Scuola di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

IMPLEMENTAZIONE E TESTING DI UN CLUSTER DI DATABASE SU UNA RETE DI PARI

IMPLEMENTATION AND TESTING OF A PEER NETWORK DATABASE CLUSTER

LINDA LUCIANO

Relatore: *Lorenzo Bettini*
Correlatore: *Correlatore*

Anno Accademico 2016-2017

INDICE

ELENCO DELLE FIGURE

"Inserire citazione"
— Inserire autore citazione

1

DEFINIZIONE DEL PROBLEMA

(replicazione dei dati - introduzione)

1.1.1 *Cluster di database*

Un cluster è una raccolta di componenti che garantisce scalabità e disponibilità distribuendone i costi. Un cluster di database (SQL usa il termine cluster di catalogo) è una collezione di database gestiti da una singola istanza di un server database in esecuzione. Un'istanza è la raccolta di memoria e processi che interagiscono con un database, cioè l'insieme di file fisici che effettivamente memorizzano i dati.[1] A tal fine, è possibile creare un cluster di database per applicazioni enterprise high-end, memorizzando e elaborando informazioni sui nodi.
L'architettura per un cluster di database è distinta da come le responsabilità dei dati sono condivise tra i nodi di calcolo.

Seguono due dei vantaggi principarli offerti dal clustering, specialmente in un ambiente di database di alto volume:

- *Fault tolerance* (tolleranza di guasti): in caso di guasto del singolo server, il cluster offre un'alternativa, poiché esiste più di un server o istanza per gli utenti a cui connettersi.
- *Load balancing* (bilanciamento del carico): la funzionalità di clustering è generalmente impostata per consentire agli utenti di essere assegnati automaticamente al server con il minor carico.[1]

Ci sono differenti tipi di architetture clustering che si diversificano da come vengono memorizzati i dati e allocate le risorse. La prima modalità di clustering è conosciuta come architettura "*shared-nothing*" (SN). È un'architettura di elaborazione distribuita in cui ogni nodo/server è totalmente indipendente e autonomo, pertanto nessuno dei nodi condivide memoria o archiviazione del disco. Più generalmente, non esiste un unico punto di

contesa nel sistema.[3] Il partizionamento è tale che ogni nodo possiede un sottoinsieme dei dati, ovvero ogni nodo ha accesso esclusivo su quel particolare sottoinsieme. [2]))

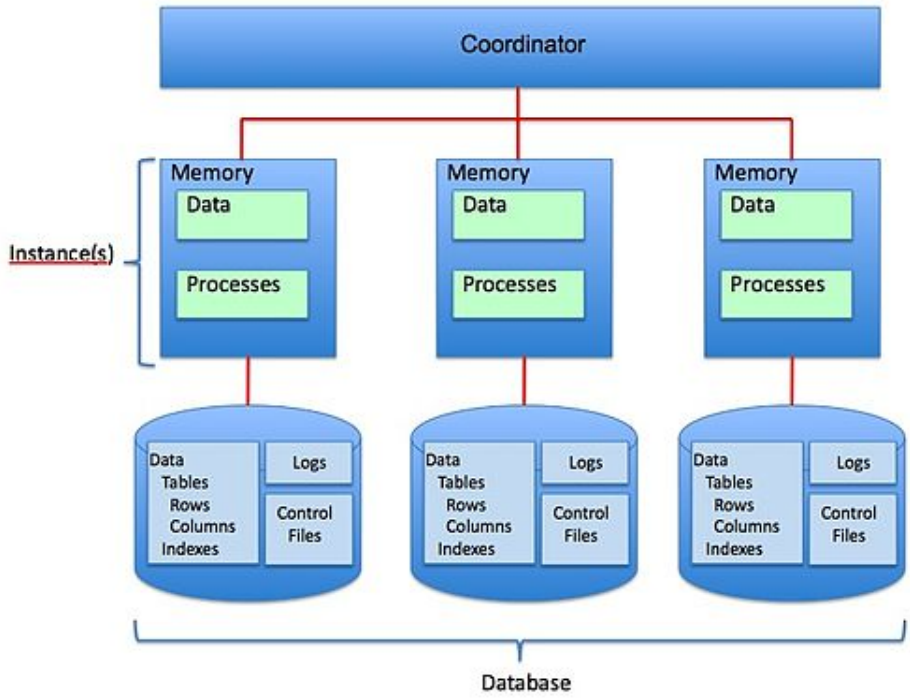

Figura 1: Architecture Shared Nothing [4]

I vantaggi dell'architettura SN rispetto a un'entità centrale che controlla la rete (un'architettura basata su controller) riguarda l'eliminazione di qualsiasi singolo punto di guasto, consentendo funzionalità di auto-riparazione (*self-healing*) e fornendo un vantaggio nell'offrire aggiornamenti non distruttivi.[5] *Shared-nothing* è anche noto come "*database sharding*". In generale, un sistema SN divide i suoi dati in vari nodi su database diversi o può richiedere a ciascun nodo di mantenere la propria copia dei dati dell'applicazione utilizzando un qualche tipo di protocollo di coordinamento.[3]

Si oppone a quest'ultima, l'architettura nota come "*shared-disk*" (disco condiviso), in cui tutti i dati vengono memorizzati centralmente in un unico disco e sono accessibili da tutti i nodi di cluster.[4] In questo tipo di struttura quindi più istanze di database vengono raggruppate in un singolo database sul disco. Nei sistemi di dischi condivisi, i blocchi (o

pagine) di dati su disco possono avere un solo proprietario.((L'architettura *shared-disk* è un esempio di *Synchronous multi-master*, ovvero ogni istanza del database può scrivere (cioè è un master) in modo sincrono.[4] **DA CHIEDERE E DA CONTROLLARE ULTIME FRASI. GUARDA COMMENTO per il sito**

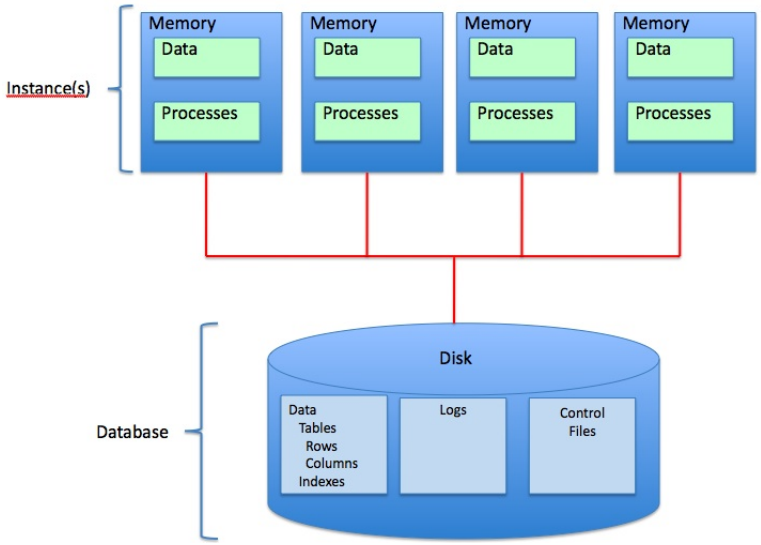

Figura 2: Architecture Shared Disk - Shared Everything [4]

In un'architettura SD, grandi reti di computer possono operare su un singolo set di dati senza la necessità di replicare o bloccare quel set di dati.[4] *Shared disk* ha due vantaggi:

- ogni processore ha la propria memoria,
- il bus di memoria non è un collo di bottiglia (contrariamente all'architettura "*shared-everything*"),
- il sistema offre un modo semplice per fornire un certo grado di tolleranza agli errori.

