Sisteme Distribuite – Teorie
4. Apel de procedura la distanta

Modelul client-server vs. RPC

- Client-server:
  - Construit in jurul I/O
  - Toate comunicatiile sunt construite in send/receive
  - Calculul distribuit seamana cu calculul centralizat
- RPC permite apel la proceduri localizate pe alte masini

Principiul RPC

- Cand un proces pe o masina A apeleaza o procedura de pe masina B,
  - Procesul apelant de pe A este suspendat, si
  - Executia procedurii apelate are loc la B.
- Informatia poate fi transportata de la apelant la apelat in parametrii si se poate intoarce prin rezultatul procedurii
- Programatorul nu vede nici o transmitere de mesaje sau I/O

Probleme

- Procedura care apeleaza si procedura apelata ruleaza in masini diferite, se executa in spatii de adresare diferite -> complicatii
- Parametrii si rezultatele trebuie sa fie transmise, ceea ce nu este simplu, in special daca masinile u sunt identice
- Ambele masini pot sa cedeze si fiecare din esecurile posibile cauzeaza probleme diferite

Operatii RPC de baza (1)

- Idea din spatele RPC este aceea de a face in asa fel incat apelului la distanta sa arate cat mai apopriat de cel local
  - adica RPC sa fie transparent: procedura care apeleaza sa nu fie constienta ca procedura apelata este executat intr-o masina diferita, sau viceversa
- Ex: citirea unei date dintr-un fisier
  - Instr-un sistem traditional (mono-procesor) :
    - Rutina de read este extrasa din biblioteca de editorul de legaturi si inserat in programul obiect
    - Procedura de citire este un fel de interfata intre codul utilizator si sistemul de operare
  - Cu RPC:
    - Cand *read* este o procedura la distanta (ex.una care va rula pe masina serverului de fisiere), o versiune diferita a *read*, numita stub a clientului, este pusa in biblioteca.
    - …

Operatii RPC de baza (2)

- Ex. Citirea unei date dintr-un fisier.
  - Cu RPC:
    - …
    - Spre deosebire de read-ul original, nu pune parametrii in registrii si cere nucleului sa ii furnizeze data.
    - Impacheteaza parametrii intr-un mesaj si cere nucleului sa trimita mesajul la server
    - Ca urmare a apelului la expediere, stub-ul clientului apeleaza receive si se blocheza pana cand raspunsul vine inapoi
    - …

Operatii RPC de baza (3)

- Ex. Citirea unei date dintr-un fisier.
  - Cu RPC:
    - …
    - Cand mesajul ajunge la server, nucelul il paseaza la un *stub al serverului* care este legat de serverul adevarat.
    - Stub-ul serverului este in apel de receive si blocat in asteptarea mesajelor
    - Stubul serverului despacheteza parametrii din mesaj si apoi apeleaza procedura server in mod uzual.
    - Din punct de vedere al serverului, pare a fi apelat direct de client – parametrii si adresele de returnare sunt pe stiva si nimic nu pare ne-uzual.
    - Serverul efectueaza lucrul sau si returneaza rezultatele la apelant in mod uzual: in cazul dat, serverul va completa un buffer cu data solicitata – acest buffer va fi intern stub-ului server.
    - …

Operatii RPC de baza (4)

- Ex. Citirea unei date dintr-un fisier.
  - Cu RPC:
    - …
    - Cand stubul server obtine inapoi controlul dupa ce apelul s-a terminat, impacheteaza rezultatul (bufferul) intr-un mesaj si apeleaza send pentru al returna la client. Apoi merge inapoi in ciclul sau pentru a apela receive asteptand urmatoarele mesaje
    - Cand mesajul vine inapoi la masina client, nucleul observa ca este adresat procesului client (partii stub a acelui proces).
    - Mesajul este copiat in bufferul de asteptare si procesul client este deblocat
    - Stubul client inspecteaza mesajul, despacheteaza rezultatul, il copiaza la apelant
    - Cand apelantul obtine controlul ca urmare a apelului la read, ceea ce stie este faptul ca data este dsponibila.

Operatii RPC de baza - pasii

1. Procedura client apeleaza stub-ul clientului.
2. Stub-ul clientului construieste un mesaj si informeaza nucleul.
3. Nucleul expediaza mesaj la nucelul la distanta
4. Nucleul la distanta ofera mesajul stub-ului serverului.
5. Stubul serverului impacheteaza parametrii si apeleaza serverul.
6. Serverul efectueaza procedura si returneaza rezultatul la stub.
7. Stub-ul server il impachetaza intr-un mesaj si informeaza nucleul
8. Nucleul la distanta expediaza mesajul la nucluel clientului.
9. Nucleul clientului da mesajul la stubul clientului.
10. Stubul despacheteaza rezultatul si il returneaza la client.

