singolo server di StoRM WebDAV, rispetto agli altri nodi sottoposti allo stesso tipo di carico, in pochi giorni il nuovo deployment con la delega delle richieste `GET` a NGINX è stato ufficialmente adottato su tutto il cluster utilizzato dall'esperimento ATLAS. Tale aggiornamento di tutti i nodi ha confermato una netta diminuzione del carico sulla CPU, come visibile nella Fig. 3.5.

In seguito agli sviluppi che hanno portato alla delega delle richieste `GET`, è stato realizzato un proof-of-concept della delega a NGINX anche per quanto riguarda le richieste `PUT`. Il caso di una richiesta `PUT` è più complesso perché con la scrittura di un file deve essere calcolato il suo checksum Adler-32 [35]. Per implementare questa funzionalità non si è potuto utilizzare soltanto NGINX, ma è stato necessario utilizzare OpenResty [36],

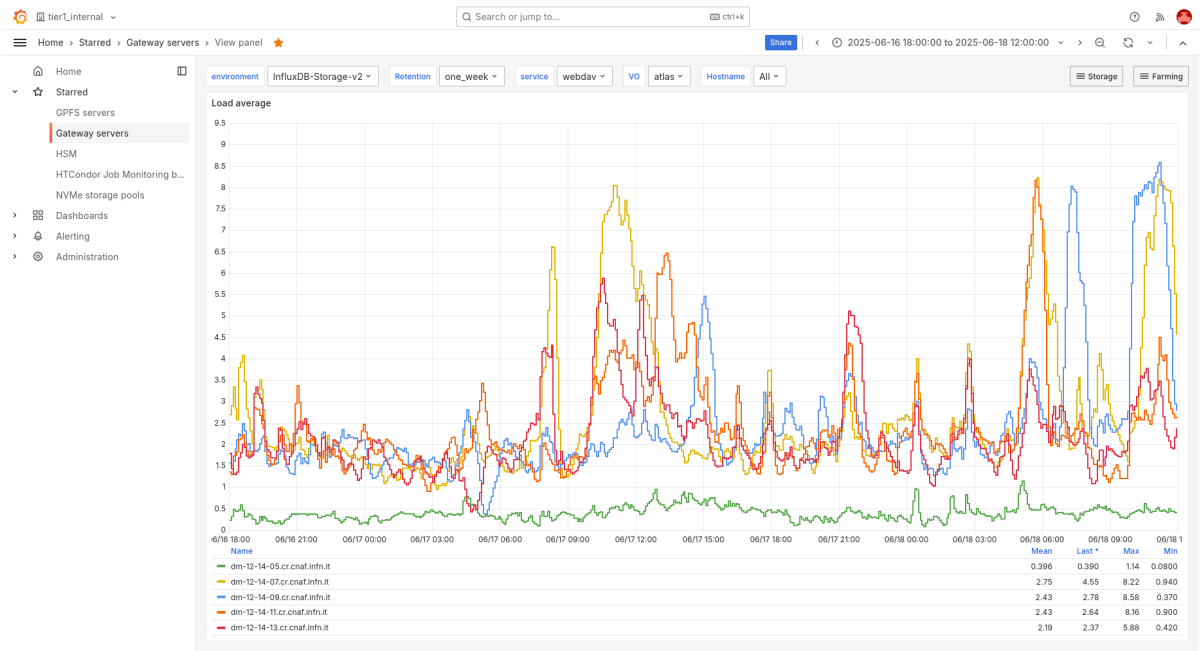


Figura 3.4: Load average sui server di StoRM WebDAV di ATLAS dopo l'aggiornamento di uno dei server alla versione con delega delle GET a NGINX (linea verde).

un fork di NGINX che permette di implementare in Lua [37] alcune funzionalità aggiuntive. Intercettando quindi con un piccolo script Lua il traffico di rete delle richieste `PUT`, è stato possibile calcolare e salvare il checksum negli attributi estesi del file.

3.3 Continuous Integration

La Continuous Integration (CI) è una pratica software che richiede l'invio frequente delle modifiche effettuate a una repository condivisa ed eseguire poi controlli continui e automatici che il codice pubblicato sia funzionante. Il repository con il codice sorgente di StoRM WebDAV si trova su GitHub e quindi è stato possibile definire dei workflow di GitHub Actions, la piattaforma di Continuous Integration di GitHub. Tali workflow sono stati definiti per eseguire una serie di controlli e azioni a seconda degli eventi che ne hanno innescato l'esecuzione.

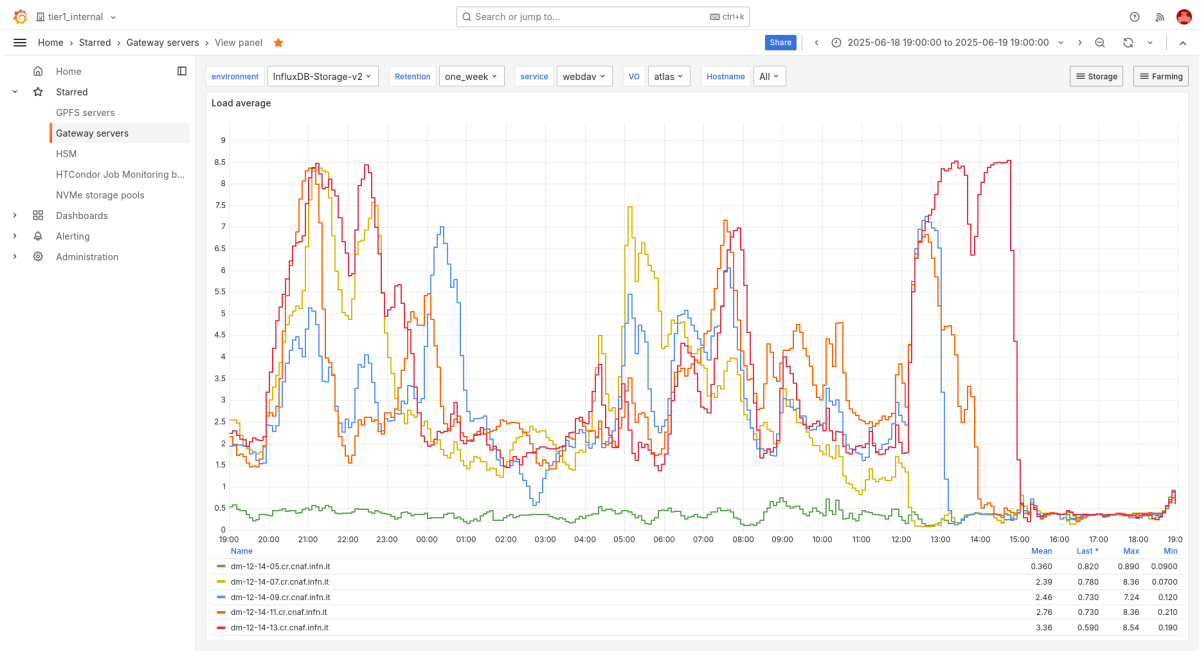


Figura 3.5: Grafico che illustra il netto calo di load average grazie all'aggiornamento di tutti i server di StoRM WebDAV del cluster dedicato ad ATLAS.

È stato definito, per esempio, un workflow che si occupa di verificare la corretta compilazione del codice e la corretta esecuzione degli unit test.

Utilizzando questo meccanismo di workflow, a ogni commit viene anche eseguita una suite di test creata utilizzando Robot Framework [38].

Un altro workflow esegue l'analisi statica del codice per ogni commit su un branch per cui esiste una Pull Request, integrandosi con il servizio SonarQube Cloud [39] dove è stato configurato un corrispondente progetto.

Per gestire i commenti sul copyright viene usato REUSE [40], come suggerito dal CERN Open Source Program Office [41], e un workflow si occupa di controllare che il progetto sia sempre conforme alle specifiche REUSE.

Per uniformare la formattazione del codice viene usato Spotless [42] e un workflow controlla che tutti i commit siano formattati correttamente.

Per velocizzare il rilascio di nuove versioni di StoRM WebDAV è stato sviluppato un nuovo workflow di GitHub Actions per creare gli RPM e automatizzare i rilasci. In

particolare:

- gli RPM vengono creati ogni volta che si fa un `git push`;
- ogni volta che si fa un merge sul branch principale, gli RPM vengono pubblicati nella repository `nightly`: la versione viene calcolata a partire da `git describe`;
- quando si crea un tag riferito a un commit nella forma `v<x>.<y>.<z>-<nome>`, gli RPM vengono pubblicati nella repository `beta` con la versione `<x>.<y>.<z>~<nome>`;
- quando si crea un tag riferito a un commit nella forma `v<x>.<y>.<z>`, gli RPM vengono pubblicati nella repository `stable` con la versione `<x>.<y>.<z>`.

Durante lo sviluppo si è cercato di rendere questo workflow il meno dipendente possibile da GitHub, usando per esempio script Bash invece di *action* di GitHub. È stato infatti possibile riutilizzare la maggior parte del codice per sviluppare la CI del modulo NGINX, il cui sorgente si trova su un'istanza di GitLab gestita dall'INFN.

Quando viene creato un tag in occasione del rilascio di una versione stabile viene creata una release su GitHub; inoltre viene caricata l'immagine Docker su Docker Hub e sul GitHub Container Registry.

3.4 Mini Data Challenge 2025

Dal 1 al 3 luglio 2025 è stata effettuata una Mini Data Challenge con la collaborazione dell'esperimento CMS. Per una adeguata comparazione dei risultati, l'esperimento ha ripetuto i suoi test sia nella configurazione con NGINX come reverse proxy e sia senza.

In entrambi i casi, quindi anche senza ricorrere alla soluzione con reverse proxy, grazie solamente agli sviluppi effettuati, StoRM WebDAV è riuscito a saturare le schede di rete dei server. Ogni server è dotato, infatti, di due schede di rete da 25 Gbps in bonding con load balancing XOR.

