Folosirea de tehnici scalabile, peer-to-peer, pentru creşterea disponibilităţii serviciului oferit de reţelele SIP

Teză de doctorat

Voichiţa Almăşan
voichita.iancu@cs.pub.ro

Îndrumător: **Prof. Dr. Ing. Iosif Ignat** Iosif.Ignat@cs.utcluj.ro

Universitatea Tehnică Cluj-Napoca, Facultatea de Automatizări şi Calculatoare, Catedra de Calculatoare
Iulie 2011

Cuvinte cheie: serviciu larg disponibil, toleranţă la defecte, scalabilitate, echilibrare a sarcinii, peer-to-peer, SIP.

Rezumat

1 Introducere

Teza de fata, intitulata "Folosirea de tehnici scalabile, peer-to-peer, pentru cresterea disponibilitatii serviciului oferit de retelele SIP", prezinta principalele probleme ale unui serviciu oferit in cadrul unui sistem distribuit, intre care am identificat ca principala problema disponibilitatea serviciului. In urma identificarii si analizarii aspectelor care au impact asupra disponibilitatii serviciilor, se propun solutii bazate pe o infrastructura logica peer-to-peer, descentralizata, combinate pe alocuri si cu tehnici DNS. De-a lungul intregii lucrari, argumentam, inclusiv prin evaluarea performantelor contributiilor propuse, de ce consideram descentralizarea ca fiind cheia solutionarii disponibilitatii in sisteme distribuite. De asemenea, subliniem cum anume poate fi ea folosita, prin intermediul unui DHT de tip peer-to-peer, pentru a oferi un serviciu distribuit cu o larga disponibilitate. Asadar, in lucrarea de fata prezentam modul de obtinere pentru un serviciu de o larga disponibilitate prin utilizarea de tehnici de descentralizare. In acelasi timp, propunem si un mecanism de transparenta in ceea ce priveste descentralizarea, pentru clientii sistemului, prin oferirea catre acestia a unei imagini unice a sistemului (Single System Image). Tehnica generica de conferire a largii disponibilitati prin descentralizare este validata, in final, prin aplicarea si evaluarea acesteia in cazul unui server SIP.

Capitolul 2 realizeaza descrierea contributiilor deja existente in domeniului in care se plaseaza teza de fata. Printre altele, acest capitol isi propune sa defineasca, in viziunea noastra, notiunile cu care opereaza aceasta teza, asa cum sunt: (1) *toleranta la defecte*; (2) *scalabilitatea*; (3) *echilibrarea incarcarii*; (4) *securitatea sistemului distribuit*; (5) *larga disponibilitate a unui serviciu*, componenta a securitatii; (6) *elemente de consens distribuit*; (7) *elemente de teoria jocurilor*; (8) *bazele de date*; (9) *retelele peer-to-peer*, si in particular DHT-ul Chord; (10) *retelele SIP* (*S*ession *I*nitiation *P*rotool), protocolul de semnalizare utilizat in retelele VoIP, pentru care modul de functionare, componentele si rolurile acestora sunt prezentate in detaliu; si, nu in ultimul rand, (11) *stadiul actual al convergentei si compatibilitatii protocoalelor peer-to-peer si SIP*.

Urmatoarele 4 capitole prezinta in amanunt contributiile originale pe care le propunem in urma cercetarii desfasurate in cadrul studiilor doctorale. In capitolul 3 propunem o modalitate generica, originala, de distribuire si *descentralizare* a unui serviciu, spre a-l face pe acesta larg disponibil. In acest capitol explicam si integrarea in solutia finala a contributiilor prezentate detaliat in capitolele 4 si 5.

In capitolul 4 vom prezenta solutia noastra pentru obtinerea unei baze de date total descentralizate, care sa ofere o buna persistenta a datelor. Am insistat asupra acestei probleme in teza de fata, intrucat consideram ca stocarea datelor referitoare la clienti este o problema importanta, comuna tuturor sistemelor distribuite. Tot in capitolul 4, vom prezenta o solutie bazata pe votare si consens distribuit, care trateaza aspectul malitiozitatii unor noduri din cadrul unui sistem distribuit descentralizat. Problema consensului apare in orice sistem distribuit care nu beneficiaza de un server central de autentificare. Serverul central ar fi fost o entitate ce ar fi redus surplusul de disponibilitate a serviciului, anuland din beneficiile descentralizarii. Nu in ultimul rand, tot in capitolul 4, vom prezenta, la final, moduri de interogare de catre clienti a serviciilor distribuite, si in particular a bazelor de date care trebuie sa fie larg disponibile. Ne propunem ca acei clienti sa fie cat mai putin impactati de timpul de obicei mare, necesar transmiterii raspunsului. Ideea de baza a reducerii impactului de comunicare asupra clientului o reprezinta decuplarea.

In capitolul 5 vom prezenta modele de auto-extindere a retelei peer-to-peer, auto-extindere ce are drept scop principal o mai buna toleranta la defecte. Primul asemenea model de auto-extindere a infrastructurii logice, peer-to-peer, pentru serviciul generic descentralizat pe care dorim sa il oferim, urmareste conferirea unui factor de replicare mai ridicat pentru anumite date. Datele in cauza nu sunt alese la intamplare, ci sunt acele date in legatura cu care se decide, in mod descentralizat, ca sunt critice pentru existenta sistemului. Cea de-a doua modalitate de auto-extindere prezentata are ca scop principal o mai buna echilibrare a incarcarii nodurilor care alcatuiesc sistemul, avand ca o consecinta colaterala pozitiva o scalabilitate crescuta a serviciului oferit.

In capitolul 6 prezentam modul in care tehnica generica, prezentata in capitolul 3 se poate adapta pentru a face ca un serviciu particular, un server de prezenta SIP, sa devina larg disponibil.

In capitolul 7 se face o evaluare a tuturor contributiilor prezentate in capitolele anterioare ale lucrarii. Pornind de la acest capitol, vom putea argumenta, in capitolul 8, utilitatea contributiilor originale prezentate in scopul acestei teze, precum si performantele lor in a imbunatati disponibilitatea serviciile distribuite. Tot in capitolul de final 8, vom trage concluziile relative la avantajele si neajunsurile solutiilor originale propuse. Bineinteles, am indicat si principalele directii de imbunatatire si de dezvoltare ulterioara pe care am dori sa le urmam, pentru obtinerea, cu impact de performanta cat mai mic, a unui serviciu care sa ofere o disponibilitate cat mai mare clientilor sai.

2 Necesităţile şi problemele identificate în contextul disponibilităţii serviciilor

In acest capitol ne propunem sa descriem stadiul actual al cercetarii in domeniile aflate in stransa relatie cu subiectul de interes al acestei teze: obtinerea unei largi disponibilitati pentru un serviciu distribuit, cu aplicatii pentru un server SIP. Se descrie un tip foarte raspandit de sisteme distribuite, si anume sistemele distribuite de stocare a datelor. Prezentam acest tip de sisteme aici pentru ca sunt vitale pentru functionarea oricarui alt tip de sistem distribuit. In continuare se realizeaza o descriere a retelelor peer-to-peer, un tip particular de sistem distribuit, care a fost proiectat initial pentru stocare de date, evoluand mai apoi dincolo de aceasta functionalitate initiala. Motivul pentru care ne-am oprit atentia asupra lor este faptul ca retelele peer-to-peer au evoluat dincolo de scopul lor initial tocmai datorita faptului ca sunt mai flexibile si functioneaza pe un model *descentralizat*. Descrierea se incheie cu o analiza a modalitatilor in care retelele peer-to-peer ar putea rezolva problemele de disponibilitate enuntate in teza. Tot in capitolul de introducere dorim sa ne dedicam analizei bibliografice a protocolului SIP, care genereaza un tip specific de arhitectura de sistem distribuit, si a implementarilor care combina acest protocol cu conceptul de peer-to-peer. In finalul studiului bibliografic se prezinta cateva modalitati si tehnici deja existente prin care unui serviciu distribuit i se poate spori disponibilitatea. Prezentam aici elementele de la care am pornit in cercetarea noastra pentru gasirea unei metode generice si scalabile, care sa sporeasca disponibilitatea unui serviciu, inclusiv a serviciului SIP.