La distinzione tra i due tipi è diventata confusa di recente con l'introduzione di distribuzione della cache. In questa configurazione, i dati sono ancora gestiti centralmente, ma controllati da un potente "server virtuale" composto da molti server che lavorano insieme come uno.[2]

1.1.2 *Rete di pari*

Peer-to-peer networking (P2P) è un modello di comunicazione decentralizzato in cui ciascuna parte ha la stessa responsabilità per l'elaborazione dei dati e ciascuna parte può avviare una sessione di comunicazione. A differenza del modello *client/server*, in cui il client effettua una richiesta di servizio e il server soddisfa la richiesta, il modello di rete P2P, noto anche come *peer networking*, consente a ciascun nodo di funzionare sia come client che come server.[6]
Quando una rete P2P viene stabilita su Internet, è possibile utilizzare un server centrale per indicizzare i file oppure stabilire una rete distribuita in cui la condivisione dei file viene suddivisa tra tutti gli utenti della rete che memorizzano un determinato file. Le dimensioni della rete e i file disponibili consentono di condividere enormi quantità di dati.
Le prime reti P2P come Napster utilizzavano il software client e un server centrale, mentre reti successive come Kazaa e BitTorrent eliminavano il server centrale e dividevano i compiti di condivisione tra più nodi per liberare la larghezza di banda.

Seguono i vantaggi di una rete *peer-to-peer*:

- se un dispositivo collegato interrompe la connessione, il servizio non termina a differenza del modello *client-server*,
- è possibile configurare i computer in gruppi di lavoro *peer-to-peer* per consentire la condivisione di file e altre risorse su tutti i dispositivi. *Peer networking* consente di condividere facilmente i dati in entrambe le direzioni, sia per i *download* sul computer che per gli *upload* dal computer,
- su Internet, le reti peer-to-peer gestiscono un volume elevato di traffico di condivisione file distribuendo il carico su più computer. Poiché non si basano esclusivamente sui server centrali, le reti P2P possono scalare meglio e sono più resistenti delle reti *client-server* in caso di guasti o colli di bottiglia del traffico,
- Le reti *peer-to-peer* sono relativamente facili da espandere. Con l'aumentare del numero di dispositivi nella rete, aumenta la potenza della rete P2P, poiché ogni computer aggiuntivo è disponibile per l'elaborazione dei dati.[7]

Le reti *peer-to-peer* sono vulnerabili agli attacchi di sicurezza. Poiché ogni dispositivo partecipa al traffico di routing attraverso la rete, gli

hacker possono facilmente lanciare attacchi *denial of service*. Il software P2P funge da server e client, il che rende le *peer networking* più vulnerabili agli attacchi remoti rispetto alle reti client-server. I dati corrotti possono essere condivisi su reti P2P modificando i file già presenti in rete per introdurre codice dannoso.[7]

1.1.3 *Sistemi di ridondanza disco (RAID)*

RAID, acronimo di *redundant array of independent disks*, insieme ridondante di dischi indipendenti (originariamente *redundant array of inexpensive disks*), è una tecnologia che permette di memorizzare dati su più dischi rigidi in un computer (o collegati ad esso) in modo da garantire una gestione sicura dei dati[8]. I dispositivi RAID sono convenienti per sistemi che abbiano necessità di grandi quantità di dati continuamente disponibili.

Il RAID, con modalità differenti a seconda del tipo di configurazione, trae vantaggio dai principi di ridondanza dei dati e di parallelismo in modo da ottenere:

- incrementi di prestazioni (in lettura/scrittura);
- aumenti nella capacità di memorizzazione disponibile;
- miglioramenti nella tolleranza ai guasti, ne segue migliore affidabilità[10]. Il RAID rende il sistema resiliente alla perdita di uno o più hard disk, permettendo di sostituirli senza l'interruzione del servizio.

I volumi RAID vengono percepiti dal sistema operativo come una singola unità, indipendentemente dal numero di componenti che li costituiscono.

Il RAID funziona mettendo i dati su più dischi e consentendo operazioni di input/output (I/O) di sovrapporsi in modo equilibrato. Poiché l'utilizzo di più dischi aumenta il tempo medio tra i guasti, memorizzare i dati ridondantemente aumenta la tolleranza agli errori.

I dati vengono suddivisi in "*stripes*", ovvero in sezioni di stessa lunghezza, detta l'unità del sezionamento e scritti su differenti dischi. Quando si richiede una lettura di dimensione superiore all'unità di sezionamento, diverse implementazioni di diversi sistemi RAID distribuiscono l'operazione su più dischi in parallelo, aumentando le prestazioni. Ad esempio, se abbiamo sezioni da 1 bit e un array di D dischi, le sequenze di dati lunghe almeno D bit sfruttano tutti i dischi. **preso tutto da wikipedia**

RAID hardware e software

Il RAID può essere implementato sia con hardware dedicato che con software specifico.

Nel primo caso si tratta di unità di controllo che gestiscono tutto autonomamente, facendo in modo che il sistema operativo veda un disco normale. Nel secondo caso, è il sistema operativo che associa i dischi e li gestisce usando una forma di ridondanza attraverso un normale controller (ATA, SCSI, Fibre Channel o altro).

Le unità di controllo RAID sono più costose di quelle normali; tuttavia, se non si creano altri tipi di problemi, hanno il vantaggio di non creare difficoltà al sistema operativo.

Controllore RAID

Un controller RAID è un dispositivo hardware o un programma software utilizzato per gestire unità disco fisso (HDD) o unità SSD (*Solid State Drive*, SSD) in un computer o un array di archiviazione in modo da funzionare come unità logica.

Un controller offre un livello di astrazione tra un sistema operativo e le unità fisiche. Un controller RAID presenta gruppi a applicazioni e sistemi operativi come unità logiche per le quali è possibile definire schemi di protezione dei dati. Poiché il controller ha la possibilità di accedere a più copie di dati su più dispositivi fisici, ha la capacità di migliorare le prestazioni e proteggere i dati in caso di crash di sistema.

Nel RAID hardware, un controller fisico viene utilizzato per gestire l'array RAID. Il controller può assumere la forma di una scheda PCI o PCI Express (PCIe), progettata per supportare un formato di unità specifico come SATA o SCSI (alcuni controller RAID possono anche essere integrati con la scheda madre.)