Dal punto di vista delle prestazioni dei trasferimenti invece, come auspicabile, l'utilizzo di NGINX ha portato a una complessiva diminuzione dei tempi di completamento delle richieste `GET`, come visibile nella Fig. 3.6.

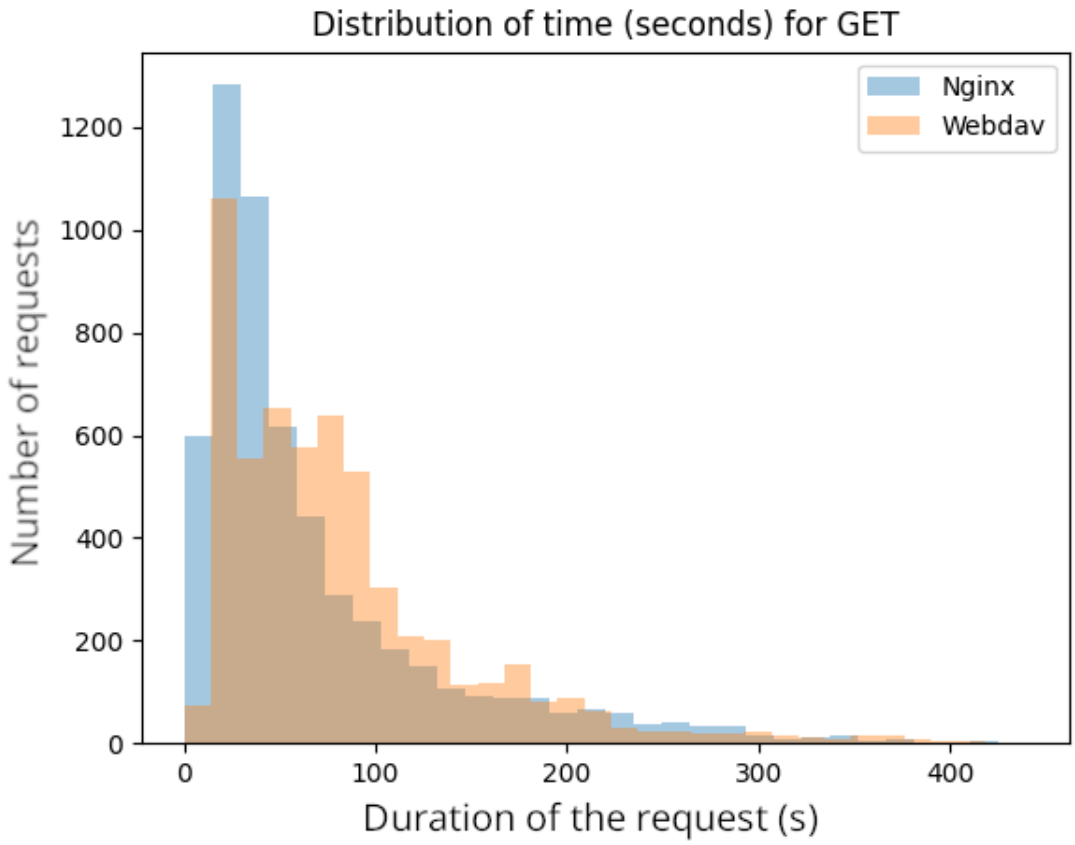


Figura 3.6: Grafico della distribuzione delle durate dei trasferimenti dati delle richieste GET.

Questa Mini Data Challenge ha permesso di confermare i miglioramenti dovuti agli sviluppi effettuati e all'utilizzo di NGINX. Inoltre ha fatto emergere delle limitazioni dovute all'utilizzo del bonding delle schede di rete con load balancing XOR. In particolare, queste limitazioni sembrerebbero essere dovute al bilanciamento non perfetto tra le due schede di rete e all'overhead del protocollo TCP per inviare per esempio i pacchetti `ACK`. I file sono infatti archiviati su vari nodi GPFS quindi, oltre al traffico di rete verso il client che ha richiesto per esempio un file, c'è anche il traffico dovuto alla lettura del file dai nodi GPFS. Quando una delle due schede di rete si satura, la connessione rallenta per il meccanismo di *congestion control* di TCP. Per esempio, quando un client fa una

richiesta `GET` (traffico in uscita) viene generato anche traffico in ingresso per la lettura dai nodi GPFS. Non essendo necessariamente queste due connessioni gestiste dalla stessa scheda di rete, può capitare che una delle due si saturi non permettendo di sfruttare al massimo il full-duplex.

3.5 Virtual thread

StoRM WebDAV utilizza un'architettura multithread dovuta a Jetty. Jetty utilizza almeno un thread per ogni richiesta (in alcuni casi anche più di uno) [43]. Occupandosi StoRM WebDAV di grandi trasferimenti di dati, la maggior parte delle richieste sono molto lunghe e I/O-bound. Si è notato che quando il numero di richieste contemporanee aumentava molto, il load average superava il numero di core del processore; di conseguenza i server diventavano molto meno responsivi. Il load average di Linux considera, oltre al numero di processi in esecuzione e in attesa di essere eseguiti, anche quelli bloccati per esempio per operazioni di I/O; quindi riflette in modo generico il carico di tutto il sistema e non solo delle CPU [44]. In particolare la percentuale di idle era molto alta; questo potrebbe essere dovuto a eccessivi context switch tra i vari thread utilizzati da StoRM WebDAV.

Dalla versione 21, Java supporta i *virtual thread* [45]. Normalmente i thread di Java utilizzano un thread del sistema operativo per tutto il tempo della loro esistenza. I thread virtuali invece, pur essendo sempre istanze di `java.lang.Thread`, non sono legati a thread di sistema. I virtual thread vengono eseguiti comunque su un thread di sistema operativo, ma quando chiamano un'operazione I/O bloccante, la runtime Java ne sospende l'esecuzione fino al completamento dell'operazione, liberando in questo modo il thread di sistema, che può essere utilizzato per eseguire altri thread virtuali. I virtual thread sono quindi pensati per task che passano molto tempo bloccati, per esempio per operazioni di I/O. Non permettono al codice di essere eseguito più velocemente, quindi la latenza non cambierà, ma permettono di scalare, aumentando il throughput.

Spring Boot supporta i virtual thread dalla versione 3.2. Visto che lo use case per cui è stato sviluppato il supporto ai thread virtuali è proprio quello di software simili a StoRM WebDAV, è stato aggiunto il supporto ai virtual thread anche a quest'ultimo.

Capitolo 4

SciTags

In questo capitolo viene presentata l'iniziativa SciTags e come è stato aggiunto il supporto al flow marking a StoRM WebDAV considerando anche il deployment che utilizza NGINX come reverse proxy.

4.1 Introduzione

SciTags (scientific network tags) [46] è un'iniziativa che promuove l'identificazione dei domini scientifici e della tipologia di attività a livello di rete. Nel caso di WLCG si tratta di identificare l'esperimento (ad esempio ATLAS o CMS) e l'attività ad alto livello (produzione, analisi, data challenge, etc.) in modo che i provider delle reti della ricerca e dell'educazione, come il GARR, possano collezionare queste informazioni e correlarle ad altri dati che hanno a disposizione.

Il traffico può essere marcato in due modi:

- inviando dei pacchetti UDP, chiamati *firefly*, oppure
- usando il campo *flow label* dell'header IPv6 dei pacchetti di rete.

Nel primo caso viene inviato un pacchetto UDP, contenente le informazioni sull'esperimento e l'attività che hanno generato il traffico, all'inizio e alla fine di ogni trasferimento alla porta 10514 dell'host remoto.

Nel secondo caso viene invece sfruttato il campo flow label dell'header IPv6 per aggiungere informazioni riguardo l'esperimento e l'attività che stanno generando il traffico. In particolare, l'identificatore dell'attività viene messo nei bit 24-29 e quello dell'esperimento nei bit 14-22 in ordine inverso per permettere futuri aggiustamenti del limite dei bit. I rimanenti 5 bit (12-13, 23, 30-31) sono usati per entropia: vengono impostati a valori casuali per ogni flusso dati.

Ogni flusso è identificato da un identificatore obbligatorio per l'esperimento, detto anche *virtual organization*, e un identificatore facoltativo per l'attività. Questi identificatori sono mappati staticamente in un flow registry [47].

Quando con una richiesta HTTP si inizia un trasferimento dati, i due identificatori sono inseriti nell'header HTTP `SciTag`:

```
<experimentId> << 6 | <activityId>
```

dove `<<` è l'operatore di bit shift a sinistra e `|` è l'OR bit-a-bit.

Il valore risultante è un intero positivo a 16 bit compreso tra 64 e 65536 (limiti esclusi perché non esistono attività con ID 0).

Per esempio, per un'attività di consolidamento dati (ID: 4) dell'esperimento ATLAS (ID: 2), l'header sarà `SciTag: 132`.

4.2 Integrazione in StoRM WebDAV

StoRM WebDAV ha aggiunto il supporto all'header `SciTag` nella versione 1.5.0 [48] sfruttando il daemon *flowd* [49].

Quest'ultimo offre una serie di plugin per ottenere le informazioni sugli identificatori dei flussi dati e un insieme di backend per marcare il traffico. In particolare, StoRM WebDAV utilizza il plugin `np_api` che permette a *flowd* di utilizzare una *named pipe* per ricevere i dati. Il backend `udp-firefly` di *flowd* viene poi usato per inviare i pacchetti UDP firefly.