Transmiterea parametrilor (1)

- Stubul clientului: preia parametrii, ii impacheteaza intr-un mesaj, si-i trimite la stub-ul server.
- Impachetarea parametrilor intr-un mesaj este numita *parameter marshaling (insiruirea parametrilor)*
- Exemplu: *sum(i,j):* doi parametrii intregi si returneaza suma lor aritmetica. Stubul clientului:
  - Preia cei doi parametrii si-l pune intr-un mesaj
  - De asemenea pune si numele sau numarul procedurii care trebuie apelata deoarece serverul poate sa suporte mai multe apeluri si este necesar a fi indicata care procedura a serverului este referita
  - … (vezi cei 10 pasi)

Transmiterea parametrilor

- Model functioneaza bine atata cat masinile client si server sunt identice si toti parametrii si rezultatele sunt tipuri scalare, precum intregi, caractere si Booleene.
- Tipuri de masini multiple => probleme potentiale!
  - Exemplu: anumite masini (ex. Intel), numara bitii de la dreapta la stanga (format numit *little endian*), pe cand altele (ex. Sun), ii numara in send invers (format numit *big endian**)
  - Solution: *a standard* has been agreed upon for representing each of the basic data types

*numele formatelor sunt date dupa politicienii din Calatoriile lui Guliver care au ajuns in razboi legat de care capat de ou sa fie spart.

Utilizarea standardelor in transmiterea de parametrii

- Un standard de retea sau o forma canonica pentru intregi, caractere, Booleene, numere in virgula mobila,
- Cere tuturor expeditorilor sa converteasca reprezentarile lor interne in aceasta forma la insiruire
- Problema: cateodata ineficient
  - Ex: marshalizare in little endian, dar conversatia se face intre doua big endian
- A doua abordare: clientul utilizeaza formatul sau nativ si indica in primul bit al mesajului care format este.
  - Stubul server converteste daca este necesar

De unde vin procedurile stub?

- in numeroase sisteme bazate pe RPC, sunt generate automatic
- data o specificatie a procedurii server si regului de codare, formatul mesajului este unic determinat
- Un compilator citeste specificatiile serverului si genereaza un stub al clientului care impacheteaza parametrii in formatul de mesaj aprobat oficial
- Similar, compilatorul paote se asemenea produce in stub server care despacheteaza mesajele si apeleaza serverul
- Ambele proceduri stub sunt generate dintr-o singura specificatie formala a serverului
  - Faca viata mai usoara pentru programatori,
  - Reduce sansele de eroare
  - Face sistemul transparent relativ la dieferntele interned de reprezentare a datelor

Parametrii pointeri si referinte?

- Exemplu: stubul clientului cunoaste ca parametrul secund pointeaza la o matrice de caractere si cunoaste cat este de mare este matricea
- Prima strategie:
  - Copiaza matricea intr-un mesaj si o expediaza la server
  - Schimbari la server utilizand pointerul (ex. Stocarea datelor in matrice) afecteaza direct bufferul mesajului la stub serverului.
  - Cand serverul termina, mesajul original poate fi trimis inapoi la stubul client care apoi il copiaza inapoi la client
  - Referinta-prin-apel a fost inlocuita prin copiere/reconstituire.

Parametrii pointeri si referinte?

- Optimizare:
  - Daca stubul stie ca bufferul este un paramateru de intrare sau un parametru de iesire la server, una dintre copii poate fi eliminate
  - Daca matricea este de intrare la server (ex. Intr-un call la scriere) nu este necesar sa fie copiat inapoi
  - Daca este de iesire, nu este necesar sa fie transmis initial.
  - Modalitatea de a le califica este cea prin intermediul specificarii formale a procedurii server
- Specificatia formala a unei proceduri
  - Scrisa intr-un anumit limbaj de specificare
  - Spune care sunt parametrii
  - Care sunt intrarile si care sunt iesirile (sau ambele),
  - Care sunt marimile lor (maxime)