Un controller RAID può anche essere solo software, utilizzando le risorse hardware del sistema host. Il RAID basata su software generalmente fornisce funzionalità simili a RAID *hardware-based*, ma la sua prestazione è tipicamente inferiore a quella delle versioni hardware.[13]

Livelli RAID

La caratteristica fondamentale che identifica una configurazione RAID è, come citato in precedenza, l'array, che rappresenta il tipo di collegamento logico che c'è tra i vari dischi. Con tale criterio viene determinato il livello RAID, ovvero la configurazione della tipologia di RAID e stabilito il

numero minimo di hard disk che sono necessari per attivarlo. A seconda del livello RAID sono implementate diverse caratteristiche operative per ottenere maggiori prestazioni o una maggiore sicurezza dei propri dati oppure entrambe le condizioni.
Si distinguono sei livelli, da `0` a `5`. Questo sistema numerato consente di differenziare le versioni e di scegliere come diffondere i dati attraverso l'array ed è stato suddiviso in tre categorie: livelli RAID standard, nidificati e non standard[8] (segue la descrizione dei livelli standard e qualche nidificato).

LIVELLI RAID STANDARD

- `RAID 0`: livello privo di ridondanza. Si occupa di unire due o più dischi, all'interno dei quali i dati vengono suddivisi equamente (tramite striping o sezionamento), in modo da bilanciare anche il carico di operazioni di lettura e scrittura che li riguardano. Livello che consente di realizzare un disco virtuale di grandi dimensioni, più efficiente, ma la rottura di uno dei dischi porta alla perdita di tutti i dati.
`RAID 0` è noto anche con il nome di *block striping*.**bibliografia commentata**

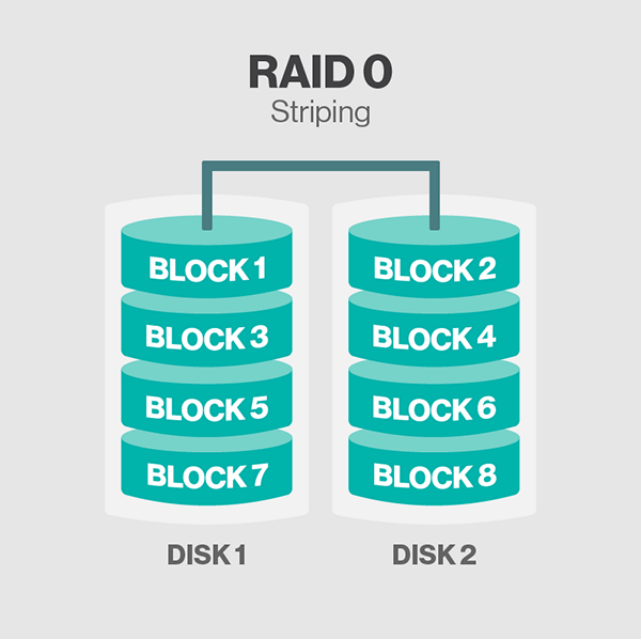

Figura 3: Sezionamento senza ridondanza - Questa configurazione ha sezionamento, ma nessuna ridondanza dei dati. Offre le migliori prestazioni, ma nessuna tolleranza agli errori.[8]

- `RAID 1`: livello che si occupa di unire assieme due o più dischi riproducendo fedelmente gli stessi dati. Questa configurazione mantiene quindi almeno una copia esatta di tutti i dati, detta "*mirror*". In questo caso, la rottura di un disco non pregiudica l'utilizzo dei dati che sono disponibili nel disco o nei dischi rimanenti. Più precisamente, l'affidabilità aumenta linearmente al numero di dischi presenti: un sistema con `N` dischi è in grado di resistere alla rottura di `N-1` componenti.
  La lettura delle prestazioni è migliorata poiché entrambi i dischi possono essere letti contemporaneamente. La scrittura delle prestazioni è la stessa di quella per il singolo disco.[8]
  `RAID 1` è conosciuto anche come *disk mirroring*.

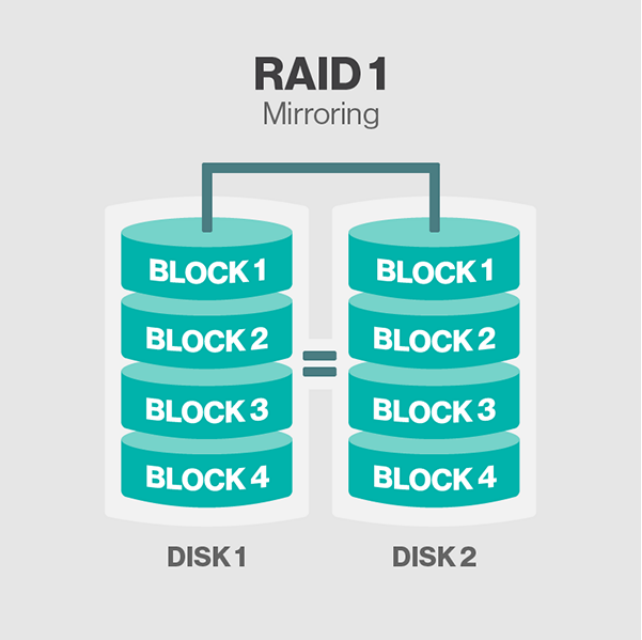

Figura 4: Replicazione - Questa configurazione è costituita da almeno due unità che duplicano la memorizzazione dei dati. Non c'è sezionamento. [8]

- `RAID 2`: livello che divide i dati al livello di bit (invece che di blocco) e usa un *codice di Hamming* per la correzione d'errore che permette di correggere errori su singoli bit e di rilevare errori doppi. Questi dischi sono sincronizzati dal controllore, in modo tale che la testina di ciascun disco sia nella stessa posizione in ogni disco.[10]

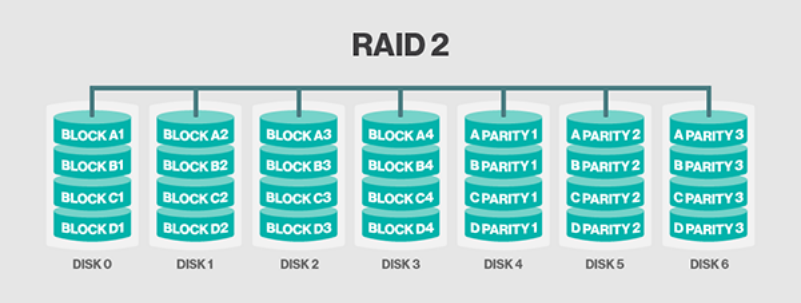

Figura 5: Sezionamento a livello di bit - Questa configurazione ha alcuni dischi che memorizzano le informazioni di errore di verifica e correzione (ECC). [8]

- `RAID 3`: livello che si occupa di unire assieme almeno tre o più dischi, all'interno dei quali i dati vengono suddivisi equamente, in modo da bilanciare anche il carico di operazioni di lettura e scrittura che li riguardano. Dedicano uno di questi dischi al contenimento di un sistema di codici di controllo, che permettono di ricostruire i dati nel caso in cui uno degli altri dischi si rompa. Le informazioni ECC vengono utilizzate per rilevare gli errori. Il recupero dei dati viene effettuato calcolando l'esclusiva `OR` (`XOR`) delle informazioni registrate sulle altre unità.[8]