In concluzie, bibliografia acestei teze se refera atat la sistemele distribuite si la proble-

mele lor, cat si la modurile in care retelele peer-to-peer pot fi utilizate pentru a diminua sau chiar elimina dintre aceste probleme. S-a facut, in contextul studiului bibliografic, o exemplificare concreta pentru protocolul SIP, a problemelor care apar in sistemele distribuite in general. De asemenea, am analizat cateva metode existente de a face larg disponibil un serviciu distribuit. Am observat, insa, ca aceste metode nu sunt suficient de *generice*, de *flexibile* sau de *scalabile*, oferind, astfel, un motiv si o directie pentru teza de fata. Pornind de aici, in capitolele ce urmeaza vom defini elementele componente, precum si modul lor de asamblare, pentru o platforma generica de distribuire si descentralizare a unui serviciu, in scopul obtinerii unei mai largi disponibilitati pentru acesta. Ne bazam in acest demers pe neajunsurile metodelor identificate si analizate in capitolul de bibliografie a tezei, dar si pe idei din acelasi capitol, utilizate in conjunctie cu avantajele retelelor peer-to-peer. Rezultatul final al muncii de cercetare relative la un serviciu larg disponibil generic va fi validat aplicandu-l unui server SIP, particular.

3 Creşterea flexibilităţii, scalabilităţii şi echilibrarea încărcării unui serviciu, prin descentralizare transparentă

In capitolul de fata ne propunem sa descriem o solutie generica, aplicabila oricarui serviciu de retea, care prin *distribuirea* si *descentralizarea* elementelor componente, ofera o *larga disponibilitate* (“high availability”) clientilor sai. In acelasi timp, dorim sa mentinem imaginea de sistem sau *serviciu unitar* (SSI – *S*ingle *S*ystem Image) fata de clientii serviciului. De aceea, ne propunem ca descentralizarea sa se faca astfel incat ea sa fie *transparenta* pentru clienti, nepunandu-le acestora probleme. De asemenea, am dori ca, pentru a imbunatati un serviciu generic spre a fi unul larg disponibil, modificarile legate de interfata serviciului sa fie minime. In acest scop, descriem modul de interactiune intre noile componente, *descentralizate*, ale serviciului distribuit devenit de-acum larg disponibil. In fond, capitolul de fata prezinta modul original de asamblare, in vederea obtinerii unui serviciu larg disponibil, a unor contributii propri, ce vor fi descrise in capitolele 4 si 5, si evaluate in capitolul 7. Prezentam si cateva concluzii preliminare in ceea ce priveste modelul propus pentru distribuirea si descentralizarea serviciului. Subliniem din nou faptul ca ne-am propus aceasta descentralizare in scopul unei mai prompte si totodata corecte deserviri a clientilor serviciului.

Modelul de repartizare intre noduri a muncii acestora. Vom propune solutia noastra in ceea ce priveste pastrarea fata de lumea exterioara a imaginii de serviciu unitar si totodata, in mod transparent, in spatele acestei imagini, sa distribuim si descentralizam cat mai mult arhitectura unui serviciu centralizat generic. Am incercat, in solutia de fata, ca surplusul de comunicare introdus din cauza distribuirii sa fie cat mai mic posibil. Elementele folosite de noi in acest scop au fost: (1) un element de *distribuire a incarcarii*, ce va fi numit in continuare *“load balancer”*, sau *“dispatcher”*, care va incerca sa distribuie in mod egal si transparent incarcarea intre nodurile pe care dorim sa distribuim serviciul; si (2) o *retea peer-to-peer* bazata pe Chord, imbunatatita, care va fi prezentata succint in acest capitol, urmand ca avantajele sale sa fie detaliate tot in aceasta teza. Componenta originala numita *load balancer* va controla accesul la reteaua Chord, asa cum rezulta din figura 1. Modelul general original de distribuire a unui serviciu, prin descentralizare, in vederea cresterii disponibilitatii sale, este

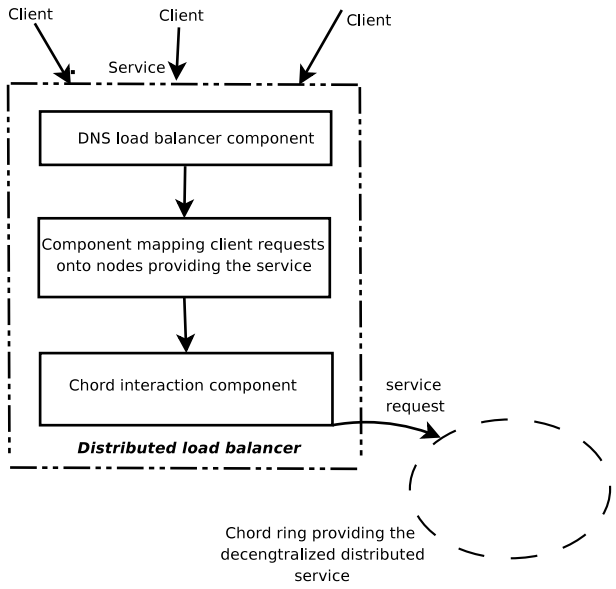

Figura 1: Arhitectura individuala a unui nod de echilibrare a incarcarii.

prezentat schematic in figura 2, fiind descris pe larg in teza. Schema de fata ar trebui sa fie suficient de generica pentru a face larg disponibil orice serviciu, cu modificari minime.

Asadar, in acest capitol se prezinta, pe scurt, viziunea noastra originala asupra elementelor necesare descentralizarii unui serviciu, precum si modul original ales pentru combinarea lor, toate in scopul obtinerii unui serviciu larg disponibil. Ne-am bazat viziunea pe avantajele de flexibilitate si dinamicitate caracteristice retelelor peer-to-peer, pe care le-am optimizat in maniera proprie. Am mentionat, de asemenea, pozitia retelei peer-to-peer in cadrul arhitecturii imbunatatite de serviciu distribuit, pentru a-i imbunatati disponibilitatea fata de clientii sai. In capitolele care urmeaza, si anume capitolele 4 si 5, vom detalia modul de functionare al acestor elemente originale, gandite ca niste componente independente. Aceste componente sunt, deci, refolosibile si in alte contexte. Am descris, dealtfel, in capi-

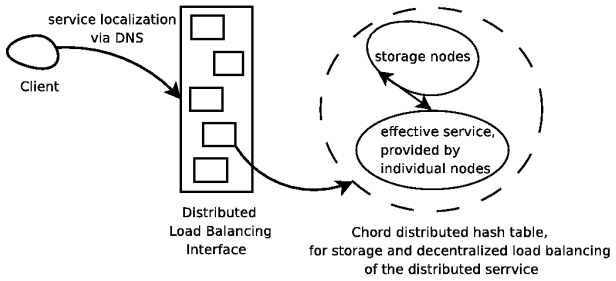

Figura 2: Metoda generala de distribuire si descentralizare a unui serviciu, pentru a-l face larg disponibil.

tolul 6, modul concret de imbinare si utilizare a acestor componente, conform sablonului prezentat in acest capitol, pentru a obtine un serviciu SIP larg disponibil.

4 Baza de date descentralizată pentru stocarea sigură şi persistentă a datelor unui serviciu larg disponibil

In capitolul 3 am prezentat descrierea generica a modelului de descentralizare a unui serviciu, in scopul obtinerii unei disponibilitati cat mai largi pentru acesta. Cu aceasta ocazie am subliniat faptul ca, pentru a asigura disponibilitatea serviciului in orice moment, trebuie ca *in orice moment* sa avem la dispozitie datele aferente fiecarui client al acelui serviciu. De aceea, este esentiala o *baza de date* care sa ofere o disponibilitate cat mai larga de acces la datele clientilor. Pentru aceasta, ne-am propus sa implementam *o baza de date distribuita*, tot *descentralizata*, care consideram ca reprezinta *o conditie necesara* pentru larga disponibilitate a oricarui serviciu.

In capitolul de fata prezentam solutia originala, propusa in cadrul acestei teze, pentru a obtine o baza de date distribuita descentralizata si de incredere, ce se bazeaza tot pe o retea Chord. Astfel, in sectiunea 4.1 a acestui capitol, prezentam arhitectura si modelul de interactiune propuse de noi, pentru a avea o baza de date cu un grad mare de disponibilitate. Baza de date propusa asigura o buna persistenta a datelor aferente clientilor serviciului distribuit, fara de care acesta nu ar putea fi larg disponibil. Pe urma prezentam o imbunatatire, elaborata tot in cadrul tezei de fata, pe care o propunem pentru baza de date larg disponibila. Imbunatatirea consta intr-o crestere a gradului de incredere in datele stocate, pe baza unor tehnici de consens in sisteme distribuite, adaptate pentru stocarea datelor bazata pe Chord. Nu in ultimul rand, prezentam o noua contributie originala, si anume viziunea noastra asupra modului ideal de proiectare a unui client pentru un serviciu larg disponibil. In particular, ne referim la clienti pentru o baza de date, ale caror cereri trebuie sa fie tratate in mod cat mai eficient si mai rapid de catre server-ul de baze de date. In general fie spus, pentru a optimiza timpul de raspuns perceput de client din partea server-ului, propunem decuplarea logica a elementului de interogare de restul clientului. O astfel de implementare de client a fost utilizata intern, in cadrul serviciului distribuit larg disponibil, pentru a optimiza comunicarea dintre nivelul care ofera propriu-zis serviciul si nivelul de stocare a datelor. Rolul si pozitia acestor 2 nivele au fost descrise in capitolul 3.

