1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

CORSO DI LAUREA MAGISTRALE IN
INFORMATICA E INNOVAZIONE DIGITALE

RELAZIONE PROGETTO

Applicazioni Distribuite e Cloud Computing

Realizzato da:
Galantini Edoardo

N° Matricola: 324692

a.a. 2023/2024

Specifica del problema

Il progetto prevede lo sviluppo di un'applicazione distribuita utilizzando il linguaggio Erlang per la gestione dei fogli di calcolo. I fogli di calcolo possono includere più schede (Tab), ognuna delle quali può essere accessibile e condivisibile con chiunque possieda i permessi necessari.

Analisi del problema

I punti che andiamo a trattare sono la creazione dei fogli e dei rispettivi tab e in che modo rendere l'applicazione distribuita.

Per poter realizzare l'applicazione si è pensato di considerare ogni foglio come una tabella mnesia, popolata da record di tipo (spreadsheet, {table,riga,colonna}).

Per la memorizzazione del foglio proprietario viene utilizzata un'altra tabella chiamata owner popolata da record (owner, {foglio,pid}).

Per consentire la memorizzazione dei permessi (lettura/scrittura) delle celle di un foglio si utilizza un'altra tabella mnesia denominata policy popolata da record di tipo (policy, {pid,foglio,politica}).

Infine, la tabella format per memorizzare il formato del foglio, utile per alcune funzioni che vedremo successivamente.

Per rendere l'applicazione adatta per un ambiente distribuito e tollerante agli errori si è deciso di usare Mnesia. È utile perché:

- È progettato per essere utilizzato in sistemi distribuiti. Può replicare i dati su più nodi, permettendo a diverse istanze del database di essere sincronizzate automaticamente.

- Supporta sia la persistenza dei dati su disco sia la memorizzazione dei dati in RAM. Questo consente un accesso rapido ai dati in memoria e la possibilità di memorizzare dati a lungo termine su disco.

- Offre supporto per transazioni ACID (Atomicità, Consistenza, Isolamento, Durabilità), garantendo che le operazioni di lettura e scrittura siano affidabili e consistenti.

- Lo schema del database può essere modificato dinamicamente senza dover fermare il sistema. Ciò significa che le tabelle possono essere aggiunte, modificate o rimosse al volo.

- Essendo parte del framework Erlang/OTP, Mnesia è strettamente integrato con le altre componenti di Erlang, beneficiando delle sue capacità di concorrenza e tolleranza agli errori.

Scelte di progettazione

Per la progettazione dell'applicazione abbiamo deciso di utilizzare il DBMS Mnesia, un sistema di gestione di database distribuito, progettato per applicazioni scritte in Erlang. È parte integrante dell'ecosistema Erlang/OTP e offre caratteristiche uniche che lo rendono particolarmente adatto per sistemi distribuiti e ad alta disponibilità.

Mnesia

1) Generic server

I behaviour in Erlang sono un potente strumento per strutturare e standardizzare il codice, facilitando lo sviluppo di applicazioni concorrenti e fault-tolerant, forniscono un'interfaccia chiara e un insieme di funzionalità comuni che possono essere facilmente riutilizzate e mantenute.
Nel nostro caso abbiamo implementato il generic server. I motivi per il quale è stato deciso di implementarlo sono i seguenti:

- Gestione di I/O:
Il modulo csv_genServer gestisce operazioni di lettura e scrittura su file, che sono tipicamente operazioni bloccanti e possono richiedere tempo. Implementando gen_server in questo modulo, possiamo gestire queste operazioni in modo asincrono, migliorando la reattività del sistema.

- Gestione del Timeout:
Le funzioni to_csv/3 e from_csv/2 implementano già una logica di timeout. Questa logica si adatta perfettamente al modello di gen_server, che ha un meccanismo integrato per gestire timeout nelle chiamate sincrone (call). Utilizzando gen_server, possiamo semplificare e rendere più robusta la gestione dei timeout.

- Manutenibilità e Scalabilità:
Implementare gen_server rende il codice più manutenibile e scalabile. Le chiamate asincrone (cast) e sincrone (call) offerte da "behaviour" semplificano la gestione delle richieste concorrenti, migliorando la scalabilità del sistema.

2) Qlc (Query List Comprehensions) per query complesse

In Erlang, qlc è un modulo che permette di eseguire query sui record e altri tipi di dati strutturati in modo simile alle query SQL, ma all'interno del linguaggio Erlang. Il file di inclusione qlc.hrl contiene definizioni e macro utili per lavorare con le query list comprehensions.
Il file qlc.hrl è incluso nei moduli Erlang per fornire accesso alle funzioni e alle macro di qlc, facilitando la scrittura di query in modo dichiarativo. Le macro definite in qlc.hrl aiutano a rendere più leggibili e gestibili le query list comprehensions.