StoRM WebDAV supporta quattro tipi di richieste per i trasferimenti dati, che sono interessanti ai fini del tracciamento: `GET`, `PUT`, TPC in push mode e TPC in pull mode. Nel caso delle TPC in pull mode, il client invia a StoRM WebDAV l'header

`TransferHeaderSciTag`; l'endpoint attivo in questo modo non marca il traffico, ma effettua una richiesta `GET` con il valore dell'header `SciTag` uguale a quello dell'header `TransferHeaderSciTag` ricevuto nella richiesta `COPY`. Negli altri casi, invece, StoRM WebDAV legge il valore dell'header `SciTag` per estrarre gli ID dell'esperimento e dell'attività e scrivere nella named pipe di *flowd* (`/var/run/flowd`) una linea strutturata in questo modo:

```
state protocol source_ip source_port dest_ip dest_port exp act
```

dove:

- `state` è `start`;
- `protocol` è `tcp`;
- `source_ip` e `source_port` sono l'indirizzo IP e la porta della sorgente del trasferimento dati;
- `dest_ip` e `dest_port` sono l'indirizzo IP e la porta della destinazione del trasferimento dati;
- `exp` e `act` sono gli ID dell'esperimento e dell'attività estratti dall'header `SciTag`.

In tutti i casi l'header `TransferHeaderSciTag` viene ignorato se è presente l'header `SciTag`.

Un esempio di una TPC in push mode tra i server A e B è mostrato nella Fig. 4.1:

1. il client invia una richiesta `COPY` al server A specificando il server B come `Destination` e con un header `SciTag`;
2. StoRM WebDAV sul server A estrae gli ID dell'esperimento dell'attività dall'header `SciTag`, scrive le informazioni necessarie nella pipe di *flowd* con lo stato `start` e inizia il trasferimento dati verso il server B. Nello stesso momento, *flowd* invia il pacchetto UDP firefly al server B;
3. i dati vengono trasferiti tra i server A e B;

4. una volta che il trasferimento è stato completato, StoRM WebDAV scrive le informazioni sulla pipe di *flowd* con lo stato `end` e *flowd* invia il corrispondente pacchetto UDP firefly al server B.

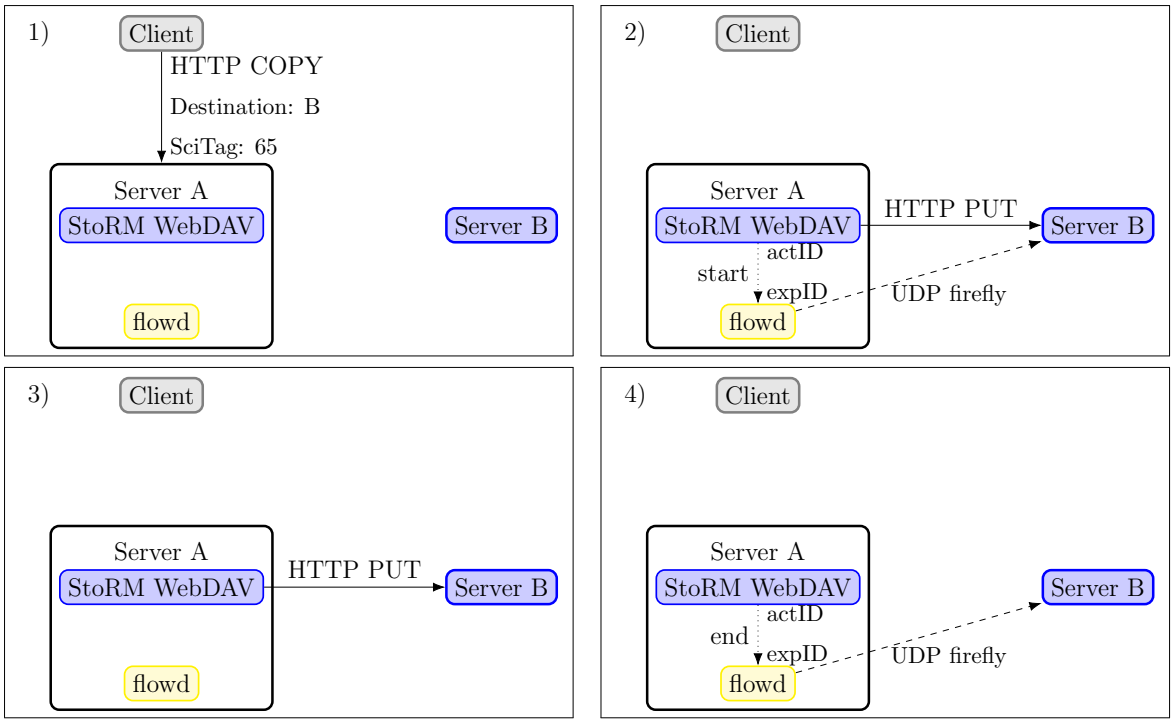


Figura 4.1: Esempio di una TPC in push mode.

L'attivazione di questa feature è facoltativa perché richiede l'installazione di *flowd*. Per semplificare l'adozione, il modulo Puppet di StoRM [50] permette con una singola flag di attivare il supporto a SciTags oltre che installare e configurare *flowd*.

Nel caso venga usato NGINX come reverse proxy è necessario inviare le informazioni riguardanti il client (indirizzo IP e porta sorgente e destinazione) a StoRM WebDAV in modo che possa comunicare queste informazioni a *flowd*, che le inserirà nel pacchetto UDP firefly. Queste informazioni vengono inserite da NGINX nell'header `Forwarded` [51].

```
http {
 map $remote_addr $forwarded_for {
  ~^[0-9.]+$          "for=\"$remote_addr:$remote_port\"";
```

```
  ~^[0-9A-Fa-f:.]+$   "for=\"[$remote_addr]:$remote_port\"";
  default             "for=unknown";
 }
 map $server_addr $forwarded_by {
  ~^[0-9.]+$          "by=\"$server_addr:$server_port\"";
  ~^[0-9A-Fa-f:.]+$   "by=\"[$server_addr]:$server_port\"";
  default             "host=unknown";
 }
}
server {
 location / {
  proxy_pass http://storm-webdav;
  proxy_pass_header Server;
  proxy_http_version 1.1;
  proxy_set_header Connection "";
  proxy_set_header Forwarded "$forwarded_by;$forwarded_for;host=$http_host";
 }
}
```

Questo header viene poi gestito dal `ForwardedHeaderFilter` di Spring Boot che fa l'override dei metodi `getRemoteHost()` e `getRemoteAddr()`. Poiché al momento questo filtro ignora il parametro `by` è stato sviluppato un ulteriore filtro in modo da fare l'override anche dei metodi `getLocalAddr()` e `getLocalPort()`.

Per fare le richieste HTTP verso i server passivi delle TPC viene usata la libreria Apache HttpClient 5, che, se da una parte rende molto semplice effettuare richieste HTTP, dall'altra complica molto l'ottenimento delle informazioni sul socket usato per il trasferimento che contiene informazioni che è necessario inserire nei pacchetti UDP firefly. Per ovviare a questo problema è stata estesa la classe `DefaultClientTlsStrategy` che definisce come passare da una connessione non criptata a una TLS.

Codice sorgente 4.1: Classe TpcTlsSocketStrategy utilizzata per ottenere informazioni sui socket usati per le TPC

```
1 public class TpcTlsSocketStrategy extends DefaultClientTlsStrategy {
```

```
2
3   public TpcTlsSocketStrategy(SSLContext sslContext) {
4     super(sslContext);
5   }
6
7   @Override
8   public SSLSocket upgrade(
9       Socket socket,
10      String target,
11      int port,
12      Object attachment,
13      HttpContext context)
14      throws IOException {
15    SSLSocket s = super.upgrade(socket, target, port, attachment, context);
16    SciTag scitag = (SciTag) context.getAttribute(SciTag.SCITAG_ATTRIBUTE);
17    if (scitag != null) {
18      SciTagTransfer scitagTransfer =
19          new SciTagTransfer(
20              scitag,
21              s.getLocalAddress().getHostAddress(),
22              s.getLocalPort(),
23              s.getInetAddress().getHostAddress(),
24              s.getPort());
25      scitagTransfer.writeStart();
26      context.setAttribute(
27          SciTagTransfer.SCITAG_TRANSFER_ATTRIBUTE,
28          scitagTransfer);
29    }
30    return s;
31  }
```

Nel caso si deleghi il trasferimento dati a NGINX, StoRM WebDAV non può sapere quando si conclude e di conseguenza non può inviare il firefly con `state end`. Per ovviare a questo problema, si è sfruttata la capacità di logging di NGINX per scrivere la riga necessaria nella pipe di *flowd*.

```
server {
 location /.storm-webdav/internal/get {
```

```
  internal;
  alias /;
  sendfile on;
  tcp_nopush on;
  keepalive_timeout 65;
  tcp_nodelay on;
  if ($upstream_http_x_scitag_actid) {
   access_log /var/run/flowd flowd;
  }
  add_header Server $upstream_http_server;
 }
}

user storm;

http {
 log_format flowd 'end tcp $server_addr $server_port '
  '$remote_addr $remote_port '
  '$upstream_http_x_scitag_actid $upstream_http_x_scitag_expid';
}
```

Capitolo 5

Tracing utilizzando eBPF

In questo capitolo viene presentata la tecnologia eBPF e come è possibile, utilizzando la libreria libbpf e in particolare il wrapper per Rust, fare tracing delle performance dei programmi. Per tracing si intende il catturare informazioni sull'esecuzione dei programmi, in particolare in tempo di esecuzione delle funzioni. L'utilizzo di questa tecnologia non richiede modifiche al codice dei programmi di cui si vogliono misurare le prestazioni. Inoltre, a volte può capitare che alcuni colli di bottiglia siano difficilmente replicabili in ambiente di test e il poter generare dei flamegraph [52] direttamente sui server di produzione senza interruzioni del servizio può essere molto utile.