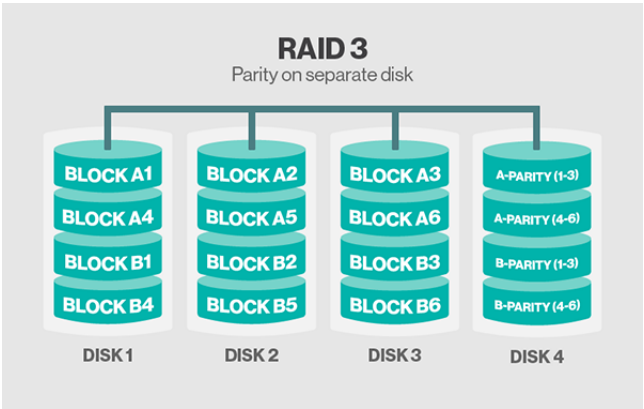

Figura 6: Sezionamento a livello di byte con disco di parità - Questa configurazione utilizza la rigatura e dedica un'unità a memorizzare informazioni di parità. [8]

- `RAID 4`: livello simile al livello tre, con la differenza che i dati vengono distribuiti in modo più efficiente tra i dischi, ma rimane compito di un disco separato il sistema di codici di controllo che permette la ricostruzione dei dati, chiamati "blocchi di parità".
  Questo livello utilizza grandi sezionamenti, il che significa che è possibile leggere i record da un'unica unità.[8]

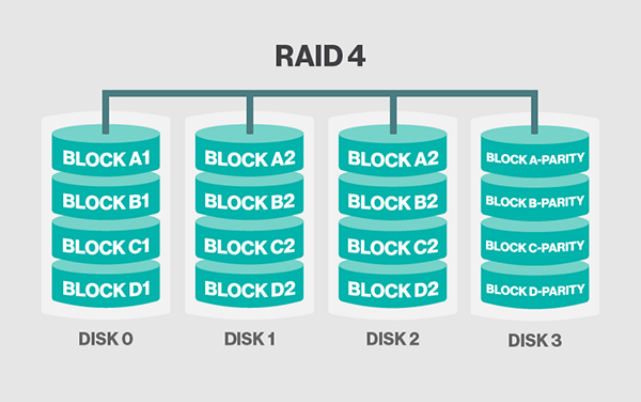

Figura 7: Sezionamento a livello di blocco con disco di parità [8]

- `RAID 5`: livello basato su livello di blocco con parità che risiedono su ciascuna unità. L'architettura dell'array consente alle operazioni di lettura e scrittura di coprire più unità. Ciò determina prestazioni migliori di quelle di un'unità singola, ma non altrettanto elevata di quella di un array `RAID 0`.[8]

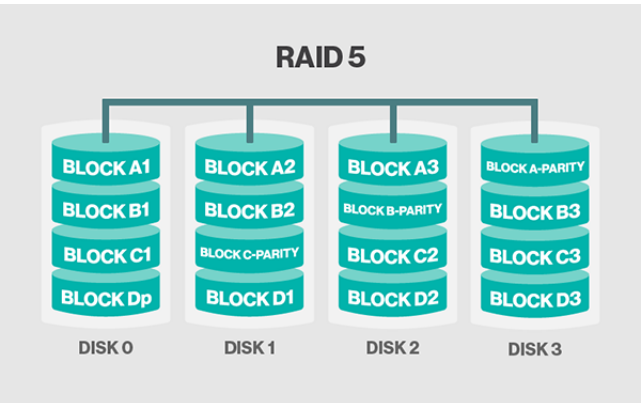

Figura 8: Sezionamento a livello di blocco con parità distribuita [8]

- `RAID 6`: livello simile a `RAID 5`, ma include un secondo schema di parità distribuito attraverso le unità nell'array. L'utilizzo di una parità aggiuntiva consente all'array di continuare a funzionare anche se due dischi non funzionano contemporaneamente. Tuttavia, questa protezione supplementare è più costosa. Le matrici RAID `6` hanno un costo superiore a gigabyte (GB) e spesso hanno prestazioni di scrittura più lente degli array `RAID 5`.[8]

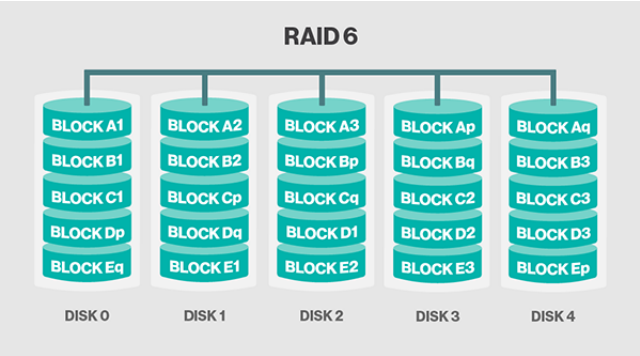

Figura 9: Sezionamento a livello di blocco con doppia parità distribuita [8]

LIVELLI RAID NIDIFICATI ANNIDATI I livelli Annidati sono dei tipi di livelli più complessi ottenuti dalla combinazione di alcuni livelli RAID Standard. Esempi classici sono le configurazioni RAID `0+1` o `10`.

1.1.4 *Codice di correzione errore (Erasure Coding)*

La codifica di cancellazione, noto come *Erasure Coding* (EC) è un metodo di protezione dei dati, i quali vengono suddivisi in frammenti, estesi e codificati con pezzi di dati ridondanti e memorizzati su un insieme di posizioni o supporti di memorizzazione diversi.

L'obiettivo della codifica di cancellazione è quello di consentire di ricostruire i dati che vengono danneggiati utilizzando le informazioni sui dati memorizzati altrove nell'array. Lo svantaggio della codifica di cancellazione è che può essere più intenso della CPU e che può tradursi in una maggiore latenza.[11]

La codifica di cancellazione è utile con la presenza di grandi quantità di dati e tutte le applicazioni o sistemi che devono tollerare i guasti, come sistemi di array a dischi, griglie di dati, applicazioni di archiviazione distribuite. Un caso comune di utilizzo corrente per la codifica di cancellazione è in un sistema object-based cloud storage[11].

Come funziona

La codifica di cancellazione crea una funzione matematica per descrivere un insieme di numeri in modo che possano essere controllati per l'accuratezza e recuperati in caso di perdita. Questo è il concetto fondamentale dei metodi di codifica di cancellazione, implementati più frequentemente utilizzando i codici *Reed-Solomon*[11].