4.1 Construirea unei baze de date descentralizate, folosind o reţea Chord

Persistenta datelor, furnizata cel mai bine de catre *o baza de date distribuita*, este necesara in cazul oricarei aplicatii distribuite, si deci cu atat mai mult in cazul unui serviciu distribuit, in scopul de a nu pierde date referitoare la clientii acestuia. In plus, necesitatea unei baze de date distribuite si *descentralizate* se impune pentru a asigura cu succes stocarea, necesara aplicatiilor care proceseaza date. Din pacate, in prezent nu exista foarte multe solutii care sa garanteze o persistenta mare a datelor, intr-un mod transparent pentru aplicatie. Prin combinarea aspectelor de arhitectura pe care le-am descris si prin prezentarea de rezultate experimentale in capitolul 7, munca noastra in acest sens urmareste sa dovedeasca faptul

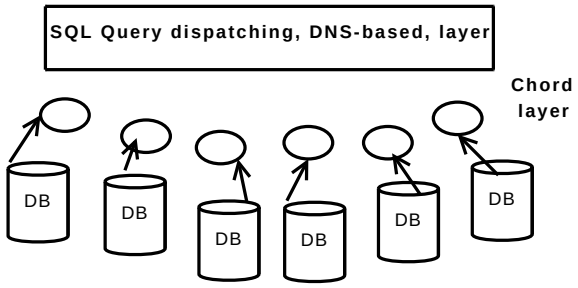

Figura 3: Arhitectura pe nivele a bazei de date distribuite.

ca DHT-ul Chord reprezinta o solutie viabila pentru problema stocarii datelor persistente, motiv pentru care consideram ca munca noastra are un impact in domeniul stocarii persistente, pe care ne propunem sa il exploatam si extindem si prin dezvoltari ulterioare. Modelul propus de noi poate fi vazut, reprezentat schematic, in figura 3.

Exista mai multe *avantaje* ale solutiei originale propuse de noi in teza, dintre care am dori sa subliniem: (1) faptul ca solutia propusa introduce un timp suplimentar de executie relativ mic, in comparatie cu bazele de date traditionale, centralizate; (2) faptul ca, in cazul in care avem un numar nu foarte mare de peer-i de stocare, conectati printr-o retea de mare viteza, performantele prototipului dezvoltat nu sunt influentate nici de catre dimensiunea retelei, nici de catre volatilitatea nodurilor; (3) faptul ca prin solutia dezvoltata de noi se ascunde locatia exacta a datelor si a copiilor lor fata de orice entitate exterioara bazei de date descentralizate, prin intermediul entitatii de distribuire a interogarilor SQL, pe care am definit-o in cadrul arhitecturii noastre; si (4) faptul ca solutia noastra garanteaza un model de tranzactii atomice pentru toate interogarile SQL suportate.

In ce priveste performantele solutiei propuse, pe viitor am dori sa *imbunatatim* aceasta baza de date distribuita si descentralizata, prin: (1) eliminarea necesitatii implicarii campului de cheie primara in orice interogare; (2) introducerea posibilitatii de a actualiza datele deja existente, folosind interogari de tip UPDATE, si adaptarea protocolului de coerenta in consecinta; (3) investigarea altor solutii *ne-statice* de echilibrare a sarcinii, pentru perfectionarea entitatii de distribuire a interogarilor SQL. Ne dorim, de asemenea, sa testam si performantele prototipului propus prin compararea rezultatelor obtinute in anumite conditii de stres cu cele obtinute de alte baze de date distribuite, care asigura si ele o anumita persistenta, asa cum este MySQL cluster, deoarece pana in prezent am comparat performantele solutiei noastre doar cu versiuni nedistribuite ale bazelor de date.

4.2 Îmbunătăţirea modalităţilor de stocare şi a securităţii informaţiei, pe baza teoriei jocurilor

Asa cum am afirmat deja, serviciile distribuite trebuie sa isi mentina niste date referitoare la fiecare client, dintre care unii clienti ar putea fi importanti, implicit datele lor fiind critice. Din acest motiv, ne dorim sa imbunatatim gradul de incredere al stocarii, prin proiectarea unui mecanism de stocare sigura, care sa mentina datele nealterate chiar si in prezenta unor

noduri malitioase, sau care nu functioneaza conform specificatiilor.

Propunem un algoritm original de vot distribuit, in spiritul consensului bizantin, algoritm in cadrul caruia se schimba foarte putine mesaje suplimentare intre noduri, performantele sale fiind evaluate in cadrul capitolului 7. Acest algoritm propus reprezinta solutia noastra la problema mentinerii *gradului de incredere* si a functionalitatii corecte a unei baze de date distribuite si descentralizate, in ciuda unor posibile atacuri de securitate. Algoritmul nostru poate identifica si poate exclude din randul nodurilor de stocare orice nod care nu functioneaza conform cerintelor de functionare ale acestui sistem distribuit. Deoarece algoritmul se afla in legatura doar cu nivelul Chord, ceea ce inseamna ca nu are nevoie sa cunoasca alte detalii despre baza de date distribuita din care face parte si DHT-ul Chord, solutia propusa de noi ar putea fi usor utilizata pentru a oferi un grad sporit de incredere pentru orice sistem distribuit descentralizat, care in mod normal ar fi avut nevoie de o autoritate centrala de autentificare.

Pe viitor, am dori sa investigam si mai in detaliu aspecte legate de durata necesara pentru ca topologia distribuita sa convearga la o topologie ce nu contine nici un nod malitios si, probabil, sa gasim chiar o formula matematica pentru a determina gradul de incredere, care sa sporeasca ritmul de convergenta al topologiei. Ne-am dori, de asemenea, sa adresam mai in detaliu si aspectul persistentei datelor care apartineau nodurilor malitioase, inainte ca acestea sa devina malitioase in urma vreunor atacuri, deoarece aceste noduri ar fi putut fi responsabile de date ce sunt critice pentru functionarea sistemului, si pentru care, in consecinta, ar fi de dorit sa luam masuri suplimentare ca sa nu le pierdem.

4.3 Propunere de tratare decuplată, la nivelul clientului, a interogărilor adresate unei baze de date

Chiar daca un serviciu distribuit beneficiaza de stocarea datelor clientilor sai intr-o baza de date rapida, totusi trebuie ca serviciul distribuit sa isi ia masurile necesare pentru cazul in care baza de date ar fi suprasolicitata de interogari ale clientilor pe care ii deserveste, crescand astfel timpul de raspuns pentru serviciul distribuit si descentralizat, si, in consecinta performantele acestuia.