3) Tabelle mnesia di tipo bag

Le tabelle di tipo bag in Mnesia sono utili quando si ha bisogno di archiviare più record con la stessa chiave primaria. Le tabelle di tipo bag in Mnesia hanno le seguenti caratteristiche: Multipli record con la stessa chiave primaria: A differenza delle tabelle di tipo set, dove ogni chiave primaria deve essere unica, le tabelle di tipo bag permettono di avere più record con la stessa chiave primaria. Nessun duplicato di record identici: Sebbene si possano avere più record con la stessa chiave primaria, non possono esserci duplicati esatti (record identici) con la stessa chiave primaria.

Scelte di altro genere

Si è deciso di implementare le funzioni che utilizzano il parametro timeout (nello specifico: set/6 , to_csv/3, from_csv/2) in modo tale che quest'ultimo indicasse il tempo, in millisecondi, dopo il quale il nuovo valore inserito o la nuova tabella creata venissero eliminati.

Implementazione

Vengono riportate alcune funzioni rilevanti per ogni modulo del progetto.

1) La funzione new del modulo spreadsheeet crea su tutti i nodi di una rete un nuovo foglio e lo popola con valori undef. Se esiste già un foglio con lo stesso nome riporta errore, altrimenti salva il pid del nodo che ha invocato la funzione.

```
% Definizione funzione new(TabName, N, M, K)
new(TabName, N, M, K) ->
    mnesia:start(),
    TabelleLocali = mnesia:system_info(tables),
    % Controllo dell'esistenza di TabName
    case lists:member(TabName, TabelleLocali) of
        true -> {error, invalid_name};
        false ->
            % Creazione dello schema delle tabelle
            SpreadsheetFields = record_info(fields, spreadsheet),

            NodeList = [node()]++nodes(),

            mnesia:create_table(TabName, [
                {attributes, SpreadsheetFields},
                {disc_copies, NodeList},
                {type, bag}
            ]),
            % Si popola il foglio con k tabelle di n righe e m colonne
            populate_spreadsheet(TabName, K, N, M),
            % Si crea una nuova tabella in cui si specifica che il nodo è proprietario del foglio (tabella)
            populate_owner_table(TabName),
            % Si salvano le informazioni in base al formato della tabella
            populate_format_table(TabName, K, N, M)
    end
.
```

Esempio di foglio creato con la funzione new/4 avente i seguenti parametri:
- Nome: foglio1
- Num. Tab: 5
- Righe: 3
- Colonne: 2

```
1   foglio1,5,1,[undef,undef]
2   foglio1,5,2,[undef,undef]
3   foglio1,5,3,[undef,undef]
4   foglio1,1,1,[undef,undef]
5   foglio1,1,2,[undef,undef]
6   foglio1,1,3,[undef,undef]
7   foglio1,2,1,[undef,undef]
8   foglio1,2,2,[undef,undef]
9   foglio1,2,3,[undef,undef]
10  foglio1,4,1,[undef,undef]
11  foglio1,4,2,[undef,undef]
12  foglio1,4,3,[undef,undef]
13  foglio1,3,1,[undef,undef]
14  foglio1,3,2,[undef,undef]
15  foglio1,3,3,[undef,undef]
```

2. La funzione create_table/0 del modulo distribution crea lo schema base delle tabelle Mnesia; crea la tabella proprietario, policy e formato in tutti i nodi della rete.

```
% Definizione funzione create_table()
create_table() ->
    NodeList = [node()] ++ nodes(),
    % Creazione dello schema delle tabelle e di quest'ultime
    mnesia:create_schema(NodeList),
    % Si esegue l'avvio di Mnesia
    start_remote(),
    OwnerFields = record_info(fields, owner),
    PolicyFields = record_info(fields, policy),
    FormatFields = record_info(fields, format),
    mnesia:create_table(owner, [
        {attributes, OwnerFields},
        {disc_copies, NodeList}
    ]),
    mnesia:create_table(policy, [
        {attributes, PolicyFields},
        {disc_copies, NodeList},
        {type, bag}
    ]),
    mnesia:create_table(format, [
        {attributes, FormatFields},
        {disc_copies, NodeList},
        {type, bag}
    ]),
    % Viene eseguito lo stop di Mnesia per tutti i nosi
    distribution:stop()
.
```