5.1 eBPF

eBPF è una tecnologia del kernel Linux che permette di estenderne le funzionalità [53]. In particolare è possibile scrivere dei programmi eBPF e collegarli a specifici hook del kernel. Il modello di esecuzione dei programmi eBPF è event-driven, cioè vengono eseguiti quando il kernel o un'applicazione passa determinati punti di hook.

Nella mia tesi triennale [54] ho illustrato come poter utilizzare il framework eXpress Data Path (XDP) per collegare programmi eBPF alla ricezione di pacchetti di rete. XDP può essere utilizzato per implementare load balancer, firewall o monitorare il traffico di rete.

eBPF può essere anche utilizzato per fare tracing delle performance di programmi. È infatti possibile collegare programmi eBPF all'entrata e all'uscita di funzioni.

La Fig. 5.1 mostra il processo di compilazione e caricamento di programmi eBPF.

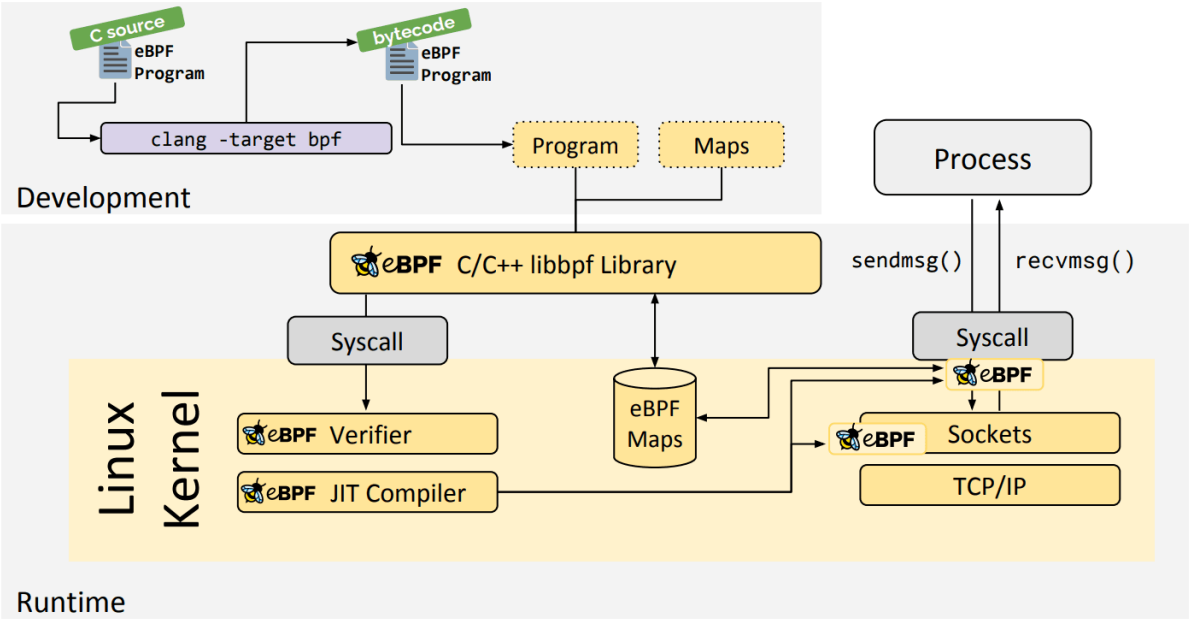


Figura 5.1: Schema della compilazione e del caricamento di programmi eBPF (Immagine di eBPF.io distribuita con licenza Creative Commons BY 4.0)

Il kernel Linux richiede che i programmi eBPF siano caricati come bytecode. Clang permette di compilare i programmi eBPF scritti in C in bytecode. Il bytecode può essere poi caricato nel kernel Linux, che si occupa dei due passi successivi.

Per prima cosa avviene una verifica per controllare che il programma soddisfi alcune condizioni, per esempio che non contenga loop infiniti, non acceda ad aree di memoria non inizializzate, non legga aree di memoria arbitraria o, nel caso di programmi che interagiscono con pacchetti di reti, non acceda fuori dai limiti del pacchetto.

Successivamente il kernel compila Just-in-Time il bytecode nel codice macchina dell'architettura su cui viene eseguito, permettendo ai programmi eBPF di essere efficienti quanto il codice compilato nativamente.

Le mappe permettono ai programmi eBPF di comunicare tra loro e con lo user space. Esistono diversi tipi di mappe: hash table, array, ring buffer, stack trace, etc.

5.2 BPF CO-RE e libbpf

Una delle limitazioni che in passato era presente nei programmi eBPF era la loro dipendenza da versioni specifiche del kernel. Una prima soluzione a questo problema è stata quella di compilare il programma eBPF direttamente sulla macchina in cui si voleva eseguire questi programmi. Per fare ciò sono state sviluppate librerie che semplificano questa operazione come per esempio BPF Compiler Collection (BCC) [55]. Lo svantaggio di questo approccio è che il programma finale deve includere, per compilare il programma eBPF, LLVM e Clang, che sono dipendenze abbastanza pesanti. Inoltre ogni volta che si vuole avviare un programma eBPF avviene la compilazione, operazione che può risultare pesante.

Per ovviare a queste limitazioni è stato sviluppato BPF CO-RE (Compile Once - Run Everywhere) [56]. In particolare sono state introdotte le BTF type information, che includono tutte le informazioni necessarie per la portabilità. La libreria libbpf [57] utilizza queste informazioni per adattare il programma eBPF alla versione del kernel a cui deve essere connesso.

Per utilizzare questa funzionalità per prima cosa è necessario estrarre le informazioni dal kernel su cui si sta sviluppando il progetto. È possibile fare ciò con il seguente comando:

```
bpftool btf dump file /sys/kernel/btf/vmlinux format c > vmlinux.h
```

Clang è stato migliorato per aggiungere al file oggetto dei programmi eBPF alcune informazioni aggiuntive. Per esempio aggiunge le rilocazioni BTF che catturano descrizioni ad alto livello delle informazioni a cui il programma vuole accedere. Per esempio, se un programma eBPF accede al campo `task_struct->pid`, Clang registrerà il fatto che si sta provando ad accedere al campo di nome `pid` di tipo `pid_t` all'interno di uno `struct task_struct`. In questo modo, se il campo viene spostato a un offset differente o dentro `struct` o `enum` anonimi nella versione kernel in cui si sta provando a usare il programma eBPF, è possibile fare la cosiddetta *field offset relocation*.

La libreria libbpf utilizza queste informazioni al momento del caricamento di un programma eBPF: prende il file oggetto del programma eBPF, lo processa per effettuare le rilocazioni necessarie a seconda del kernel su cui si sta eseguendo e avvia il collegamento e la verifica del programma eBPF. Questa libreria è parte del codice del kernel Linux e sono disponibili anche dei wrapper in Rust [58] e in Go [59], di cui è disponibile anche una versione scritta in Go puro [60] senza l'utilizzo di CGo.

Un'applicazione eBPF è composta da uno o più programmi eBPF, mappe e variabili globali (condivise tra tutti i programmi eBPF). libbpf metta a disposizione delle API che i programmi userspace possono usare per manipolare i programmi eBPF e gestire le varie fasi di un programma eBPF. Un'applicazione eBPF ha le seguenti fasi:

Fase di apertura libbpf apre il file oggetto eBPF in modo da sapere quali programmi eBPF, mappe e variabili globali contiene. In questa fase è possibile impostare le variabili globali.

Fase di caricamento libbpf crea le mappe, risolve le varie rilocazioni, verifica e carica i programmi eBPF. In questa fase è possibile impostare le mappe senza incorrere in race condition poiché i programmi non sono ancora stati eseguiti.

Fase di collegamento libbpf collega i programmi eBPF ai vari hook di sistema (entrata e uscita di funzione, arrivo di pacchetti di rete, etc.). I programmi vengono quindi eseguiti in modo event-driven e possono comunicare con lo userspace utilizzando le mappe e le variabili globali.

Fase di distruzione libbpf scollega i programmi eBPF e libera le risorse.

Gli skeleton eBPF sono un'interfaccia alternativa per interagire con gli oggetti eBPF che astraggono le API messe a disposizione da libbpf. Sono dei file che, oltre a contenere il bytecode eBPF, in modo da non avere altri file da distribuire oltre all'applicazione userspace, mettono a disposizioni funzioni per le varie fasi del programma eBPF e per interagire con le mappe e le variabili globali. Poiché questi skeleton vengono generati a partire dall'oggetto eBPF, contengono strutture dati che elencano tutti programmi eBPF, le mappe e le variabili globali; in questo modo si può evitare di doverli cercare per nome,

evitando così errori nel caso venissero rinominate nel codice sorgente dell'applicazione eBPF.

5.3 libbpf-rs

Nei seguenti esempi verrà utilizzato Rust e la libreria libbpf-rs. Per creare un nuovo progetto basta eseguire:

```
cargo init
cargo add libbpf-rs
cargo add --build libbpf-cargo
```

Il seguente codice contiene due programmi eBPF di tipo `uprobe` e `uretprobe`; questi tipi di programmi vengono eseguiti rispettivamente quando si invoca una funzione e quando si fa il `return`.

Codice sorgente 5.1: Esempio di due programmi eBPF collegati all'invocazione e all'uscita di una funzione

```
1  #include "vmlinux.h"
2  #include <bpf/bpf_tracing.h>
3
4  SEC("uprobe//bin/bash:readline")
5  int BPF_UPROBE(readline_enter) {
6      bpf_printk("Bash readline called");
7      return 0;
8  };
9
10 SEC("uretprobe//bin/bash:readline")
11 int BPF_URETPROBE(readline_exit) {
12     bpf_printk("Bash readline returned");
13     return 0;
14 };
15
16 char LICENSE[] SEC("license") = "GPL";
```

In particolare, sono state usate le macro `BPF_UPROBE` e `BPF_URETPROBE` per definire i due programmi eBPF: il primo collegato all'invocazione alla funzione `readline` dell'eseguibile `/bin/bash` e il secondo all'uscita della stessa.