Punctul slab in marea parte a sistemelor distribuite este faptul ca nodurile care le compun sunt conectate prin intermediul unor legaturi care nu sunt sigure. Peste aceste legaturi ne-am dori sa putem comunica in mod sigur, deoarece schimbarea de mesaje si solicitarea unor anumite servicii reprezinta ideea centrala pentru functionarea oricarui tip de sistem distribuit. Din aceste considerente, putem afirma cu destula certitudine ca legaturile de retea dintre nodurile unui sistem distribuit reprezinta componentele cele mai sensibile ale acestora, ele avand si probabilitatea cea mai mare de esec. Un tip de esec impotriva caruia este greu de luptat il reprezinta *lipsa disponibilitatii*, care ar fi putut fi generata fie de catre un atac de refuz al acordarii serviciului (*D*enial *o*f *S*ervice – DoS), fie din cauza unei proiectari necorespunzatoare a intregului sistem. Acest tip de esec s-ar putea datora fie unei gatuiri pe parcursul unei conexiuni de retea, fie unei supraincarcari a unui nod din partea caruia solicitam un serviciu. Cum gatuirile de retea pot fi impiedicate prin metode de imbunatatire a calitatii serviciilor (*Q*uality *o*f *S*ervice – QoS), supraincarcarea nodurilor ramane problema cu care ne preocupam aici, incercand sa o rezolvam utilizand unele metode de decuplare a

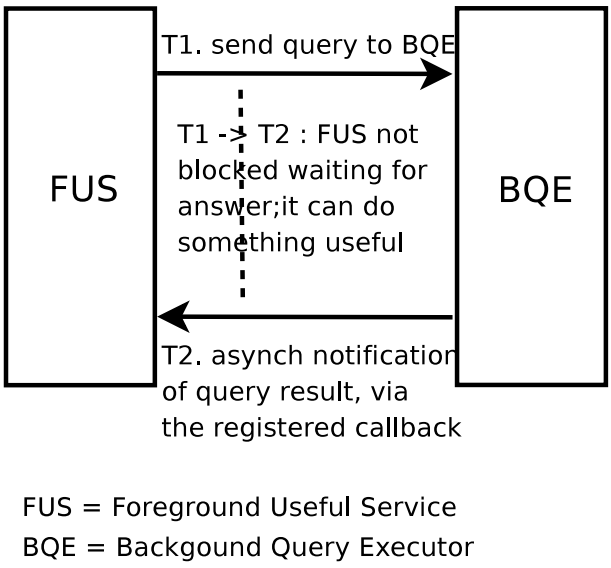

Figura 4: Descrierea schematica a mecanismului de decuplare.

entitatilor si, nu in ultima instanta, de echilibrare a sarcinii.

In teza descriem un nivel de decuplare din cadrul server-elor, care le separa pe acestea in serviciul util si in entitatea secundara de executie a interogarilor si care utilizeaza un mecanism de tip apel de procedura la distanta (*R*emote *P*rocedure *C*all – RPC). Schematic, aceasta decuplare poate fi vazuta in figura 4. Cele doua nivele descrise interactioneaza utilizand o interfata simpla, ce ofera functionalitati similare cu cele oferite de catre DMA (*D*irect *M*emory *A*ccess) la nivelul hardware. Am evaluat performantele prototipului obtinut de noi pentru decuplarea executiei interogarilor in capitolul 7, si am estimat si timpul suplimentar introdus prin aceasta decuplare. Se poate observa faptul ca nivelul de decuplare are performante bune, timpul suplimentar introdus fiind nesemnificativ in comparatie cu durata executiei efective a interogarii. De asemenea, se poate observa faptul ca arhitectura propusa de noi minimizeaza si riscul anumitor atacuri de tip DoS.

In completarea solutiilor preexistente, care se concentreaza mai ales pe optimizari ce se pot face la nivelul executiei interogarilor pe server-ele vizate si pe optimizari la nivelul legaturilor de retea, solutia noastra a incercat sa adreseze cat mai generic problema decuplarii executiei interogarilor de catre un serviciu distribuit. Testarea s-a realizat pe un sistem distribuit VoIP, si anume un server SIP. Motivul utilizarii unui server SIP a fost doar faptul ca aceste tipuri de server-e au nevoie sa efectueze foarte des multe interogari, atat de baze de date, cat si ale sistemului DNS.

Pentru a imbunatati rezultatele obtinute de aceasta versiune a prototipului nostru, in urmatoarea sa versiune dorim sa optimizam anumite aspecte, cum ar fi modul efectiv de solicitare a executiei interogarilor, prin mai multa interactiune cu arhitectura de intrare-iesire de nivel jos, cand se transmite interogarea. Dorim sa realizam, de asemenea, mai multe masuratori ale prototipului nostru pe sisteme multiprocesor si multi-core, in medii de executie cat mai apropiate de cele reale. In plus, deoarece dorim sa adresam o gama multipla de inte-

rogari, ne propunem ca pe viitor sa definim si un limbaj generic de formulare a interogarilor si a raspunsurilor la acestea.

5 Exploatarea dinamicităţii reţelelor peer-to-peer, pentru o mai bună stocare a datelor

In acest capitol ne propunem sa exploatam in detaliu dinamicitatea intrinseca a retelelor peer-to-peer de tip DHT, propunand solutii originale, ca extensii pentru DHT-ul Chord. Reamintim faptul ca acest DHT este utilizat pentru a implementa o baza de date descentralizata, pentru a fi *larg disponibila*. Baza de date in cauza este prezentata in amanunt in capitolul 4. Ceea ce aduce nou acest capitol sunt completari originale ale protocolului Chord, cu scopul de: (1) a spori siguranta de stocare prin sporirea *tolerantei la defecte* a sistemului, asigurandune astfel ca nu se pierd date, asa cum vom vedea in sectiunea 5.1, precum si (2) cu scopul de a *echilibra*, in mod transparent si *scalabil*, incarcarea individuala a nodurilor din reteaua de stocare, asa cum se prezinta in sectiunea 5.2. Ambele aspecte prezentate in acest capitol se bazeaza pe capacitatea de a se auto-extinde nelimitat si in mod transparent, pe care o au DHT-urile. Aceste sisteme distribuite reusesc sa inglobeze orice nod care este "dispus" sa indeplineasca rolul de peer.

Asadar, in capitolul de fata se prezinta contributii originale reprezentand extensii ale Chord, bazate pe capacitatea sa de auto-extindere. Ambele contributii sporesc toleranta la defecte precum si echilibrarea incarcarii, avantaje aplicabile si nodurilor din cadrul bazei de date prezentate in capitolul 4. Astfel, acest capitol adreseaza probleme de disponibilitate prin: (1) toleranta la defecte, (2) echilibrarea incarcarii si (3) scalabilitate ale bazei de date aferente serviciului distribuit. Combinat cu capitolul 4, capitolul de fata prezinta modificarile care au trebuit aduse unui DHT Chord clasic, spre a-l adapta cat mai bine la functia de stocare cu grad de disponibilitate si de incredere cat mai mare. In final, toate acestea ajuta ca modelul de descentralizare a unui serviciu distribuit, prezentat in capitolul 3, sa transforme acel serviciu intr-unul larg disponibil, fapt ce reprezinta obiectivul central al acestei teze.

La fel ca in cazul elementelor prezentate in capitolul 4, fiecare contributie originala prezentata in acest capitol poate fi utilizata impreuna cu celelalte elemente nou-introduse de aceasta teza. In particular, aceste elemente pot fi utilizate in conjunctie cu baza de date descentralizata, prezentata in capitolul 4. De asemenea, ele pot fi utilizate si individual, in alte tipuri de sisteme decat cele pentru care ele au fost initial proiectate, in cadrul acestei teze. Din acest motiv, elementele in cauza au fost descrise independent unul de celalalt, in ciuda permanentei interactiuni care are loc intre ele, conform modelului descris in capitolul 3.

5.1 Îmbunătăţirea siguranţei de stocare a unei reţele Chord, prin auto-extindere

In ceea ce priveste stocarea datelor referitoare la clientii mai importanti, la care ne-am referit deja ca fiind "date critice" sau "date sensibile", discutam in amanunt in teza ca pentru acestea ar fi necesar un grad de replicare mai mare, pentru a spori certitudinea ca aceste date nu se vor pierde in urma esecurilor individuale ale nodurilor.

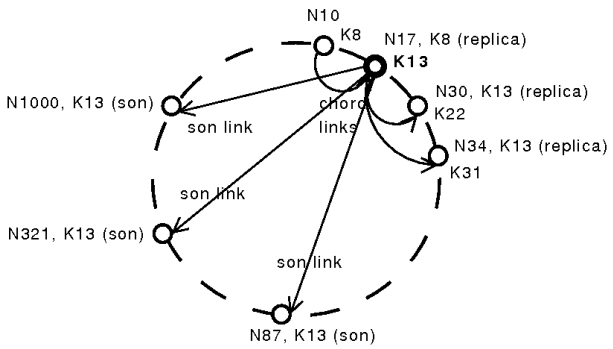

Figura 5: Stocarea utilizand un DHT de tip peer-to-peer, imbunatatit.

