Università degli Studi di Ferrara
Corso di Laurea in Informatica

Profiling, analisi delle prestazioni e proposte per l'ottimizzazione del RDBMS MySQL utilizzato dal progetto DIRAC/LHCbDIRAC.

Relatore:
Dott. Luca Tomassetti

Laureando:
Alberto Mesin

Anno Accademico 2012-2013

Indice

Introduzione

Il lavoro presentato in questa tesi riguarda lo studio, l'analisi e la formulazione di proposte per il miglioramento del database di back-end del progetto DIRAC/LHCbDIRAC. LHCbDIRAC, basato su DIRAC, è il sistema di sottomissione per l'accesso all'infrastruttura distribuita Grid per l'esperimento LHCb del CERN. Ad esso è affidata la gestione dei job di Produzione, Merge, Ricostruzione degli Eventi e Analisi per i dati sperimentali e simulati. Il sistema utilizza un RDBMS MySQL per la gestione di numerosi database. La volontà di passare ad un motore relazionale e transazionale per la definizione schemi e la possibilità che, in un recente futuro, il DBMS possa rappresentare un serio limite alle prestazioni del sistema stesso hanno reso necessario questo studio. Il lavoro svolto si è concentrato sul profiling di un singolo schema relazionale per il quale sono stati utilizzati metodi di analisi e fornite soluzioni ai problemi riscontrati il quanto più possibile generali e per tanto validi per l'intero sistema.

La tesi si compone di quattro capitoli. Nel primo capitolo viene fatta una breve descrizione di cosa siano le Grid e quali sono le problematiche legate al loro sviluppo e uso, in particolare viene posto l'accento sul problema dell'ottimizzazione nell'uso delle risorse dei sistemi completamente distribuiti. Nel secondo capitolo vengono descritte struttura e funzionamento del sistema DIRAC, e anche in questo caso si cerca di concentrarsi sulle caratteristiche peculiari che lo distinguono dagli altri middleware oltre che, ovviamente, sul ruolo ricoperto, in esso, dal database oggetto di studio. Nel terzo capitolo sono discussi gli strumenti e metodologie impiegate nell'analisi dello schema relazionale TransformationDB, vengono presentati i risultati e esposti i problemi riscontrati ad ogni livello. Nel quarto ed ultimo capitolo viene affrontato il problema del deployment delle istanze del DBMS del progetto LHCbDIRAC. È presentata e motivata, confrontandola con le alternative considerate oltre che con l'attuale configurazione, la soluzione che si ritiene più consona.

Capitolo 1

Grid Computing

Molti problemi odierni, soprattutto nel campo della ricerca, richiedono l'impiego di ingenti risorse computazionali. Localizzare in un singolo sito le risorse necessarie in certi casi risulta sconveniente se non addirittura impossibile, oltre che quasi sempre inefficiente.

I sistemi di Grid Computing permettono di realizzare una struttura di risorse computazionali distribuite geograficamente. Un sistema Grid deve permettere l'accesso coordinato alle risorse distribuite da parte degli utenti. Le comunità di utenti delle strutture Grid vengono definite Virtual Organizations (VO). Il Grid Computing è un servizio finalizzato alla condivisione della potenza di calcolo e della capacità di storage sull'infrastruttura Internet. Le istituzioni e gli individui che mettono a disposizione le proprie risorse computazionali per un fine comune fanno parte della stessa VO.

Questo tipo di applicazione sta dando un grande contributo alla ricerca scientifica permettendo a varie istituzioni in tutto il mondo di analizzare e stoccare grandi quantità di dati. Attualmente esistono centinaia di Grid al mondo, ognuna creata per rispondere alle esigenze di un determinato progetto di ricerca o di un gruppo di utenti. Tuttavia già si comincia a pensare ad una Grid estesa a livello globale che le riunisca tutte in un unico sistema distribuito.

L'obbiettivo finale di una Grid è quello di fornire un servizio simile a quello della rete di distribuzione elettrica. L'utente finale deve poter disporre di un'interfaccia con la quale richiedere le risorse (potenza di calcolo, storage di dati, software applicativi, ecc...) senza doversi curare di come e dove queste risorse vengono rese disponibili. Un sistema di Grid Computing serve a fornire un ambiente di calcolo data-intensive le cui applicazioni devono accedere facilmente, e in maniera affidabile, a grandi quantità di dati geograficamente distribuiti.

Le tecniche di Grid Computing sono ormai in grado di fornire all'utente l'astrazione di un SuperComputer con capacità di calcolo e storage impensabili per una singola macchina o cluster.

Figura 1.1: Rappresentazione grafica, concettuale, della distribuzione geografica di risorse e utenti nei sistemi Grid.

1.1 Cos'è il MiddleWare

Le moderne architetture di Grid Computing prevedono un livello software, definito middleware, fra le risorse computazionali, siano esse di storage o di calcolo, e gli utenti delle VO. Il Middleware è tipicamente formato da una collezione di software (Resource Broker, Storage Broker, Network Monitor, ecc...) che forniscono all'utente l'interfacciamento trasparente con l'infrastruttura Grid. La funzione prima del middleware è quella di fare il match fra le risorse richieste dai Job e quelle che lo stato della Grid rende disponibili al momento, in modo da aumentare l'efficienza globale del sistema e non solo soddisfare le singole richieste degli utenti.

1.1.1 Grid senza MiddleWare

BOINC è un apprezzato software per la realizzazione di Grid che segue un'architettura server-client. Forse il più conosciuto progetto di calcolo distribuito, e collaborativo, realizzato su BOINC è SETI@home, tuttavia lo stesso CERN sta utilizzando BOINC per due progetti Grid: LHC@home e LHC@home Test4Theory. La struttura delle Grid realizzate con BOINC è

stellata, un unico nodo al centro-stella (Server) fa da dispatcher per i job da sottomettere ai nodi foglia (Client). Un client fa riferimento ad un unico Server da cui riceve l'input e a cui ritorna l'output.

Non è obbiettivo di questa tesi discutere il funzionamento delle Grid BOINC-style, tuttavia sottolineare la differenza dello scenario da queste offerto con quello delle Grid, complemante distribuite, gestite e utilizzate dalle VOs, dove l'utenza e il controllo delle risorse è inter-istituzionale, può dare una migliore visione di quali problemi comporti affrontare la schedulazione in queste ultime. In una Grid stellata è facile individuare nel/nei Server/s (nodo centrale) un'entità in grado di tenere sotto controllo le risorse e il carico del sistema; è quindi possibile fare in modo che questa agisca da scheduler, per l'intero sistema, applicando tecniche e algoritmi poco differenti da quelli classici. È altresì facile pensare che il Server sia gestito da una singola istituzione che può definire in libertà le proprie politiche in qualunque ambito.

1.1.2 La necessità del Resource Broker

Volendo modellare le risorse distribuite di una Grid come un unico Super-Computer si ripropone uno dei problemi fondamentali dei Sistemi Operativi; quello della schedulazione e gestione delle risorse condivise.

Il Resource Broker è un componente fondamentale del MiddleWare nei sistemi Grid. Esso ha l'onere di assegnare le risorse ai vari Jobs (altrimenti detti Gridlets) in modo compatibile sia con i requisiti del Job stesso che con le esigenze dettate dallo stato del sistema. Anche se lo scheduling è un problema classico dell'informatica per cui sono stati sviluppati, nel corso degli anni, diversi algoritmi e tecniche, nessuno degli approcci precedenti si è rilevato efficace sull'architettura Grid. La differenza principale delle Grid, rispetto alle altre architetture computazionali precedenti, sta nell'impossibilità di implementare un vero e proprio scheduler centrale in grado di avere l'effettivo controllo su tutte le risorse e su tutti i Job. Un ostacolo alla realizzazione di un classico scheduler centralizzato è costituito dall'eterogeneità, propria della Grid, delle risorse e dei Job; entrambi possono appartenere a diverse istituzioni, ognuna delle quali può stabilire differenti: criteri di schedulazione e accesso alle risorse (priorità), politiche di sicurezza, modelli di affidabilità/diponibilita delle risorse, requirements minimi per i propri job. Le continue oscillazioni nel carico e nella quantità di risorse disponibili, l'impossibilità di realizzare un vero controllo centralizzato, l'eterogeneità dei Job sottomessi al sistema rendono la realizzazione di uno scheduling efficiente molto più complicata che nei sistemi localizzati classici. Le risorse che, tipicamente, il Resource Broker deve amministrare includono: i sistemi di calcolo (normalmente Worker Node o WN), i sistemi di Storage (normalmente Sto-

rage Element o SE) per cui può appoggiarsi ad un Storage Broker e la rete di connessioni (normalmente la comune infrastruttura Internet pubblica coadiuvata, alle volte, da qualche trunk privato, a banda larga, fra le istituzioni maggiori) per cui può servirsi di un Network Monitor.

1.1.3 Storage Broker

Lo Storage Broker (SB) è un filesystem logico distribuito, spesso basato nelle sue implementazioni sull'architettura client-server. Questo strato software fornisce agli utenti l'astrazione di una unica gerarchia di nomi logici per i files. In accordo con la configurazione e le politiche decise dalla Virtual Organization, il Resource Broker provvede quella che viene definita "data grid", una file-library persistente e, contemporaneamente, file-system distribuito. Il Storage Broker fornisce un'interfaccia uniforme alle disomogenee risorse di archiviazione fornite, attraverso la rete, dalla grid. Fra le mansioni del Storage Broker vi è quella di implementare il namespace logico, globale, per i files e mantenere i metadati relativi ai data-objects. Normalmente le implementazioni del SB si appoggiano ad un database relazionale, detto Metadata Catalog, per mantenere metadati, relativi agli oggetti gestiti, come: users, groups, resources, collections, ecc... Questo rende possibile, attraverso delle interrogazioni alla base di dati, rintracciare un data-object o ottenere facilmente più informazioni a riguardo.

Dovendo lavorare su diverse piattaforme, sia hardware che software, il Storage Broker è implementato come un software di alto livello, le cui funzionalità si basano, a loro volta, su altri software di livello più basso. Il Resource Broker, che in certe architetture Grid può essere costituito da più software, ha il compito fondamentale di associare ad ogni data-object, tipicamente file o directory, un identificativo univoco e, a secondo di chi, e quando, lo richiede determinare, di volta in volta, la migliore istanza e metodo d'accesso.

1.2 Grid e SuperComputer

Un'altra soluzione adottata nel caso in cui si abbiano da affrontare problemi computazionali complessi è quella dei SuperComputer. A differenza delle ingenti spese da sostenere per acquistare l'hardware dedicato usato nei supercomputer, le Grid consentono di realizzare potenti sistemi di calcolo usando materiale normalmente reperibile sul mercato. Buona parte delle Grid è costituita da materiale per PC. La Grid consente quindi una maggiore scalabilità in caso si vogliano aumentare le capacità computazionali del sistema. L'essere pensato per risorse eterogenee le consente al MiddleWare

di poter usufruire di ogni risorsa messa a disposizione dalla VO. Adottare hardware di uso generico consente, inoltre, di poterlo riutilizzare più facilmente per altri scopi. Lavorare su architetture, software come hardware, ben conosciute e documentate aiuta nello sviluppo e nel testing del software da sottomettere al sistema.

Punto debole delle Grid è la scarsa larghezza di banda e l'alta latenza che possono riscontrarsi nei collegamenti fra i nodi del sistema, si dice infatti che le Grid sono formate da nodi debolmente accoppiati. Certi problemi di calcolo richiedono uno scambio frequente di dati fra i vari processi di calcolo paralleli; in questo caso della memoria condivisa fra due thread sulla stessa macchina è una soluzione decisamente migliore rispetto alla rete Internet pubblica o qualsiasi altra rete IP. Inoltre, in situazioni dove un singolo processo di calcolo dipende fortemente da un'altro avere uno scheduler centralizzato è necessario.

Capitolo 2

DIRAC

Tecnicamente DIRAC è un middleware pensato per comunità, medio-grandi, di utenti che devono accedere a risorse distribuite. In realtà la definizione di middleware, seppur tecnicamente giusta e per questo utilizzata in questo documento, va un poco stretta al progetto DIRAC che si autodefinisce come Interware. Il design di DIRAC è stato pensato per permettergli di formare uno strato intermedio tra le varie risorse computazionali, anche disomogenee, disponibili e le Virtual Organization (VO), al fine di fornire agli utenti di queste ultime un accesso: unico, trasparente, affidabile e ottimizzato. DIRAC non si sostituisce al Middleware preesistente, come ad esempio gLite, della Grid, ma lo sfrutta, modellandolo come una Resource, al fine di fornire servizi di più alto livello all'utenza. Il particolare design del progetto DIRAC gli permette di fornire un'ottimizzazione del rendimento globale del sistema intervenendo solo a livello utente, senza quindi andare a toccare la struttura e l'organizzazione delle risorse sottostanti.

2.1 Caratteristiche generali

Il modello di Workload Management System che utilizza i Pilot Jobs, introdotto da DIRAC, è stato largamente adottato da varie infrastrutture di Grid-Computing per la sua capacità di aggregare, in un singolo sistema, risorse di differente natura (Grids, Cluster, Clouds) in maniera completamente trasparente all'utente.

L'architettura di DIRAC prevede la cooperazione di vari Distributed Services e Light Agents, implementati sullo stesso framework DISET seguendo gli standard di sicurezza per il Grid Computing. I Pilot Agents permettono di implementare un efficiente Workload Management Systems (WMS). Il

workload prodotto dall'intera comunità è ottimizzato attraverso l'uso di una Task Queue centrale.

Il Production Management System di DIRAC funziona sfruttando i servizi di Workload e Data Management. Questo permette di effettuare una sottomissione dei Job pilotata dai dati, realizzando un workflow di arbitraria complessità. DIRAC fornisce, in questo ambito, un set completo di strumenti.

DIRAC fornisce un Workload Management Systems specificatamente progettato per essere resistente ai continui fallimenti e all'instabilità tipica degli ambienti Grid. Il progetto include un versatile Data Management System (DMS), ottimizzato per il trasferimento affidabile dei dati, che automatizza i task di trasferimento dati effettuati di ruotine.

DIRAC, implementato come Grid middleware per l'esperimento LHCb, è stato il primo ad usare i Pilot Jobs. Tale caratteristica fornisce un'interfaccia omogenea a risorse computazionali disomogenee. Contemporaneamente, i Pilot Jobs consentono di ritardare le decisioni di scheduling all'ultimo momento utile, permettendo perciò di prendere in considerazione le effettive condizioni di esecuzione sulla risorsa selezionata e le eventuali richieste dell'ultimo momento effettuate al sistema.

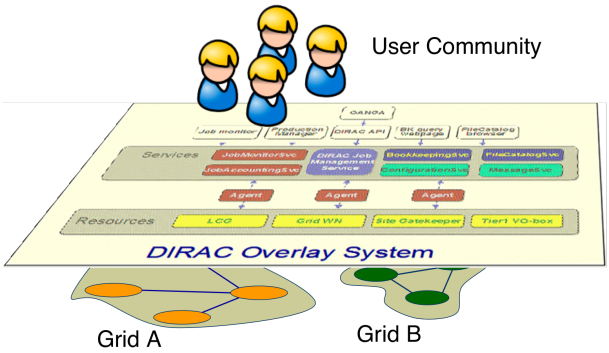

Figura 2.1: Rappresentazione dell'astrazione fornita da DIRAC che permette di nascondere all'utente l'implementazione delle varie risorse computazionali distribuite.

Il Workload Management System di DIRAC fornisce un unico meccanismo di scheduling per i jobs anche se questi hanno profili molto diversi fra loro. Per ottenere un livello globale di ottimizzazione, il WMS organizza i Job pendenti in Task Queues, indipendentemente dal fatto che essi appartengano a singoli utenti o a processi di produzione. Le Task Queues sono composte da Job che hanno requisiti simili. Le priorità delle varie code vengono assegnate rispettando le politiche definite dalla VO. La sottomissione

dei Pilots ed il Job-Matching avvengono seguendo un approccio statistico che tiene conto della priorità delle code.

2.2 Il framework DISET

DISET è il framework si cui si basano i servizi di comunicazione, autorizzazione ed autenticazione forniti da DIRAC. Il DISET tradizionale fornisce meccanismi di Remote Procedure Call (RPC) e trasferimento files. Un normale meccanismo RPC tuttavia non è del tutto adatto per il framework Executor. Il normale modello RPC non prevede la possibilità, da parte del Server, di inviare dati al Client in maniera asincrona. Inoltre ogni chiamata, da parte del Client, richiede l'instaurazione di una nuova connessione che deve passare attraverso tutto l'handshake SSL. Statisticamente si è notato che l'overhead SSL consuma la maggior parte della banda impegnata dalle RPC.

Si è deciso di basare il framework *Executor* su un nuovo DISET con maggiori capacità. La nuova versione del framework prevede il supporto alle connessioni persistenti e alle chiamate asincrone. Qualsiasi componente del sistema può riutilizzare una connessione precedentemente aperta per inviare e ricevere messaggi. I Services possono inviare dati ai Clients senza attendere che questi facciano polling verso di loro. L'eliminazione della prassi del polling aumenta ulteriormente il throughput. Anche se solo il Client può aprire la connessione verso il Service questo, poi, la può utilizzare in modo asincrono. La possibilità esclusiva di aprire la connessione è l'unica differenza, dal punto di vista della comunicazione, che differenzia Clients e Services.

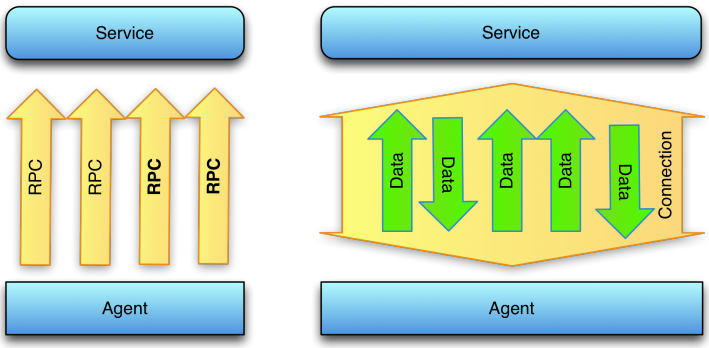

Figura 2.2: Rappresentazione grafica delle connessioni permanenti nel FrameWork DISET; principale differenza con RPC.

2.3 Struttura e Deployment

I componenti principali in cui si può dividere DIRAC sono: i Database, i Servizi (Services), gli Agenti (Agents). Queste entità, combinate in modo da interagire, formano i Sistemi (Systems) di DIRAC.

- **Database**: Mantengono persistente lo stato dei Sistemi. Vengono usati da Servizi e Agenti che vi accedono usandoli come una sorta di memoria condivisa.

- **Servizi**: Sono i componenti passivi che restano in ascolto delle richieste in arrivo dai Client. In base alle richieste ricevute possono: interrogare il database di Back-End, per fornire dati, o inserirvi le informazioni relative alle richieste ricevute. Un Servizio può comportarsi come Client nei confronti di altri Servizi, dello stesso o di altri Sistemi di DIRAC.

