Tema 1 - Simularea activităţii unui cluster

Adriana Drăghici, Dan Dragomir
adriana.draghici at cs pub ro, dan.dragomir at cs pub ro

March 12, 2014

1 Introducere

Un cluster este o colecţie de noduri (calculatoare) individuale conectate între ele, care lucrează împreună. Într-un cluster există mai multe tipuri de noduri:

- Front-End Processors (FEPs) - coordonează accesul utilizatorilor la cluster
- Noduri computaţionale - prelucrează cererile utilizatorilor
- Noduri de stocare - stochează datele din cluster (în unele arhitecturi de cluster funcţiile de stocare şi de prelucrare sunt executate pe acelaşi nod)

În linii mari activitatea unui cluster poate fi descrisă astfel:

- utilizatorii trimit FEP-urilor job-urile care vor fi prelucrate de cluster; fiecare job este format din mai multe task-uri
- FEP-urile distribuie task-urile din job-urile primite către nodurile computaţionale din cluster; deoarece nodurile sunt echipate cu procesoare multi-core mai multe task-uri pot fi atribuite unui singur nod
- nodurile computaţionale încep execuţia task-urilor interogând nodurile de stocare în funcţie de necesităţile task-ului

2 Enunţ

Pentru această temă va trebui să implementați, folosind limbajul Python, o variantă distribuită a algoritmului de eliminare gaussiană pentru a rezolva un sistem de ecuații liniare $A*x = b$. Pentru detalii despre eliminarea gaussiană consultați pagina Wikipedia [1].

Pe baza scheletului și a infrastructurii de testare puse la dispoziție, vom rula apoi algoritmul în cadrul unui cluster simulat. Fiecare nod al clusterului va fi simulat printr-un obiect al clasei `Node`, pe care voi trebuie să o completați atât cu codul algoritmului, cât și cu codul necesar pentru comunicarea între noduri. Rularea în paralel a nodurilor va fi făcută, bineînțeles, cu ajutorului thread-urilor. Fiecare obiect `Node` va trebui să creeze cel puțin un thread pentru a putea executa algoritmul distribuit și comunica cu celelalte noduri. Deoarece presupunem ca nodurile simulate conțin procesoare multi-core este permis să creați oricâte thread-uri doriți în cadrul unui obiect `Node`.

Sistemul de ecuații liniare va fi distribuit între nodurile clusterului. Vom presupune că numărul de noduri ale clusterului este egal cu numărul de ecuații ale problemei. Fiecare nod va stoca o linie a matricii A și elementul b aflat pe aceeași linie. Partea de stocare a clusterului va fi simulată cu ajutorul obiectelor de tip

[1] `http://en.wikipedia.org/wiki/Gaussian_elimination`

`Datastore`. Fiecare obiect `Node` va avea asociat un astfel de obiect care va conține linia matricii și elementul b pentru nodul respectiv.

Datorită arhitecturii clusterului (fig. 1) un nod are acces direct doar la datastore-ul său. Pentru a afla alte elemente ale matricii, el va trebui să comunice cu nodul care stochează acele elemente. *Comunicarea între thread-urile celor două noduri poate fi făcută prin orice metodă doriți, însă va trebui să fie sincronizată corect.*