Problema stocarii cu un grad de redundanta diferit, in functie de datelor stocate, a fost adresata si in contextul aplicatiilor peer-to-peer care ruleaza peste grid. In acest context, ideea principala din spatele solutiilor elaborate a fost ca aplicatia de nivel inalt sa solicite un grad de redundanta, care la randul sau se traduce, la nivelul overlay-ului peer-to-peer utilizat, printr-un numar de noduri fizice de stocare diferite. Daca serviciul de partaj de date pe grid isi da seama ca nu are destui peer-i instantiati pentru a asigura gradul de redundanta cerut, acesta va realiza integrarea unor noi peer-i in overlay-ul peer-to-peer, interogand si interactionand cu sistemul de gestiune a resurselor grid-ului. Rezervarea nodurilor suplimentare solicitate, necesare asigurarii gradului dorit de redundanta, se face in maniera best-effort, deoarece nimeni nu ne poate asigura faptul ca la un moment dat *grid-ul insusi* are disponibile toate resursele dorite. In urma rezervarii nodurilor, serviciul peer-to-peer de stocare va fi extins astfel incat sa le cuprinda si pe acestea, urmand ca mai apoi datele sa poata fi stocate si pe noile noduri, in conformitate cu gradul de redundanta solicitat.

Avantajul unei solutii ca cea prezentata succint mai sus, utilizata pentru auto-extinderea la cerere a unui serviciu de partaj de date pe grid, il constituie faptul ca nu trebuie rezervate de la inceput toate nodurile de care serviciul distribuit ar putea avea nevoie din partea grid-ului, ceea ce duce la un consum de energie mai mic din partea aplicatiei luate in ansamblu, dar si la o mai eficienta partajare a resurselor grid-ului. Cele 2 dezavantaje pe care le vedem in ceea ce priveste arhitectura distribuita de autoextindere la cerere sunt: (1) faptul ca aceasta solutie nu se poate adapta usor spre a fi folosita pentru un set oarecare de noduri, care nu apartin neaparat unui grid; si (2) faptul ca aceasta solutie nu este intim conectata cu dinamica topologiei peer-to-peer, lucru care ar putea fi de dorit, intrucat ar fi avantajos sa putem reconfigura mai rapid si mai simplu intreaga topologie distribuita, in conditiile volatilitatii nodurilor. Avand in vedere aceste devavantaje identificate, si tinand cont de avantajele mentionate deja pentru o asemenea solutie de auto-extindere la cerere a unui serviciu distribuit, propunem un model original, proiectat si implementat de catre noi, care poate fi utilizat in scopul auto-extinderii unei infrastructuri de stocare distribuite si descentralizate, ce se bazeaza pe paradigma peer-to-peer. Mai mult chiar, nivelul de replicare pentru acest model a fost proiectat in special pentru a se afla intr-o cat mai stransa legatura cu DHT-ul utilizat, pentru ca solutia noastra sa poata folosi usor mecanismele intrinseci de stabilizare ale infras-

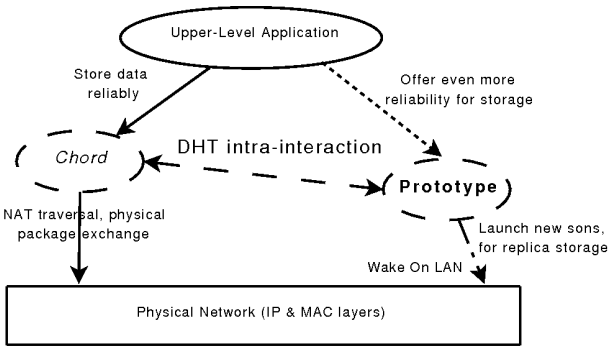

Figura 6: Interactiuni intre diversele entitati implicate in stocarea persistenta a datelor.

tructurii DHT, pentru cazurile de noduri volatile. Solutia noastra beneficiaza, deci, atat de avantajele de "regenerare de la sine" ale infrastructurii Chord, cat si de mecanismul special proiectat de noi, care este capabil sa extinda serviciul de stocare distribuit si descentralizat si pe alte noi noduri, asa cum se poate vedea schematic in figura 5. Rezulta un prototip care are o interactiune stransa cu DHT-ul, asa cum reiese din figura 6, in scopul de a reusi sa stocheze intr-o mainera mai eficienta date sensibile, carora nu le este permis sa dispara din cadrul infrastructurii de stocare coordonate de catre DHT.

Consideram ca principala contributie obtinuta prin munca originala descrisa in detaliu in teza, care este mai apoi evaluata in capitolul 7, este faptul de a fi gasit o maniera de a rezolva problema pierderii datelor critice, prin stocarea acestora pe nodurile nou-integrate in cadrul infrestructurii peer-to-peer. Bineinteles, infrastructura astfel obtinuta nu este nici ea 100% sigura, dar cu toate acestea ea ofera un grad de siguranta mult mai mare decat un DHT clasic. Surplusul de siguranta este obtinut exploatand un alt aspect important continut de munca noastra, si anume utilizarea unui grad de replicare variabil, in functie de nevoile fiecarei date individuale, ceea ce permite sistemului sa integreze noi noduri de stocare *doar* cand acest lucru este necesar, fiind dictat de catre dinamica interna a aplicatiei de nivel inalt cu care sistemul distribuit si descentralizat de stocare interactioneaza. Dealtfel, unul dintre rationamentele crearii prototipului nostru a fost sa dovedeasca, prin intermediul masuratorilor pe care le-am efectuat asupra sa in capitolul 7, costurile implicate in raport cu performantele oferite de o asemenea infrastructura generica de auto-extindere, care implica integrarea de noi noduri in infrastructura logica de tip peer-to-peer doar in momentul in care acestea sunt absolut necesare. Am vazut faptul ca timpul suplimentar introdus de catre aceasta solutie, in cadrul unui set de noduri conectate printr-o retea locala, este destul de mic, mai ales daca luam in considerare si beneficiile oferite de mecanismul de replicare imbunatatit, ceea ce ne permite sa speram ca acest prototip reprezinta un bun candidat pentru a indeplini necesitatile de stocare ale datelor sensibile din cadrul unui sistem descentralizat cu o larga distributie geografica.

Pe viitor, ne-am dori sa ne concentram atentia asupra adaptarii prototipului nostru pentru a rula pe o infrastructura logica avand o mai larga distribuire geografica, pentru a putea masura performantele sale si intr-un asemenea context. De asemenea, ne-am dori sa reali-

zam niste experimente mai detaliate referitoare la influenta pe care o are dimensiunea infrastructurii logice de tip peer-to-peer asupra probabilitatii de a pierde datele stocate in mod redundant. Inainte de a realiza asemenea experimente, am putea face si unele optimizari, pentru a maximiza avantajele utilizarii modelului nostru de auto-extindere, cum ar fi: (1) investigarea unor posibilitati de a reduce numarul de mesaje schimbate de catre peer-i, cum ar fi cazul mesajelor schimbate intre peer-ul fiu si parintele, sau cazul mesajelor schimbate intre posesorul unei chei si succesorii acestuia, care pastreaza si ei copii ale datelor aferente cheii; si (2) paralelizarea cererilor de lansare a noi noduri, ce vor fi integrate ca peer-i membri ai infrastructurii Chord, in cazul cererilor ce se refera la chei diferite, aflate in gestiunea nodului curent.

5.2 Auto-extinderea DHT-ului Chord, în scopul echilibrării încărcării

O problema care apare in cadrul oricarui sistem distribuit, si implicit in cadrul serviciilor distribuite si descentralizate, este aceea a gasirii unei modalitati eficiente de utilizare a tuturor nodurilor din sistem, astfel incat sa se evite supraincarcarea. In particular, ne preocupa rezolvarea problemei supra-incarcarii si suprasolicitarii unui nod, din ale carui indatoriri am dori sa pasam si altor noduri, in cazul in care nodul in discutie depaseste un anumit prag de incarcare. In sectiunea 5.1 descriem un mecanism de *auto-extindere*, utilizat pentru a spori siguranta de stocare in special pentru anumite date sensibile, prin cresterea gradului de redundata al acestora. Cunoscand proprietatile de dispersie ale DHT-ului, descrise in studiul bibliografic al acestei teze, vom descrie in amanunt modul in care auto-extinderea unei infrastructuri peer-to-peer ar putea fi benefica pentru echilibrarea incarcarii nodurilor care o compun, fapt ce a fost si evaluat in capitolul 7.

Datorita faptului ca *functia de dispersie* este cea care este utilizata pentru a lua decizia referitoare la apartenenta unor date la un peer, spre a fi responsabil de stocarea lor, putem privi aceasta functie ca reprezentand ea insasi un *load balancer distribuit*, pozitionat in fata retelei Chord, jucand un rol similar cu cel al inregistrarilor de tip DNS NAPTR si SRV pozitionate in fata unui serviciu distribuit de retea, asa cum am vazut in capitolul 3. Ne-am dorit, deci, sa exploatam beneficiile oferite de functia de dispersie, spre a rezolva problema echilibrarii incarcarii nodurilor.