- **Agenti**: Sono i componenti attivi, che girano ininterrottamente e invocano periodicamente l'esecuzione dei loro metodi. Gli agenti sono le entità che animano l'intero sistema, attraverso l'esecuzione effettiva delle azioni e l'invio di richieste, sia ad altre parti di DIRAC che a servizi esterni.

- **Sistemi**: Fornisce, al resto del sistema DIRAC, una soluzione per una data classe di operazioni. Implementa funzionalità complesse e di alto livello. Esempi di Sistemi possono essere il Workload Management System o il Transformation System.

Per ottenere un'installazione, completa, di DIRAC è richiesta la cooperazione di differenti Sistemi. Un set di Sistemi DIRAC in grado di fornire, all'utente finale, tutte le funzionalità complete è detto Setup. Tutte le installazioni di Clients DIRAC devono fare riferimento ad un preciso Setup. Un Setup può comprendere più installazioni Server. Ogni installazione Server appartenente ad una istanza di DIRAC, che può essere condivisa da più Setup. I Setup rappresentano il massimo livello nella gerarchia dei componenti di DIRAC. Essi combinano assieme più istanze di Sistemi. Una certa comunità di utenti potrà avere più Setup di DIRAC a propria disposizione. Ad esempio si potrebbero avere Setup separati per: Produzione, Certificazione e Test. Una istanza di un certo Sistema può appartenere ad uno o più Setup. Conseguentemente, più setup possono trovarsi a condividere la stessa istanza di un Sistema. I setup multipli, di una stessa comunità di utenti, condividono le stesse informazioni di Configurazione, che gli permettono di accedere alle stesse risorse computazionali. Ogni istanza di Sistema o Setup deve avere un

nome univoco. La mappatura dei Sistemi, nei Setup, è descritta nella sezione `/DIRAC/Setups` della Configurazione dell'installazione DIRAC.

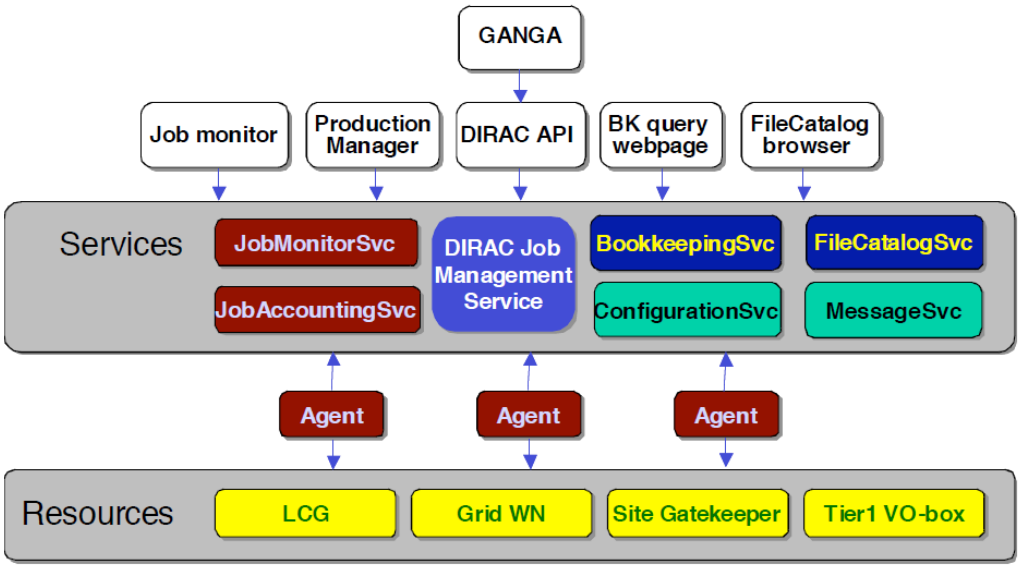

Figura 2.3: Layout schematico di DIRAC e delle sue relazioni con i componenti esterni della Grid.

2.4 Workload Management System

Lo scheduling effettuato dal Workload Management System implementa il paradigma dei Pilot Jobs Generici. Questa tecnica di scheduling risolve diversi problemi d'uso legati alla tipica instabilità delle risorse messe a disposizione delle Computer-Grids. In particolare aiuta la gestione delle attività degli utenti all'interno di grandi Virtual Organizations, come possono essere quelle legate agli esperimenti su LHC.

Integrare, nello stesso sistema, la gestione delle attività di Produzione e di Analisi consente di ottimizzare il workload globale di una Virtual Organization. Questo viene realizzato attraverso un'applicazione più efficiente delle politiche definite dalla VO, nel rispetto delle priorità assegnate ai tasks di differenti utenti e gruppi.

Le attività di Produzione e Analisi consistono nell'esecuzione di un certo numero di applicazioni sull'infrastruttura di computing distribuito. Queste applicazioni vengono formalmente definite "WorkLoad".

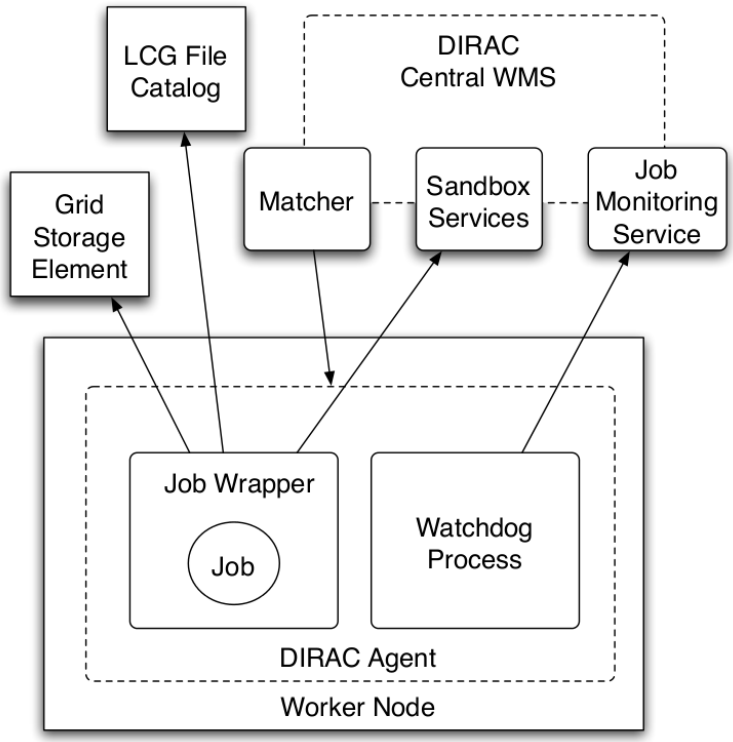

Figura 2.4: Gestione del workload, da parte di DIRAC, sul Worker Node.

Il normale ciclo di vita di un workload per il WMS di DIRAC consiste di diversi steps.

Quando un Job viene sottomesso al WMS di DIRAC il *Job Reciver* gli assegna un *Job ID* e lo inserisce, assieme al Proxy dell'utente a cui appartiene, nel *Job Database*. Durante il processo di sottomissione il servizio di *SandBox* provvede all'upload di qualsiasi file di Input necessario all'applicazione. Il *Data Optimizer*, interpellato dal *Job Reciver*, interroga l'*LFC File Catalog* in merito ai files di input al fine di individuare il migliore Storage Element (SE). Il *Data Optimizer* provvede, inoltre, ad inserire il Job in una *Task Queue*. A questo punto l'*Agent Director* invia, attraverso LCG, un *Pilot Agent* avente i medesimi requirements del Job Utente. Spetta all'*Agent Monitor* controllare lo stato del *Pilot Job* e, quando necessario, richiederne la risottomissione. Quando un *Pilot Job* viene messo in esecuzione, su un Worker Node (WN), provvede all'installazione DIRAC ed esegue un Agent che, a sua volta, richiede un particolare Job di un certo Utente. Il servizio *Matcher* ha il compito di confrontare i requisiti del Job con le caratteristiche del WN fornite dall'Agent. Una volta che un job viene inviato al WN tutti i

software necessari, non ancora presenti, vengono installati localmente. I link ai software già installati sono creati, localmente al job, durante l'installazione DIRAC. L'Agent crea, dinamicamente, il *Job Wrapper* usando le informazioni relative al WN. Il *Job Wrapper* scarica la *Input SandBox* e provvede a fornire l'accesso ai file di Input. L'applicazione richiesta dal job viene avviata in un processo figlio, un processo *WatchDog* è avviato in parallelo per fornire l'heart-beat al servizio di *Job Monitoring*. Quest'ultimo monitorizza il Job attraverso informazioni come l'utilizzo di memoria e CPU. Il *Job Wrapper* provvede a notificare i cambiamenti nello stato del job al servizio di *Job Monitoring*. Al termine del job il *Wrapper* gestisce l'upload dell'Output Sandbox, attraverso il servizio *SandBox*, e provvede ad inserire l'output nel *Job Database*. Una volta che il *DIRAC Agent* ha terminato la sua esecuzione il *Pilot Agent*, prima di terminare anch'esso, provvede a liberare le risorse. Durante ogni fase del *WorkFlow*, di un Job, il servizio di *Job Monitoring* è usato per aggiornare le relative informazioni nel database.

2.4.1 Pilot Jobs

I Pilot sono comuni jobs, della Grid, che vengono sottomessi, dal WMS di DIRAC, con i normali strumenti messi a disposizione dal MiddleWare sottostante. Nel caso specifico di LHCb viene usata l'infrastruttura EGEE e quindi i job sono sottomessi sfruttando il WMS gLite. Lo scopo dei Pilot Jobs e quello di riservare uno Slot temporale, su un Worker Node della Grid, e recuperare un reale workload, fra quelli in coda, per la sua esecuzione.

L'idea fondamentale alla base dei Pilot Jobs è quella di, quando questo è possibile, sfruttare per il massimo del tempo, una volta acquisite, le risorse dei Computing Element (CE). I *DIRAC Agent*, infatti, richiedo nuovi job utente fintanto che ne esistono e la risorsa computazionale è disponibile. Questo permette, per i job utente in coda, di sfruttare il tempo macchina lasciato libero da un'altro job con le stesse caratteristiche. Un tale comportamento riduce drasticamente lo start-time medio dei brevi Job di Analisi lanciati dagli utenti.

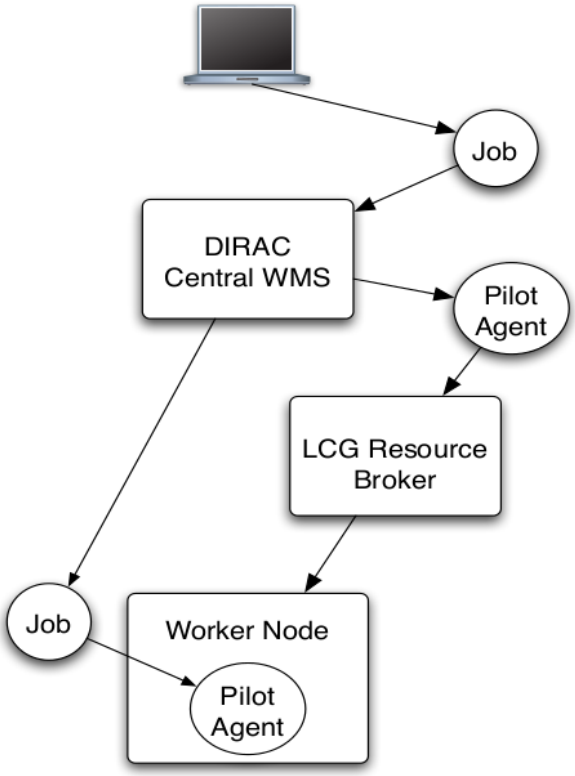

Figura 2.5: Rappresentazione schematica del paradigma dei Pilot Agent.

Agent Director e *Agent Monitor* possono essere configurati per definire come i *Pilot Agent* devono essere sottomessi. La scelta del *DIRAC Agent* da utilizzare determina la maniera in cui i job verranno recuperati dal WN. Sono previste tre modalità di sottomissione da utilizzare in base alle caratteristiche del job:

- **Resubmission**: una normale sottomissione LCG con monitoraggio degli eventuali fallimenti, quando necessario possono essere sottomessi più *Pilot Agent*.
- **Filling**: più *Pilot Agent* possono essere sottomessi, ognuno dei quali richiederà più Job dello stesso Utente, uno dopo l'altro in maniera sequenziale.
- **Multi-Threaded Filling**: segue la stessa logica adottata dal modo Filling ma, in questo caso, vengono parallelizzati due Job, per volta, sul WN.

Questa tecnica, basata sulla massimizzazione dell'uso delle risorse, rende possibile un livello di ottimizzazione non raggiungibile con i soli strumenti LCG.

2.5 Database di Back-End

Come già accennato precedentemente il database di Back-End viene usato, dal sistema, come una sorta di memoria condivisa tra la sua parte attiva, gli Agent, e quella passiva, i Service. Entrambe le parti possono inserire e leggere i dati dal DBMS. Questa struttura permette di poter meglio configurare il sistema a seconda delle necessità della comuntà di utenti.

Il database di Back-End è diviso in più schemi. Ogni Schema fa capo ad un particolare sottosistema di DIRAC, che svolge ben definite funzioni. Per ogni Schema ci saranno, quindi, ben determinati tipi di Agent e Service che ne usufruiranno. Questa divisione presenta due vantaggi:

- La suddivisone in più Schemi rappresenta una particolare forma di partizionamento verticale dell'intera base di dati. Questa caratteristica permette di distribuire il database, e quindi anche occupazione di spazio disco e carico di lavoro, su diverse macchine.

- Una eventuale riprogettazione dello Schema appartenente ad un determinato sottosistema non richiederebbe modifiche al codice, di Agent e Service, appartenente agli altri.

Esistono tre Setup di LHCbDIRAC attualmente in funzione presso il CERN, ognuna con un diverso deployment per il DBMS:

- **Produzione**

- **Certificazione**

- **Sviluppo**

L'attuale installazione di LHCbDIRAC, in produzione al CERN, utilizza un database i cui schemi sono distribuiti su tre host diversi. Ognuno dei tre host mantiene, invece, in esecuzione almeno un'istanza di Service e Agent per ogni sottosistema. Possono quindi esistere più Agent e Service che puntano alla stessa istanza di Schema. I componenti attivi e passivi, dei sottosistemi di DIRAC, sono progettati per bilanciare e sincronizzare autonomamente le proprie istanze multiple.

Dal punto di vista infrastrutturale l'installazione di produzione vede, su tutti i database server, l'impiego di un DBMS MySQL versione 5.1 in configurazione Stand-Alone.

Gli schemi previsti da DIRAC sono:

- AccountingDB
- ConfigurationDB
- DataManagementDB
- FrameworkDB
- RequestManagementDB
- ResourceStatusDB
- StorageManagementDB
- TransformationDB
- WorkloadManagementDB

Quasi la totalità degli Schemi prevede la compresenza di tabelle MyISAM, che rappresentano la grande maggioranza, e InnoDB.

Capitolo 3

Profiling e TransformationDB

Fra le richieste, formulate dal gruppo di sviluppo del progetto DIRAC, vi è quella di provvedere ad una conversione di tutte le tabelle, in tutti gli schemi, del database di Back-End all'engine InnoDB. Essendo il database diviso in più schemi, che rispecchiano la suddivisione del sistema in sotto-sistemi, è stato possibile cominciare lo studio, e i test, concentrandosi su una partizione ben definita del database di Back-End, senza toccare le altre. Questa netta divisione ha anche il vantaggio di permettere un aggiornamento parziale, e graduale, del sistema.

Lo schema, e di conseguenza sotto-sistema, prescelto come primo soggetto di studio è stato il *TransformationDB*. Anche se uno degli obbiettivi è quello di ridurre al minimo il numero di modifiche necessarie al codice, delle applicazioni che sfruttano il database, è stato da subito chiaro che questo avrebbe comportato una forte limitazione nella normalizzazione dello schema. Essendo una pratica necessaria, anche se non sufficiente, per ottenere, fra i molti vantaggi, un buon livello di prestazioni da parte del DBMS è necessario trovare un giusto compromesso tra compatibilità con l'attuale codice e Normalizzazione.delle Relazioni.

3.1 Conversione dello Schema

Lo schema attualmente in uso da LHCbDIRAC utilizza per quasi la totalità delle tabelle, in esso contenute, l'engine MyISAM. Questo engine per la memorizzazione dei dati su disco, che tuttavia conserva alcuni punti di forza a suo favore, non supporta transazioni e chiavi esterne. Solamente la tabella Transformations fa uso dell'engine InnoDB. Da subito la conversione delle tabella all'engine InnoDB ha presentato delle difficoltà.

3.1.1 La tabella TransformationTasks

Nel convertire la tabella TransformationTasks ci si è imbattuti in un problema di compatibilità tra i due engine del DBMS.

```
mysql> show create table TransformationTasks\G
*************************** 1. row ***************************
       Table: TransformationTasks
Create Table: CREATE TABLE `TransformationTasks` (
  `TaskID` int(11) NOT NULL AUTO_INCREMENT,
  `TransformationID` int(11) NOT NULL,
  `ExternalStatus` char(16) DEFAULT 'Created',
  `ExternalID` char(16) DEFAULT '',
  `TargetSE` char(255) DEFAULT 'Unknown',
  `CreationTime` datetime NOT NULL,
  `LastUpdateTime` datetime NOT NULL,
  `RunNumber` int(11) DEFAULT '0',
  PRIMARY KEY (`TransformationID`,`TaskID`),
  KEY `ExternalStatus` (`ExternalStatus`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Come si può notare, dall'output dello statement riportato sopra, la tabella prevede un campo `TaskID`, con proprietà `AUTO_INCREMENT`, e un campo `TransformationID` che, assieme, vadano a formare la chiave primaria, composta, della tabella. Questo tipo di definizione per la chiave primaria è pienamente supportato da MyISAM. L'engine InnoDB, invece, non consente di definire un campo `AUTO_INCREMENT` senza che questi funga, senza nessuna concatenazione con altri campi, da chiave primaria per la relazione. Questo comportamento è anche giustificato dal fatto che, secondo le regole della normalizzazione, un campo di tipo `AUTO_INCREMENT`, appositamente pensato per fornire un indice ordinato, è il candidato ideale per la chiave primaria in quanto: identifica univocamente una tupla ed è minimo.

Si è deciso, dopo aver consultato i responsabili dello sviluppo di DIRAC, di non modificare la chiave primaria della tabella. Grazie ad un trigger che computa, ad ogni inserimento, quale sia il nuovo TaskID, da assegnare alla tupla, è stato anche possibile evitare modifiche alla logica dell'applicazione che, altrimenti, avrebbe dovuto provvedere da sola al calcolo del seriale. In questo caso, dunque, si è preferito privilegiare la trasparenza, della modifica verso l'applicazione, rispetto ad una piena normalizzazione-ottimizzazione dello schema. Una soluzione del genere è molto simile a come vengono costruiti, e trattati, i campi `AUTO_INCREMENT` dal motore MyISAM. Ovviamente una simile scelta non introduce nessun vantaggio fra quelli derivati dall'adozione di una vera chiave `AUTO_INCREMENT` (`SERIAL`).