Cazul general al unui pointer la o structura de date arbitrara

- Exemplu: un pointer la un graf complex
- Abordare:
  - Pasarea pointerului la stubul serverului si generraea unui cod special in procedura server pentru utilizarea pointerilor
  - Un pointer este de-refentiat (pus intr-un registru si indirectat prin registru) prin expedierea unui mesaj inapoi la client cerandu-l sa i-l dea si sa-l trimita (caz read) sau sa-l stocheze (write) de la / la adresa referita
  - Metoda este foarte ineficienta:
    - Exemplul unui server de fisiere care stocheza octeti in buffer prin expedeirea fiecaruia intr-un mesaj separat

Legaturi dinamice

- Intrebare: cum localizeaza clientul serverul?
- O metoda este adresa de retea a serverului
  - Abordarea este extrem de inflexibila!
    - Daca serverul se muta sau daca serverul este replicat si daca interfata se schimba, programe numeroase trebuie gasite si recompilate
- Pentru a evita aceste probleme, anumite sisteme distribuite utilizeaza *legatura dinamica* pentru a pune cap-la-cap clientii si serverele.

Specificarea formala a serverului

- Exemplu: in specificatie se indica numele serverului (ex. file_server), numarul de versiune (ex 3.1), o lista de proceduri oferita de server (read, write, create, si delete).
- Pentru fiecare procedura, tipurile parametrilor
- Fiecare parameter este specificat ca fiind un parametru in, out sau in-out (directia este relativa la server)
  - Un parametru in, precum un nume de fisier, este trimis de la client la server
  - Un parametru out precum buf in read, este expediat de la server la client. Buf indica locul unde serverul pune data solicitata de client
  - Un parametru in-out poate fi trimis de client la server, modificat si expediat inapoi la client (copiere/recuperare)

Binder (liant, legator) si register

- Cand server incepe executia, apelul la initializare din afara ciclului principal *exporta* interfata serverului, adica.:
  - Serverul expediaza un mesaj la un program numit *binder*, pentru a-si face existenta cunoscuta
- Acest proces de expediere a mesajului este referit ca *inregistrarea* serverului.
- Pentru a se inregistra, serverul ofera binderului
  - Numele sau,
  - Numarul sau de versiune,
  - Un identificator unic, tipic pe 32 biti, si
  - *handle* utilizat pentru localizarea sa.
    - Acesta este idenpendent de sistem, si poate fi o adresa Ethernet, o adresa IP, o adersa X.500, un identificator de proces, sau altceva.
  - Alte informatii, precum autentificarea
- Un server poate de asemenea sa se dez-registreze de la binder cand nu mai este pregatit sa ofere servicii

Interfata binderului

Call	*Input*	*Output*
Register	Name,version,handle, unique id	
Deregister	Name,version,unique id	
Lookup	Name, version	Handle, unique id

Cum localizeaza clientii serverul

1. Apeleaza una dintre procedurile la distanta pentru prima data, fie de exemplu, read
2. Stubul clientului observa ca nu este legat la un server si trimite un mesaj la binder cerand sa importe versiunea 3.1 a interfetei file_server.
3. Binderul verifica daca cel putin un server a exportat o inetrfata cu acest nume si cu aceasta versiune
4. Daca nici unul dintre serverele care ruleaza este dispus sa suporte aceata interfata, apelul esueaza
5. Daca exisat un server adecvat, binderul ofera handle-ul si identificatorul unic stubului client.
6. Stubul clientului utilizeaza handle ca adresa pentru a expedia mesajul dorit
7. Mesajul contibe paranetrii si identificatorul unic pe care nucleul serverului le utilizeaza pentru a directa mesajelor sosite la serverul corect in conditiile in care mai multe servere ruleaza pe acea masina