3. La funzione to_csv/2, del modulo csv_manager, permette di salvare su disco un foglio già creato in formato csv. È stata implementata tramite l'utilizzo del generic server behaviour.

```
to_csv(TableName, FileName) ->
    gen_server:call(?MODULE, {to_csv, TableName, FileName}).
```

```
handle_call({to_csv, TableName, FileName}, _From, State) ->
    Reply = to_csv_impl(TableName, FileName),
    {reply, Reply, State};
```

```
% Definizione funzione to_csv_impl(TableName, FileName)
to_csv_impl(TableName, FileName) ->
    mnesia:start(),
    TabelleLocali = mnesia:system_info(tables),
    case lists:member(TableName, TabelleLocali) of
        false -> {error, invalid_name_of_table};
        true ->
            % Si apre il file in scrittura
            {ok, File} = file:open(FileName, [write]),
            % Vengono estratti dati dalla tabella Mnesia
            Records = ets:tab2list(TableName),
            % Si convertono i dati in CSV
            CsvContent = records_to_csv(Records),
            % Viene eseguita la scrittura su file
            file:write(File, CsvContent),
            % Si chiude il file
            file:close(File)
    end
.
```

Scelte implementative

L'applicazione è suddivisa in più moduli.

Modulo Spreadsheet:
Un foglio di calcolo è una "matrice" di NxM celle e ogni cella può contenere un qualsiasi tipo di dato primitivo o il valore undef.

Contiene le seguenti funzioni:
new(name)-> spreadsheet | {error, reason}

- Crea un nuovo foglio di nome "name" di dimensioni NxM di K tab.
- Assegna il processo creatore come proprietario del foglio.
- I parametri N, M, K sono definiti di default nel modulo.

new(name, N, M, K)-> spreadsheet | {error,reason}

- Crea un nuovo foglio di nome "name" di K tab.
- Ogni tab ha dimensioni NxM.
- Assegna il processo creatore come proprietario del foglio.

share(spreadsheet, AccessPolicies)-> bool
- Il proprietario del foglio può condividere il foglio in "lettura" o "scrittura" con altri processi.

- AccessPolicies è una lista di {Proc,AP} dove:
  - Proc è un Pid/reg_name.
  - AP = read | write.
- Le policy di accesso ad un foglio possono cambiare in qualsiasi momento.

get(spreadsheet, tab, i, j, val)-> Value | undef

- Legge il valore della cella (i,j) che appartiene al tab del foglio spreadsheet.

set(spreadsheet, tab, i, j, k, val)-> bool

- Scrive il valore della cella (i,j) che appartiene al tab del foglio spreadsheet.

get(spreadsheet, tab, i, j, val, timeout)-> Value | undef | timeout

- Uguale alla funzione get/4 però con un timeout entro il quale l'esecuzione termina subito.

set(spreadsheet, tab, i, j, val, timeout)-> bool | timeout

- Uguale alla funzione set/4 però con un timeout entro il quale l'esecuzione termina subito.

info(name)-> Spreadsheet_info

- Ritorna le informazioni del foglio name. Deve contenere: il numero di celle x tab e i permessi di lettura e scrittura.

Modulo csv_manager:

Il modulo è stato progettato per semplificare le conversioni tra fogli e file csv. Inoltre, viene utilizzato il generic_server behaviour per una migliore gestione della concorrenza e fault-tolerance.

Le funzioni implementate sono le seguenti:

to_cvs(spreadsheet,filename)-> ok | {error,reason}

- Esporta in csv il foglio con il nome filename.csv.

from_cvs(filename)-> spreadsheet | {error,reason}

- Crea il foglio dal filename.csv.

to_cvs(name,filename, timeout)-> ok | {error,reason} | timeout

- Uguale alla funzione to_cvs/1 però con un timeout entro il quale l'esecuzione termina subito.

from_cvs(name, filename, timeout)->spreadsheet | {error,reason} | timeout

- Uguale alla funzione from_cvs/2 però con un timeout entro il quale l'esecuzione termina subito.

Modulo distribution:

Il modulo serve per la realizzazione di un ambiente distribuito con il DBMS Mnesia, utilizzando funzioni per la creazione di tabelle Mnesia (su disco), il suo avvio e la sua terminazione.

Contiene le seguenti funzioni:

create_table()-> {atomic, ok} | {error, reason}

- Crea tabelle del DB in tutti i nodi presenti nella rete.

start()-> ok | {error, reason}

- Avvia Mnesia nel nodo locale.

stop()-> ok | {error, reason}

- Termina Mnesia nel nodo locale.

Modulo measurements:

Il modulo serve per testare alcune misure del programma sotto stress.

average_lookup_time(name, n_nodi)-> {atomic, value} | error

- Misura in microsecondi il tempo medio di ricerca con n nodi.

average_setup_time(Foglio, N)-> {atomic, value} | error

- Misura in microsecondi il tempo medio di configurazione con n nodi.