In termini matematici, la protezione offerta dalla codifica di cancellazione può essere rappresentata in forma semplice dalla seguente equazione:

$$n = k + m$$

dove:

- la variabile `k` è la quantità originale di dati o simboli
- la variabile `m` indica i simboli aggiuntivi o ridondanti che vengono aggiunti per fornire protezione dai guasti
- la variabile `n` è il numero totale di simboli creati dopo il processo di codifica di cancellazione[11]
- la variabile `r`, chiamata velocità di codice, è definita nel seguente modo:

$$r = \sqrt{\frac{k}{n}}$$

Ad esempio, in una configurazione `10` di `16`, o EC `10/16`, sei simboli supplementari (`m`) saranno aggiunti ai `10` simboli di base (`k`). I `16` frammenti di dati (`n`) saranno diffusi su `16` unità, nodi o posizioni geografiche. Il file originale potrebbe essere ricostruito da `10` frammenti verificati.[11]

I codici di cancellazione, noti anche come codici di correzione degli errori di avanzamento (FEC), sono stati sviluppati più di 50 anni fa. Da quel momento sono emersi diversi tipi. In uno dei tipi più comuni, *Reed-Solomon*, i dati possono essere ricostruiti utilizzando qualsiasi combinazione di simboli `k` o pezzi di dati, anche se i simboli `m` sono persi o non sono disponibili. Ad esempio, in EC `10/16`, sei unità, nodi o posizioni geografiche potrebbero essere persi o non disponibili e il file originale sarà ancora recuperabile.[11]

1.2 HARDWARE UTILIZZATO PER GLI ESPERIMENTI

Inserire foto hardware con descrizione

1.3 SOFTWARE UTILIZZATO PER GLI ESPERIMENTI

1.3.1 *PosgreSQL*

PostgreSQL è un potente sistema *Open Source* di database relazionale (DBMS, *Database Management System*) cioè è un sistema software progettato per consentire la creazione e manipolazione efficiente di database, ovvero di collezioni di dati strutturati. Ha più di 15 anni di sviluppo attivo e un'architettura collaudata che ha guadagnato una notevole reputazione per l'affidabilità, l'integrità dei dati e la correttezza.

PostgreSQL è un sistema di gestione dei database relazionale (*Object-Relational*) e ha il compito di gestire le operazioni di archiviazione e di salvaguardia dell'integrità dei dati allocati.
Funziona su tutti i principali sistemi operativi, tra cui Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), e Windows.[12]

Seguono le caratteristiche principali di PostgreSQL; le funzionalità introdotte nella recente versione 9.1 :

- elevata aderenza agli standard SQL;
- architettura client-server con una gamma completa di driver e di client;
- progettato in modo altamente concorrente, evitando che i processi in scrittura blocchino i processi in lettura;
- altamente configurabile ed estendibile, consentendo svariati tipi di applicazioni;
- elevate scalabilità e prestazioni, unite a un ampio spettro di possibilità per tarare la configurazione;
- sofisticato ottimizzatore delle query, adeguato per la business intelligence;
- supporto completo per Java, Python, Perl, PHP e molti altri linguaggi, sia per le procedure interne al server di database che per l'accesso da parte di client;
- elevata affidabilità, con una vasta serie di caratteristiche per durabilità e alta disponibilità;

- tipi di dati avanzati, come ad esempio GIS, full text search, e molti altri;
- internazionalizzazione, codifiche multibyte e collation. [13]

PostgreSQL è progettato per essere estensibile, poiché è possibile definire i propri tipi di dati, tipi di indici e lingue funzionali. Offre un ricco set di strumenti per gli sviluppatori in modo da gestire l'accesso simultaneo ai dati. Inoltre vi è la possibilità di ottimizzarlo per soddisfare le proprie esigenze, tramite uno sviluppo di un plugin personalizzato.

Un database di classe enterprise, PostgreSQL vanta funzionalità sofisticate come il controllo della concorrenza multiversione, il ripristino in tempo reale (*point in time recovery*), *tablespaces*, replica asincrona, transazioni nidificate (*savepoints*), backup in linea/a caldo, un sofisticato query planner/optimizer, ed il *write ahead logging* per una maggiore tolleranza ai guasti.
È estremamente scalabile sia nella quantità pura di dati che può gestire sia nel numero di utenti concorrenti che può ospitare. Esistono sistemi Active PostgreSQL in ambienti di produzione che gestiscono oltre 4 terabyte di dati.[12]

Alcuni limiti generali di PostgreSQL sono inclusi nei punti riportati di seguito:

- Dimensione massima del database: Illimitato
- Dimensione massima Tabella: `32 TB`
- Dimensione massima Riga: `1,6 TB`
- Dimensione Massima campo: `1 GB`
- Numero Massimo Righe per tabella: Illimitato
- Numero Massimo colonne per la tabella : `250 - 1600` a seconda dei tipi di dati delle colonne
- Numero Massimo indici per la tabella: Illimitato[13]

MVCC (*Multi-Version Concurrency Control*) è una tecnica avanzata per migliorare le prestazioni del database in un ambiente multiutente, mantenendo la coerenza dei dati. A differenza della maggior parte degli altri sistemi di database che utilizzano i *locks* per il controllo della concorrenza, Postgres mantiene la coerenza dei dati utilizzando questo modello. Ciò

significa che durante l'interrogazione di un database ogni transazione vede un'istantanea di dati (una versione del database), indipendentemente dallo stato corrente dei dati sottostanti. Questo protegge la transazione dalla visualizzazione di dati incoerenti che potrebbero essere causati da altri eventuali aggiornamenti simultanei delle transazioni sulle stesse righe di dati, fornendo l'isolamento della transazione per ciascuna sessione del database.

La differenza principale tra i modelli multiversione e di blocco è che nei blocchi MVCC acquisiti per la query (lettura) i dati non sono in conflitto con i blocchi acquisiti per la scrittura dei dati e quindi la lettura non blocca mai la scrittura e la scrittura non blocca mai la lettura.[12]

Caratteristiche PostgreSQL

PostgreSQL ha il pieno supporto per le subquery (incluse sottotitoli nella clausola FROM), i livelli di isolamento delle transazioni di lettura e serializzabili. E mentre PostgreSQL ha un catalogo di sistema completamente relazionale che supporta più schemi per database, il suo catalogo è accessibile anche attraverso lo schema di informazioni definito nello standard SQL.

Le funzionalità di integrità dei dati includono le chiavi primarie (combinate), le chiavi estranee con aggiornamenti e cancellazioni a cascata, i controlli dei vincoli, i vincoli unici e non i vincoli nullo.

Ha anche una serie di estensioni e funzionalità avanzate. PostgreSQL supporta indici composti, unici, parziali e funzionali che possono utilizzare qualsiasi metodo di archiviazione B-tree, R-tree, hash o GiST.

Altre funzionalità avanzate includono l'ereditarietà delle tabelle, i sistemi di regole e gli eventi del database. L'ereditarietÃ di tabella mette un orientamento orientato all'oggetto sulla creazione della tabella, consentendo ai progettisti di database di derivare nuove tabelle da altre tabelle, trattandole come classi di base. Ancora meglio, PostgreSQL supporta sia l'ereditarietÃ singola che quella multipla in questo modo.