La libreria libbpf-cargo permette di integrare la creazione degli skeleton eBPF con lo strumento Cargo di Rust. Per fare ciò è necessario creare un build script che utilizza lo `SkeletonBuilder` messo a disposizione da questa libreria.

Codice sorgente 5.2: Esempio di build script per generare lo skeleton

```
1  use std::env;
2  use std::path::PathBuf;
3
4  use libbpf_cargo::SkeletonBuilder;
5
6  const SRC: &str = "src/bpf/trace.bpf.c";
7
8  fn main() {
9      let out = PathBuf::from(
10         env::var_os("CARGO_MANIFEST_DIR")
11             .expect("CARGO_MANIFEST_DIR must be set in build script"),
12     )
13     .join("src")
14     .join("bpf")
15     .join("trace.skel.rs");
16
17     SkeletonBuilder::new()
18         .source(SRC)
19         .build_and_generate(&out)
20         .unwrap();
21     println!("cargo:rerun-if-changed={SRC}");
22 }
```

Il build script genererà automaticamente al momento della build un file skeleton `example.skel.rs` che semplifica l'interazione con questi programmi da Rust.

Codice sorgente 5.3: Esempio di programma Rust per aprire, caricare e collegare i programmi eBPF

```
1  use crate::example::ExampleSkelBuilder;
```

```
2  use libbpf_rs::skel::{OpenSkel, Skel, SkelBuilder};
3  use std::mem::MaybeUninit;
4
5  mod example {
6      include!(concat!(
7          env!("CARGO_MANIFEST_DIR"),
8          "/src/bpf/example.skel.rs"
9      ));
10 }
11
12 fn main() {
13     let example_builder = ExampleSkelBuilder::default();
14     let mut open_object = MaybeUninit::uninit();
15     let open_skel = example_builder.open(&mut open_object).unwrap();
16     let mut skel = open_skel.load().unwrap();
17     skel.attach().unwrap();
18     loop {}
19 }
```

Si può notare come gli skeleton semplifichino molto l'apertura, il caricamento e il collegamento di programmi eBPF. In particolare, `ExampleSkelBuilder` è uno `struct` che implementa il `trait SkelBuilder` che ci permette di aprire il file oggetto eBPF. È possibile poi usare lo skeleton aperto per caricare i programmi eBPF. Infine il metodo `attach` permette di collegare in automatico tutti i programmi eBPF.

È possibile compilare ed eseguire il programma con i seguenti comandi:

```
cargo build
sudo ./target/debug/ebpf-test
```

L'output di `bpf_printk` viene scritto nel trace log del kernel, quindi per visualizzarlo è necessario usare:

```
sudo cat /sys/kernel/debug/tracing/trace_pipe
```

Interagendo con un altro terminale è possibile notare che viene scritto `Bash readline called` ogni volta che viene presentato un prompt, mentre quando si fa Invio viene scritto `Bash readline returned`.

```
bash-42 [014] ...11 685.400167: bpf_trace_printk: Bash readline called
bash-42 [014] ...11 692.091196: bpf_trace_printk: Bash readline returned
```

5.4 blazesym

Per fare tracing delle performance è necessario per prima cosa ottenere l'elenco delle funzioni. Per ottenerlo è possibile utilizzare la libreria blazesym [61].

È possibile aggiunge questa libreria usando:

```
cargo add blazesym
```

Il seguente programma utilizza la libreria blazesym per aprire un file ELF preso come argomento. Tutti i simboli vengono aggiunti in un vettore e per ogni simbolo viene scritto il tipo (funzione o variabile), il nome e l'indirizzo.

Codice sorgente 5.4: Esempio di utilizzo della libreria blazesym per ottenere i simboli di funzione di un eseguibile

```
1  use std::{env, ops::ControlFlow};
2
3  use blazesym::inspect::{
4      Inspector,
5      source::{Elf, Source},
6  };
7
8  fn main() {
9      let args: Vec<String> = env::args().collect();
10     let file_path = &args[1];
11     let src = Source::Elf(Elf::new(file_path));
12     let inspector = Inspector::new();
13     let mut sym_infos = Vec::new();
14     inspector
15         .for_each(&src, |sym| {
16             sym_infos.push(sym.to_owned());
17             ControlFlow::Continue(())
18         })
19         .unwrap();
```

```
20      for sym in sym_infos {
21          println!("[{:?}] {} ({:#X})", sym.sym_type, sym.name, sym.addr);
22      }
23  }
```

È possibile eseguire questo programma usando:

```
$ cargo run -- ./example_debug
[Function] main (0x40047A)
[Function] add (0x400466)
```

5.5 Collegare i programmi eBPF a hook

Per fare tracing delle performance di un programma è necessario collegare un programma eBPF all'entrata e all'uscita delle funzioni di interesse.

Per fare ciò non è possibile utilizzare il metodo `attach`, usato nel Codice sorgente 5.3, che prova a collegare in automatico tutti i programmi eBPF, perché non è pensabile inserire i nomi delle funzioni di cui fare tracing in modo hardcoded.

libbpf-rs permette di collegare in modo manuale vari programmi eBPF a specifici hook.

Per prima cosa è necessario creare due mappe, la prima sarà di tipo stack trace (`BPF_MAP_TYPE_STACK_TRACE`) e servirà a salvare gli stack delle chiamate di funzione; l'altra sarà di tipo ring buffer (`BPF_MAP_TYPE_RINGBUF`) e verrà utilizzata per inviare le informazioni sulle chiamate di funzione dai programmi eBPF allo userspace.

Due programmi eBPF vengono usati per ottenere le informazioni sullo stack (utilizzando la funzione helper `bpf_get_stackid`), il tempo (`bpf_ktime_get_ns`), il PID e il TID (`bpf_get_current_pid_tgid`) e se è una chiamata o un'uscita di funzione.

I singoli eventi vengono inviati tramite il ring buffer in modo da poter essere elaborati dal programma in userspace.

Codice sorgente 5.5: Esempio di due programmi eBPF collegabili all'entrata e all'uscita di funzioni

```
1  #include "vmlinux.h"
2  #include <bpf/bpf_tracing.h>
```

```
3  #define PERF_MAX_STACK_DEPTH 127
4
5  struct trace_info {
6      u32 stackid;
7      u8 event_type;
8      u64 time;
9      u32 pid;
10     u32 tid;
11 };
12
13 struct trace_info _trace_info = {0};
14
15 struct {
16     __uint(type, BPF_MAP_TYPE_STACK_TRACE);
17     __type(key, u32);
18     __uint(value_size, PERF_MAX_STACK_DEPTH * sizeof(u64));
19     __uint(max_entries, 10000);
20 } stack_traces SEC(".maps");
21
22 struct {
23     __uint(type, BPF_MAP_TYPE_RINGBUF);
24     __uint(max_entries, 10000);
25 } events SEC(".maps");
26
27 SEC("uprobe")
28 int BPF_UPROBE(start) {
29     struct trace_info info = {};
30     info.stackid = bpf_get_stackid(ctx, &stack_traces, 0 | BPF_F_USER_STACK);
31     info.event_type = 0;
32     info.time = bpf_ktime_get_ns();
33     u64 pid_tgid = bpf_get_current_pid_tgid();
34     info.pid = pid_tgid >> 32;
35     info.tid = pid_tgid & 0xFFFFFFFF;
36     if (info.stackid > 0) {
37         bpf_ringbuf_output(&events, &info, sizeof(info), 0);
38     }
39     return 0;
40 }
```

```
41
42 SEC("uretprobe")
43 int BPF_URETPROBE(end) {
44     struct trace_info info = {};
45     info.stackid = bpf_get_stackid(ctx, &stack_traces, 0 | BPF_F_USER_STACK);
46     info.event_type = 1;
47     info.time = bpf_ktime_get_ns();
48     u64 pid_tgid = bpf_get_current_pid_tgid();
49     info.pid = pid_tgid >> 32;
50     info.tid = pid_tgid & 0xFFFFFFFF;
51     bpf_ringbuf_output(&events, &info, sizeof(info), 0);
52     return 0;
53 }
54
55 char LICENSE[] SEC("license") = "GPL";
```

Il programma in userspace salva tutti i simboli di tipo funzione trovati utilizzando la libreria blazesym in un vettore. Utilizza poi lo `SkelBuilder` per aprire il file oggetto eBPF e collega, per ogni simbolo di funzione del programma di cui si vuole fare tracing, i due programmi eBPF precedenti, usando il metodo `attach_uprobe_with_opts`. Successivamente fa polling degli eventi inviati al ring buffer. Ogni volta che un evento viene ricevuto, si accede alla mappa degli stack trace e per ogni simbolo viene usato il `Symbolizer` di blazesym per ottenere i nomi delle funzioni dello stack.