Astfel, am elaborat si mai apoi evaluat un model de echilibrare a sarcinii, prin imbunatatirea unui DHT Chord, permitand acestui DHT sa se adapteze mai bine in conditiile necesitatii stocarii unor cantitati crescande de date. Am realizat aceasta imbunatatire pentru Chord prin adaugarea unei capacitati suplimentare de stocare, in momentul in care ea a fost necesara, asa cum reiese din figura 7. Aceasta capacitate de stocare este materializata fie de: (1) noi peer-i integrati in DHT prin intermediul mecanismului de auto-extindere; fie de (2) spatiu fizic de stocare, cum ar fi adaugarea unui hard disk suplimentar sau a unui hard disk mai mare pe nodul suprasolicitat. Ambele aceste metode de echilibrare fac ca DHT-ul sa accepte mai usor si mai repede sa stocheze datele clientilor sai, din punctul acestora de vedere.

Principala contributie pe care o aduce solutia prezentata succint mai sus o reprezinta faptul de a *imbunatati* capacitatea naturala a infrastructurii de tip DHT de a echilibra incarcarea nodurilor membre, prin faptul de a permite acestei infrastructuri sa se auto-extinda.

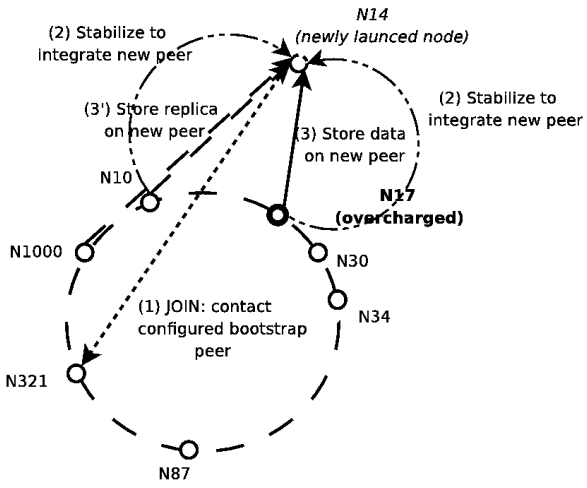

Figura 7: Adaugarea unui nou peer Chord, pentru a ajuta un peer supraincarcat.

Auto-extinderea se realizeaza prin adaugarea in principal de peer-i suplimentari, sau si de elemente fizice de stocare, ceea ce duce la o mai buna echilibrare a incarcarii intre peer-i si deci la evitarea aparitiei de peer-i supraincarcati sau suprasolicitati. Mecanismul proiectat de noi este complementar celui oferit implicit de DHT-urile Chord obisnuite, acesta fiind de asemenea transparent din punctul de vedere al clientilor DHT-ului, asa cum sunt si mecanismele interne ale acestuia. Originalitatea mecanismului nou-introdus consta in faptul ca reuseste sa extinda infrastructura in mod dinamic, modificand-o exact acolo unde este nevoie, prin utilizarea unui set maximal de peer-i disponibili, fapt ce face ca intregul sistem Chord imbunatatit sa devita mai scalabil si deci mai capabil decat inainte de a oferi o larga disponibilitate fata de clientii sai.

Pe viitor, ne-am dori sa ne concentram atentia asupra adaptarii, testarii si evaluarii modelului de echilibrare a sarcinii prezentat, pentru a rula pe o infrastructura logica distribuita geografic. De asemenea, ne-am dori sa gasim mijloace de a suprasolicita peer-ii cu cat mai multe cereri, spre a vedea posibile puncte slabe pe care le are solutia prezentata de echilibrare a incarcarii, si a vedea in ce momente sistemul ar putea deveni indisponibil sau partial indisponibil, pentru a fi constienti de modurile de a ataca sistemul prin atacuri de tip DoS, si in consecinta a putea preveni asemenea atacuri. Un alt aspect asupra caruia am dori sa ne concentram atentia este determinarea celei mai bune pozitii pentru nodul nou-lansat, astfel incat el sa preia cat mai multe din indatoririle nodurilor supraincarcate, aspecte pe care dorim sa le analizam in contextul a diverse situatii de distributie pentru cheile si pentru dimensiunile datelor.

6 Aplicarea modelului de descentralizare a unui serviciu pentru un server SIP

In capitolul 3 se prezinta modul generic de imbinare a elementelor prezentate in capitolele 4 si 5. Pe baza capitolelor anterioare, in capitolul de fata dorim sa validam modelul generic de descentralizare a unui serviciu, spre a-l putea folosi pentru un server SIP de presence cu disponibilitate larga. Astfel, descriem modul original in care am reusit sa facem ca serviciul de presence oferit intr-o retea SIP sa fie larg disponibil. Pentru a obtine aceasta larga disponibilitate am utilizat, bineinteles, *tehnicile peer-to-peer* definite pe parcursul acestei teze.

Scopul ce rezulta din chiar titlul tezei de fata este acela de a realiza, si mai apoi valida, o distribuire si o descentralizare a unui server SIP, in vederea cresterea disponibilitatii acestuia. S-a vorbit deja, in capitolul 3, despre metodologia generica de a distribui un serviciu in scopul de a-l face larg disponibil clientilor sai, fara a modifica ceva din modul de oferire al serviciului, in particular fara a schimba ceva in *protocolul de comunicare client-server* definit de serviciul respectiv. Analizand cu atentie modelul de distribuire generica a unui serviciu, se observa faptul ca, in cazul unor protocoale orientate pe conexiune, va trebui ca, daca mesajul initial al clientului a trecut printr-un nod din load balancer si mai apoi a ajuns la nodul Chord care ofera si serviciul respectiv, toate mesajele din partea acelui client sa treaca prin load balancer inainte de a ajunge la nodul Chord. Pe de alta parte, din cauza orientarii pe conexiune a serviciului, load balancer-ul nu va putea realoca un client unui nou peer Chord la primirea unui mesaj oarecare, decat eventual in situatia speciala de esec a primului nod Chord, situatie ce va fi transparenta pentru client, dar ar putea implica eventuale retransmisii din partea acestuia. Intr-o asemenea situatie se realizeaza o incarcare inutila a load balancer-ului, pentru cazul general, in care peer-ul responsabil de un client nu paraseste reteaua, iar scopul nostru este tocmai sa evitam incarcarea excesiva a unei componente a sistemului, care ar putea duce la gatuirea acestuia. De aceea, deoarece componenta protocolului SIP pentru stabilirea de noi sesiuni se bazeaza tocmai pe existenta unei conexiuni, *sesiunea* insasi, din cauza ca *nu ne propunem sa modificam protocolul SIP in aceasta lucrare*, vom exeplifica distribuirea unui serviciu SIP in scopul unei largi disponibilitati prin *descentralizarea si distribuirea in scopul unei mult mai largi disponibilitati a unui server SIP de presence*, serviciu ce se preteaza foarte bine la descentralizarea folosind Chord si metoda de echilibrare a incarcarii descrisa pe larg in capitolul 3. Motivul pentru care nu ne-am propus sa implementam si distribuirea unui proxy de stabilire a sesiunii si de routare SIP este: (1) pe de-o parte, faptul ca, desi acest lucru este posibil folosind metoda generica de distribuire a unui serviciu pentru obtinerea unei largi disponibilitati, ar implica un numar mare de mesaje suplimentare intre nodul Chord ce va deservi un client SIP si acel client SIP; iar (2) pe de alta parte, faptul ca daca dorim sa evitam acele mesaje suplimentare ar fi necesare schimbari in protocolul SIP, cum ar fi introducerea de noi campuri in mesajele de stabilire si de incheiere a sesiunii (INVITE, respectiv BYE sau CANCEL), modificari pe care nu dorim sa le facem, pentru ca am pierde compatibilitatea cu clientii SIP deja existenti. Asadar, in continuare ne vom ocupa de distribuirea, in mod transparent pentru clienti, in scopul obtinerii unei largi disponibilitati a serviciului, a unui presence server, folosind modelul generic prezentat in capitolul 3. Preluand ideea din acest capitol si adaptand-o pentru SIP, cheia reprezentand ID-ul Chord al unui client SIP va fi obtinuta aplicand functia de hash a DHT-ului Chord pe *URI*-ul (*U*niversal *R*esource *I*dentifier) clientului SIP in cauza.