Il sistema di regole, chiamato anche il sistema di riscrittura delle query, consente al progettista del database di creare regole che identificano operazioni specifiche per una determinata tabella o vista e le trasformano dinamicamente in operazioni alternative quando vengono elaborate.

Il sistema eventi è un sistema di comunicazione interprocesso in cui i messaggi e gli eventi possono essere trasmessi tra i client utilizzando i comandi `LISTEN` e `NOTIFY`, consentendo sia la semplice comunicazione peer to peer sia un coordinamento avanzato sugli eventi del database.

Poiché le notifiche possono essere rilasciate da trigger e stored procedure, i client PostgreSQL possono monitorare eventi di database come gli aggiornamenti, gli `insert` o le eliminazioni di tabella quando vengono eseguiti.[12]

Elevata personalizzazione

PostgreSQL esegue procedure memorizzate in più di una dozzina di linguaggi di programmazione, tra cui Java, Perl, Python, Ruby, Tcl, C / C ++ e il proprio PL / pgSQL, simile a PL / SQL di Oracle. I trigger e le stored procedure possono essere scritti in C e caricati nel database come libreria, permettendo una grande flessibilità nell'estensione delle sue funzionalità. Allo stesso modo, PostgreSQL include un *framework* che consente agli sviluppatori di definire e creare i propri tipi di dati personalizzati insieme a funzioni di supporto e operatori che definiscono il loro comportamento.

Il codice sorgente di PostgreSQL è disponibile sotto una licenza libera open source: la licenza PostgreSQL. Questa licenza dà la libertà di utilizzare, modificare e distribuire PostgreSQL in qualsiasi forma, sorgente aperta o chiusa.[12]

Non esiste un unico formato per tutti i software di replica. È necessario capire le proprie esigenze e come si adattino diversi approcci. Ad esempio, ecco due estremi nello spazio del problema di replica: Hai alcuni server collegati a una rete locale che vuoi mantenere sempre la corrente per scopi di failover e di bilanciamento del carico. Qui si considererebbero soluzioni sincrone, desiderose e quindi prive di conflitti. I tuoi utenti prendono una copia locale del database con loro sui computer portatili quando lasciano l'ufficio, apportano modifiche mentre sono lontani e hanno bisogno di unire quelli con il database principale quando tornano. Qui si desidera un approccio asincrono e pigro di replica e sarÃ costretto a considerare come gestire i conflitti nei casi in cui lo stesso record sia stato modificato sia sul server master che su una copia locale. Questi sono problemi di replica del database, ma il modo migliore per risolverli Ã¨ molto diverso. E come si puÃ²*vederedaquestiesempi*, *lareplicahamolteterminologiespecifichechedovraicapireperco*

Hot Standby / Streaming Replication Ã¨ disponibile a partire da PostgreSQL 9.0 e fornisce una replica binaria asincrona a uno o piÃ¹`standby`.`Glistandbypossonoanc`

(spiegazione cinci -audio-)

(http://www.slony.info/images/Slony-I-concept.pdf

Replicating schema changes is an often discussed problem and only ver y fe w database systems provide the necessary hooks to implement

it. PostgreSQL does not provide the ability to define triggers called on schema changes, so a transparent way to replicate schema changes is not possible without substantial work in the core PostgreSQL system. Moreover, ver y often database schema chages are not single, isolated DDL statements that can occur at any time within a running system. Instead they tend to be groups of DDL and DML statements that modify multiple database objects and do mass data manipulation like updating a new column to its initial value. The Slony-I replication system will have a mechanism to execute SQL scr ipts in a controlled fashion as part of the replication process.

1.3.2 *Pglogical*

Pglogical è un sistema logico di replica implementato come estensione di PostgreSQL. Completamente integrato, non richiede alcun triggers o programmi esterni. Questa alternativa alla replica fisica è un metodo altamente efficiente per replicare i dati utilizzando un modello di *publish/subscribe* per la replica selettiva.[14]

Vantaggi

I vantaggi offerti da Pglogical sono i seguenti:

- Replica sincrona
- Replica ritardata
- Risoluzione dei conflitti configurabili
- Capacità di convertire lo standby fisico in una replica logica
- Può pubblicare i dati da PostgreSQL a un abbonato Postgres-XL
- Le sequenze possono essere replicate
- Nessun trigger significa ridurre il carico di scrittura sul Provider
- Nessuna re-esecuzione di SQL significa overhead e latenza ridotti per il Sottoscrittore
- Il sottoscrittore non è in ripristino di riposo caldo, in modo da poter utilizzare tavoli temp, non sbloccati o normali

- Non è necessario annullare le query per consentire alla replica di continuare la riproduzione
- Il sottoscrittore (*subscriber*) può avere diversi utenti e protezione, indici diversi, impostazioni di parametri diversi
- Replica solo un database o un sottoinsieme di tabelle, noto come set di replica (*Replication Sets*)
- Replicare in versioni o architetture di PostgreSQL, consentendo aggiornamenti a bassa o zero-downtime
- Più server a monte in un singolo subscriber per l'accumulo di cambiamenti.[14]

Casi di uso

I diagrammi che seguono descrivono i gestori di database delle funzioni che sono in grado di eseguire con pglogical:

Come funziona pglogical?

Pglogical utilizza le funzioni di Decodifica Logica aggiunte da 2ndQuadrant (e disponibili da PostgreSQL 9.4). Pglogical funziona ancora più veloce con PostgreSQL 9.5 e successive, con bassi overhead su entrambi i provider e abbonati.

Pglogical si basa molto sulle caratteristiche introdotte nell'ambito dello sviluppo BDR, tra cui:

- Decodifica logica
-
- Slot di replica
- Lavoratori di sfondo statico
- Origini di replica
- Impegnano timestamp
- Messaggi WAL logici.[14]

Migrare e aggiornare PostgreSQL con tempi di inattività quasi a zero

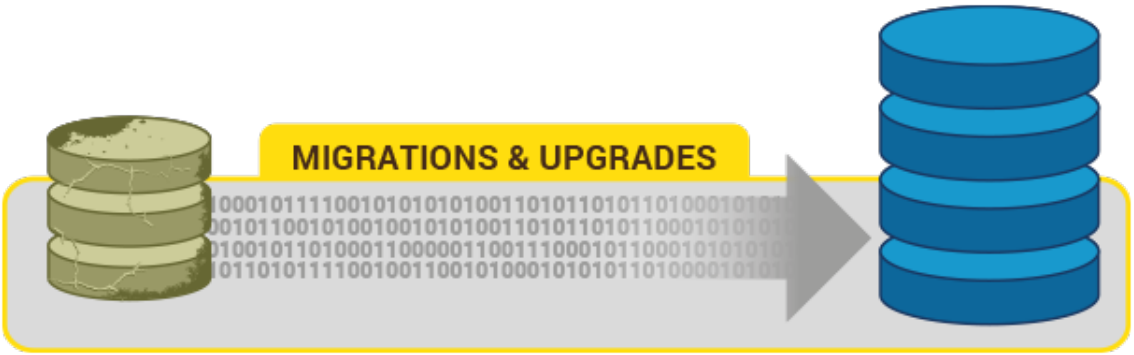

Figura 10: Migrazione e aggiornamenti PostgreSQL [14]