```
mysql> show create trigger count_TransformationTasks_TaskID \G
*************************** 1. row ***************************
               Trigger: count_TransformationTasks_TaskID
              sql_mode: NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`root`@`localhost`
TRIGGER count_TransformationTasks_TaskID
BEFORE INSERT ON TransformationTasks
FOR EACH ROW
SET NEW.TaskID=(SELECT @last := IFNULL(MAX(TaskID) + 1,1)
FROM TransformationTasks
WHERE TransformationID=NEW.TransformationID)
  character_set_client: utf8
  collation_connection: utf8_general_ci
    Database Collation: latin1_swedish_ci
1 row in set (0.00 sec)
```

È da notare che, con questa implementazione, la cancellazione della tupla con il maggior valore di TaskID comporta, contrariamente a quanto succede con la proprietà `AUTO_INCREMENT`, il riutilizzo di quest'ultimo da parte del successivo inserimento. Questo comportamento è perfettamente compatibile con quanto accadeva, precedentemente, con la tabella gestita da MyISAM.

3.1.2 Definizione delle Chiavi Esterne

Uno degli indubbi vantaggi nell'adozione dell'engine InnoDB è la possibilità di definire delle Chiavi Esterne, le quali, a loro volta, forniscono garanzie sulla consistenza dei dati. Questo sposta una buona parte della complessità dall'applicazione al DBMS.

L'adozione di, corrette, politiche sulle chiavi esterne consente di automatizzare, ulteriormente, e rendere atomiche operazioni che, altrimenti, l'applicazione deve eseguire passo-passo, controllandone gli esiti. Basti pensare, anche in uno schema con sole tredici relazioni come il TransformationDB, quanti controlli, e conseguenti azioni, possa comportare la cancellazione, o l'inserimento, di un record in una tabella con un'engine che non garantisca integrità relazionale nei dati.

Ovviamente, dal momento che si può riporre maggior fiducia nel corretto comportamento della base di dati, l'applicazione può, e deve, essere purgata da tutto quel codice di controllo, prima necessario, che ora produce solo inutile everhead nel sistema.

Per le azioni di inserimento, la mera definizione delle Chiavi Esterne comporta già il massimo grado di sgravio per l'applicazione. D'altra parte, l'uso di chiavi esterne senza la definizione, su di esse, di corrette politiche complica, per l'applicazione, le operazioni di aggiornamento e cancellazione delle tuple. Se prima, in un contesto dove la base di dati non gestiva i vincoli di

referenziazione fra le relazioni, l'applicazione risultava libera di eseguire le operazioni di `UPDATE` e `DELETE`, come pure le `INSERT`, sulle varie tabelle, nell'ordine che preferiva, ora, con l'adozione dei vincoli relazionali, le operazioni debbono essere svolte nell'ordine preciso dettato dai vincoli.

La sola adozione delle chiavi esterne, tanto più se senza la definizione di adeguate politiche, non comporta nessun beneficio al sistema, anzi rischia solo di aumentarne la complessità totale, quando l'applicazione non si affida ai nuovi automatismi forniti dalla modifica alla base di dati.

Il medesimo ragionamento fatto per le chiavi esterne, e le politiche a loro applicate, vale per le Transazioni. Le complesse, e quindi potenzialmente fonte di bug o comportamenti instabili, procedure, applicate in caso di errore, per riportare in uno stato consistente la base di dati ora possono essere sostituite da un semplice statement `ROLLBACK`.

Un esempio pratico

Durante lo studio dello Schema TransformationDB ci si è imbattuti nel caso della tabella TransformationFileTasks. Le uniche operazioni che avevano come soggetto questa tabella erano `INSERT`. Nel codice dell'applicazione non appariva nessun statement per la cancellazione delle tuple dalla tabella.

Un controllo nel database di produzione ha rivelato che effettivamente la tabella, che aveva raggiunto il ragguardevole traguardo delle 67.141.809 righe, conteneva tuple ormai inutili.

La definizione di Foreign Key e relative politiche in cancellazione avrebbero evitato una simile situazione. Questo dimostra quale sia il vantaggio, anche in termini di velocità di sviluppo e test, dell'affidarsi ad un corretto uso delle funzioni messe a disposizione da un RDBMS, rispetto al voler gestire la consistenza dei dati lato applicazione.

3.2 Replicazione del Carico

Ottenere un'installazione completa [1] di DIRAC in locale e, soprattutto, produrre per essa un carico di lavoro sufficiente a simulare, anche in un eventuale rapporto di scala, quello che avviene in produzione è abbastanza problematico. Si è cercato allora la maniera di riprodurre, il più efficacemente

[1] Anche quando si vuole studiare un solo sotto-sistema di DIRAC, dato che questo interagisce con gli altri, è necessario avere un sistema, il quanto più possibile, completo per poter pensare di riprodurre, efficacemente, il comportamento di un'installazione reale in produzione

possibile, il carico, verso il DBMS, generato da installazioni già in funzione presso il CERN.

Per ottenere una buona riproduzione del carico di un database, come per un qualsiasi altro sistema, che possa produrre dei risultati significativi, si necessita di:

- un metodo per ricreare esattamente la condizione iniziale del sistema, nel caso specifico un normalissimo Dump dello Schema.
- una soluzione per riprodurre le stessa serie di eventi occorsi nel sistema che si vuole studiare, nel caso specifico questo è stato ottenuto con Log e Tool dedicati.

3.2.1 Slow e General Log di MySQL

In principio è stata valutata l'idea di utilizzare, per la replicazione del carico, i log generati direttamente da DIRAC. Anche se questo Log, ottenuto attraverso il componente chiamato `gLogger`, contiene le Query fatte dal sistema presenta alcuni problemi:

- il log presenta una gran quantità di informazioni che non riguardano le Query o comunque, in qualche maniera, il DBMS.
- il formato del log non è standard o conosciuto e, di fatto, non esiste nessun tool, necessario vista la mole di dati, per la sua analisi.
- si è riscontrata nei log, probabilmente a causa di una errata gestione dei buffer, la presenza di righe inconsistenti per troncamenti o porzioni di diverse entries concatenate in una sola riga.

MySQL fornisce, nativamente, due tipi distinti di log attivabili, e configurabili, a Run-Time: Slow e General log [2] . La configurazione dei log può essere fatta attraverso delle variabili, per conoscerne il valore si possono usare gli statements:

```
mysql> show variables like '%log%';
+---------------------------+-------------------------------------+
| Variable_name             | Value                               |
+---------------------------+-------------------------------------+
...
| general_log               | OFF                                 |
| general_log_file          | /var/lib/mysql/stellascura.log      |
```

[2]Dalla versione 5.1.6 i log possono anche essere fatti su una tabella interna del database, per permetterne la consultazione attraverso statement SQL.

```
...
| log_slow_queries          | OFF                                |
...
| slow_query_log            | OFF                                |
| slow_query_log_file       | /var/lib/mysql/stellascura-slow.log |
...
+---------------------------+------------------------------------+
41 rows in set (0.00 sec)

mysql> show global variables like 'long_query%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

I valori sono modificabili con statements del tipo:

```
set global <variable_name> = <value>;
```

La variabile globale `long_query_time` è settata sia a livello globale che di sessione utente, negli statement che si riferiscono alla variabile globale è sempre necessario usare lo specificatore `global`. La modifica del valore di una di queste variabili ha effetto immediato ma per le variabili `*_query_log_file` può essere necessario uno statement `flush logs;`. Il valore della variabile `long_query_time` è da intendersi espresso in secondi.

Il General Log, una volta attivato, raccoglie tutte le Query pervenute al DBMS. Il log prodotto ha un formato simile al seguente: [3].

```
/usr/sbin/mysqld, Version: 5.5.31-0ubuntu0.12.04.2 ((Ubuntu)). started with:
Tcp port: 3306  Unix socket: /var/run/mysqld/mysqld.sock
Time                 Id Command    Argument
130701 17:09:04    39 Connect   root@localhost on TransformationDB
...
                   39 Query     select @@version_comment limit 1
130701 17:09:08    39 Query     select * from Transformations
130701 17:09:10    39 Quit
...
```

Lo Slow Log è più configurabile, esso infatti raccoglie solamente le Query il cui tempo di esecuzione supera il valore di `long_query_time`. Il prodotto è diverso da quello del General:

```
/usr/sbin/mysqld, Version: 5.5.31-0ubuntu0.12.04.2 ((Ubuntu)). started with:
Tcp port: 3306  Unix socket: /var/run/mysqld/mysqld.sock
```

[3] Può variare leggermente con la versione di MySQL

```
Time                 Id Command    Argument
...
# Time: 130701 16:53:12
# User@Host: root[root] @ localhost []
# Query_time: 0.237943  Lock_time: 0.094079 Rows_sent: 0  Rows_examined: 0
SET timestamp=1372690392;
select * from Transformations;
...
```

Analisi dei Log di Produzione

Purtroppo non è stato possibile, sia per una questione di dimensioni che per evitare di influire troppo sulle prestazioni del sistema, ottenere un Dump del database di produzione. Una replicazione esatta del carico in questione è stata quindi impossibile. Tuttavia, visto che lo si può fare senza dover interrompere il sevizio e con un overhead minimo, è stato possibile ottenere dei Log che, comunque, sono tornati utili per ricavare diverse informazioni sul tipo di carico sostenuto in Produzione.

Per eseguire l'analisi dello Slow Log si è utilizzato il tool `pt-query-digest` della suite Percona ToolKit. Questo tool pratica una sofisticata, e configurabile anche per mezzo di scripting Perl, analisi dei log di MySQL. L'analisi parte da eventi provenienti dallo Slow Log, il caso da noi preso in considerazione, o dalla `PROCESSLIST`. È possibile eseguire analisi arbitrariamente complesse ordinando e raggruppando secondo vari parametri le Query. Caratteristica distintiva del tool è la capacità di determinare il fingerprint delle query e, quindi, raggrupparle secondo questo criterio. Quest'ultima caratteristica alleggerisce di molto il compito di chi deve analizzare i log.

Grazie a questo tool è possibile determinare quali siano le Query che più impegnano il database e che, quindi, andrebbero ottimizzate o, se possibile, diminuite in frequenza di esecuzione. Il prolisso output del programma è diviso in due parti:

- un riepilogo in cui viene riportata la lista, ordinata, delle Query che hanno ottenuto più match per il criterio specificato,
- la singola analisi di ogni Query, presente nella lista, con tutti gli attributi richiesti.

A titolo esemplificativo viene riportata l'analisi di un Log di produzione. Nel Log sono presenti tutte le Query eseguite dal DBMS, quindi anche quelle che interessano Schemi appartenenti a sottosistemi, di DIRAC, diversi dall'oggetto di analisi. È possibile costruire filtri su vari parametri associati agli eventi. Il listato seguente propone il `dump` di una struttura associata, da `pt-query-digest`, ad un evento:

```
$VAR1 = {
  Query_time => '0.033384',
  Rows_examined => '0',
  Rows_sent => '0',
  Thread_id => '10',
  Tmp_table => 'No',
  Tmp_table_on_disk => 'No',
  arg => 'SELECT col FROM tbl WHERE id=5',
  bytes => 103,
  cmd => 'Query',
  db => 'db1',
  fingerprint => 'select col from tbl where id=?',
  host => '',
  pos_in_log => 1334,
  ts => '071218 11:48:27',
  user => '[SQL_SLAVE]'
};
```

Il tool, se non diversamente specificato, limita il report a 20 query o, dove non si dovesse arrivare a tale quota, al 95% dei fingerprints totali. L'output viene presentato allo `stdout` e un indicatore di avanzamento, oltre ai normali errori, allo `stderr`.

```
pt-query-digest --filter '$event->{db} eq "ProductionDB"' \
--limit=100% 22_slow.log > analisi
```

Di seguito viene riportata una parte del report:

```
# 656.8s user time, 8.7s system time, 35.94M rss, 226.77M vsz
# Current date: Tue Jul  2 21:42:11 2013
# Hostname: d3.fe.infn.it
# Files: 22_slow.log
# Overall: 2.26M total, 78 unique, 181.83 QPS, 0.96x concurrency _________
# Time range: 2013-04-16 18:31:40 to 21:59:14
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time         11924s     1us    114s     5ms   273us   411ms    18us
# Lock time          6336s       0     96s     3ms    44us   368ms       0
# Rows sent          4.46M       0   9.77k    2.06    0.99   89.70       0
# Rows examine     730.34M       0  10.82M  338.18    0.99  22.27k       0
# Query size       142.43M       6  79.15k   65.95  212.52  409.31   26.08

# Profile
# Rank Query ID           Response time   Calls  R/Call  V/M   Item
# ==== ================== =============== ====== ======= ===== ===========
#    1 0xB6181EDD2FB888A8 6400.6257 53.7%  11330  0.5649 27.07
     SELECT TransformationFiles
#    2 0x874051C92A574963  878.2083  7.4%  30592  0.0287 40.62
     UPDATE TransformationFiles
```

```
#    3 0xAFA91D2D2DCA4EA4  804.9874  6.8%      9 89.4430  8.12
     SELECT TransformationFiles
...
# Query 1: 0.91 QPS, 0.52x concurrency, ID 0xB6181EDD2FB888A8 at byte 313313910
# This item is included in the report because it matches --limit.
# Scores: V/M = 27.07
# Time range: 2013-04-16 18:31:44 to 21:58:30
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count          0   11330
# Exec time     53   6401s   248us     62s   565ms      1s      4s    71ms
# Lock time     57   3631s    29us     62s   321ms   167us      4s    54us
# Rows sent      8 389.69k       0     847   35.22  329.68  102.63       0
# Rows examine  11  81.01M       0  55.29k   7.32k  38.40k  12.88k  874.75
# Query size     1   2.20M     203     204  203.98  202.40       0  202.40
# String:
# Databases    ProductionDB
# Hosts        volhcb24.c... (3848/33%)... 2 more
# Users        Dirac
# Query_time distribution
#   1us
#  10us
# 100us  #####
#   1ms  ##################################
#  10ms  ####################################################
# 100ms  ################################################################
#    1s  ###########
#  10s+  #
# Tables
#    SHOW TABLE STATUS FROM ‘ProductionDB‘ LIKE ’TransformationFiles’\G
#    SHOW CREATE TABLE ‘ProductionDB‘.‘TransformationFiles‘\G
# EXPLAIN /*!50100 PARTITIONS*/
SELECT TransformationID,FileID,Status,TaskID,TargetSE,UsedSE,ErrorCount,
LastUpdate,InsertedTime,RunNumber FROM TransformationFiles
WHERE ‘Status‘ = "Unused" AND ‘TransformationID‘ = "19732" LIMIT 10000\G
...
```

Utilizzando i tool `grep` e `awk` è stata realizzata una veloce analisi del General Log di Produzione. In particolare si è sfruttato l'identificativo di connessione che, questo tipo di log, inserisce in ogni riga contenete una query. È quindi semplice, contando il numero di righe che riportano lo stesso identificativo, avere la stima di come la connessione sia sfruttata.

È risultato che, in un periodo di circa 2 ore e 45 minuti descritto dal log raccolto:

- si sono effettuate, verso il DBMS, circa 329511 connessioni.
- ogni connessione ha servito una media di 4,88 Query.

È evidente che il sistema di gestione, in pool, delle connessioni del progetto DIRAC è affetto da un malfunzionamento.

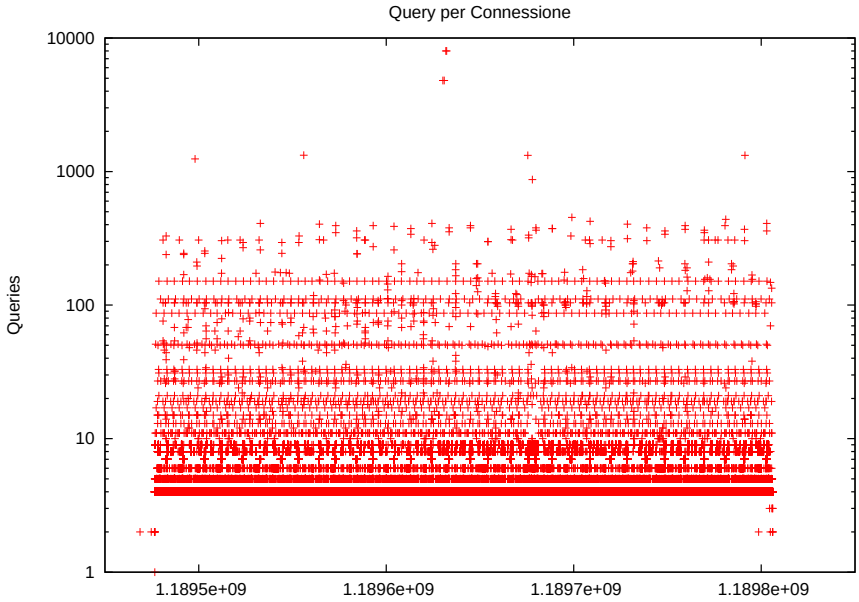

Figura 3.3: Numero di Query effettuate per connessione al DBMS, di produzione, nell'arco di 2:45 ore.

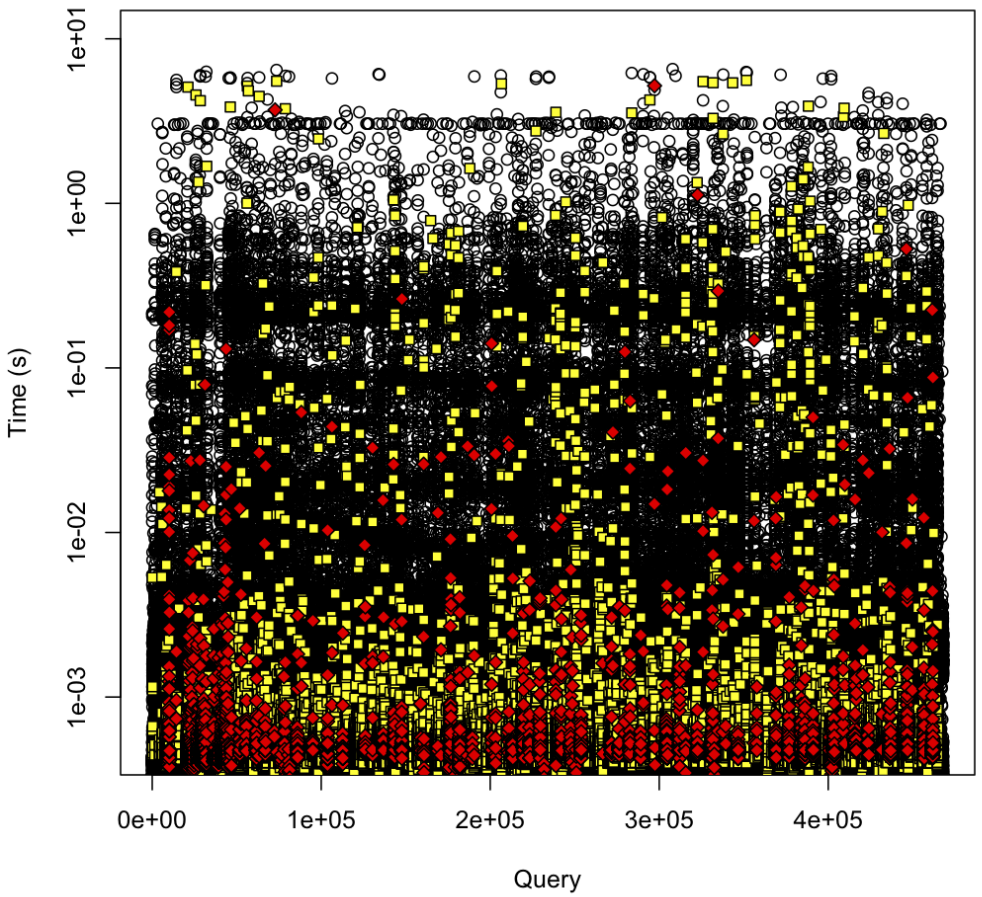

Figura 3.4: Tipo di Queries e relativo tempo di esecuzione nei Log di Produzione catturati. SELECT, UPDATE, INSERT.