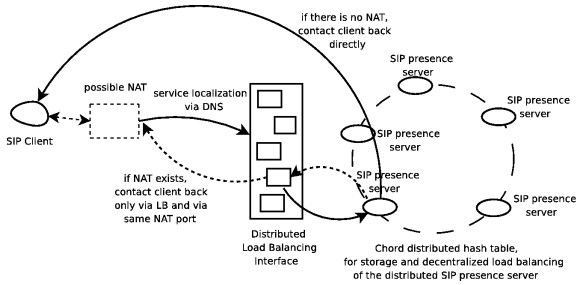

Figura 8: Distribuirea dinamica si echilibrata a clientilor intre mai multe server-e de presence SIP, aflate pe un inel Chord.

In cazul distribuirii dinamice a unui presence server, vom avea un load balancer de tipul celui descris in capitolul 3 in fata inelului Chord pe care ruleaza server-e de presence. Fiecare nod care este trecut in intrarea DNS aferenta serviciului de presence server, adica fiecare nod care indeplineste functionalitatea de *echilibrare a incarcarii*, va functiona dupa urmatorul algoritm simplu, care este ilustrat schematic si in figura 8: (1) va prelua mesaje de la clientii SIP de presence, prin intermediul unei componente minimale de presence server SIP; (2) pentru fiecare client SIP de presence va parsa header-ul mesajului SIP, pentru a identifica URI-ul UAC-ului (*U*ser *A*gent *C*lient) SIP curent si eventual al UAC-ului a carui stare se doreste a se afla, cat si metoda SIP prezenta in mesaj: PUBLISH sau SUBSCRIBE; (3) daca metoda este PUBLISH, URI-ul ce se va folosi in continuare este cel al clientului sursa, iar daca metoda este SUBSCRIBE, URI-ul folosit este celalalt URI care apare in mesaj; (4) pe baza URI-ului SIP care trebuie utilizat, se va calcula cheia sa in inelul Chord, folosind functia de hashing a acestui DHT; (5) nodul dispatcher curent trimite, prin intermediul componentei sale Chord, intregul mesaj al clientului catre peer-ul Chord responsabil de cheia acestui client, peer pe care ruleaza si presence server-ul ce deserveste clientul; (6) presence server-ul de pe peer-ul care a primit mesajul de presence raspunde corespunzator clientului, astfel: (6.a) daca clientul se afla in spatele unui NAT (*N*etwork *A*ddress *T*ranslation), atunci server-ul de presence *retine* acest lucru relativ la clientul in cauza, in *baza de date distribuita* bazate pe Chord, in datele persistente ale acestuia, iar raspunsul va fi trimis trecand din nou prin nodul dispatcher; sau (6.b) altfel, raspunsul va fi trimis direct catre client, pentru ca acesta sa poata contacta deacum inainte direct nodul pe care ruleaza server-ul de presence responsabil pentru cererea sa. Se observa ca NAT-ul pune probleme comunicarii elegante de tip peer-to-peer, problema cu care, dealtfel, se confrunta toate protocoalele peer-to-peer existente.

In urma receptionarii unui mesaj de tip PUBLISH, daca s-au primit si mesaje de tip SUBSCRIBE pentru clientul care a facut PUBLISH, clientii care s-au inscris ca sa afle starea de prezenta a acestuia vor fi notificati de catre server-ul de presence printr-un mesaj de tip NOTIFY. Notificarea clientilor se face fie direct, daca clientul in cauza nu se afla in spatele unui NAT, fie folosind dispatcher-ul, in caz contrar, si anume exact nodul din dispatcher prin care clientul in cauza a contactat serviciul de server presence, din cauza ca, in majoritatea cazurilor, doar acelui nod din intreaga arhitectura a server-ului de presence NAT-ul ii permite sa contacteze inapoi clientul.

Nu am mai insistat aici asupra modului de imbinare a componentelor individuale, prezentate de-a lungul acestei teze in capitolele 4 si 5, intrucat aceasta respecta ceea ce s-a definit in capitolul 3. Dorim sa mentionam ca si aplicarea modelului generic pentru a crea un serviciu SIP larg disponibil reprezinta o contributie originala, dezvoltata in cadrul acestei teze.

7 Rezultatele evaluării unui serviciu descentralizat pentru a deveni larg-disponibil

In acest capitol vom prezenta, in ordine, rezultatele evaluarii experimentale a elementelor pe care le-am descris deja in capitolul 4, si in capitolul 5, precum si in capitolele 3 si 6. Facem acest lucru pentru a putea ajunge la o concluzie legata de eficienta tehnicilor si metodelor utilizate pentru descentralizarea generica a unui serviciu, in scopul sporirii disponibilitatii sale. Elementele pe care am urmarit sa le imbunatatim, pentru a ajunge la un serviciu larg disponibil, si care vor fi evaluate in cele ce urmeaza, sunt: (1) persistenta datelor pe care serviciul in cauza le stocheaza, punandu-le la dispozitia clientilor sai; (2) decuplarea componentei serviciului care realizeaza interogari, imposibil de evitat, cum ar fi interogari de DNS; (3) gradul de incredere si siguranta datelor pastrate de catre serviciu, chiar si in conditiile unor amenintari de securitate sau a unor noduri care functioneaza incorect; (4) toleranta la defecte si scalabilitatea, prin auto-extindere, pentru a putea asigura nevoile sporite de persistenta, pe care le au datele critice sau sensibile; (5) echilibrarea sarcinii, statica sau dinamica prin auto-extindere, in scopul mentinerii disponibilitatii largi a serviciului, chiar si in contextul cresterii solicitarii acestuia de catre clienti; si, nu in ultimul rand, (6) combinarea optimizarilor realizate asupra tolerantei la defecte a serviciului, a scalabilitatii sale, a echilibrarii incarcarii sale, precum si a gradului de persistenta a datelor cu care serviciul opereaza, in scopul obtinerii unei solutii cat mai generice, de descentralizare, ce se poate aplica oricarui serviciu centralizat, spre a-i spori disponibilitatea.

Mentionam ca toate aspectele evaluate in acest capitol sunt *ortogonale*, ceea ce are ca o consecinta faptul ca ideile originale introduse sunt mai usor de urmarit daca le discutam si descriem separat. In acest capitol ne-a fost mai usor chiar si sa le evaluam separat, fara ca in acest mod sa negam in vreun fel posibilitatea ca ele sa fie utilizate impreuna. Asa cum am mentionat deja, impreuna, aceste aspecte ofera toate premisele pentru ca serviciul are le utilizeaza, distribuit in mod descentralizat, sa devina cat mai larg disponibil pentru clientii sai.

7.1 Rezultatele evaluării bazei de date descentralizate

7.1.1 Evaluarea propriu-zisă a bazei de date descentralizate

In aceasta sectiune vom evalua prototipul descris pe larg in capitolul 4.1. Ne intereseaza in special cat de potrivita este infrastructura logica Chord pentru a gestiona si controla o baza de date distribuita si descentralizata, principala noastra preocupare fiind timpul de raspuns suplimentar pe care aceasta solutie il introduce, in comparatie cu timpul de raspuns al unui server de baze de date traditional, centralizat, care ruleaza pe un singur nod.