Per interpretare i byte che arrivano dal ring buffer come lo `struct trace_info` generato nello skeleton verrà usata la libreria `plain` che è possibile aggiungere usando:

```
cargo add plain
```

Codice sorgente 5.6: Esempio di collegamento di programmi eBPF a tutte le funzioni di un eseguibile

```
1 use std::{
2     env, mem::MaybeUninit, num::NonZeroU32, ops::ControlFlow,
3     time::Duration,
4 };
5
6 use blazesym::{
```

```
7       Pid, SymType,
8       inspect::{
9           Inspector,
10          source::{Elf, Source},
11      },
12      symbolize::{Input, Sym, Symbolized, Symbolizer, source::Process},
13  };
14  use libbpf_rs::{
15      Link, MapCore, MapFlags, RingBufferBuilder, UprobeOpts,
16      skel::{OpenSkel, Skel, SkelBuilder},
17  };
18  use plain::Plain;
19  mod trace {
20      include!(concat!(
21          env!("CARGO_MANIFEST_DIR"),
22          "/src/bpf/trace.skel.rs"
23      ));
24  }
25  #[allow(clippy::wildcard_imports)]
26  use trace::*;
27
28  unsafe impl Plain for trace::types::trace_info {}
29
30  fn main() {
31      let args: Vec<String> = env::args().collect();
32      let file_path = &args[1];
33      let src = Source::Elf(Elf::new(file_path));
34      let inspector = Inspector::new();
35      let mut sym_infos = Vec::new();
36      inspector
37          .for_each(&src, |sym| {
38              if sym.sym_type == SymType::Function {
39                  sym_infos.push(sym.to_owned());
40              }
41              ControlFlow::Continue(())
42          })
43          .unwrap();
44      let example_builder = TraceSkelBuilder::default();
```

```
45      let mut open_object = MaybeUninit::uninit();
46      let open_skel = example_builder.open(&mut open_object).unwrap();
47      let skel = open_skel.load().unwrap();
48      let object = skel.object();
49      let mut links: Vec<Link> = vec![];
50      for sym in sym_infos {
51          for prog in object.progs_mut() {
52              let opts = UprobeOpts {
53                  retprobe: prog.name().to_str().unwrap() == "end",
54                  ..Default::default()
55              };
56              let link = prog
57                  .attach_uprobe_with_opts(
58                      -1,
59                      src.path().unwrap(),
60                      sym.file_offset.unwrap().try_into().unwrap(),
61                      opts,
62                  )
63                  .expect("Failed to attach eBPF program");
64              links.push(link);
65          }
66      }
67      let symbolizer = Symbolizer::new();
68      let handle_events = |data: &[u8]| {
69          let mut trace_info = trace::types::trace_info::default();
70          plain::copy_from_bytes(&mut trace_info, data).expect("Wrong size");
71          let stacks = &skel.maps.stack_traces;
72          match stacks
73              .lookup(&trace_info.stackid.to_ne_bytes(), MapFlags::empty())
74          {
75              Ok(Some(stack)) => {
76                  let valid_addrs = stack
77                      .chunks_exact(8)
78                      .map(|chunk| {
79                          u64::from_ne_bytes(chunk.try_into().unwrap())
80                      })
81                      .filter(|&addr| addr != 0)
82                      .collect::<Vec<_>>();
```

```
83                  let src = blazesym::symbolize::source::Source::Process(
84                      Process::new(Pid::Pid(
85                          NonZeroU32::new(trace_info.pid)
86                              .expect("Negative PID"),
87                      )),
88                  );
89                  match symbolizer
90                      .symbolize(&src, Input::AbsAddr(&valid_addrs))
91                  {
92                      Ok(syms) => {
93                          for sym in syms {
94                              match sym {
95                                  Symbolized::Sym(Sym { name, .. }) => {
96                                      println!("{name}");
97                                  }
98                                  Symbolized::Unknown(reason) => {
99                                      println!("<no-symbol> ({reason})")
100                                 }
101                             }
102                         }
103                     }
104                     Err(e) => {
105                         eprintln!("Failed to symbolize addresses: {e}");
106                     }
107                 }
108             }
109             Ok(None) => {
110                 eprintln!("Stack id {} not found!", trace_info.stackid);
111             }
112             Err(e) => {
113                 eprintln!(
114                     "Failed to lookup stack id {}: {e}",
115                     trace_info.stackid
116                 );
117             }
118         }
119         println!("-----");
120         0
```

```
121     };
122     let mut builder = RingBufferBuilder::new();
123     builder
124         .add(&skel.maps.events, handle_events)
125         .expect("Failed to add RingBuffer");
126     let ringbuf = builder.build().unwrap();
127     loop {
128         ringbuf
129             .poll(Duration::from_millis(100))
130             .expect("Error polling")
131     }
132 }
```

5.6 Serializzare i dati nel Trace Event Format

Il Trace Event Format [62] è un formato JSON per rappresentare i dati di tracing.

Nello specifico, è composto da un oggetto JSON con la proprietà obbligatoria `traceEvents` che è un array di eventi. Ogni evento è un oggetto JSON con le seguenti proprietà:

name è il nome dell'evento;

cat è la lista separata da virgole delle categorie degli eventi;

ph è il tipo dell'evento, rappresentato da una singola lettera (per esempio `B` per l'inizio di un evento e `E` per la fine);

ts è il timestamp dell'evento;

tts è il timestamp del thread (opzionale);

pid è il process id;

tid è il thread id;

args sono gli argomenti aggiuntivi per l'evento.

Nel caso di eventi di tipo `E` solo i campi `pid`, `tid`, `ph` e `ts` sono obbligatori.

Per serializzare i dati in JSON verrà usata la libreria Serde JSON [63].

```
cargo add serde -F derive
cargo add serde_json
```

Viene utilizzata la `struct TraceEvent` che deriva il trait `Serialize` di Serde per salvare i singoli eventi. Viene usato un vettore di questi `struct` per salvare la lista degli eventi avvenuti. Ogni volta che viene ricevuto un evento dal ring buffer:

- se il vettore è vuoto, tutte le funzioni sullo stack vengono aggiunte come eventi di inizio; questo viene fatto perché il tracing potrebbe iniziare quando il programma che si vuole tracciare è già stato avviato;
- se è un evento di uscita da una funzione, viene aggiunto un evento di fine al vettore;
- se è un evento dovuto all'invocazione di una funzione, viene aggiunto un evento di inizio con le informazioni della funzione appena invocata che è quindi la prima dello stack.

Quando viene interrotta l'esecuzione del programma di tracing usando Ctrl+C, la libreria Serde si occupa di serializzare il vettore in JSON.

Codice sorgente 5.7: Programma che utilizza Serde JSON per serializzare i dati in Trace Event Format

```
1  use std::{
2      env,
3      mem::MaybeUninit,
4      num::NonZeroU32,
5      ops::ControlFlow,
6      sync::{
7          Arc,
8          atomic::{AtomicBool, Ordering},
9      },
10     time::Duration,
11 };
12
13 use blazesym::{
14     Pid, SymType,
15     inspect::{
```

```
16          Inspector,
17          source::{Elf, Source},
18      },
19      symbolize::{Input, Sym, Symbolized, Symbolizer, source::Process},
20  };
21  use libbpf_rs::{
22      Link, MapCore, MapFlags, RingBufferBuilder, UprobeOpts,
23      skel::{OpenSkel, Skel, SkelBuilder},
24  };
25  use plain::Plain;
26  use serde::Serialize;
27  mod trace {
28      include!(concat!(
29          env!("CARGO_MANIFEST_DIR"),
30          "/src/bpf/trace.skel.rs"
31      ));
32  }
33  #[allow(clippy::wildcard_imports)]
34  use trace::*;
35
36  unsafe impl Plain for trace::types::trace_info {}
37
38  #[derive(Serialize)]
39  struct TraceEvent {
40      #[serde(skip_serializing_if = "Option::is_none")]
41      name: Option<String>,
42      cat: String,
43      ph: char,
44      ts: u64,
45      #[serde(skip_serializing_if = "Option::is_none")]
46      tts: Option<u64>,
47      pid: u32,
48      tid: u32,
49  }
50
51  fn main() {
52      let args: Vec<String> = env::args().collect();
53      let file_path = &args[1];
```

```
54      let src = Source::Elf(Elf::new(file_path));
55      let inspector = Inspector::new();
56      let mut sym_infos = Vec::new();
57      inspector
58          .for_each(&src, |sym| {
59              if sym.sym_type == SymType::Function {
60                  sym_infos.push(sym.to_owned());
61              }
62              ControlFlow::Continue(())
63          })
64          .unwrap();
65      let example_builder = TraceSkelBuilder::default();
66      let mut open_object = MaybeUninit::uninit();
67      let open_skel = example_builder.open(&mut open_object).unwrap();
68      let skel = open_skel.load().unwrap();
69      let object = skel.object();
70      let mut links: Vec<Link> = vec![];
71      for sym in sym_infos {
72          for prog in object.progs_mut() {
73              let opts = UprobeOpts {
74                  retprobe: prog.name().to_str().unwrap() == "end",
75                  ..Default::default()
76              };
77              let link = prog
78                  .attach_uprobe_with_opts(
79                      -1,
80                      src.path().unwrap(),
81                      sym.file_offset.unwrap().try_into().unwrap(),
82                      opts,
83                  )
84                  .expect("Failed to attach eBPF program");
85              links.push(link);
86          }
87      }
88      let mut events = vec![];
89      let symbolizer = Symbolizer::new();
90      {
91          let handle_events = |data: &[u8]| {
```

```
92              let mut trace_info = trace::types::trace_info::default();
93              plain::copy_from_bytes(&mut trace_info, data)
94                  .expect("Wrong size");
95              if trace_info.event_type == 1 {
96                  if !events.is_empty() {
97                      events.push(TraceEvent {
98                          name: None,
99                          cat: String::from(""),
100                         ph: 'E',
101                         ts: trace_info.time,
102                         tts: None,
103                         pid: trace_info.pid,
104                         tid: trace_info.tid,
105                     });
106                 }
107                 return 0;
108             }
109             let stacks = &skel.maps.stack_traces;
110             match stacks.lookup(
111                 &trace_info.stackid.to_ne_bytes(),
112                 MapFlags::empty(),
113             ) {
114                 Ok(Some(stack)) => {
115                     let valid_addrs = stack
116                         .chunks_exact(8)
117                         .map(|chunk| {
118                             u64::from_ne_bytes(chunk.try_into().unwrap())
119                         })
120                         .filter(|&addr| addr != 0)
121                         .collect::<Vec<_>>();
122                     let src = blazesym::symbolize::source::Source::Process(
123                         Process::new(Pid::Pid(
124                             NonZeroU32::new(trace_info.pid)
125                                 .expect("Negative PID"),
126                         )),
127                     );
128                     match symbolizer
129                         .symbolize(&src, Input::AbsAddr(&valid_addrs))
```

```
130                 {
131                     Ok(syms) => {
132                         if !events.is_empty() {
133                             if let Some(Symbolized::Sym(Sym {
134                                 name,
135                                 ..
136                             })) = syms.first()
137                             {
138                                 events.push(TraceEvent {
139                                     name: Some(name.to_string()),
140                                     cat: String::from(""),
141                                     ph: 'B',
142                                     ts: trace_info.time,
143                                     tts: None,
144                                     pid: trace_info.pid,
145                                     tid: trace_info.tid,
146                                 });
147                             }
148                         } else {
149                             for sym in syms.iter().rev() {
150                                 if let Symbolized::Sym(Sym {
151                                     name,
152                                     ..
153                                 }) = sym
154                                 {
155                                     events.push(TraceEvent {
156                                         name: Some(name.to_string()),
157                                         cat: String::from(""),
158                                         ph: 'B',
159                                         ts: trace_info.time,
160                                         tts: None,
161                                         pid: trace_info.pid,
162                                         tid: trace_info.tid,
163                                     });
164                                 }
165                             }
166                         }
167                     }
```

```
168                         Err(e) => {
169                             eprintln!(
170                                 "Failed to symbolize addresses: {e}"
171                             );
172                         }
173                     }
174                 }
175                 Ok(None) => {
176                     eprintln!(
177                         "Stack id {} not found!",
178                         trace_info.stackid
179                     );
180                 }
181                 Err(e) => {
182                     eprintln!(
183                         "Failed to lookup stack id {}: {e}",
184                         trace_info.stackid
185                     );
186                 }
187             }
188             0
189         };
190         let mut builder = RingBufferBuilder::new();
191         builder
192             .add(&skel.maps.events, handle_events)
193             .expect("Failed to add RingBuffer");
194         let ringbuf = builder.build().unwrap();
195         let running = Arc::new(AtomicBool::new(true));
196         let r = running.clone();
197         ctrlc::set_handler(move || {
198             r.store(false, Ordering::SeqCst);
199         })
200         .expect("Error setting Ctrl-C handler");
201         while running.load(Ordering::SeqCst) {
202             ringbuf.poll_raw(Duration::from_millis(100));
203         }
204     }
205     println!(
```

```
206             "{{\"traceEvents\": {}}}",
207             serde_json::to_string(&*events).unwrap()
208         );
209     }
```