3.2.2 Percona Playback

Per la replicazione del carico si sono usati Dump e Log provenienti dai sistemi di *Sviluppo* e *Validazione*. La scelta dello strumento da utilizzare è ricaduta su percona-playback, un tool OpenSource appositamente pensato per questo scopo. La particolarità di percona-playback, rispetto ad altri tool come mysqlslap [4], sta nella capacità di accettare come input direttamente i

[4]mysqlslap è un tool per l'emulazione del carico fornito dallo stesso progetto MySQL

log prodotti dal DBMS. Anche se è in grado di generare carico da quasi tutti i tipi di log, forniti da MySQL e Fork, l'input migliore, e di default, resta lo Slow Log. Per ottenere una completa riproduzione del carico lo Slow Log deve essere catturato impostando la variabile `long_query_time` a 0 (zero) secondi, in maniera da non tralasciare nessuna Query.

Percona Playback è stato pensato con una struttura che supporta i plug-in, questa caratteristica lo rende configurabile e, se necessario, modificabile a piacimento. Nella categoria `db`, dei plug-in, sono presenti le librerie client che è possibile utilizzare per connettersi al DBMS. Per il momento sono selezionabili due librerie:

- `libmysqlclient`: bloccante
- `libdrizzle`: non bloccante

Nel corso dei test sono state utilizzate entrambe. Si è potuto notare come l'unico parametro, effettivamente, affetto da variazioni significative sia il wall-clock, nettamente più basso per `libdrizzle`. Si è quindi deciso di svolgere la maggior parte dei test utilizzando `libdrizzle`, al fine di ottimizzare i tempi, e, alla bisogna, usare `libmysqlclient` per confermare i risultati o fugare dubbi di sorta.

Altra caratteristica, che rende l'uso del tool facilmente adattabile ai vari contesti e alle relative esigenze, è l'adozione dei flussi standard per input e output. L'output, di percona-playback, è diviso sui due flussi forniti dal Sistema Operativo, che vengono utilizzati per riportare:

- **stdout** risultati delle misure andate a buon fine.
- **stderr** motivazioni per cui la riproduzione di uno statement non è stata considerata significativa oltre che, in maniera convenzionale, eventuali errori del processo utente.

L'uso dei due flussi facilita la consultazione dell'output, che raggiunge velocemente dimensioni notevoli, oltre a permettere, ovviamente, di conservare solo la parte che interessa. Anche per l'input può essere, al posto di un regular-file sul filesystem, utilizzato lo stdin. Questa caratteristica consente l'integrazione con altri applicativi attraverso ridirezioni e pipe.

L'output presenta un report per ogni Query replicata con successo.

```
...
thread 0 slower query was run in  63 microseconds instead of 2
  <--
# User@Host: Dirac[Dirac] @ localhost []
# Query_time: 0.000002  Lock_time: 0.000000 Rows_sent: 2  Rows_examined: 46
  -->
...
```

La causa più frequente di insuccesso, come è possibile osservare dallo stdout, è il differente numero di tuple tornate dallo statement preso in considerazione. Questa caratteristica rende le misure effettuate molto affidabili, in quanto vengono riportate solo se le condizioni originarie sono perfettamente riproducibili, ma riduce fortemente il numero di campioni. Il problema si accentua ancora di più in una situazione, come quella presa in considerazione, dove buona parte dei Log contiene Query riguardanti Schemi diversi da quello di interesse. Al termine del test viene, sempre allo stdout, presentato un riepilogo generale.

```
...
Detailed Report
----------------
SELECTs  : 981091 queries (575659 faster, 405432 slower)
INSERTs  : 176059 queries (148220 faster, 27839 slower)
UPDATEs  : 265528 queries (238439 faster, 27089 slower)
DELETEs  : 9436 queries (2728 faster, 6708 slower)
REPLACEs : 1408 queries (1224 faster, 184 slower)
DROPs    : 0 queries (0 faster, 0 slower)


Report
------
Executed 8307488 queries
Spent 00:09:44.667763 executing queries versus an expected 00:15:25.805826 time.
993700 queries were quicker than expected, 7232342 were slower
A total of 1939677 queries had errors.
Expected 2411177 rows, got 19580 (a difference of 2391597)
Number of queries where number of rows differed: 964533.

Average of 8307488.00 queries per connection (1 connections).
```

Rovescio della medaglia del'alta flessibilità, e configurabilità, di percona-playback è la necessità di specificare molte opzioni alla riga di comando:

```
percona-playback --query-log-file ./input_test/slow.log --mysql-host 127.0.0.1 \
    --mysql-username <user> --mysql-password <passwd> \
    --mysql-schema TransformationDB \
    --dispatcher-plugin thread-pool --thread-pool-threads-count 50 \
    2> ./output_test/stderr > ./output_test/stdout &
```

Risultati e Considerazioni

I test sono stati effettuati mettendo a confronto diverse versioni di MySQL con entrambi gli schemi del TransformationDB. Le versioni, di MySQL, prese in considerazione sono state:

- **5.1.69** versione, fornita dai repository ufficiali di CentOS, della stessa serie presente, al momento, in produzione.
- **5.6.10-12** le più recenti versioni GA disponibili nel periodo in cui sono stati effettuati i test.
- **5.5.32** versione GA più simile al DBonDemand, fornito dal servizio IT del CERN, attualmente in uso nell'installazione di Certificazione. Questa versione è anche quella su cui si basa il codice del Percona XtraDB Cluster attualmente disponibile.

Il primo risultato degno di nota è la, sostanziale, parità di risultati, nei test effettuati con il vecchio schema prevalentemente MyISAM, tra le varie versioni. Questo risultato è giustificato dal fatto che MySQL ha abbandonato, dalla versione 5.5 compresa, MyISAM, come engine di default, in favore di InnoDB.

Le differenze, fra le varie versioni del DBMS, rilevate con i test effettuati sul nuovo schema, convertito al motore InnoDB, sono state minime. I motivi di questi risultati, in apparenza contrastanti con quanto dichiarato da Oracle rispetto alle prestazioni delle nuove versioni, sono da ricercare nella direzione presa dallo sviluppo di MySQL. Le versioni più recenti sono pensate, con particolare attenzione, per scalare al meglio sulle nuove architetture. In particolare le nuove release di InnoDB sono in grado di sfruttare meglio, ottenendone significativi benefici in termini di prestazioni, i moderni hardware che consentono un forte parallelismo in esecuzione. Negli stessi grafici forniti da Oracle è possibile notare come non vengano presi in considerazione i sistemi dotati di un numero di core inferiore a sei. Purtroppo le macchine messe a nostra disposizione non ricadono nella categoria considerata da Oracle.

Nella recente versione, GA, di MySQL l'engine InnoDB è stato implementato per migliorare il supporto alla concorrenza sfruttando, al posto dei *mutex* come precedentemente avveniva, lo scheduler del Sistema Operativo. Nei test condotti è stato possibile verificare che, anche in presenza di architetture hardware datate, l'aggiornamento del Kernel Linux, dalla versione 2.6.32, fornita con CentOS-6, alla 3.0.78, Main-Line fornita del repository ELRepo, ha comportato un sensibile miglioramento nelle prestazioni del DBMS.

3.3 Proposte e considerazioni

Il miglioramento di un sistema complesso, o anche solo di una parte di esso, come può essere DIRAC, e il suo database di Back-End, comporta

l'operazione di scelte ben ponderate. Il primo passo, dopo aver studiato il sistema e averne individuato i punti passibili di miglioramento, è quello di produrre una lista di possibili vie da percorrere. Ogni modifica o variazione non va considerata isolatamente ma, bensì, valutata in relazione alle altre componenti del sistema. Bisogna, altresì, tenere conto che nulla può essere ottenuto a costo zero; la soluzione di un problema implica, nella maggior parte dei casi se non sempre, la creazione di altre problematiche collaterali che, ovviamente, devono essere di entità e/o rilevanza minore rispetto al problema originale.

3.3.1 Tabelle statistiche per lo schema TransformationDB

L'adozione dell'engine InnoDB per la memorizzazione di tutte le tabelle, presenti nel database di Back-End, è uno dei requirement posti dai responsabili del progetto DIRAC/LHCbDIRAC. I vantaggi del passare ad un sistema di memorizzazione dati ACID [5] compilant, quando adeguatamente sfruttato, sono ovvi. Tuttavia anche InnoDB presenta dei svantaggi rispetto ad altri engine; uno dei problemi noti è la sua lentezza dell'eseguire statement che contengano la funzione `COUNT(*)`, soprattutto se eseguiti su tabelle di grandi dimensioni. Questa istruzione viene, tuttavia, usata sovente per creare dei report riassuntivi sui dati, spesso proprio quando questi sono in quantità considerevole. Nel carico proposto al TransformationDB si individuano due di queste Query, i cui fingerprint sono:

```
SELECT ‘ExternalStatus‘, COUNT(*) FROM ‘TransformationTasks‘
WHERE ‘TransformationID‘ = "<ID>" GROUP BY ‘ExternalStatus‘
ORDER BY ‘ExternalStatus‘;

SELECT ‘TransformationID‘, ‘Status‘, COUNT(*) FROM ‘TransformationFiles‘
WHERE ‘TransformationID‘ = "<ID>" GROUP BY ‘TransformationID‘, ‘Status‘
ORDER BY ‘TransformationID‘, ‘Status‘;
```

Entrambe compaiono nella Top-15 delle Query maggiormente effettuate in Produzione, rispettivamente in 11°, occupando il DBMS per l'1,5% del tempo totale, e 13° posizione, con l'1,4% del tempo. Ci si attende che questi dati peggiorino con il passaggio a InnoDB delle tabelle.

Nell'ambito della realizzazione del sistema di produzione per il progetto SuperB si ha avuto modo di sperimentare, con successo, una soluzione ad un problema simile. Quello che viene proposto è la realizzazione di una relazione, popolata tramite trigger sulla tabella ora oggetto di interrogazione, che mantenga in essa i valori attualmente ottenuti attraverso le query oggetto

[5] ACID - Atomicità, Coerenza, Isolamento e Durabilità

di studio. Questo, oltre a spostare alcune Query di selezione su una tabella diversa diminuendo le contese, ridurrebbe lo statement ad una normale SELECT su una tabella scarsamente popolata.

Ovviamente il rovescio della medaglia è quello di appesantire, a causa dei trigger e degli aggiornamenti sulla nuova tabella, un poco le operazioni di inserimento, cancellazione e modifica sulle relazioni originali. In questo caso, quindi, è importante valutare, almeno l'ipotetica, fattibilità verificando il carico offerto da queste operazioni. Risulta da subito chiaro che per la tabella `TransformationFiles` una tale operazione introdurrebbe, anche nell'ottica della conversione InnoDB, più svantaggi che vantaggi; un'operazione di `UPDATE` su questa tabella rappresenta, infatti, il 7.4% del tempo totale di utilizzo del DBMS, piazzandosi al secondo posto fra tutte.

Tabella ExternalStatus

La tabella ExternalStatus vuole essere solo un esempio di implementazione, applicato al caso più significativo preso in considerazione, per la tecnica sopra descritta. Non vi è stato modo, a causa dell'eccessiva differenza del profilo tra i carichi di Certificazione/Sviluppo e quello di Produzione, di ottenere risultati certi sulla sua efficacia in termini di prestazioni. Creazione della tabella:

```
create table ExternalStatus(
    TransformationID int(11) primary key,
    nCreated tinyint unsigned default 0,
    nSubmitted tinyint unsigned default 0,
    nReserved tinyint unsigned default 0,
    nWaiting tinyint unsigned default 0,
    nRunning tinyint unsigned default 0,
    nDone tinyint unsigned default 0,
    nCompleted tinyint unsigned default 0,
    nFailed tinyint unsigned default 0,
    nKilled tinyint unsigned default 0,
    foreign key( TransformationID ) references
     TransformationTasks( TransformationID )
     on delete cascade
) engine = InnoDB;
```

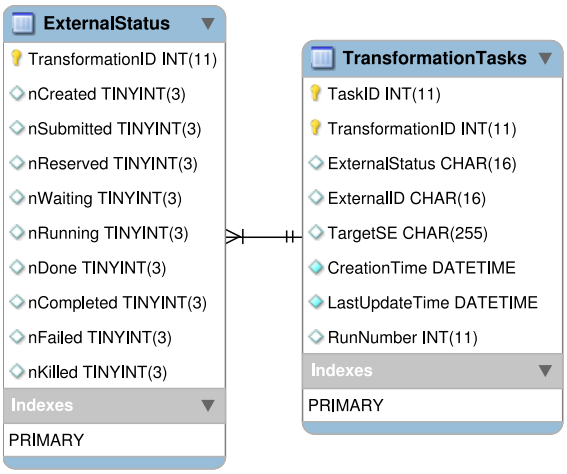

Figura 3.5: Diagramma EER della tabella statistica ExternalStatus.

Si rendono necessari due trigger per popolare la tabella:

```
delimiter //
CREATE TRIGGER TransformationTasks_before_insert
BEFORE INSERT ON TransformationTasks
FOR EACH ROW
begin
    SET NEW.TaskID= (SELECT @last := IFNULL(MAX(TaskID) + 1,1)
    FROM TransformationTasks WHERE TransformationID=NEW.TransformationID);

    if( ( select count(*) from ExternalStatus
    where TransformationID = NEW.TransformationID ) = 0 )
    then
        case NEW.ExternalStatus
            when 'Created' then insert into
            ExternalStatus ( TransformationID, nCreated )
            values ( NEW.TransformationID, 1 );
            when 'Submitted' then insert into
            ExternalStatus ( TransformationID, nSubmitted )
            values ( NEW.TransformationID, 1 );
...
        end case;
    else
        case NEW.ExternalStatus
            when 'Created' then update ExternalStatus
            set nCreated = nCreated + 1
            where TransformationID = NEW.TransformationID;
            when 'Submitted' then update ExternalStatus
            set nSubmitted = nSubmitted + 1
            where TransformationID = NEW.TransformationID;
```

```
...
        end case;
    end if;
end; //

create trigger TransformationTasks_before_update
before update on TransformationTasks
for each row
begin
    case OLD.ExternalStatus
        when 'Created' then update ExternalStatus
        set nCreated = nCreated - 1
        where TransformationID = NEW.TransformationID;
        when 'Submitted' then update ExternalStatus
        set nSubmitted = nSubmitted - 1
        where TransformationID = NEW.TransformationID;
...
    end case;

    case NEW.ExternalStatus
        when 'Created' then update ExternalStatus
        set nCreated = nCreated + 1
        where TransformationID = NEW.TransformationID;
        when 'Submitted' then update
        ExternalStatus set nSubmitted = nSubmitted + 1
        where TransformationID = NEW.TransformationID;
...
    end case;
end; //
delimiter ;
```

A questo punto, dato che le `SELECT` con clausola il valore del campo non saranno più così frequenti, è possibile eliminare l'indice su `ExternalStatus` di `TransformationTasks`:

```
ALTER TABLE TransformationTasks DROP INDEX ExternalStatus;
```

Ovviamente una simile modifica, anche se la maggior parte del cambiamento le rimane trasparente, non può essere fatta senza una modifica all'applicazione. Il codice deve essere modificato per usare la nuova query ed interpretarne il risultato:

```
SELECT * FROM ExternalStatus WHERE TransformationID = '<ID>';
```

La modifica, comunque, è davvero minima e coinvolge un solo metodo.

3.3.2 Performance Schema

Come è facile capire, una corretta ottimizzazione di una Base di Dati, e del RDBMS che la gestisce, dipende fortemente dal carico che queste devono

sopportare. L'analisi dei Log è, in questo ambito, importante e fornisce, a vari livelli, molte informazioni sugli aspetti critici, o comunque passibili di miglioramento, sia del DBMS che dell'applicazione che lo sfrutta.

Tuttavia l'uso dei Log si rivela insufficiente come unico metodo per l'analisi dello stato di un DBMS. I Log presentano, infatti, dei limiti:

- Un Log in forma classica, sia sotto forma di file regolare che di tabella in un database, occupa tanto più spazio quanto più deve essere elevata la sua granularità. Spesso buona parte, quando non la totalità, delle informazioni nei log viene scartata, comunque dopo aver occupato spazio (disco) e tempo (processore), per essere stata raccolta e processata.

- Soprattutto nella forma di file regolari i Log garantiscono una maniera asincrona per analizzare il comportamento del DBMS. Questa caratteristica è ottima per un'analisi che deve avvenire in tempi successivi agli eventi oggetto d'interesse. È però difficile, o quantomeno molto scomodo, implementare su queste informazioni un controllo diretto da parte dell'applicazione che genera il carico.

Dalla versione, GA, 5.5 [6] compresa è presente, nel DBMS MySQL e nei suoi Fork, un nuovo strumento per la misurazione, istantanea, delle prestazioni. Il `performance_schema` si presenta come una collezione di tabelle, non scritte su disco e quindi volatili, che forniscono misurazioni degli eventi. Con evento viene intesa una qualsiasi operazione, compiuta dal DBMS, misurabile in termini di tempo. Le operazioni misurabili vengono definite "instruments". L'attivazione del `performance_schema`, che si avvale per popolare le tabelle di istruzioni inserite direttamente nel codice sorgente del DBMS, può essere fatta solo all'avvio del demone attraverso il parametro omonimo. Per utilizzare il `performance_schema` non è necessaria nessuna modifica agli statement, l'adozione è quindi trasparente all'applicazione. Nemmeno la maniera in cui gli statement vengono eseguiti varia, la raccolta di informazioni è completamente parallela e trasparente al normale funzionamento. Il DBMS alloca la memoria per il funzionamento del `performance_schema` in maniera statica, l'allocazione dinamica non è usata per evitare eccessivi overhead [7]. Il codice del `performance_schema` può crashare, e quindi smettere di funzionare, senza influenzare il resto del DBMS. I risultati delle misurazioni, poiché presentati sotto forma di relazioni, possono essere consultati con Query SQL,

[6] Anche se il funzionamento rimane simile, la quantità di strumenti presenti nelle serie GA 5.6 è nettamente superiore.