Legare dinamica: avantaje vs. dezavantaje

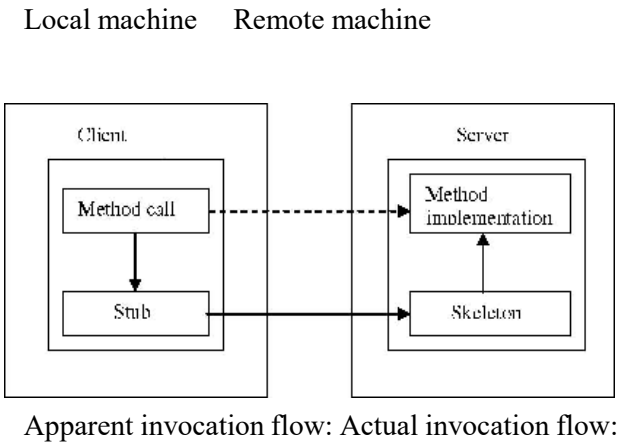
- Avantaje:
  - Poate trata servere multiple care suporta aceeasi interfata
  - Binderul poate imparstia clientii aleator la servere pt.incarcare balansata
  - Binderul poate chestiona periodic serverele, dez-registrand automat oricare servere care nu raspunde, pentru a atinge un anumit grad de tolerare a esecurilor
  - Binderul poate asista la eutentificare
    - Un server poate specifica, de ex., ca poate utiliza numai o lsita specificata de utilizatori, in care caz binderul refuza utilizatorii care nu sunt pe lista
  - Binderul poate de asemenea verifica daca atat clientul cat si serverul utilizeaza aceeasi versiune de interfata
- Dezavantaje:
  - Surplusul extra in timp pentru exportarea si importarea interfetelor.
  - Deoarece numeroase proecese client au viata scurta si fiecare proces de legare trebuie starta de fiecare data, efectul poate fi semnificativ
  - Intr-un sistem distribuit mare, binderul poate deveni sursa de gatuire
    - Sunt necesari binderi multiplii.
      - Cand o interfata este inregistrata/dez-registrata, un nr, substantial de mesaje este necesar pentru a tine toti binderili sincronizati si la zi, creand astfel un nou surpuls

Protocol orientat conexiune vs fara conexiune

- Protocol orientat-conexiune:
  - In momentul in care clientul este legat la server, o conexiune este stabilita intre ei
  - Tot traficul, in ambele directii, utilizeaza aceasta conexiune
  - Avantaje: comunicarea devine mai usoara.
    - Cand un nucleu expediaza un mesaj, nu-si face probleme ca poate fi pierdut sau ca trebuie sa primeasca o instiintare (tratat de software-ul care suporta conexiunea)
  - Dezavantaj: pierdere in performanta.
    - Tot software-ul extra produce incetinire.
  - Avantajul principal (nu se pierd pachete) nu este strict necesar in LAN care sunt de incredere
- Drept consecinta, majoritatea sistemelor distribuite care sunt construite pentru utilizare intr-o singura cladire sau un campus utolizeaza protocoale fara conexiune.

Remote Method Invocation (RMI)

- Invoca metode ale obiectelor la distanta (adica objecte localizate in alte sisteme)
- Detaliile de retelistatica cerute explicit de programarea cu streamuri si socluri dispar si faptul ca obiectul este localizat la distanta este aproape transparent programatorului OO
- Programul server care controleaza obiectul la distanta inregistreaza interfata cu un serviciu de numire astfel facand interfata accesibila programelor client
- Interfata contine semanturile pentru metodele obiectelor pe care serverul doreste sa le faca disponibile public
- Programul client poate utiliza acelsi serviciu de numire pentru a obtine o referinta la aceasta interafta in forma unui stub
- Stubul este un surogat local ('stand-in'/ placeholder) pentru obiectul la distanta
- Pe sistemul la distanta va fi un alt surogat, scheletul.

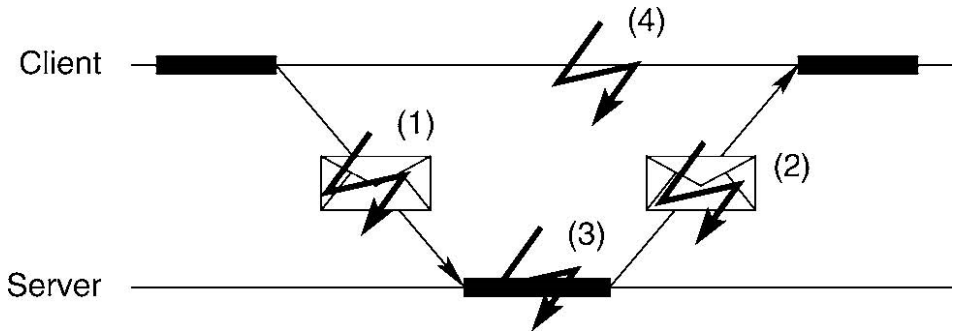


Detalii RMI

- Cand programul client invoca o metoda a obiectului la distanta, apare clientului ca si cum metoda este invocata direct asupra obiectului
- O metoda echivalenta este apelata de stub
- Stubul forwardeaza apelul si orice parametrii scheletului de la masina la distanta
- In Java numai tipurile de primitive si acele tipuri de referinte care implementeaza interfata *Serializable* pot fi utilizati ca si parametrii – serializarea acestor parametrii este numita marshalling (insiruire).
- La receptionarea sirului de octeti, scheletul converteste acest sir in apelul original al metodei si parametri asociati (deserializarea parametrilor fiind numita unmarshalling)
- La final, scheletul apeleaza implementarea metodei de la server.