Testing

Per ogni nodo della rete che farà parte del sistema distribuito, eseguire il comando:
erl-sname "nome_nodo"-setcoockie "nome_cookie"

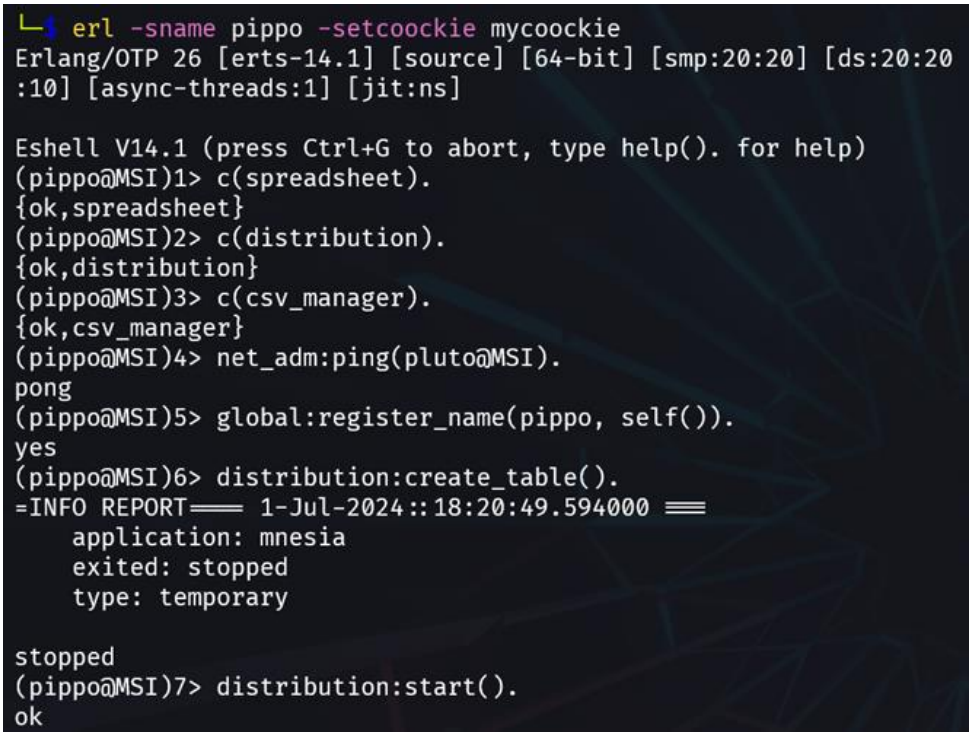
Compilare i moduli con i comandi (per ogni nodo):
c(spreadsheet).
c(distribution).
c(csv_manager).
c(measurements).

Far sì che i nodi si mettano in contatto (esempio):
shell 1> net_adm:ping(pippo@MSI).
shell 2> net_adm:ping(pluto@MSI).

Salvare globalmente il nome di ogni nodo:
shell 1> global:register_name(pippo, self()).
shell 2> global:register_name(pluto, self()).

Eseguire programma in maniera distribuita; per creare tabella Mnesia su tutti i nodi della rete:
distribution:create_table().

Per avviare DBMS Mnesia su tutti i nodi della rete (oppure interrompere):
distribution:start().
distribution:stop().



A questo punto è possibile utilizzare tutti i metodi implementati:

1) Funzione per creare un nuovo foglio:
   - Nome: foglio1
   - Tab: 8
   - Righe: 8
   - Colonne: 3

```
(pippo@MSI)8> spreadsheet:new(foglio1, 8, 8, 3).
ok
```

2) Funzione per condividere i permessi (lettura o scrittura) a un nodo specifico:
   - Nome: foglio1
   - Nodo: pippo (proprietario)
   - Politica: scrittura

```
(pippo@MSI)9> spreadsheet:share(foglio1, {pippo, write}).
ok
```

3) Funzione per modificare un valore di una cella:
   - Nome: foglio1
   - Tab: 1
   - Riga: 1
   - Colonna: 1
   - Valore: 100

```
(pippo@MSI)10> spreadsheet:set(foglio1, 1, 1, 1, 100).
ok
```

4) Funzione per aggiungere una nuova riga:

```
(pippo@MSI)11> spreadsheet:add_row(foglio1, 1).
ok
```

5) Funzione per l'avvio del generic server e per la conversione di un foglio in CSV.
   - Nome: foglio1
   - File destinazione: foglio1.csv

```
(pippo@MSI)15> csv_manager:start_link().
{ok,<0.278.0>}
(pippo@MSI)16> csv_manager:to_csv(foglio1, "foglio1.csv").
ok
```

6) Funzione per misurare il tempo medio di ricerca con N nodi:

```
(owner@LAPTOP-560JLB7T)20> measurements:average_lookup_time(foglio1, 1000).
{microsecond,80.54}
```