[7] Oracle dichiara, ottimisticamente, un overhead compreso tra lo 0 e il 15%. Nei nostri test è stato misurato tra il 15 e il 20%.

in tempo reale da parte dell'applicazione stessa. Questa caratteristica consente di realizzare applicazioni in grado di analizzare, efficacemente, il carico del DBMS su cui vanno ad operare. Anche se con le nuove versioni gli "instruments", inseriti nel codice, aumenteranno, già dalle versioni attuali è possibile ottenere informazioni dettagliate [8] e configurabili su:

- Waits
- Looks
- File I/O
- Mutex
- Cache

Gli "instruments" possono essere configurati in: nome, tipo, numero e unità di misura ritornata (di default cicli di clock). La configurazione può essere eseguita dinamicamente modificando, attraverso normali statement SQL, le tuple della tabella `setup_instruments` nel `performance_schema` [9].

Percona Monitoring Plugins

I grafici rendono facile la lettura di dati. Poter monitorare i parametri di un sistema con un semplice colpo d'occhio accresce la probabilità di individuare situazioni sconvenienti, e magari anche le relative cause, prima che degenerino in, veri e propri, problemi. Percona Monitoring, fornito come plug-in per MySQL e build-in in Percona Server, è un tool che permette il monitoraggio, nelle intenzioni degli sviluppatori dello stesso livello di quello fornito con la versione Enterprise di MySQL, del DBMS attraverso l'integrazione con Nagios o Cacti.

3.3.3 Partizionamento delle Tabelle

Uscendo dalla pura astrazione fornita dal DBMS si possono applicare tecniche di ottimizzazione anche ad un livello più basso. Le tabelle sono dati memorizzati su file che, a loro volta, risiedono su FileSystem [10]. Partizionamento è un termine, dal significato abbastanza ampio, che può descrive

[8] Viene supportata una risoluzione dell'ordine dei picosecondi.

[9] I nomi del performance schema sono maiuscoli per certe release della versione 5.5 e minuscoli per la 5.6

[10] Con i nuovi DBMS è possibile utilizzare partizioni di disco grezze come tablespace per i DB, sebbene questa tecnica fornisca, in certi casi, notevoli benefici; in questo documento non verrà presa in considerazione in quanto contrastante con i requirement.

diverse tecniche le quali, però, hanno tutte uno scopo comune; aumentare le prestazioni del DMBS attraverso l'aumento del numero, e conseguentemente diminuzione delle singole dimensioni, delle relazioni e, quindi, dei file (a volte detti tablespace) utilizzati per memorizzare in dati.

Un vantaggio introdotto, dal partizionamento, è la diminuzione della concorrenza, fra i thread, per l'accesso ai dati che si trovano su partizioni, se non addirittura file, differenti. Tale tecnica, su tabelle largamente popolate e i cui dati sono molto richiesti in ordine casuale, può essere spinta fin al punto di utilizzare diversi dispositivi fisici (dischi) per ospitare i tablespace, portando dunque il partizionamento, e quindi parallelismo nell'accesso ai dati, a livello hardware.

Altro vantaggio introdotto dal partizionamento è quello di permettere, comunque, alle istruzioni di lavorare su set di dati ridotti, rispetto alla totalità delle tuple contenute nella partizione originale.

Va da subito chiarito che, qualunque sia la tecnica adottata, non sempre un partizionamento introduce miglioramenti alle prestazioni. Partizionare i dati comporta un overhead per il DBMS che, necessariamente, deve essere superato, in termini di prestazioni, dai vantaggi del lavorare su set di dati ridotti e con minore concorrenza. Il metodo, e l'eventuale parametro, di partizionamento devono essere ben ponderati. In linea generale si deve essere sicuri che, almeno la maggior parte quando non tutti, gli statement che più impegnano il DBMS vadano a lavorare, ognuno, su una sola partizione alla volta. È ovvio che tutte le operazioni che coinvolgono due, o più, partizioni sono penalizzate e, potenzialmente, a rischio di Dead-lock.

Anche se esistono diverse tecniche per ottenerli, i partizionamenti si possono dividere in due macro-categorie:

p. Verticale

Lo Schema di una tabella, che per la logica relazionale dovrebbe essere unica, viene diviso in più parti, distribuendo i campi sulle varie partizioni, che conterranno il medesimo numero di tuple. Si tratta, in buona sostanza di un partizionamento per colonne. Questa tecnica è indicata quando a far lievitare la dimensione dei dati sono, prevalentemente, i valori di determinati campi che, magari, non vengono spesso utilizzati. Risulta altresì valida quando una tabella presenta un numero molto elevato di campi che, soventemente, non vengono utilizzati, dall'applicazione tutti assieme. L'uso di questo tipo di partizionamento non richiede obbligatoriamente il supporto da parte del DBMS.

p. Orizzontale

Tutte le partizioni contengono i medesimi campi (hanno lo stesso Schema). Si stabilisce una regola, tipicamente sul valore di un campo o indice, per determinare in quale partizione le varie tuple vanno memorizzate. Questa tecnica

è indicata per relazioni dove è solamente il numero di tuple ad aumentare la dimensione della tabella. In sostanza si tratta di un partizionamento per righe. Al fine di ottenere un partizionamento trasparente all'applicazione è necessario che vi sia un supporto da parte del DBMS.

Il progetto DIRAC adotta già, in alcuni casi, un partizionamento verticale per separare, dal resto della relazione, grossi campi di tipo `TEXT` o `BLOB`. Ad esempio, nello Schema TransformationDB preso in considerazione in questa analisi, si nota il caso delle tabelle `TransformationTasks` e `TaskInputs`.

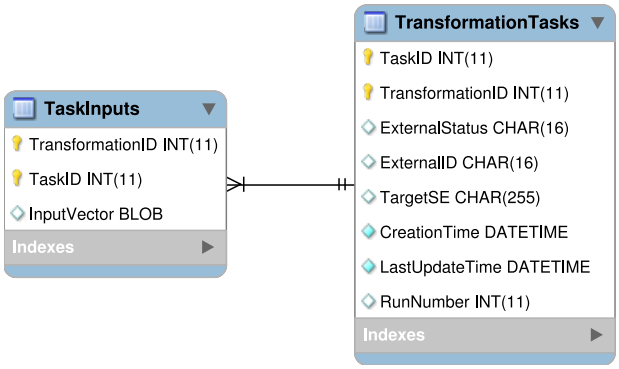

Figura 3.6: La tabella TaskInputs risulta non normalizzata, dovrebbe infatti non esistere, tuttavia alleggerisce di un campo BLOB TransformationTasks.

Le nuove versioni di MySQL hanno largamente migliorato, a diversi livelli, il supporto al partizionamento. Uno dei maggiori svantaggi, ancora presenti, nell'uso del partizionamento Orizzontale è l'impossibilità di definire chiavi esterne con tabelle, InnoDB, partizionate. Non è, infatti, possibile definire foreign keys su una tabella partizionata come, pure, non è possibile, per una qualsiasi tabella, definire foreign keys che referenzino campi di una tabella partizionata. Sebbene gli effetti collaterali, dell'uso di partizionamento orizzontale, siano, per il momento, in contrasto con i requirements del progetto, si possono ottenere comunque alcuni dei vantaggi, in termini di prestazioni, fra quelli sopra descritti sfruttando la possibilità di mantenere un TABLESPACE per tabella InnoDB, come succede di default per MyISAM. I vantaggi ottenuti con l'uso dell'opzione `innodb_file_per_table` [11] (default dalla versione 5.6.6) sono multipli e, in buona parte, ricadono fra quelli necessari al progetto LHCbDIRAC:

[11] Per le versioni 5.6 e successive è possibile definire il path del TABLESPACE direttamente dallo statement di creazione della tabella: CREATE TABLE ... DATA DIRECTORY = absolute_path_to_directory

- La memorizzazione delle tabelle su diversi dispositivi fisici può migliorare le prestazioni di I/O e, inoltre, può essere sfruttata per facilitare le operazioni di back-up e restore.
- Con un file per ogni tabella risulta più facile recuperare lo spazio dopo un `TRUNCATE` o un `OPTIMIZE TABLE` (o sintassi equivalente).
- Avere i file distinti a livello di tabella facilita la migrazione da un'istanza ad un'altra del RDBMS.
- La compressione può essere abilitata separatamente per ogni TABLESPACE.
- Ogni operazione, compiuta su una specifica relazione, inficia meno le prestazioni di quelle effettuate su altre tabelle.

Le recenti versioni di MySQL, e Fork che di esso incorporano continuamente il codice, includono il supporto, per poter sfruttarne al meglio le prestazioni, ai dischi a stato solido (SSD). Tabelle InnoDB che sono soggetto di un gran numero di statement di selezione, possono essere memorizzate su questo tipo di supporti, privi di meccanica, al fine di migliorare le prestazioni del sistema.

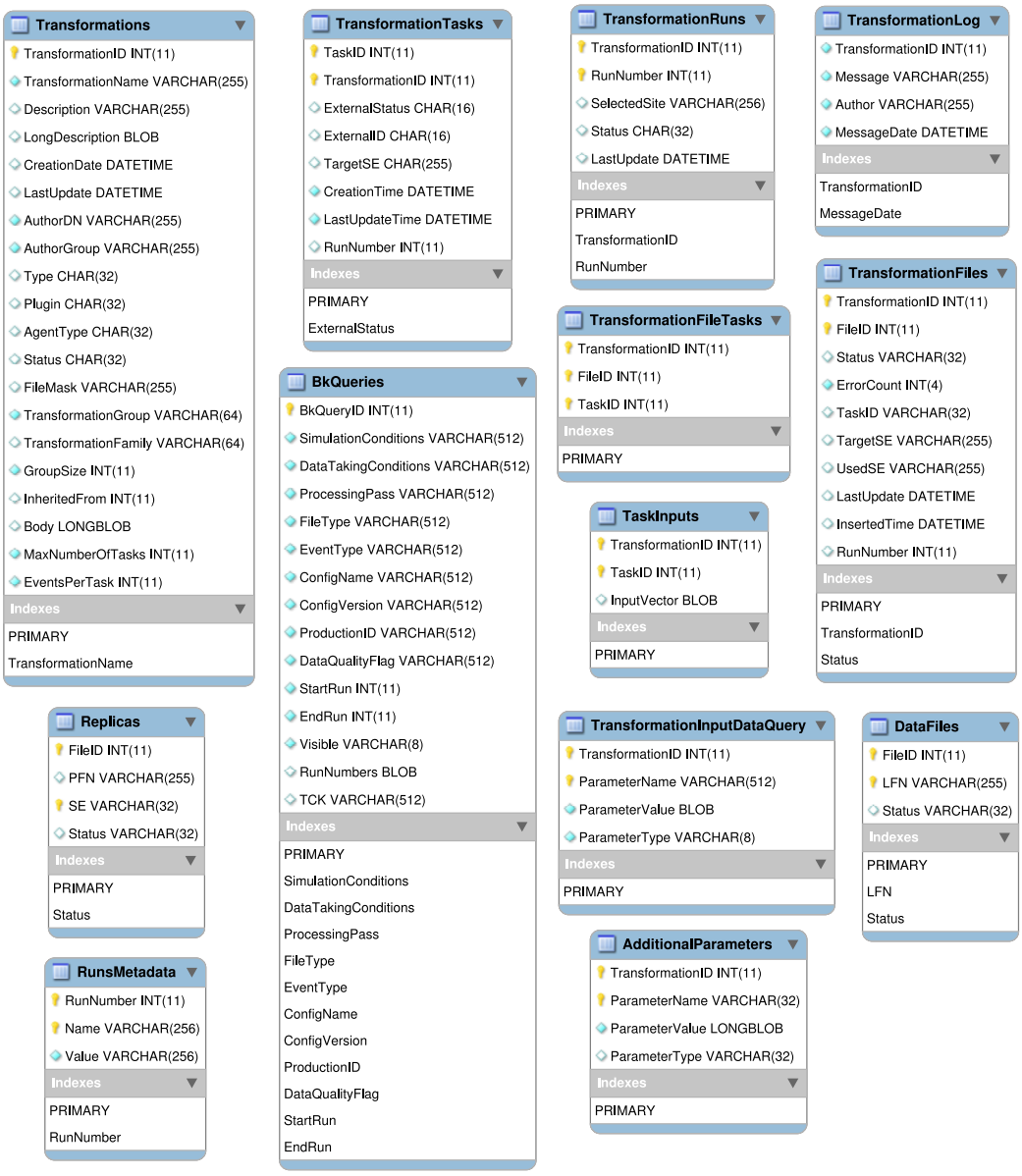

Figura 3.1: Diagramma EER dello schema TransformationDB attualmente in uso da LHCbDIRAC.

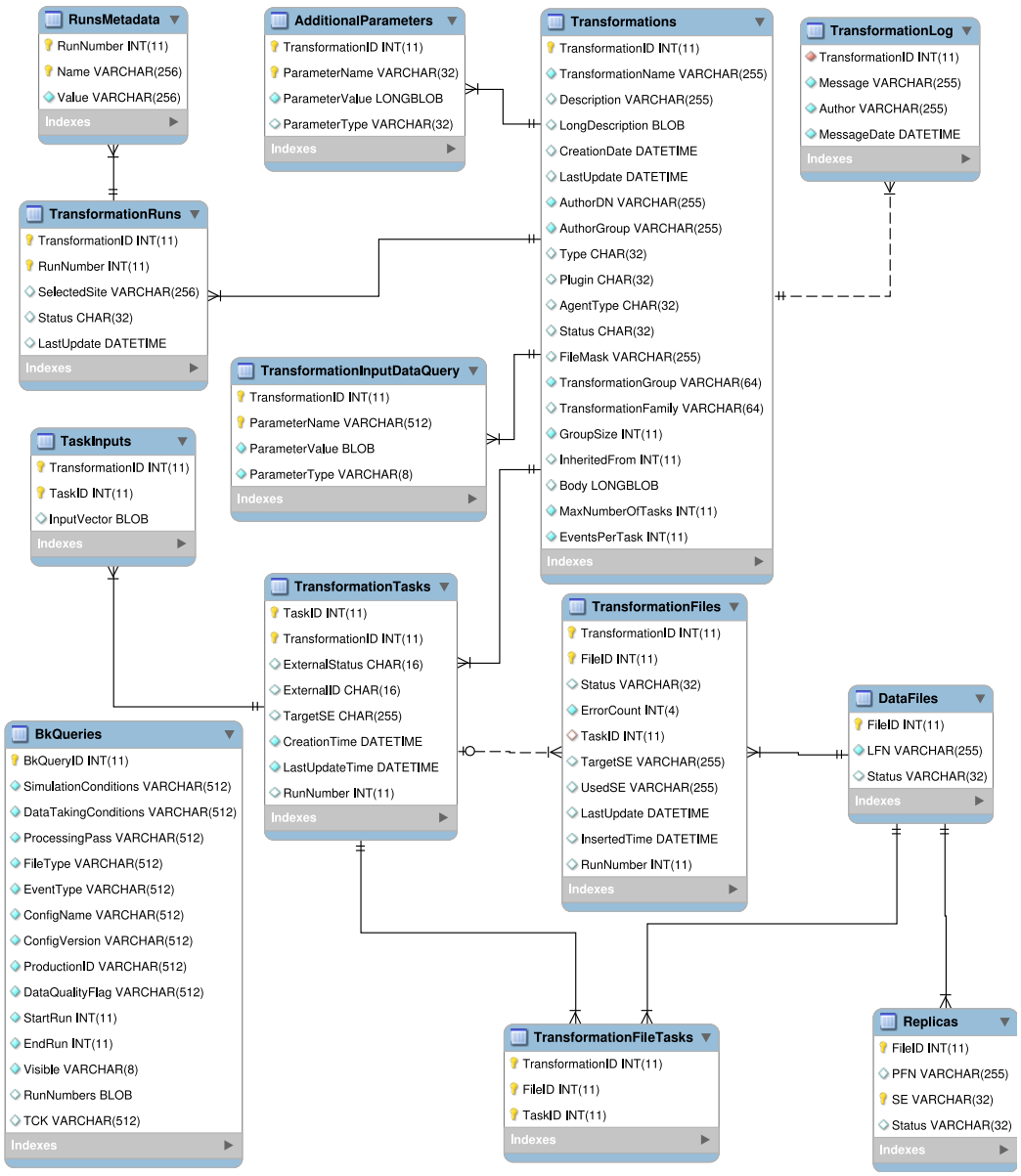

Figura 3.2: Diagramma EER dello schema TransformationDB, convertito all'engine InnoDB, nel quale sono riportati i vincoli relazionali.

Capitolo 4

DataBase Deployment

Attualmente l'installazione di LHCbDIRAC, su cui grava tutta la produzione MonteCarlo e l'analisi dei dati per l'esperimento LHCb del CERN, si affida a delle semplici installazioni stand-alone di una, ormai obsoleta, versione della serie 5.1 di MySQL. Il design del database di back-end di DIRAC permette, grazie alla divisione in più schemi, uno per sottosistema del middleware, un buon partizionamento verticale su più macchine.

Una semplice installazione stand-alone, che sfrutta per i tablespace una normale partizione fisica, è del tutto inadeguata per un sistema che deve garantire alta disponibilità e affidabilità. In pratica l'attuale sistema di produzione/analisi di LHCbDIRAC dipende dal funzionamento di macchine che sono poco più che normali PC, anche a livello di configurazione.

Vista la criticità del servizio, che non può essere interrotto, e l'importanza dei dati, risulta più che ragionevole pensare ad una installazione, per il RDBMS, di tipo Cluster. Tale soluzione dovrà garantire.

- Replicazione dei Dati.
- Una maggiore affidabilità del servizio (eliminazione dello SPOF).

4.1 Percona XtraDB Cluster

Percona XtraDB Cluster è un'alternativa alle soluzioni di Clustering e Replicazione messe a disposizione direttamente da MySQL, in grado di fornire una buona scalabilità e alta disponibilità (High Availability). Percona XtraDB Cluster integra Percona Server, un fork di MySQL che si propone di scalare meglio sui nuovi hardware e fornire prestazioni più stabili, con la libreria Galera di MySQL per fornire alta disponibilità con un miglior rapporto fra risorse impegnante e prestazioni ottenute.

4.1.1 Panorama delle caratteristiche

Percona Cluster si propone di garantire, rispetto alle soluzioni standard di MySQL:

- Minore downtime e maggiore disponibilità (HA).
- Minore investimento per ottenere un struttura che garantisca High Availability.
- Minore tempo (e quindi costo) per la formazione dei Database Administrators (DBA).
- Nessun investimento necessario in software di terze parti per ottenere alta disponibilità.

Le caratteristiche distintive di Percona Cluster sono:

- Capacità di realizzare replicazione sincrona
- Supporto alla replicazione con più nodi master
- Replicazione parallela
- Aggiornamento/Re-sincronizzazione automatica per i nodi

4.1.2 Compatibilità con MySQL e vantaggi

Altra caratteristica saliente di Percona Cluster è quella di essere completamente compatibile con il RDBMS MySQL, su cui si basa e di cui include continuamente il nuovo codice e le features [1]. Questo rende la migrazione da MySQL a Percona completamente trasparente all'applicativo che non deve essere minimamente modificato.