Esecuri in comunicare

- pierderea de mesaje
- caderea unui proces



1. Pierderea mesajului cerere
2. Pirederea mejajului raspuns
3. Caderea serverului
4. Caderea clientului

Tipuri de esecuri (1/2)

1. Pierderea unui mesaj cerere:
   - Clientul trebuie sa retransmita mesajul dupa un timeout
   - Problema: clientul nu poate diferentia diferitele tipuri de esecuri
     - Ex: daca mesajul rezultat este cel care a fost pierdut, o retransmitere a mesajului cerere poate rezulta in executarea de doua ori a procedurii la distanta
     - Ex: proceduri care necesita mult timp vs. selectia unui timeout scurt
2. Pierderea unui mesaj raspuns:
   - Clientul retrasnmite cererea dupa un timeout.
   - Problema: daca serverul nu recunoaste ce s-a intamplat, executa procedura din nou

Tipuri de esecuri (2/2)

3. Caderea serverului:
   - Daca serverul cade datorita unei erori, trebuie determinat daca executia partiala a procedrii a produs deja efecte in stare.
     - Ex. Daca baza de date a fost modificata in timpul procedurii, problema recuperarii si a continuarii din momentul esecului nu este triviala
4. Caderea clientului:
   - Un proces client care cade in timpul executiei unui RPC este referit ca o *invocare orfana*
   - Problema: ce face serverul u rezultatele si unde trebuie sa le trimita

Semantica esecurilor

- Aplicatii diferite pot avea cerinte diferite in ceea ce priveste calitatea serviciului (QoS) in termeni de detectie a esecurilor si recuperare

Semantica eseculuui	*Operatie fara esec*	*Pierdere mesaj*	*Cadere server*
Maybe (Poate)	Executie: 1 Rezultat: 1	Executie: 0/1 Rezultat: 0	Executie: 0/1 Rezultat: 0
At-least-once (Cel putin odata)	Executie: 1 Rezultat: 1	Executie: >=1 Rezultat: >=1	Executie: >=0 Rezultat: >=0
At-most-once (Cel mult odata)	Executie: 1 Rezultat: 1	Executie: 1 Rezultat: 1	Executie: 0/1 Rezultat: 0
Exactly once (Exact odata)	Executie: 1 Rezultat: 1	Executie: 1 Rezultat: 1	Executie: 1 Rezultat: 1

Semantica *Maybe*

- Referita ca si *efortul cel mai bun*.
- Nu ofera mecanisme pentru tratarea esecurilor
- Ex. RPC poate fi executat o data sau ninciodata la partea server
- Clientul receptioneaza cel mult un rezultat
- Nu ofera garantii
- Atata timp cat nu apar esecuri, RPC este adersat adecvat

Semantica *At-least-once*

- RPC va fi executat la partea server cel putin odata in cazul pierderii de mesaje
- Dupa un timeout, clientul repeta RPC pana cand receptioneaza un raspuns de la server
- O procedura poate fi efectuata de mai multe ori la server
- E posibil ca un client sa receptioneze raspunsuri multiple datorate exeutiei repetate
- Nu ofera o confirmare daca serverul cade
- Adecvata pentru proceduri idempotente care nu cauzeaza schimbarea starii la server si pot fi executate mai mult decat o data fara a produce pagube

Semantica *At-most-once*

- Procedura va fi executata cel mult odata – atat in cazul pierderii de mesaje cat si a caderii serverului
- Daca serverul nu cade, sunt garantate exact o executie si exact un rezultat
- Necesita un protocol complex de gestiune a mesajelor si numerotare

Semantica *Exactly once*

- Cazul ideal, nu este usor de atins
- Invocarea de catre un client va rezulta intr-o singura executie la partea server si va fi furmizat un singur rezultat
- Dezirabila pentru tranzactii bancare
- Cazul cel mai simplu: operatii idempotente
  - Cazul unei simple infomrari care doar citeste data de la serverul de la distanta fara a schimba starea serverului
    - Executia repetata si numeroase mesaje rezultat nu sunt o problema