Accumulare le
modifiche provenienti da server di database scartati in un data warehouse

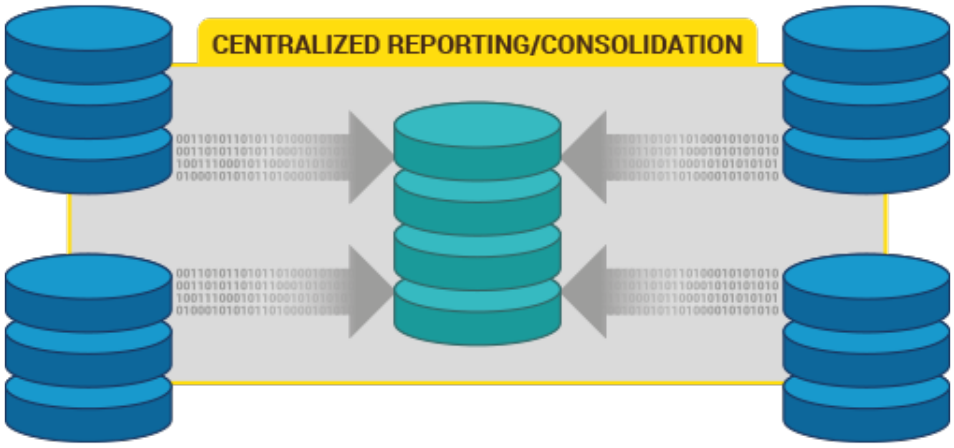

Figura 11: Aggregazione [14]

Copiare tutti o una selezione di tabelle di database ad altri nodi di un cluster

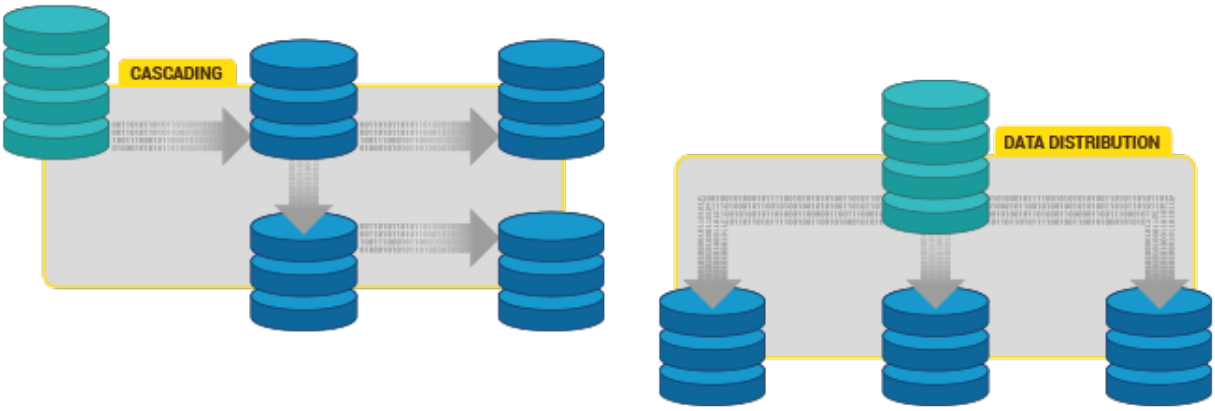

Figura 12: A cascata e distribuzione dati [14]

Le modifiche del database in tempo reale ad altri sistemi

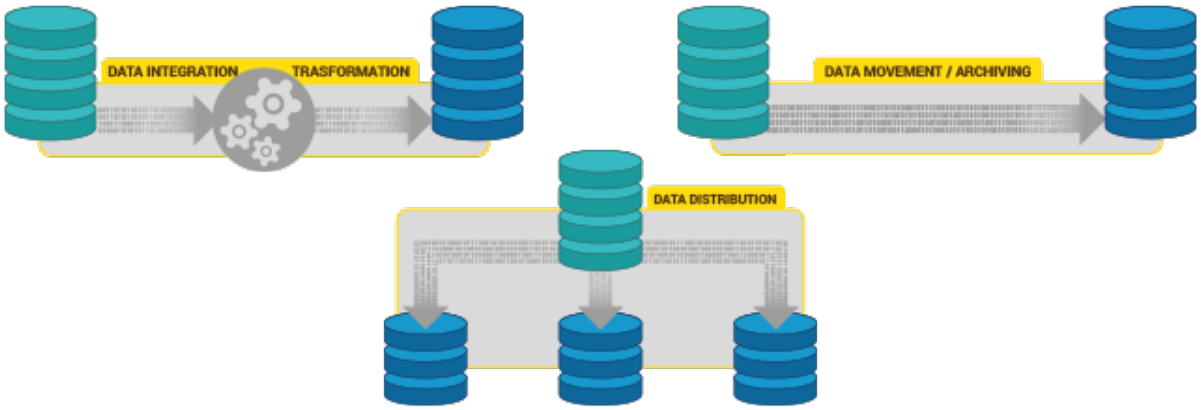

Figura 13: A cascata e distribuzione dati [14]

Replica pglogical Bi-Directional Replication (BDR)?

No. pglogical non fornisce funzionalitÃ complete di replica multi-master e un supporto di modifica dello schema coerente, come fa la BDR. Pglogical incorpora le funzionalitÃ di BDR e le lezioni apprese da BDR per produrre una soluzione piÃ¹ *semplicepiÃ¹ semplicedautilizzareperlareplicaunidirezion*

Lo sviluppo di BDR continuerÃ per quelli che richiedono piena capacitÃ multi-master, riutilizzando gran parte del codice da pglogical.[14]

`DA METTERE? DIFFERENZA TRA PG LOGICAL E BDR` Replica pglogical Bi-Directional Replication (BDR)? No. pglogical non fornisce funzionalitÃ complete di replica multi-master e un supporto di modifica dello schema coerente, come fa la BDR. Pglogical incorpora le funzionalitÃ di BDR e le lezioni apprese da BDR per produrre una soluzione piÃ¹ *semplicepiÃ¹ semplicedautilizzareperlareplicaunidirezionale, utilizzabileda*