Il motivo principale che ha portato a testare questo software è stata la capacità di realizzare un cluster con più master. In una simile configurazione è possibile eseguire statement di modifica dei dati (INSERT, UPDATE, DELETE, TRUNCATE) su tutti i nodi master indifferentemente, come avviene per quelli di selezione (SELECT). Inoltre avere più master elimina lo SPOF [2] naturalmente rappresentato dal singolo master. Fra master è implementata la replica sincrona, in grado di garantire che ogni transazione, il cui COMMIT termina con successo, sia stata replicata.

[1] L'assoluta compatibilità con MySQL è garantita anche dalla versione stand-alone di Percona Server che, in questo lavoro di tesi, non è stata presa in considerazione solo per motivi di tempo.

[2] SPOF - Single Poin Of Failure.

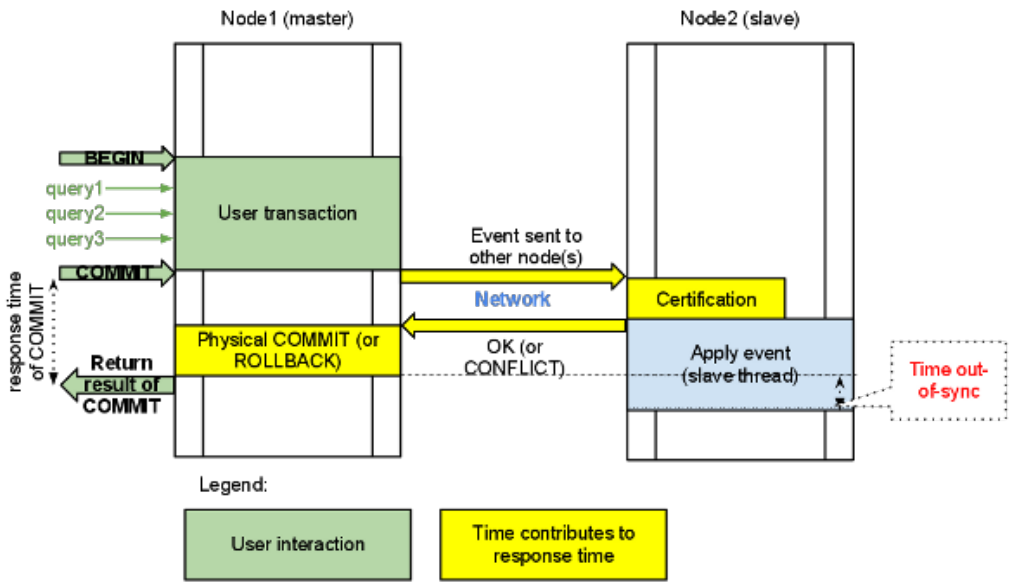

Figura 4.1: Schema temporale della replicazione sincrona effettuata da Percona XtraDB Clsuter; si può notare come la validazione sugli altri nodi avvenga al commit e il piccolo lasso temporale in cui i server sono fuori sincrono.

In realtà la replicazione effettuata da Percona Cluster non è, propriamente, sincrona. Viene infatti definita "virtually synchronous replication". Esiste un breve periodo di tempo, successivo alla validazione della transazione da parte dei nodi, in cui i dati non sono consistenti su tutto il cluster. In questo, breve, lasso di tempo lo stesso statement, di selezione, potrebbe dare risultati discordanti se eseguito su nodi diversi. Al costo di un, piccolo, peggioramento nelle prestazioni delle `SELECT`, che verranno messe in wait durante l'applicazione degli eventi sul nodo, si può, settando la variabile `wsrep_causal_reads` ad `ON`, rendere la replicazione completamente sincrona.

Fornendo, in questo contesto, delle funzionalità aggiuntive, lo statement `COMMIT` può incontrare tipi di problemi differenti da quelli riscontrabili in una normale installazione stand-alone. Il fallimento della fase di certificazione, sugli altri nodi, viene segnalato tornando un errore `DEADLOCK` o `LOCK TIMEOUT`. È buona pratica modificare l'applicazione in modo che possa reagire a questi "nuovi" errori.

Delle normali installazioni stand-alone di MySQL, poste in configurazione di replicazione, permetterebbero di eseguire modifiche dei dati solo su un'unico master, che risulterebbe sgravato "solamente" dalle SELECT, uniche operazioni eseguibili sugli slave senza perdere consistenza. Tipicamente gli slave

vengono messi in modalità read-only per evitare il pericolo di inconsistenze. Inoltre l'aumento nel numero dei nodi comunque non eliminerebbe il singolo SPOF rappresentato dal master. Solo dalla versione 5.6 MySQL supporta una modalità di replicazione semi-sincrona in cui, per ogni transazione chiusa con successo, viene garantito che almeno uno slave abbia effettivamente replicato.

4.1.3 Split-Brain e Quorum

L'adozione di Galera Lib, e quindi anche di Percona Cluster che su essa si basa, imporrebbe l'uso di un numero minimo di tre nodi per evitare la Situazione dello Split-Brain. Bisognerebbe sempre dividere il Sistema in un numero di partizioni dispari. Per partizioni, in questo ambito, si intende parti del Cluster che hanno un buona probabilità di fallire, o venire isolate dal resto del sistema, indipendentemente. Altra buona regola è quella di rendere le potenziali partizioni di uguale cardinalità (stesso numero di nodi) e con pari probabilità di fallimento/isolamento. Ad esempio tre macchine sotto lo stesso switch (che diventa uno SPOF) rappresentano un, minimo, buon partizionamento, ognuna delle macchine ha la stessa probabilità di guastarsi o essere isolata dalle altre provocando una perdita di pari peso al sistema. Anche un cluster di sei macchine, sebbene sia un numero pari, può essere ben partizionato collegando due macchine per ogni switch della rete; in questo caso i tre switch garantiscono un numero dispari di partizioni.

Ogni volta che un nodo si aggiunge al Cluster un contatore, comune a tutti i nodi, viene incrementato. Simmetricamente, ogni volta che un nodo si disconnette, non per fallimento o isolamento ma in maniera regolare, dal Cluster il sistema decrementa il contatore. Quando uno o più nodi del Cluster non rispondono per un determinato periodo di tempo, normalmente dell'ordine di qualche secondo, si rende necessario un voto. Tutti i nodi ancora connessi comunicano il voto agli altri. I voti vengono sommati e si calcola la percentuale di votanti rispetto al contatore totale dei nodi. Se si supera il Quorum (50%) la partizione rimane attiva; resta in stato Primary e i suoi nodi accettano Query. Se il Quorum non viene superato la partizione capisce di essere quella con meno nodi e diventa passiva; passa in stato non-Primary e i suoi nodi non accettano Query. Questo comportamento è reso necessario onde evitare che due partizioni, rimaste isolate tra loro ma ancora raggiungibili dai clients, possano, rimanendo attive contemporaneamente, rendere inconsistente la base di dati.

Il problema dello Split-Brain si presenta quando nessuna delle due partizioni, di un sistema, causate da un'isolamento o da un guasto, è in grado di raggiungere il Quorum e, quindi, tutto il sistema passa in stato passivo. Se,

ad esempio, siamo in presenza di un Cluster con solo due nodi e uno si guasta l'altro otterrà dal voto, solo il suo, una percentuale del 50% e, non passando il Quorum, diverrà passivo. È facile notare che per un nodo è impossibile distinguere se la controparte è isolata o caduta, quindi deve comportarsi sempre nella stessa maniera. Nel caso i due nodi siano isolati fra loro, ma ancora funzionanti, si avranno due partizioni passive per evitare l'inconsistenza dei dati.

Se ne deduce che un sistema di tipo fail-over può essere implementato con un numero minimo di 3 nodi.

4.1.4 Configurazione a due nodi master

La configurazione realizzata è caratterizzata da soli due nodi master, nonostante la documentazione ufficiale consigli almeno tre nodi per un uso in produzione. Tuttavia con due nodi si ottiene lo stesso una replicazione e attraverso l'uso di un nodo fittizio del cluster, realizzato con Galera Arbitrator, è possibile ottenere la piena High Availability del sistema attraverso l'eliminazione del problema dello Split-Brain. L'aggiunta di un ulteriore nodo al Cluster, qualora si optasse per le tre macchine, è semplice e richiede di seguire i passi descritti per il secondo server.

Regole del Firewall e SELinux

Tutti i sistemi operativi moderni mettono a disposizione, di default, un servizio di FireWall abbastanza restrittivo. Questa politica è, di norma, un ottimo punto di partenza per tutte le possibili installazioni. Quando si vuole attivare un nuovo servizio, rendendolo disponibile all'esterno dell'host, su un sistema si devono definire delle nuove regole per permettervi l'accesso.

La documentazione di Percona Cluster consiglia di disabilitare SELinux ed affidarsi ai classici metodi dei sistemi *POSIX* per gestire la sicurezza; per ottenere questa configurazione sui sistemi el6 è sufficente editare il file `/etc/selinux/config` andando a settare la variabile SELINUX al valore - disabled:

```
SELINUX=disabled
```

Questo garantirà che al successivo boot il sistema non abiliti Security-Enhanced Linux, come avviene di default. Una volta riavviata la macchina si può usare il comando `sestatus` per controllare lo stato di SELinux; l'output desiderato è del tipo: `SELinux status: disabled`

Perché un'stanza di Percona possa partecipare ad un Cluster la macchina, sulla quale andrà a girare, deve permettere l'accesso alle porte TCP 3306, 4444, 4567 e 4568. Per ottenere una simile configurazione sui sistemi el6

bisogna editare il file `/etc/sysconfig/iptables` come mostrato di seguito:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
### Regole per le porte del Cluster Percona ###
-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 4567 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 4444 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 4568 -j ACCEPT
##############################################
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Riavviare il servizio iptables con il comando `service iptables restart`, se le regole sono state ben definite l'output sarà del tipo:

```
iptables: Flushing firewall rules:                          [  OK  ]
iptables: Setting chains to policy ACCEPT: filter           [  OK  ]
iptables: Unloading modules:                                [  OK  ]
iptables: Applying firewall rules:                          [  OK  ]
```

È possibile verificare ulteriormente la comprensione delle regole da parte del firewall di sistema con il comando `service iptables status` che dovrebbe fornire un output simile al seguente:

```
Table: filter
Chain INPUT (policy ACCEPT)
num  target  prot opt source      destination
1    ACCEPT  all  --  0.0.0.0/0   0.0.0.0/0  state RELATED,ESTABLISHED
2    ACCEPT  icmp --  0.0.0.0/0   0.0.0.0/0
3    ACCEPT  all  --  0.0.0.0/0   0.0.0.0/0
4    ACCEPT  tcp  --  0.0.0.0/0   0.0.0.0/0  state NEW tcp dpt:22
5    ACCEPT  tcp  --  0.0.0.0/0   0.0.0.0/0  state NEW tcp dpt:3306
6    ACCEPT  tcp  --  0.0.0.0/0   0.0.0.0/0  state NEW tcp dpt:4567
7    ACCEPT  tcp  --  0.0.0.0/0   0.0.0.0/0  state NEW tcp dpt:4444
8    ACCEPT  tcp  --  0.0.0.0/0   0.0.0.0/0  state NEW tcp dpt:4568
9    REJECT  all  --  0.0.0.0/0   0.0.0.0/0  reject-with icmp-host-prohibited

Chain FORWARD (policy ACCEPT)
```

```
num  target  prot opt source      destination
1    REJECT  all  --  0.0.0.0/0   0.0.0.0/0   reject-with icmp-host-prohibited

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
```

Installazione su el6

Percona mette a disposizione un repository per le distribuzioni che adottano il gestore di pacchetti yum, l'installazione e l'aggiornamento su Enterprise Linux e derivate, tra cui CentOS e Scientific Linux, risulta quindi facilitata. Aggiunta del repository Percona fra quelli del sistema:

```
rpm -Uhv \
http://www.percona.com/downloads/percona-release/percona-release-0.0-1.x86_64.rpm
```

Installazione dei software necessari usando yum:

```
yum install -y Percona-XtraDB-Cluster-server \
Percona-XtraDB-Cluster-client xtrabackup
```

Le due macchine usate per il test sono d1.fe.infn.it e d2.fe.infn.it (da ora in poi d1 e d2 per comodità). Questi host non hanno un indirizzo IP statico all'interno della LAN dell'INFN di Ferrara. Nonostante, nelle guide ufficiali, sia richiesto di usare gli indirizzi IP nei file di configurazione e non gli hostname, anche utilizzando quest'ultimi, per identificare le macchine, il cluster funziona perfettamente comunque. Tuttavia in produzione è sconsigliabile dipendere da un DNS, per l'identificazione dei nodi, e optare per degli indirizzi statici. I due file di configurazione, qui riportati in versione minimale e senza nessun ulteriore file tuning di sorta, sono davvero semplici e poco differenti fra le varie macchine. Nelle distribuzioni el6 il file di configurazione, compatibilmente con quello di MySQL, è sito in `/etc/my.cnf`.

File di configurazione per d1:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path alla libreria Galera
wsrep_provider=/usr/lib64/libgalera_smm.so

# URL di connessione al Cluster
# deve contenere gli indirizzi/hostname di nodo#1, nodo#2
# (Meglio specificare indirizzi IP se statici)
wsrep_cluster_address=gcomm://d1.fe.infn.it,d2.fe.infn.it
```

```
# BinLog settao a ROW per Galera Lib
binlog_format=ROW

# L'engine MyISAM è ancora supportato solo in maniera sperimentale
default_storage_engine=InnoDB

# Piccolo File Tuning per le performance caldamente consigliato
innodb_locks_unsafe_for_binlog=1

# Richiesto da Galera Lib
# Serve a cambiare la maniera in cui InnoDB gestisce il
# LOCK della tabella in caso di campo AUTOINCREMENT.
innodb_autoinc_lock_mode=2

# Nodo #1 (Meglio specificare indirizzo IP se statico)
wsrep_node_address=d1.fe.infn.it

# imposta il metodo di replicazione SST
wsrep_sst_method=xtrabackup

# Nome del Cluster
wsrep_cluster_name=LHCbFE

# Account utente per SST
wsrep_sst_auth="<sstuser>:<sstpasswd>"
```

File di configurazione per d2:

```
[mysqld]

datadir=/var/lib/mysql
user=mysql

# Path alla libreria Galera
wsrep_provider=/usr/lib64/libgalera_smm.so

# URL di connessione al Cluster
# deve contenere gli indirizzi/hostname di nodo#1, nodo#2
# (Meglio specificare indirizzi IP se statici)
wsrep_cluster_address=gcomm://d1.fe.infn.it,d2.fe.infn.it

# BinLog settao a ROW per Galera Lib
binlog_format=ROW

# L'engine MyISAM è ancora supportato solo in maniera sperimentale
default_storage_engine=InnoDB

# Piccolo File Tuning per le performance caldamente consigliato
```

```
innodb_locks_unsafe_for_binlog=1

# Richiesto da Galera Lib
# Serve a cambiare la maniera in cui InnoDB gestisce il
# LOCK della tabella in caso di campo AUTOINCREMENT.
innodb_autoinc_lock_mode=2

# Nodo #2 (Meglio specificare indirizzo IP se statico)
wsrep_node_address=d2.fe.infn.it

# imposta il metodo di replicazione SST
wsrep_sst_method=xtrabackup

# Nome del Cluster
wsrep_cluster_name=LHCbFE

# Account utente per SST
wsrep_sst_auth="<sstuser>:<sstpasswd>"
```

Il file di configurazione del primo nodo ad essere avviato, nel caso specifico d1, dovrebbe contenere il parametro `wsrep_cluster_address` settato a `gcomm://` rispetto alla configurazione sopra mostrata. Tuttavia questa opzione, una volta avviato il demone, andrebbe riportata al valore precedente, poiché creerebbe problemi durante un successivo riavvio quando altri nodi siano stati già avviati. Una soluzione, che non prevede di ri-editare il file, è forzare l'opzione alla riga di comando:

```
service mysql start --wsrep-cluster-address="gcomm://"
```

In questa maniera si comunica al demone che può fare il bootstrap senza bisogno di connettersi a nessun altro nodo del Cluster.

Si necessita ora di definire una password per l'utente root di MySQL, di default l'installazione non la prevede. Gli statement e comandi che si possono usare sono quelli tipici di MySQL. Ad esempio è possibile usare mysqladmin.

```
mysqladmin -u root -p password
<password>
<password>
```

Bisogna definire l'utenza per SST [3] , per farlo bisogna connettersi al DBMS come root ed usare i seguenti statement:

[3]SST - Percona definisce State Snapshot Transfer l'operazione che comporta la copia completa dei dati da un nodo donatore (Donor) a uno appena entrato a fare parte del Cluster (Joiner), che necessita di sincronizzarsi. Quando il delta dei dati di un nodo non supera la *gcache* massima del cluster viene applicato il metodo Incremental State Transfer (IST) meno gravoso per il sistema.