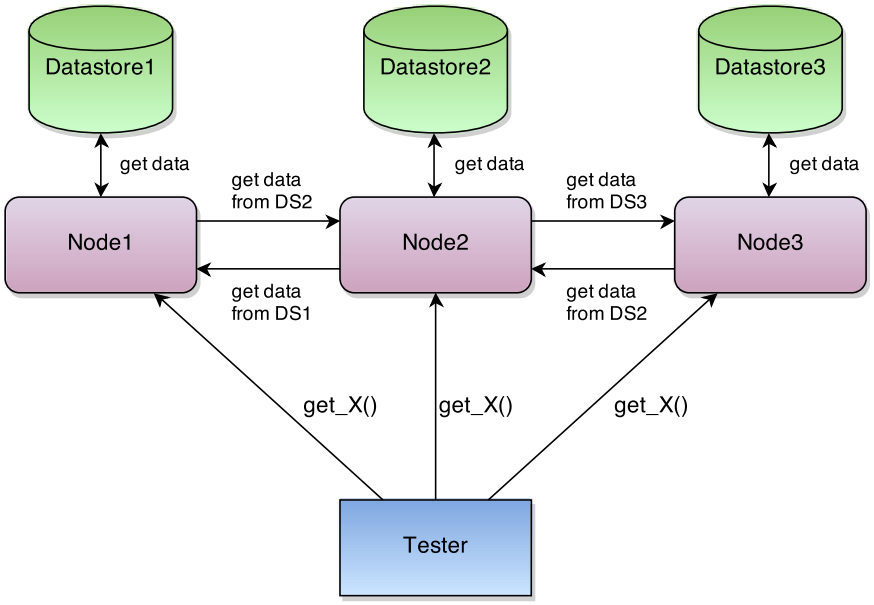

Figure 1: Arhitectura clusterului

Accesul unui nod la datele stocate în datastore-ul său se va face prin intermediul metodelor puse la dispoziție de clasa `Datastore`. Fiecare nod va trebui să apeleze metodele datastore-ului lui din thread-urile sale. Infrastructura de testare va verifica aceste accese și va rejecta orice acces care nu vine dintr-un thread creat în nodul de care aparține datastore-ul. Pentru ca datastore-ul să cunoască thread-urile care aparțin nodului său acestea vor trebui înregistrate cu metoda `register_thread` (fig. 2).

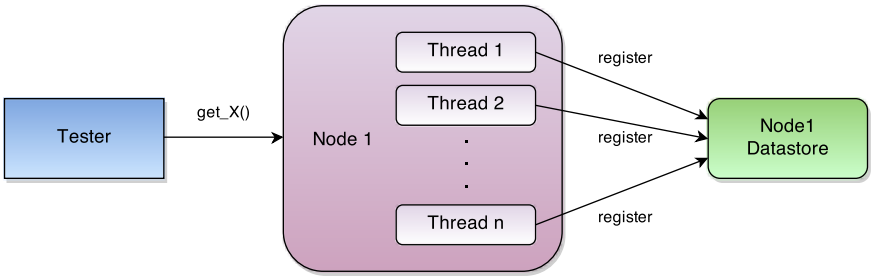

Figure 2: Inregistrarea la datastore

În mod normal un cluster este folosit pentru a rezolva o problema de dimensiuni mari, care nu ar încăpea pe un singur nod. Din acest motiv, un nod nu poate cache-ui întreaga problemă. Pentru a simula acest lucru, fiecare cerere pentru un element din datastore, fie ca este necesar nodului local sau reprezintă răspunsul pentru o cerere a unui nod remote, trebuie să fie citit din datastore. Este **interzis** să cache-uiţi conţinul datastore-ului în cadrul clasei `Node`.

Exemple:

- În prima iterație, nodurile cu ID-urile 1..N-1 vor avea nevoie de linia 0. Este interzis ca nodul 0 să citească această linie din datastore-ul său o singură dată, la începutul iterației, să o stocheze într-un câmp al clasei și apoi să raspundă celolalte noduri citind date din acest câmp. El va trebui să interogheze datastore-ul pentru fiecare cerere primită. În exemplul prezentat, va citi linia din datastore de N-1 ori.
- La o anumită iterație un nod are nevoie să calculeze un factor cu care va înmulți linia pivotului pentru a obține zero-uri pe linia sa. Acest factor este un raport între două valori. O dată cerute cele două valori de la datastore-uri și calculat factorul, acesta poate fi refolosit de oricâte ori și nu va fi considerată cache-uire.