2

DEFINIZIONE DEL PROGETTO

2.1.1 *Architettura del progetto*

concept Provider-Subscriber - da scegliere dove inserirlo Master to multiple cascaded slaves The basic structure of the systems combined in a Slony-I installation is a master with one or more slaves nodes. Not all slave nodes must receive the replication data directly from the master. Every node that receives the data from a valid source can be configured to be able to forward that data to other nodes. There are three distinct ideas behind this capability. The first is scalability. One database, especially the master that receives all the update transactions from the client applications, has only a limited capability to satisfy the slave nodes queries during the replication process. In order to satisfy the need for a big number of read-only slave systems it must be possible to cascade. The second idea is to limit the required networ k bandwidth for a backup site while keeping the ability to have multiple slaves at the remote location. The third idea is to be able to configure failover scenar ios. In a master to multiple slave configuration, it is unlikely that all slave nodes are exactly in the same synchronization status when the master fails. To ensure that one slave can be promoted to the master it is necessary that all remaining systems can agree on the status of the data. Since a committed transaction cannot be rolled back, this status is undoubtly the most recent sync status of all remaining slave nodes. The delta between this one and every other node must be easily and fast generated and applied at least to the new master (if thatâs not the same system) before the promotion can occur. Nodes, Sets and forwarding The Slony-I replication system can replicate tables and sequence numbers. Replicating sequence numbers is not unproblematic and is discussed in more detail in section 2.3. Table and sequence objects are logically grouped into sets. Every set should contain a group of objects that is independant from other objects originating from

the same master. In shor t, all tables that have relationships that could be expressed as foreign key constraints and all the sequences used to generate any ser ial numbers in these tables should be contained in one and the same set. Figure 1 illustrates a replication configuration that has 2 data sets with different origins. To replicate both date sets to NodeC it is not required that Node C really communicates with the origin of Set 1. This scenario has full redundancy for every node. Obviously if Node C fails, the masters of Set 1 and Set2 are still alive, no problem. If Node A fails, Node B can get promoted to the master of both sets. The tricky situation is if Node B fails. In the case Node B fails, Node C needs to get promoted to the master of Set 2 and it must continue replicating Set 1 from Node A. For that to be possible, Node A must have knowledge about Node C and its subscription to Set 1. Generally speaking, every node that stores replication log infor mation must keep it until all subscribers of the affected set are known to have replicated that data. To simplify the logic, the configuration of the whole networ k with all nodes, sets and subscriptions will be forwarded to and stored on all nodes. Because the sets,anode is not subscribed to must not even exist in its database, this does not include the infor mation about what tables and sequences are included in any specific set.

Simulazione di un filesystem distribuito (Dati e Metadati)

CONSIDERAZIONI STATISTICHE SULLA RIDONDANZA SUL DATO

CONSIDERAZIONI STATISTICHE SULLA RIDONDANZA DEL METADATO

RESILIENZA AI CAMBIAMENTI DI RETE

3

DEFINIZIONE DEL QUADRO SPERIMENTALE

DA RIVEDERE - http://www.slony.info/images/Slony-I-concept.pdf
Failover: While it is relatively easy to tell in a master to multiple slave scenar io which of the slaves is most recent at the time the master fails, it is near ly impossible to tell the actual row delta between two slaves. So in the case of a failing master, one slave can be promoted to the master, but all other slaves need to be re-synchronized with the new master. Perfor mance: Stor ing the logging infor mation in one or ver y fe w rotating log tables means that the replication engine can retrieve the actual data for one replication step with ver y fe w quer ies that select from one table only. In contrast to that a system that fetches the current values from the application tables at replication time needs to issue the same number of quer ies per replicated table and these queries will be joining the log table(s) with the application data table. It is obvious that this systems perfor mance will be reverse proportional to the number of replicated tables. At some time the complete delta to be applied, which can not be split as pointed out already, will cause the PostgreSQL database system to require less optimal than in memory hash join query plans to deal with the number of rows returned by these queries and the replication system will be unable to ever catch up unless the wor kload on the master drops significantly

LANCIO IN CONFIGURAZIONE 1

LANCIO IN CONFIGURAZIONE 2

LANCIO IN CONFIGURAZIONE 3

4

CONCLUSIONI E POSSIBILI EVOLUZIONI

UTILIZZO DI DISCHI SSD

UTILIZZO DI PROCESSORI DUAL CORE

5

ESERCIZI

1. Scrivere le possibili evoluzioni del programma

   ```
   co X: = X+2 // X: = X+1 oc
   ```

   assumendo che ciascun assegnamento è realizzato da tre azioni atomiche che caricano X in un registro (`Load R X`), incrementano il valore del registro (`Add R v`) e memorizzano il valore del registro((`Store R X`). Per ciascuna delle esecuzioni risultanti dallÕinterleaving delle azioni atomiche descrivere il contenuto dopo ogni passo della locazione condivisa X e dei registri privati, R_1 del processo che esegue il primo assegnamento ed R_2 per il processo che esegue il secondo assegnamento. Se assuma che il valore iniziale di X sia 50.

2. Si definisca il problema della *barrier synchronization* e si descrivano per sommi capi i differenti approcci alla sua soluzione. Se ne fornisca quindi una soluzione dettagliata utilizzando i semafori.

3. Considerare n api ed un orso che possono avere accesso ad una tazza di miele inizialmente vuota e con una capacità di k porzioni. L'orso dorme finchè la tazza è piena di k-porzioni, quindi mangia tutto il miele e si rimette a dormire. Le api riforniscono in continuazione la tazza con una porzione di miele finchè non si riempie; l'ape che aggiunge la k-esima porzione sveglia l'orso. Fornire una soluzione al problema modellando orso ed api come processi e utilizzando un monitor per gestire le loro operazioni sulla tazza. Prevedere che le api possano eseguire l'operazione *produce-honey* anche concorrentemente.

4. Descrivere le primitive di scambio messaggi send e receive sia sincrone che asincrone ed implementare

```
- synch_send(v:int)
- send(v:int)
- receive(x:int)
```

utilizzando le primitive di LINDA.

BIBLIOGRAFIA

[1] Techopedia - *Definition - What does Clustering mean?* (Cited on page 7.)

[2] Techopedia - *Techopedia explains Clustering* (Cited on pages 8 and 9.)

[3] Wikipedia, the free encyclopedia - *Shared nothing architecture* (Cited on page 8.)

[4] DA RIGUARDARE - *https://en.wikibooks.org/wiki/Oracle*$_{a}$*nd*$_{D}$*B2,*$_{C}$ *omparison*$_{a}$*nd*$_{C}$*om*

[5] Dave Wright - *The Advantages of a Shared Nothing Architecture for Truly Non-Disruptive Upgrades solidfire.com. 2014-09-17. Retrieved 2015-04-21* (Cited on page 8.)

[6] SearchNetworking - *peer-to-peer (P2P)* (Cited on page 10.)

[7] Lifeware - *Introduction to Peer-to-Peer Networks* (Cited on pages 10 and 11.)

[8] SearchStorage - *RAID (redundant array of independent disks)* (Cited on pages 3, 11, 13, 14, 15, 16, and 17.)

[9] SearchStorage - *RAID controller* (Cited on pages 12 and 20.)

[10] Derek Vadala - *Managing RAID on Linux, O' Reilly, 2002* (Cited on pages 11 and 15.)

[11] SearchStorage - *erasure coding* (Cited on pages 17 and 18.)

[12] PosgreSQL - *Documentation* (Cited on pages 19, 20, 21, and 22.)

[13] 2ndQuadrant Ltd - *PostgreSQL* (Cited on pages 12 and 20.)

[14] 2ndQuadrant Ltd - *pglogical* (Cited on pages 3, 23, 24, 25, and 26.)

[15] Autore - *Titolo* - altre informazioni