```
mysql -u root -p
<password>

mysql-prompt> CREATE USER '<sstuser>'@'localhost'
           -> IDENTIFIED BY '<sstpasswd>';
mysql-prompt> GRANT RELOAD, LOCK TABLES, REPLICATION CLIENT
           -> ON *.* TO '<sstuser>'@'localhost';
mysql-prompt> FLUSH PRIVILEGES;
```

A questo punto, se i file di configurazione sono stati compilati bene e in modo coerente con l'account SST, è sufficiente avviare i demoni, uno alla volta, sulle varie macchine. L'operazione può essere fatta nella normale maniera fornita dai tool del SystemV:

```
service mysql restart
```

Se il nodo è stato configurato bene importerà tutte le impostazioni del primo server avviato, password di root compresa. Quindi riuscire a loggarsi con la password di root impostata sul primo nodo anche sugli altri è un ottimo inizio. Una volta loggati nel nodo è possibile avere informazioni sullo stato dello stesso e del Cluster con lo statement:

```
SHOW STATUS LIKE 'wsrep%';
```

Che produce un'output simile al seguente:

```
+----------------------------+---------------------------------------+
| Variable_name              | Value                                 |
+----------------------------+---------------------------------------+
| wsrep_local_state_uuid     | b0a1c52f-dcce-11e2-0800-fe7d8793e0f3  |
| wsrep_protocol_version     | 4                                     |
| wsrep_last_committed       | 3                                     |
| wsrep_replicated           | 0                                     |
| wsrep_replicated_bytes     | 0                                     |
| wsrep_received             | 2                                     |
| wsrep_received_bytes       | 212                                   |
| wsrep_local_commits        | 0                                     |
| wsrep_local_cert_failures  | 0                                     |
| wsrep_local_bf_aborts      | 0                                     |
| wsrep_local_replays        | 0                                     |
| wsrep_local_send_queue     | 0                                     |
| wsrep_local_send_queue_avg | 0.000000                              |
| wsrep_local_recv_queue     | 0                                     |
| wsrep_local_recv_queue_avg | 0.000000                              |
| wsrep_flow_control_paused  | 0.000000                              |
| wsrep_flow_control_sent    | 0                                     |
| wsrep_flow_control_recv    | 0                                     |
| wsrep_cert_deps_distance   | 0.000000                              |
```

```
| wsrep_apply_oooe           | 0.000000                                |
| wsrep_apply_oool           | 0.000000                                |
| wsrep_apply_window         | 0.000000                                |
| wsrep_commit_oooe          | 0.000000                                |
| wsrep_commit_oool          | 0.000000                                |
| wsrep_commit_window        | 0.000000                                |
| wsrep_local_state          | 4                                       |
| wsrep_local_state_comment  | Synced                                  |
| wsrep_cert_index_size      | 0                                       |
| wsrep_causal_reads         | 0                                       |
| wsrep_incoming_addresses   | d1.fe.infn.it:3306,d2.fe.infn.it:3306 |
| wsrep_cluster_conf_id      | 4                                       |
| wsrep_cluster_size         | 2                                       |
| wsrep_cluster_state_uuid   | b0a1c52f-dcce-11e2-0800-fe7d8793e0f3  |
| wsrep_cluster_status       | Primary                                 |
| wsrep_connected            | ON                                      |
| wsrep_local_index          | 0                                       |
| wsrep_provider_name        | Galera                                  |
| wsrep_provider_vendor      | Codership Oy <info@codership.com>     |
| wsrep_provider_version     | 2.5(r150)                               |
| wsrep_ready                | ON                                      |
+----------------------------+-----------------------------------------+
```

Di cui le coppie chiave valore più significative, al fine di verificare se l'installazione del cluster è andata a buon fine, sono:

- `wsrep_local_state_comment` = stato sincronizzazione nodo
- `wsrep_cluster_size` = nomero dei nodi attualmente presenti
- `wsrep_incoming_addresses` = lista nodi e porte dei server
- `wsrep_cluster_status` = tipo di nodo (Primary o Slave)
- `wsrep_connected` = stato connessione al servizio di replicazione
- `wsrep_ready` = stato servizio replicazione

Da ora è possibile su tutte le macchine, prima ad essere avviata compresa, usare i tool del SystemV, forniti dal sistema, per gestire i singoli servizi, senza aggiungere nessuna opzione aggiuntiva. Su el6 si può usare la classica sintassi:

```
service mysql start|status|restart|stop
```

Per accertarsi, in caso di riavvio della macchina, che il nodo torni "up" automaticamente si può usare il comando `chkconfig --level 2345 mysql on` su ogni host.

Il test conclusivo della fase di installazione è stato quello di: creare un database di prova connettendosi a d1, creare una tabella all'interno del database appena realizzato connettendosi a d2, popolare la tabella di test da sessioni aperte su entrambe le macchine.

Galera Arbitrator

Galera Arbitrator è un software nato per consentire di risolvere il problema dello Split-Brain nei sistemi che usano la Galera Lib e implementano un numero di nodi/partizioni pari. In pratica si tratta di un semplice demone che esplica due funzioni principali:

- partecipa al contatore dei nodi del cluster incrementandolo di una unità (per istanza avviata)

- in caso uno o più nodi vadano in timeout vota come fosse un vero nodo del Cluster

Il fatto di non essere un vero nodo, ma un semplice demone dalle basse richieste di risorse, gli permette di essere messo in esecuzione su macchine che offrono altri servizi senza, di fatto, influire sulle prestazioni del sistema. Nell'ambito del progetto LHCbDIRAC, ma non solo, potrebbe avere senso rendere attivo un'arbitrator su macchine che eseguono Agent, o Service, ritenuti prioritari rispetto ad altri che sfruttano lo stesso Cluster. Va ricordato che l'arbitrator deve poter osservare tutto il traffico del cluster, porlo su di un link con banda ristretta verso il resto dei nodi peggiorerebbe, sensibilmente, le prestazioni del Cluster.

Anche per l'Arbitrator bisogna definire una nuova regola del firewall modificando `/etc/sysconfig/iptables` similmente a quanto mostrato di seguito:

```
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
#### Regole per Galera Arbitrator (garbd) ####
-A INPUT -m state --state NEW -m tcp -p tcp --dport 4567 -j ACCEPT
```

```
#############################################
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

Per rendere effettive le modifiche si deve riavviare il servizio come visto precedentemente per i nodi. Nella nostra istallazione di prova l'arbitrator è stato posto sulla macchina d3.fe.infn.it (da ora in poi solo d3 per comodità).

Si può testare subito l'arbitrator, avviandolo come normale utente non privilegiato in foreground, con il comando:
`garbd -a gcomm://d2.fe.infn.it:4567 -g LHCbFE`
Con l'opzione `-a` viene specificato indirizzo/nome e porta di un nodo del cluster attualmente attivo. L'opzione `-g` specifica il nome del cluster, come dichiarato nei file di configurazione dei nodi. L'output non sarà riportato poiché vistosamente prolisso e sufficientemente chiaro.

Per verificare se effettivamente il cluster ha aumentato il proprio contatore dei nodi è sufficiente, dopo essersi connessi ad un nodo, usare lo statement:
`show status like 'wsrep_cluster_size';`
Si nota che il contatore è aumentato di un'unità con l'avvio dell'arbitrator.

```
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| wsrep_cluster_size | 3     |
+--------------------+-------+
1 row in set (0.00 sec)
```

Arbitrator prevede anche un'opzione, `-d`, per essere avviato come processo demone. Avviato in questa modalità il processo resiste al SIGHUP, del terminale che lo ha avviato, ma viene perso l'output e, in caso di riavvio dell'host, il servizio non sarà automaticamente rialzato. Per ovviare a questi problemi si è costruito uno script compatibile con il SystemV. Lo script è posto in `/etc/init.d/arbitrator` e il suo conteno è:

```
#!/bin/bash
#
# chkconfig: 2345 65 35
# description: Galera Arbitrator deamon for Percona Cluster
#
if [ "$(whoami)" != 'root' ];
then
echo "You don't have the Super-Cow power!";
exit 1;
fi
```

```
# Get function from functions library
. /etc/init.d/functions
#### Setup Variables ####
NODE_NAME="d2.fe.infn.it"
NODE_PORT="4567"
CLUSTER_NAME="LHCbFE"
OUTPUT_FILE="/var/lib/mysql/garbd.log"
#########################
# Start the arbitrator service
start() {
        daemon --user=mysql "/usr/bin/garbd -a gcomm://$NODE_NAME:$NODE_PORT \
        -g $CLUSTER_NAME &> $OUTPUT_FILE &"
        ### Create the lock file ###
        touch /var/lock/subsys/garbd
        success $"Arbitrator server startup"
        echo
}
# Restart the Arbitrator service
stop() {
        killproc garbd
        ### Now, delete the lock file ###
        rm -f /var/lock/subsys/garbd
        echo
}
### main logic ###
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status garbd
        ;;
  restart|reload|condrestart)
        stop
        start
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|reload|status}"
        exit 1
esac
exit 0
```

Sono stati assegnati allo script i permessi corretti con: `chmod 755 /etc/init.d/arbitrator`. Ora il servizio può essere controllato con i normali tool del SystemV forniti dal sistema:

```
service arbitrator start|status|restart|stop
```

Si è poi provveduto a renderlo attivo all'avvio con: `chkconfig --add arbitrator`. Per controllare se l'operazione è andata a buon fine si può usare il comando `chkconfig --list`, il cui output deve restituire, fra le altre, una riga simile a questa:

```
arbitrator      0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

4.1.5 Test per i campi AUTO_INCREMENT

Per gestire i campi AUTO_INCREMENT, Percona Cluster varia l' `auto_increment_offset` su ogni nodo. All'interno del singolo nodo il metodo di Locking è lo stesso normalmente usato per le, comuni, tabelle InnoDB. Nel caso di inserimenti effettuati su più nodi, contemporaneamente, viene usato il, così detto, "optimistic locking" e, in pratica, il Cluster verifica la propagabilità dell'operazione solo al momento del `COMMIT`. In caso l'inserimento non sia propagabile, a tutti i nodi, l'applicazione riceverà, in risposta al `COMMIT`, un `LOCK_ERROR`.

Anche se il comportamento del Cluster è sufficientemente lineare da rendere semplice la, necessaria, modifica all'applicazione per adattarvisi, si è ritenuto di interesse condurre alcuni, semplici, test al riguardo. A tale scopo si sono implementati degli script, in `PHP`, che, connettendosi contemporaneamente ad entrambi i nodi anche con più processi, effettuano continui inserimenti in una tabella con chiave primaria avente proprietà `AUTO_INCREMENT`.

Sono stati condotti test fino a dieci connessioni, contemporanee, per Nodo e 51200 inserimenti a connessione. Un primo notevole risultato è rappresentato dal non aver mai ottenuto Errori o Warnings, tutti gli inserimenti sono andati a buon fine, seppur con qualche deterioramento delle prestazioni in certi casi. È necessario far notare da subito che il carico, prodotto dagli script, rappresenta una condizione estrema che difficilmente si riscontrerà nell'uso reale, tanto più per periodi lunghi come quelli dei test.

Da ciò che si è potuto osservare il comportamento del Cluster, per quanto concerne il caso considerato, è descrivibile in due fasi:

- per un breve periodo di tempo, che equivale a quello di un sensato burst di inserimenti consecutivi, i thread di entrambi i nodi inseriscono, grossomodo, alla stessa velocità.
- uno dei due host mantiene la stessa velocità media della fase iniziale, mentre i thread dell'altro nodo vengono, significativamente, rallentati.

Fra i thread appartenenti allo stesso Nodo vi è, mediamente, assoluta parità in tutte le fasi.

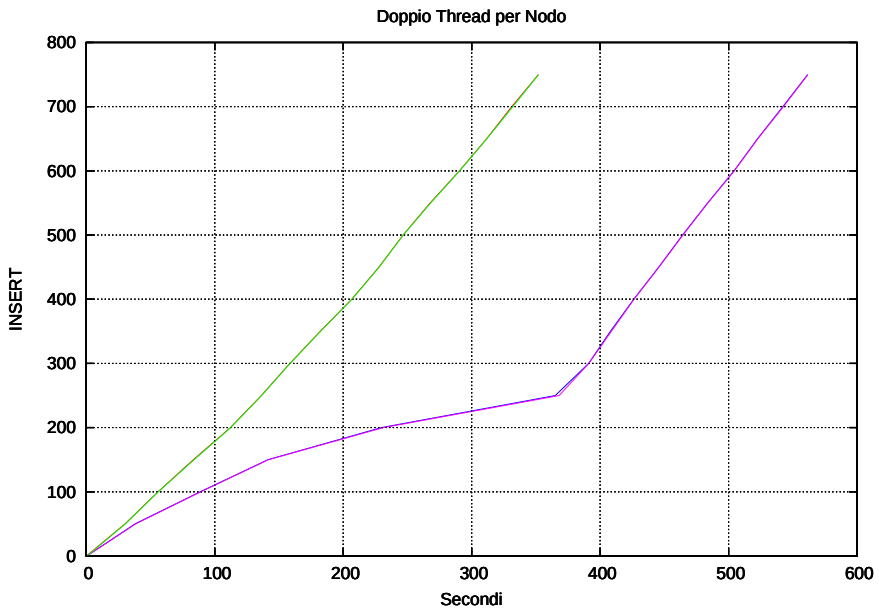

Figura 4.2: Test da 800 inserimenti con due Thread per Nodo. Le linee appartenenti ai Thread dello stesso nodo sono, per la maggior parte, sovrapposte.

Aggiungendo, nel codice che effettua le `INSERT`, degli `sleep` da mezzo secondo ogni 400 inserimenti, condizione ancora lontana dall'essere comune nei reali carichi, si è potuto notare, in un buona percentuale dei casi, la tendenza del Cluster a invertire la "priorità" assegnata agli host.

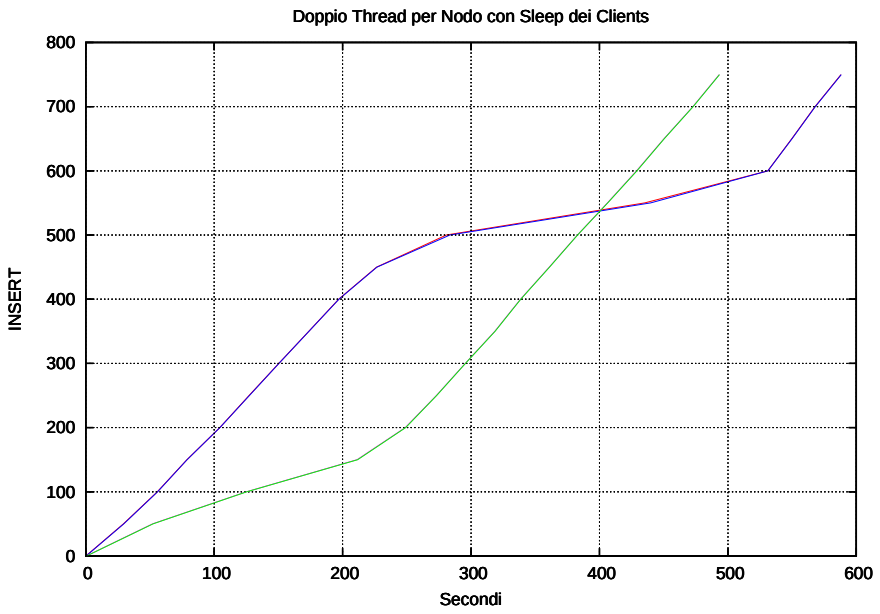

Figura 4.3: Test da 800 inserimenti con due Thread per Nodo. Da notare, nell'intorno dei 400 inserimenti per il primo nodo, lo scambio delle pendenze fra le curve. Le linee appartenenti ai Thread dello stesso nodo sono, per la maggior parte, sovrapposte.

4.2 HAproxy

Una volta ottenuto un Cluster funzionante bisogna far si che l'applicazione lo sfrutti. Il Cluster Percona non è in grado di bilanciare il carico sui suoi nodi; questa incombenza viene lasciata all'applicazione o a qualcosa per essa. Al fine di garantire la completa trasparenza del cambiamento all'applicazione si è deciso di optare per un bilanciatore del carico esterno. La scelta è caduta sul software più affermato in questo campo; HAProxy.

HAProxy è un Load-Balnacer TCP/HTTP open source, comunemente usato per aumentare le prestazioni di siti WEB e altri servizi praticando la suddivisione delle richieste totali su più server.

4.2.1 Heart-Beat da Percona Cluster

Percona Cluster viene fornito con uno script appositamente pensato per dare informazioni, sullo stato di salute dei vari nodi, a HAProxy. Lo script sfrutta xinetd, che quindi deve essere installato sui nodi, e nei sistemi el6 ha il percorso `/usr/bin/clustercheck`.

Per prima cosa è necessario creare un'utenza MySQL per clustercheck. Per fare questo basta connettersi ad uno solo dei nodi e usare gli statements:

```
GRANT PROCESS ON *.* TO 'cluster_check'@'localhost'
IDENTIFIED BY '<cluster_check_passwd>';
FLUSH PRIVILEGES;
```

Le operazioni descritte da questo punto in poi vanno eseguite su ogni nodo del cluster (nel nostro esempio d1 e d2). È necessario fare una modifica allo script `clustercheck` in maniera che possa svolgere il proprio compito senza bisogno di argomenti aggiuntivi alla riga di comando, le righe relative a `MYSQL\_USERNAME` e `MYSQL\_PASSWORD` vanno editate in accordo a quanto segue:

```
MYSQL_USERNAME="${1-cluster_check}"
MYSQL_PASSWORD="${2-<cluster_check_passwd>}"
```

Lanciando direttamente il comando, su un nodo attivo e sincronizzato, si dovrebbe ottenere un output simile al segunte:

```
HTTP/1.1 200 OK
Content-Type: text/plain
Connection: close
Content-Length: 40

Percona XtraDB Cluster Node is synced.
```

L'installazione di xinetd può essere fatta, sui sistemi el6, con yum:
`yum -y install xinetd`
Il servizio va confermato attivo a tutti i runlevel a cui lo sono i server Percona:
`chkconfig --level 2345 xinetd on`

Si aggiunge il servizio mysqlchk al file `/etc/services` appendendo la riga:
`mysqlchk 9200/tcp # mysqlchk`

Si configura il servizio mysqlchk editando il file `/etc/xinetd.d/mysqlchk` in accordo a quanto segue:

```
# default: on
# description: mysqlchk
service mysqlchk
{
# this is a config for xinetd, place it in /etc/xinetd.d/
        disable = no
        flags           = REUSE
        socket_type     = stream
        port            = 9200
        wait            = no
        user            = mysql
        server          = /usr/bin/clustercheck
        log_on_failure  += USERID
        only_from       = 192.84.144.0/24
        # recommended to put the IPs that need
        # to connect exclusively (security purposes)
        per_source      = UNLIMITED
}
```

In particolare è bene controllare che:

- l'utente usato abbia permessi sufficienti per eseguire clustercheck.
- il percorso dello script clustercheck sia corretto.
- si conceda la connessione al più stretto numero di macchine possibile (`only_from`).

A questo punto si rende necessario configurare una nuova regola per iptables, sulla porta 9200, accodata a quelle precedentemente usate per Percona Cluster.
`-A INPUT -m state --state NEW -m tcp -p tcp --dport 9200 -j ACCEPT`

Ora si può procedere ad avviare il servizio xinetd con `service xinetd start` e controllare, da un'altro nodo, che funzioni correttamente con `telnet <ip_macchina_configurata> 9200`. L'output dovrebbe essere identico, salvo le normali aggiunte fatte da `telnet`, a quello precedentemente osservato lanciando clustercheck localmente.

4.2.2 Configurazione HAProxy

Nella nostra installazione di test la macchina che ospita il bilanciatore di carico è d3. HAProxy è disponibile nei repository el6:
`yum -y install haproxy`

Per configurare HAProxy serve modificare il file `/etc/haproxy/haproxy.cfg` coerentemente a quanto segue:

```
global
    log         127.0.0.1 local2

    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

    stats socket /var/lib/haproxy/stats

defaults
    mode                    http
    log                     global
    option                  tcplog
    option                  dontlognull
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000

#---------------------------------------------------------------------
# Percona XtraDB Cluster configuration
#---------------------------------------------------------------------
listen percona_cluster 0.0.0.0:3306
mode tcp
balance  roundrobin
option httpchk

server d1 d1.fe.infn.it:3306 check port 9200
server d2 d2.fe.infn.it:3306 check port 9200
#---------------------------------------------------------------------
# Admin WEB interface
#---------------------------------------------------------------------
listen admin 0.0.0.0:9999
stats uri /
```

Evidentemente le righe riferite ai nodi del Cluster vanno modificate in accordo con la configurazione che si sta realizzando.

`server <id> <indirizzo_nodo>:3306 check port 9200`

Dove è possibile si raccomanda sempre di usare l'indirizzo IP al posto del-

l'hostname. Il parametro indicato con `<id>` altro non è che il nome con il quale l'interfaccia di amministrazione identificherà la macchina.

A questo punto si rende necessaria l'apertura di due porte sull'host che esegue il bilanciatore: la 3306 per le connessioni dei clients MySQL e la 9999 per l'accesso all'interfaccia WEB di monitoraggio HAProxy.

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 9999 -j ACCEPT
```

Infine è necessario avviare il servizio e renderlo, permanentemente, attivo ai prossimi boot dell'host:

```
chkconfig --level 2345 haproxy on
service haproxy start
```

Lo stato del Cluster, e il lavoro del bilanciatore, possono essere monitorati attraverso l'interfaccia WEB fornita da HAProxy. Sarà sufficiente connettersi all'indirizzo del'host, che ospita il bilanciatore, alla porta 9999.