È possibile quindi fare tracing di un programma usando:

```
sudo ./target/debug/ebpf-tracing-rs ./fib_debug > trace.json
```

Si può utilizzare per esempio Perfetto UI [64] per visualizzare sotto forma di flamegraph i dati raccolti. Nella Fig. 5.2 viene mostrato il risultato nel caso di un programma che calcola un numero della successione di Fibonacci in modo ricorsivo.

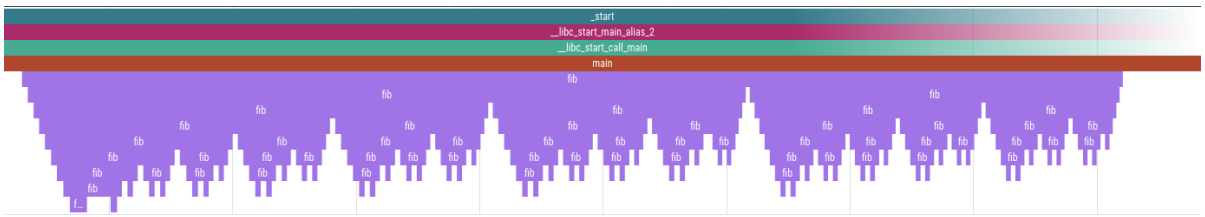


Figura 5.2: Visualizzazione di un flamegraph utilizzando Perfetto UI.

Conclusioni e sviluppi futuri

In questa tesi è stata presentata la suite di software StoRM e gli sviluppi effettuati per preparare StoRM WebDAV al Run 4 di LHC. Sono state aggiornate le versioni di Spring Boot e Jetty. Inoltre è stato aggiunto il supporto al deployment usando come reverse proxy NGINX, delegando le richieste `GET` a quest'ultimo. Grazie a questi sviluppi sono state risolte delle criticità riscontrate durante la Data Challenge del 2024 e ridotto il load average dei server di 6 volte. La stabilità è aumentata drasticamente e non avvengono più riavvii dovuti a problemi di StoRM WebDAV che prima erano invece giornalieri. Questi risultati positivi sono stati verificati grazie anche a una Mini Data Challenge svoltasi all'inizio di luglio. L'utilizzo della Continuous Integration ha velocizzato lo sviluppo e il rilascio di nuove versioni. Infine l'utilizzo dei virtual thread sembra promettente. L'ultimo sviluppo necessario per passare a un deployment senza i componenti SRM è l'aggiornamento di StoRM Info Provider che dovrà essere disaccoppiato da StoRM Backend.

Inoltre è stata presentata l'iniziativa SciTags e come è stato aggiunto il supporto al flow marking a StoRM WebDAV.

Nell'ultimo capitolo è stato illustrato come è possibile utilizzare eBPF per fare tracing delle performance di un programma senza che sia necessario modificarne il codice. È stato inoltre utilizzato Rust e le librerie libbpf-rs e blazesym per mostrare come si può effettuare tracing con eBPF e mostrare i dati sotto forma di flamegraph. Questo potrà per esempio essere utilizzato per trovare colli di bottiglia in StoRM.

Bibliografia

[1] O. Brüning, H. Burkhardt, and S. Myers, "The large hadron collider," *Progress in Particle and Nuclear Physics*, vol. 67, no. 3, pp. 705–734, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0146641012000695

[2] CERN, "Worldwide LHC Computing Grid," accessed: 1 September 2025. [Online]. Available: https://wlcg-public.web.cern.ch/

[3] I. Foster and C. Kesselman, Eds., *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998.

[4] F. Ian and K. Carl, *The History of the Grid*. IOS Press, 2011. [Online]. Available: https://doi.org/10.3233/978-1-60750-803-8-3

[5] L. dell'Agnello, T. Boccali, D. Cesini, L. Chiarelli, A. Chierici, S. Dal Pra, D. De Girolamo, A. Falabella, E. Fattibene, G. Maron, D. Michelotto, L. Morganti, A. Prosperini, V. Sapunenko, and S. Zani, "INFN Tier–1: a distributed site," *EPJ Web Conf.*, vol. 214, p. 08002, 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921408002

[6] A. Chierici, D. Michelotto, G. Sergi, A. Pascolini, and D. Lattanzio, "Moving a data center keeping availability at the top," *EPJ Web Conf.*, vol. 337, p. 01277, 2025. [Online]. Available: https://doi.org/10.1051/epjconf/202533701277

[7] A. Sciabà, "Hardware technology trends in HEP computing," *EPJ Web Conf.*, vol. 337, p. 01325, 2025. [Online]. Available: https://doi.org/10.1051/epjconf/202533701325

[8] Consortium GARR, "Storia della rete GARR," accessed: 1 September 2025. [Online]. Available: https://www.garr.it/it/infrastrutture/rete-nazionale/storia-della-rete-garr

[9] ——, "La rete GARR-T," accessed: 1 September 2025. [Online]. Available: https://www.garr.it/it/infrastrutture/rete-nazionale/rete-garr-t

[10] "Einstein Telescope Italia," accessed: 1 September 2025. [Online]. Available: https://www.einstein-telescope.it/

[11] Consortium GARR, "Rete internazionale," accessed: 1 September 2025. [Online]. Available: https://www.garr.it/it/infrastrutture/rete-internazionale

[12] S. Campana, *WLCG data challenges for HL-LHC - 2021 planning*, Sep. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.5532452

[13] Consortium GARR, "Data Centre Interconnection: 1,6 Tbps tra CERN e Bologna grazie allo spectrum sharing," accessed: 1 September 2025. [Online]. Available: https://www.garr.it/it/news-e-eventi/2213-data-centre-interconnection-1-6-tbps-tra-cern-e-bologna-grazie-allo-spectrum-sharing

[14] K. V. Ellis, "The WLCG Data Challenge," *EPJ Web Conf.*, vol. 337, p. 01327, 2025. [Online]. Available: https://doi.org/10.1051/epjconf/202533701327

[15] F. Donno, L. Abadie, P. Badino, J.-P. Baud, E. Corso, S. D. Witt, P. Fuhrmann, J. Gu, B. Koblitz, S. Lemaitre, M. Litmaath, D. Litvintsev, G. L. Presti, L. Magnoni, G. McCance, T. Mkrtchan, R. Mollon, V. Natarajan, T. Perelmutov, D. Petravick, A. Shoshani, A. Sim, D. Smith, P. Tedesco, and R. Zappi, "Storage resource manager version 2.2: design, implementation, and testing experience," *Journal of Physics: Conference Series*, vol. 119, no. 6, p. 062028, jul 2008. [Online]. Available: https://doi.org/10.1088/1742-6596/119/6/062028

[16] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The Globus Striped GridFTP Framework and Server," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, 2005, pp. 54–54.