Simularea FEP-ului este făcută de către tester. Acesta va crea obiectele `Node` și `Datastore` din clusterul simulat, va informa fiecare nod despre datastore-ul său și despre celelalte noduri din cluster, iar apoi va cere soluția problemei de la noduri (fig. 1). La creare, nodurile vor primi un ID, precum și dimensiunea problemei (N). ID-ul este reprezentat de un număr întreg între 0 și $N-1$, inclusiv. Datastore-ul fiecărui nod va fi inițializat cu linia din matricea A și elementul din b aflate pe aceeași linie cu ID-ul. De asemenea, fiecare nod va trebui să returneze un element din x împreună cu poziția acestui element în vectorul soluției.

3 Implementare

Pentru rezolvarea acestei teme va trebui să completați clasa `Node` cu implementările pentru metodele deja definite în schelet. Pe lângă acestea puteți crea bineînțeles oricâte alte metode/clase/module aveți nevoie în rezolvare. Nu modificați alte clase prezente în scheletul temei decât pentru debugging. Detaliile exacte despre metodele claselor `Node` și `Datastore` (ce trebuie să oferiți și ce este oferit) le găsiți în *docstring*-ul temei, în `doc/index.html`.

4 Testare şi notare

Tema va fi verificată automat, folosind infrastructura de testare, pe baza a 10 teste definite în directorul `tests`. Unele teste sunt rulate de mai multe ori pentru a detecta eventualele bug-uri de sincronizare. Există un timeout, specific fiecărui test, în care trebuie să se încadreze execuţia tuturor iteraţiilor testului respectiv.

Infrastructura de testare, scheletul clasei Node, precum şi documentaţia API-ului poate fi descărcată de aici [2].

Tema se va implementa în Python 2.7. Arhiva temei (fişier .zip) va fi uploadată pe site-ul cursului şi trebuie să conţină:

- fişierul `node.py` cu implementarea clasei `Node`
- alte surse `.py` folosite de soluţia voastră (nu includeţi fişierele infrastructurii de testare)
- fişierul README cu detaliile implementării temei (poate fi în engleză)

Notarea se va face în felul următor:

- $100pct$ - 10 teste a câte $10pct$
- $10pct$ - bonus, dacă numărul de thread-uri create de fiecare nod este mai mic decât numărul de request-uri paralele suportate de datastore-ul său plus 1
- $-20pct$ - depunctări

[2] `http://cs.curs.pub.ro/2013/pluginfile.php/26064/mod_assign/intro/tema1.zip`

- folosirea busy-waiting
- lipsa organizării codului, implementare inelegantă și/sau complexă
- ilizibilitatea codului, inconsistenţa stilului
- lipsa comentariilor utile din cod
- fişier README sumar ($-10pct$ pentru fișier gol sau inexistent)
- alte situaţii nespecificate aici, dar considerate inadecvate

5 Precizări

- unul din teste va necesita implementarea pivotării pentru a evita impărțirea la 0
- două dintre teste vor necesita preluarea datelor din datastore în paralel, în limita numărului de request-uri suportate
- pot exista depunctări mai mari decât este specificat în secţiunea *Notare* pentru implementări care nu respectă spiritul temei
- implementarea şi folosirea API-ul oferit este obligatorie
- toate operaţiile făcute de un nod (calcule, accesare datastore, comunicare cu alte noduri) trebuie făcute din thread-uri proprii nodurilor; atenție, metoda `get_x` este apelată de tester de pe thread-urile sale
- este interzis să cache-uiţi datele citite din datastore
- bug-urile de sincronizare, prin natura lor sunt nedeterministe; o temă care conţine astfel de bug-uri poate obţine punctaje diferite la rulări succesive; în acest caz punctajul temei va fi cel dat de tester în momentul corectării pe sistemul asistentului
- recomandăm testarea temei în cât mai multe situații de load al sistemului și pe cât mai multe sisteme (fep, cluster etc.) pentru a descoperi bug-urile de sincronizare