HAProxy version 1.4.22, released 2012/08/09

Statistics Report for pid 4100

> General process information

pid = 4100 (process #1, nbproc = 1)
uptime = 0d 0h15m59s
system limits: memmax = unlimited; ulimit-n = 8015
maxsock = 8015; **maxconn** = 4000; **maxpipes** = 0
current conns = 2; current pipes = 0/0
Running tasks: 1/4

active UP
active UP, going down
active DOWN, going up
active or backup DOWN
active or backup DOWN for maintenance (MAINT)
backup UP
backup UP, going down
backup DOWN, going up
not checked
Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.4)
- Online manual

percona_cluster

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	3 000	0		0	0	0	0	0					OPEN								
d1	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	15m59s UP	L7OK/200 in 32ms	1	Y	-	0	0	0s	-
d2	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	15m59s UP	L7OK/200 in 31ms	1	Y	-	0	0	0s	-
Backend	0	0		0	0		0	0	3 000	0	0	0	0	0	0		0	0	0	0	15m59s UP		2	2	0		0	0s	

admin

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server									
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				2	2	-	2	2	3 000	6		1 616	41 674	0	0	0					OPEN								
Backend	0	0		0	0		0	0	3 000	0	0	1 616	41 674	0	0		0	0	0	0	15m59s UP		0	0	0		0		

Figura 4.4: L'interfaccia WEB di HAProxy installato su d3.

Da ora in poi tutte le connessioni MySQL dell'applicazione potranno essere fatte all'host che fa girare HAProxy, quello che nell'installazione d'esempio è d3. È importante notare che, da ora in poi, il Cluster vedrà arrivare le connessioni dal bilanciatore; essendo le utenze MySQL associate all'host d'origine è fondamentale definirle/ridefinirle tenendo conto di questo fatto. Per fare questo è necessario connettersi direttamente ad un nodo, qualsiasi, come utente root ed usare la sintassi `GRANT`. Percona Cluster è pensato per

propagare correttamente, ai nodi, le modifiche agli account fatte con gli statement `GRANT`; le modifiche dirette alle tabelle dello schema `mysql` non sono sempre propagate e, quindi, potrebbero avere effetto locale.

A titolo esemplificativo vengono riportati gli statement usati, nell'installazione di prova, per creare un utente MySQL con cui connettersi, attraverso HAProxy, ad un database appositamente creato.

```
CREATE DATABASE clustertest;
GRANT ALL ON clustertest.* TO 'clusteruser'@'<haproxy_host>'
IDENTIFIED BY '<user_password>';
FLUSH PRIVILEGES;
```

Sarà possibile connettersi da qualsiasi macchina specificando al client MySQL i seguenti parametri:
`mysql -u clusteruser -p -h <haproxy_host> clustertest`
Per controllare a quale nodo si è effettivamente connessi si può usare lo statement:
`SHOW VARIABLES LIKE 'hostname';`

4.3 Gestore dischi LVM

Logical Volume Manager è un ulteriore strato software intermedio, nella gestione dei dischi da parte del Sistema Operativo, specificatamente pensato per rendere più flessibile la gestione dei volumi rispetto al normale partizionamento fisico. Oramai incluso di default in molte distribuzioni GNU/Linux, tra cui le el6, può essere già selezionato al momento dell'installazione (tranne che per il filesystem di boot) ed è divenuto una soluzione utilizzatissima per file e database server.

Grazie all'astrazione sui dischi, fatta da LVM, è possibile estendere i filesystem oltre le dimensioni delle singole partizioni fisiche. Il software, che permette di nascondere l'organizzazione dei dispositivi hardware, rende possibile ridimensionare e muovere i Volumi Logici anche senza smontare i filesystem o fermare i processi che li usano. Tutto questo con un overhead davvero minimo per il sistema, sopratutto su architetture moderne. LVM inoltre consente di eseguire degli snapshot dei volumi logici, questa caratteristica risulta utile, tra le altre cose, per eseguire backup minimizzando, ma non annullando, i tempi di inattività dei servizi. Altra caratteristica, in certi ambiti non meno importante, è quella di consentire l'assegnazione di nomi coerenti con la destinazione d'uso ai LogicalVolumes come ai VolumeGroups.

4.3.1 Struttura di LVM

LVM permette di raggruppare più partizioni fisiche, definite Physical Volumes, in un Volume Group. I Volume Group possono essere estesi o ridotti, in pratica vi si possono aggiungere o togliere partizioni fisiche. Ogni Volume Group può contenere uno o più Logical Volumes, anche questi ridimensionabili a piacere, sui quali possono essere costruiti FileSystem, queste entità verranno viste, dall'utente, alla stregua delle partizioni convenzionali.

4.3.2 Snap-Shot LVM

LVM permette di creare snapshot in maniera più efficiente dei tradizionali metodi di backup. Questa efficienza è ottenuta mediante l'utilizzo del metodo COW (copy-no-write). Lo snapshot iniziale di un LV conterrà hardlink agli inode del FileSystem di cui fare il backup. Solo quando qualche file verrà modificato LVM copierà veramente i dati sullo snapshot, che è un LV anch'esso. Questo rende la procedura di creazione di uno snapshot molto veloce, distribuendo la complessità del processo di copia nel tempo, e ottimizza l'occupazione di spazio sul disco.

4.3.3 Conclusioni

Una trattazione completa del funzionamento e dell'uso di LVM esula dagli scopi di questo lavoro di tesi. La documentazione liberamente disponibile in rete è completa ed esaustiva. Tale soluzione è stata citata in questo lavoro solo per darne un'idea delle capacità, al fine di motivarne il, caloroso, consiglio all'adozione, quantomeno per la partizione dati del DBMS.

4.4 Alternative e Motivazioni

Come hanno confermato recenti eventi, la situazione delle installazioni MySQL del progetto LHCbDIRAC è ormai al limite. Lo spazio sui volumi dedicati ai TableSpace comincia a scarseggiare e si teme che i DBMS possono rappresentare un "collo di bottiglia" per le prestazioni dell'intero sistema. Sono state cercate, in diverse direzioni, soluzioni che permettessero un aggiornamento "a caldo" e che, quindi, consentissero di non fermare il sistema attualmente in uso. Date le caratteristiche delle attuali installazioni, è stato impossibile individuare un metodo, di migrazione delle basi di dati, che non contemplasse:

- uno stop, anche minimo, dei servizi MySQL.

- un decadimento eccessivo, e per eccessivo tempo, delle prestazioni del sistema.
- una perdita, anche minima, di dati sensibili.

Si è quindi optato per effettuare una nuova installazione, su cui nuovi Agent e Service andranno a puntare. La vecchia installazione non accetterà più nuovi task ma sarà lasciata a drenare dalle attività in corso. I dati che potessero risultare utili saranno, a mezzo di Agent/Script appositamente progettati, copiati nella nuova base di dati.

4.4.1 Alternative per la Clusterizzazione

Dovendo, la modifica del sistema, risultare il più trasparente possibile all'applicazione, la scelta del software DBMS da poter usare ha dovuto forzatamente ricadere su prodotti MySQL compatibili.

MySQL Cluster

MySQL Cluster è una soluzione certamente più completa, di Percona XtraDB Cluster, e che permette una configurazione molto più ampia e dettagliata. Il maggior vantaggio offerto è quello di poter variare indipendentemente il numero di:

- nodi Client che accettano le connessioni e seguono le Query, di qualsiasi tipo.
- nodi Dati che si occupano della memorizzazione e gestione effettiva dei dati, essi rappresentano la vera componente Clusterizzata
- numero di repliche dei dati da mantenere.

Inoltre MySQL Cluster presenta il vantaggio di un nodo di Management, il quale permette di gestire e monitorare l'intero cluster da un solo punto. Tutta questa flessibilità, ovviamente, introduce una notevole complessità.

Una simile soluzione è stata scartata non per la sua validità tecnica, soprattutto alla luce delle attuali migliorie apportate alla versione 7.3, ma per la sua inadeguatezza alle esigenze del caso preso in considerazione. Risulta infatti chiaro, anche dalla breve descrizione sopra riportata, che MySQL è pensato per realizzare Cluster di considerevoli dimensioni. Per avere, come ottenuto con Percona Cluster, due nodi e due replicazioni, in una installazione MySQL Cluster fatta a regola d'arte, servirebbero, almeno quattro host contro i due della soluzione prescelta:

- due nodi Dati, in grado quindi di formare un node-group con due replicazioni
- due nodi Client
- il nodo di Management, come l'Arbitrator per Percona, non viene contato in quanto può essere installato su una macchina adibita anche ad altri servizi minori.

Inoltre un Cluster MySQL, essendo più configurabile, richiede maggiore attenzione ed esperienza. Adottare una simile soluzione implicherebbe la necessità di formare un DBA [4] al fine di poterla sfruttare a pieno.

È evidente come Percona permetta, nelle realizzazioni di Cluster medio-piccoli, un migliore rapporto tra prestazioni e costi, in termini di: hardware, tempi e personale. MySQL Cluster è stato scartato poiché, pensato per realizzare Cluster di grandi dimensioni, si è dimostrato sovradimensionato per i fini del progetto.

Replicazione MySQL

MySQL mette a disposizione, nativamente, un meccanismo di replicazione asincrona-semisincrona basata sul modello master-slave. In questa modalità di funzionamento esiste unico nodo, detto Master, su cui poter eseguire statement di modifica ai dati. Il master, oltre a gestire le transazioni, si occupa di propagare le modifiche a uno o più Slave. Gli Slave sono nodi su cui le modifiche dei dati avvengono solo per replicazione di quelle del Master. Uno Slave può essere usato, da un Client, solo per eseguire Query di selezione. Esistono due modalità di propagazione:

- **asincrona** non è mai possibile avere la certezza che gli slave possiedano una replica esatta dei dati sul Master.
- **semisincrona** si può essere certi che almeno uno Slave abbia, in ogni momento, la replica aggiornata dei dati.

Come già accennato nei paragrafi precedenti questa tecnica comporta due principali svantaggi:

- Il Master rappresenta, a tutti gli effetti, uno SPOF. La questione può essere risolta solo con tool di terze parti, il che complica configurazione e mantenimento.

[4] DBA - Database Administrator, figura professionale specializzata nella gestione dei DBMS.

- L'applicazione, anche se a vari livelli a seconda della configurazione adottata, deve essere modificata per gestire due tipi di connessioni a MySQL: Sola Lettura e Lettura/Scrittura.

- L'applicazione deve tenere conto del possibile disallineamento fra i dati degli Slave e quelli del Master.

Soprattutto volendo eliminare lo SPOF fornito dal singolo nodo Master, si sarebbe resa necessaria una configurazione di pari, o maggiore, complessità, rispetto alla soluzione proposta, ottenendo minore trasparenza per l'applicazione.

4.4.2 Vantaggi

Il vantaggio fin da subito palese è rappresentato dalla replica dei dati. Un sistema che garantisce la persistenza dei dati oltre il guasto fisico di un disco è, rispetto alla situazione attuale, un vistoso miglioramento.

Avere un sistema formato da due nodi garantisce la continuazione del servizio anche in caso di guasto, ad uno dei due, si ottiene quindi una situazione di Alta Affidabilità. Inoltre, questa caratteristica, può essere sfruttata per aggiornare sistema e DBMS. Si è ampiamente visto, durante il lavoro svolto, come MySQL e Fork stiano introducendo migliorie, su tutti i fronti, ad ogni nuova versione. È stato altresì dimostrato come l'effettiva capacità, da parte del Sistema Operativo, di sfruttare l'hardware influenzi le prestazioni del DBMS. Gli aggiornamenti, inoltre, garantiscono la risoluzione di bug funzionali e di sicurezza; elementi di fondamentale importanza per macchine perennemente connesse alla rete e sottoposte a carico.

Grazie alle caratteristiche di LVM, un sistema così configurato permetterebbe di aggiungere spazio disco al prezzo di un breve arresto dell'host, il tempo fisicamente necessario a collegare il nuovo dispositivo, o di eseguire uno snapshot semplicemente fermando il demone, su una singola macchina, per pochi minuti.

HAProxy fornisce la massima trasparenza verso l'applicazione costituendo, al contempo, un bilanciatore di carico intelligente che monitora i servizi su cui splitta le connessioni. Lo SPOF rappresentato, nella nostra installazione di test, da HAProxy può essere eliminato con l'adozione di un fail-over cluster (ad esempio Heartbeat e Pacemaker). La trasparenza fornita da HAproxy consente di non modificare l'applicazione [5] , questo permette di continua-

[5] L'unica modifica caldamente consigliata, come descritto precedentemente, è il controllo degli errori forniti all'eventuale fallimento dello statement COMMIT. Tuttavia questa modifica mantiene il codice perfettamente compatibile con le installazioni stand-alone.

re a distribuire DIRAC con la sua funzione di risoluzione automatica delle dipendenze.

I software, tutti OpenSource, e la configurazione adottata, sono stati scelti per rispondere al meglio alle esigenze specifiche di LHCbDIRAC. I responsabili del progetto hanno sempre curato la struttura IT in proprio, gestendo internamente gli host dove i servizi girano e senza appoggiarsi ad entità esterne. Tenendo anche in considerazione l'evidenza che, fino ad oggi, delle semplici installazioni stand-alone hanno svolto dignitosamente il compito. Si è prestata, quindi, particolare attenzione a fornire una soluzione facilmente gestibile e che richieda pochi sforzi in investimenti hardware e di formazione, in grado comunque di fornire tutte le garanzie che un progetto della portata di LHCb richiede.

Conclusioni

Il lavoro di tesi svolto ha riguardato lo studio e l'analisi del database di back-end per il progetto DIRAC/LHCbDIRAC. Si è provveduto successivamente alla formulazione di proposte per la soluzione ai problemi riscontrati. LHCbDIRAC è il sistema per la gestione dei grid job di Produzione, Merge, Ricostruzione Eventi e Analisi dell'esperimento LHCb del CERN. Un'installazione LHCbDIRAC prevede per il suo funzionamento varie istanze di RDMBS MySQL. I requirements soddisfatti sono stati molteplici. Prendendo in esame il caso particolare dello schema relazionale *TransformationDB* si sono date le linee guida per la conversione di tutto il database di back-end all'engine InnoDB. Si è implementata un'analisi delle prestazioni del DBMS e sono state proposte soluzioni genericamente valide alle criticità riscontrate.

Il lavoro svolto ha presentato molte difficoltà di varia natura. Profilare un database e formulare delle possibili soluzioni è un processo complesso e, per la natura stessa dei DBMS e dell'uso che ne viene fatto, mai uguale a se stesso. Un database è il più delle volte utilizzato da un'applicazione. Introdurre cambiamenti alla base di dati senza coinvolgere l'applicazione rischia di risultare uno spreco di risorse. L'esperienza fatta ha confermato, nel sottoscritto, l'idea che un'applicativo, qualora faccia un uso anche minimo di un DBMS, deve vedere partire la propria progettazione dalla corretta realizzazione dei modelli relazionali della base di dati. Proprio perché rappresenta, di per se, un sistema complesso in cui diversi fattori influiscono sul risultato finale, un RDBMS, qualora mal impiegato, rischia di inficiare le prestazioni dell'applicazione che lo sfrutta.

Non meno importante è la progettazione del deployment delle istanze del DMBS che si intendono usare. Come è stato possibile verificare nel corso del lavoro svolto, una perfetta trasparenza dell'architettura usata dal servizio DBMS è pressoché, impossibile. Sarebbe quindi importante, già in fase di progettazione, avere un'idea del carico che potenzialmente dovrà sopportare il sistema, in modo da poter adattare l'applicazione alle variazioni che la soluzione prescelta comporta. L'esperienza di LHCbDIRAC dimostra l'importanza di considerare il sistema nella sua interezza.

È stata realizzata, secondo le richieste dei responsabili del progetto, la migrazione dello schema *TransformationDB* all'engine InnoDB, mantenendolo il più possibile compatibile con il precedente, e sono state apportate le minime modifiche al codice necessarie ad adattare l'applicazione al cambiamento.

È stata formulata una soluzione per l'*High Availability* del DBMS adottato, caratteristica imprescindibile data l'importanza della continuità del servizio e dei dati presenti nel database. Tale soluzione è stata implementata tenendo conto delle caratteristiche dell'applicazione e del carico da essa prodotto, oltre che delle esigenze e delle caratteristiche proprie dell'esperimento LHCb. Particolare attenzione è stata posta nel reperimento di una soluzione scalabile, facilmente implementabile e mantenibile oltre che, il più possibile, trasparente. La scelta di tutte le componenti del sistema e della relativa configurazione è stata motivata e documentata. I test condotti, sull'installazione locale alla sezione dell'INFN di Ferrara, hanno dato riscontri positivi.

Bibliografia

[1] S. Paterson, University Of Glasgow *DIRAC INFRASTRUCTURE FOR DISTRIBUTED ANALYSIS*

[2] A. Tsaregorodtsev, LHCb Computing Group *Workload Management with Pilot Agents in DIRAC*

[3] Contributori di Wikipedia, 'Grid computing', Wikipedia, L'enciclopedia libera, 22 maggio 2013, 17:33 UTC, `http://it.wikipedia.org/w/index.php?title=Grid_computing&oldid=58980448` [in data 6 luglio 2013]

[4] Contributori di Wikipedia, 'Berkeley Open Infrastructure for Network Computing', Wikipedia, L'enciclopedia libera, 27 aprile 2013, 16:10 UTC, `http://it.wikipedia.org/w/index.php?title=Berkeley_Open_Infrastructure_for_Network_Computing&oldid=58487737` [in data 6 luglio 2013]

[5] Wikipedia contributors, 'GLite', Wikipedia, The Free Encyclopedia, 12 May 2013, 04:25 UTC, `http://en.wikipedia.org/w/index.php?title=GLite&oldid=554687835` [accessed 6 July 2013]

[6] DIRAC contributors, 'DIRAC Grid', `http://diracgrid.org/` [accessed 6 July 2013]

[7] Oracle-MySQL contributors, 'MySQL', `http://dev.mysql.com/doc/refman/5.6/en/replication-semisync.html` `http://dev.mysql.com/doc/refman/5.6/en/innodb-multiple-tablespaces.html` [accessed 6 July 2013]

[8] Percona-Percona contributors, 'Percona XtraDB Cluster' `http://www.percona.com/software/percona-xtradb-cluster` `http://www.mysqlperformanceblog.com/2012/07/25/percona-xtradb-cluster-failure-scenarios-with-only-2-nodes/`

http://www.percona.com/doc/percona-xtradb-cluster/features/multimaster-replication.html [accessed 6 July 2013]

[9] Percona-Percona contributors, 'Percona ToolKit' http://www.percona.com/software/percona-toolkit http://www.percona.com/doc/percona-toolkit/2.2/pt-query-digest.html [accessed 6 July 2013]

[10] Galera-Galera contributors, 'Galera Arbitrator' http://www.codership.com/wiki/doku.php?id=galera_arbitrator [accessed 6 July 2013]

Elenco delle figure