[17] L. M. Dusseault, "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)," RFC 4918, Jun. 2007. [Online]. Available: https://www.rfc-editor.org/info/rfc4918

[18] CERN TWiki, "TapeRestAPI," accessed: 1 September 2025. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/LCG/TapeRestAPI

[19] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, K. Lőrentey, and F. Spataro, "From gridmap-file to VOMS: managing authorization in a Grid environment," *Future Generation Computer Systems*, vol. 21, no. 4, pp. 549–558, 2005, high-Speed Networks and Services for Data-Intensive Grids: the DataTAG Project. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X04001682

[20] A. Ceccanti, M. Hardt, B. Wegh, A. Millar, M. Caberletti, E. Vianello, and S. Licehammer, "The INDIGO-Datacloud Authentication and Authorization Infrastructure," *Journal of Physics: Conference Series*, vol. 898, no. 10, p. 102016, oct 2017. [Online]. Available: https://doi.org/10.1088/1742-6596/898/10/102016

[21] P. P. Ricci, D. Bonacorsi, A. Cavalli, L. Dell'Agnello, D. Gregori, A. Prosperini, L. Rinaldi, V. Sapunenko, and V. Vagnoni, "The Grid Enabled Mass Storage System (GEMSS): the Storage and Data management system used at the INFN Tier1 at CNAF." *Journal of Physics: Conference Series*, vol. 396, no. 4, p. 042051, dec 2012. [Online]. Available: https://doi.org/10.1088/1742-6596/396/4/042051

[22] B. Bockelman, A. Ceccanti, F. Furano, P. Millar, D. Litvintsev, and A. Forti, "Third-party transfers in WLCG using HTTP," *EPJ Web Conf.*, vol. 245, p. 04031, 2020. [Online]. Available: https://doi.org/10.1051/epjconf/202024504031

[23] V. Vasiliadis, "Support for Open Source Globus Toolkit Ends January 2018," 2017, accessed: 1 September 2025. [Online]. Available: https://www.globus.org/blog/support-open-source-globus-toolkit-ends-january-2018

[24] CERN TWiki, "HTTP / WebDAV Third-Party-Copy Technical Details," accessed: 1 September 2025. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/LCG/HttpTpcTechnical

[25] M. Lassnig and C. Wissing, "WLCG/DOMA Data Challenge 2024: Final Report," Jun. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.11444180

[26] CERN TWiki, "Storage Space Accounting introduction," accessed: 1 September 2025. [Online]. Available: https://twiki.cern.ch/twiki/bin/view/LCG/StorageSpaceAccounting

[27] L. Field and M. Schulz, "Grid Deployment Experiences," 2005. [Online]. Available: https://cds.cern.ch/record/865688

[28] Spring Website, "Spring Boot," accessed: 1 September 2025. [Online]. Available: https://spring.io/projects/spring-boot

[29] ——, "Spring Boot support," accessed: 1 September 2025. [Online]. Available: https://spring.io/projects/spring-boot#support

[30] "Jetty." [Online]. Available: https://jetty.org/

[31] "Milton." [Online]. Available: https://milton.io/

[32] F. Giacomini, F. Agostini, L. Bassi, J. Gasparetto, R. Miccoli, and E. Vianello, "Enhancing StoRM WebDAV data transfer performance with a new deployment architecture behind NGINX reverse proxy," in *Proceedings of International Symposium on Grids & Clouds (ISGC) 2024 — PoS(ISGC2024)*, vol. 458, 2024, p. 030.

[33] INFN-CNAF, "ngx_http_voms_module." [Online]. Available: https://baltig.infn.it/cnafsd/ngx_http_voms_module

[34] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, J. Jensen, D. Kelsey, A. Frohner, D. L. Groep, W. S. de Cerff, M. Steenbakkers, G. Venekamp, D. Kouril,

A. McNab, O. Mulmo, M. Silander, J. Hahkala, and K. Lhorentey, "Managing Dynamic User Communities in a Grid of Autonomous Resources," 2003. [Online]. Available: https://arxiv.org/abs/cs/0306004

[35] L. P. Deutsch and J. loup Gailly, "ZLIB Compressed Data Format Specification version 3.3," RFC 1950, May 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1950

[36] "OpenResty." [Online]. Available: https://openresty.org/en/

[37] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, "The evolution of Lua," in *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, ser. HOPL III. New York, NY, USA: Association for Computing Machinery, 2007, p. 2–1–2–26. [Online]. Available: https://doi.org/10.1145/1238844.1238846

[38] Robot Framework Foundation, "Robot Framework." [Online]. Available: https://robotframework.org/

[39] SonarSource, "SonarQube Cloud." [Online]. Available: https://www.sonarsource.com/products/sonarcloud/

[40] Free Software Foundation Europe, "REUSE." [Online]. Available: https://reuse.software/

[41] G. Tenaglia, "Open Source at CERN and WLCG," accessed: 1 September 2025. [Online]. Available: https://indico.cern.ch/event/1506690/contributions/6341673/attachments/3011594/5310068/Open%20Source%20at%20CERN%20and%20WLCG%20-%20CodiMD.pdf

[42] DiffPlug, "Spotless." [Online]. Available: https://github.com/diffplug/spotless

[43] Webtide, "Jetty Threading Architecture," accessed: 1 September 2025. [Online]. Available: https://jetty.org/docs/jetty/12.1/programming-guide/arch/threads.html

[44] B. Gregg, "Linux Load Averages: Solving the Mystery," accessed: 1 September 2025. [Online]. Available: https://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html

[45] Oracle, "Virtual Threads," accessed: 1 September 2025. [Online]. Available: https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html

[46] G. Attebury, M. Babik, D. Carder, T. Chown, A. Hanushevsky, B. Hoeft, A. Lake, M. Lambert, J. Letts, S. McKee, K. Newell, and T. Sullivan, "Identifying and Understanding Scientific Network Flows," *EPJ Web of Conf.*, vol. 295, p. 01036, 2024. [Online]. Available: https://doi.org/10.1051/epjconf/202429501036

[47] "SciTags Flow Registry," accessed: 1 September 2025. [Online]. Available: https://scitags.org/api.json

[48] F. Agostini, L. Bassi, J. Gasparetto, F. Giacomini, R. Miccoli, and E. Vianello, "Evolving StoRM WebDAV: Delegation of file transfers to NGINX and support for SciTags," *EPJ Web Conf.*, vol. 337, p. 01182, 2025. [Online]. Available: https://doi.org/10.1051/epjconf/202533701182

[49] M. Babik and T. Sullivan, "flowd." [Online]. Available: https://github.com/scitags/flowd

[50] INFN-CNAF, "StoRM Puppet module." [Online]. Available: https://forge.puppet.com/modules/cnafsd/storm/readme

[51] A. Petersson and M. Nilsson, "Forwarded HTTP Extension," RFC 7239, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7239

[52] B. Gregg, "The flame graph," *Commun. ACM*, vol. 59, no. 6, p. 48–57, May 2016. [Online]. Available: https://doi.org/10.1145/2909476

[53] eBPF.io authors, "eBPF Documentation," accessed: 1 September 2025. [Online]. Available: https://ebpf.io/what-is-ebpf/

[54] L. Bassi, "Bilanciamento del carico per servizi di accesso ai dati a elevata efficienza utilizzando eXpress Data Path," Ph.D. dissertation, 2023. [Online]. Available: https://amslaurea.unibo.it/id/eprint/29243/

[55] IO Visor Project, "BPF Compiler Collection (BCC)," accessed: 1 September 2025. [Online]. Available: https://github.com/iovisor/bcc

[56] A. Nakryiko, "BPF CO-RE (Compile Once – Run Everywhere)," accessed: 1 September 2025. [Online]. Available: https://nakryiko.com/posts/bpf-portability-and-co-re/

[57] The kernel development community, "libbpf Overview," accessed: 1 September 2025. [Online]. Available: https://docs.kernel.org/bpf/libbpf/libbpf_overview.html

[58] "libbpf-rs." [Online]. Available: https://github.com/libbpf/libbpf-rs

[59] Aqua Security, "libbpfgo." [Online]. Available: https://github.com/aquasecurity/libbpfgo

[60] Cilium, "ebpf-go." [Online]. Available: https://ebpf-go.dev/

[61] D. Müller and Kui-Feng, "blazesym." [Online]. Available: https://github.com/libbpf/blazesym

[62] nduca and dsinclair, "Trace Event Format," accessed: 1 September 2025. [Online]. Available: https://docs.google.com/document/d/1CvAClvFfyA5R-PhYUmn5OOQtYMH4h6I0nSsKchNAySU/edit?tab=t.0#heading=h.yr4qxyxotyw

[63] E. Tryzelaar and D. Tolnay, "Serde JSON." [Online]. Available: https://github.com/serde-rs/json

[64] Perfetto Team, "Perfetto UI." [Online]. Available: https://ui.perfetto.dev/

Ringraziamenti

Un primo grandissimo ringraziamento va sicuramente al GARR che mi ha permesso, grazie alla borsa studio "Orio Carlini", di passare quasi 2 anni in un ambiente stimolante come quello dell'INFN-CNAF. Sono stato molto fortunato a poter collaborare con i tecnologi del reparto di sviluppo software del CNAF che sono stati gentilissimi ad accogliermi. Mi hanno anche permesso di presentare questi sviluppi ad alcune conferenze che sono state delle esperienze incredibili.

Ringrazio i miei relatori Renzo Davoli, Francesco Giacomini ed Enrico Vianello che mi hanno seguito durante la stesura di questa tesi.

Grazie ai miei genitori, mio fratello e i nonni per tutto l'affetto che mi fanno sentire ogni giorno.

Durante questi anni bolognesi ho incontrato coinquilini, diventati poi amici, fantastici che hanno reso questo periodo indimenticabile, in particolare: "mamma" Silvia, "queen" Greta, Francesco, Chiara, Marco e Serena.

Non posso poi dimenticarmi dei miei amici nerd e della comitiva del "gruppo montagna" (e non solo) con cui abbiamo passato troppe avventure insieme per essere elencate qui.