Server-ul descentralizat de baze de date pe care l-am proiectat si evaluat este compus dintr-un set de noduri pe care ruleaza server-e MySQL individuale, controlate fiecare de catre un peer Chord, asa cum am descris in arhitectura prezentata in sectiunea 4.1. Am variat numarul de peer-i Chord si, in consecinta, gradul de descentralizare al bazei de date distribuite, utilizand dimensiuni intre 1 si 10 noduri, care toate apartineau unei aceleasi retele locale de mare viteza, pentru a nu trebui sa ne preocupam prea mult, cel putin deocamdata, de timpul suplimentar necesar schimbului de mesaje implicat de comunicarea dintre peer-i. Avand aceasta arhitectura, precum si cate o componenta de distribuire a interogarilor care ruleaza pe fiecare peer Chord, conform arhitecturii prototipului nostru, am realizat urmatoarele experimente: (1) am masurat timpul suplimentar introdus in cazul solicitarii de catre un client a unor date din baza de date distribuita si descentralizata; si (2) am masurat timpul suplimentar introdus pentru a mentine datele corect replicate, in conditiile in care se pot insera noi date in baza de date distribuita si descentralizata. Rezultatele acestor 2 experimente sunt ilustrate in figurile 9 si 10. Trebuie mentionat faptul ca valorile urmatorilor parametri nu au fost considerate atunci cand s-a facut graficul din figurile 9 si 10: (1) latenta retelei noastre de test, care era in jurul valorii de 2ms, pentru cazul comunicarii intre oricare pereche de noduri; (2) intervalul de stabilizare predefinit, specific retelei Chord, care reprezinta intarzierea maxima admisa pana cand un peer isi da seama ca unul dintre vecinii sai s-a schimbat, interval care a fost setat la 50ms; (3) timpul necesar realizarii de catre un nod MySQL server a unei interogari de tip SQL, timp egal in medie cu 10ms, pentru cazul celor mai simple interogari pe care le-am incercat, si egal in medie cu 500ms pentru interogarile cele mai complicate pe care le-am incercat; (4) timpul necesar download-ului concret al copiei unor date de pe un nod MySQL server pe un altul, timp care este important sa fie cunoscut in cazul in care exista o volatilitate neneglijabila a peer-ilor Chord, si pe care l-am considerat ca fiind aproximativ egal cu timpul mediu necesar pentru a trimite intre peer-i un pachet IP de dimensiune medie, timp care am afirmat deja ca in cazul retelei noastre de test a fost egal cu aproximativ 2ms. Nu in ultimul rand, mentionam ca dimensiunea datelor cu care lucram, pe care le stocam si care pot fi obtinute din baza de date, este suficient de mica pentru a putea incapea intr-un singur pachet IP.

In urma efectuarii experimentelor descrise mai sus, se poate observa, deci, ca timpii suplimentari introdusi de catre solutia noastra, atat pentru preluarea datelor de catre clienti cat si pentru mentinerea persistentei datelor, sunt independenti de numarul de peer-i Chord, in cazul considerat, in care numarul acestor peer-i este relativ mic, iar volatilitatea lor este moderata. Aceste limitari ale mediului de testare au fost facute tinand cont si de modul in care ar fi alcatuit si s-ar comporta in realitate un server de baze de date distribuit si descentralizat, pentru ca in mod sigur un asemenea server nu va avea nevoie sa contina mai mult de cateva zeci de noduri, care pot fi alese pentru a se asigura o volatilitate moderata.

7.1.2 Evaluarea algoritmului de consens distribuit pentru menţinerea unor date de încredere

In aceasta sectiune ne propunem sa evaluam algoritmul de consens distribuit, descris pe larg in sectiunea 4.2. Ne vom canaliza atentia in special asupra timpului suplimentar introdus de acest algoritm distribuit, urmand sa tragem concluzia daca acest timp suplimentar este justificat de beneficiile functionale, aduse de catre algoritmul in cauza.

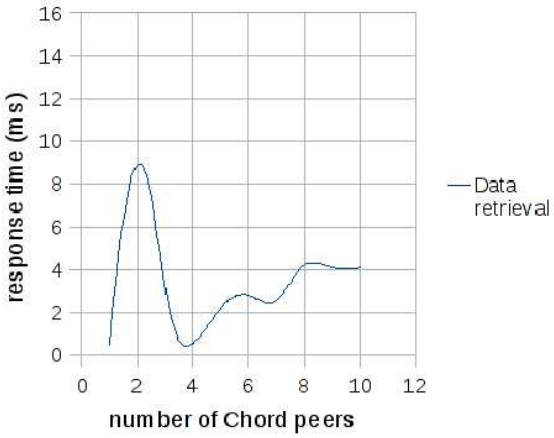

Figura 9: Evaluarea duratei preluarii de date de la baza de date.

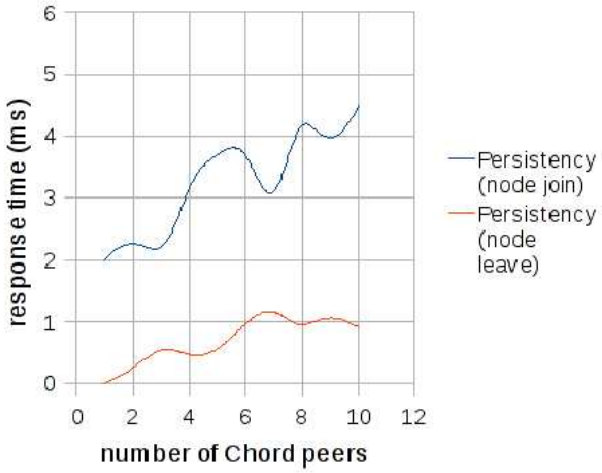

Figura 10: Evaluarea timpului suplimentar, necesar pentru mentinerea persistentei datelor.

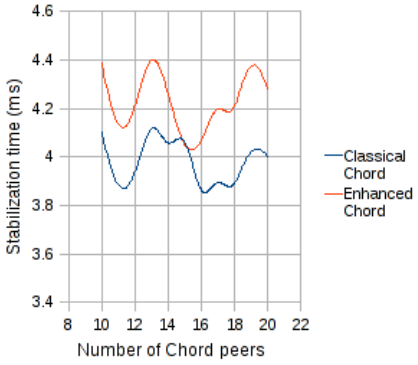

Figura 11: Evaluarea timpului suplimentar, necesar mentinerii topologiei, pe care il introduce algoritmul de consens distribuit.

Asa cum am anticipat deja in sectiunea 4.2, pentru a dovedi eficienta algoritmului nostru, am utilizat o baza de date distribuita, ce ruleaza peste infrastructura logica Chord, pe care am prezentat-o si in sectiunea 4.1, si am evaluat-o in sectiunea 7.1. Nodurile de stocare din cadrul bazei de date distribuite si descentralizate ruleaza server-e centralizate MySQL si am variat numarul de noduri de stocare, si, in consecinta, numarul de peer-i din cadrul topologiei Chord, intre 10 si 20 de noduri. Mediul nostru de test a constat intr-o infrastructura fizica non-volatila, deoarece in evaluarea algoritmului de consens nu ne-a interesat sa vedem, deocamdata, impactul volatilitatii nodurilor. De asemenea, peer-i Chord utilizati de noi au rulat in cadrul unor masini virtuale, care au fost impartite in mod egal intre cele 5 noduri fizice diferite, pe care le-am avut la dispozitie, noduri conectate printr-o retea locala de mare viteza. Experimentul realizat de noi a inceput cu 10 peer-i Chord, dintre care unul era malitios, noi dorind sa vedem in cat timp acest nod malitios este eliminat din cadrul topologiei Chord. La fiecare pas, am reinitializat mediul de testare, cu un numar de noduri care crestea de fiecare data cu 1, pana cand am obtinut 10 seturi de rezultate experimentale. Din aceste date am dorit sa obtinem informatii despre: (1) cat de mult ii ia retelei Chord sa isi mentina topologia, prin rularea periodica a algoritmului de stabilizare, pe care noi l-am combinat si cu elemente ale algoritmului nostru de consens, prin tehnici de “piggy-backing” (in figura 11); si (2) cat de mult ii ia retelei Chord sa stocheze si sa returneze datele asociate unei chei, considerand neglijabila dimensiunea datelor, avand in vedere ca pentru fiecare mesaj receptionat de un peer Chord, peer-ul in cauza trebuie sa verifice, inainte de a procesa mesajul, daca are incredere in peer-ul sursa al mesajului (in figura 12). Nu am incercat sa masuram cat timp le ia peer-ilor Chord din cadrul unei topologii sa elimine un nod malitios, deoarece, asa cum am descris si in sectiunea 4.2, este evident ca aceasta durata de timp este influentata foarte mult si de cat de multe mesaje trimite nodul malitios.

Atat din figura 11, cat si din figura 12 se poate observa faptul ca, deoarece timpii suplimentari introdusi de algoritmul proiectat de noi sunt mici, dimensiunea topologiei Chord aproape ca nu influenteaza timpul suplimentar introdus de acest algoritm de consens distribuit optimizat. Consideram ca acest lucru reprezinta un rezultat foarte bun pentru algoritmul nostru, pentru ca stim ca in general, cand este vorba de algoritmi distribuiti de consens

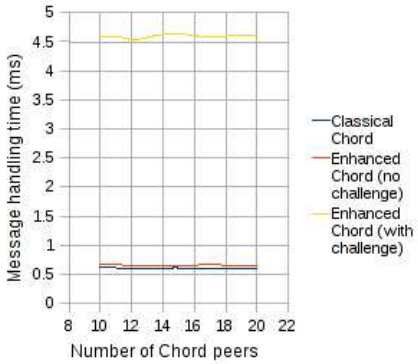

Figura 12: Evaluarea timpului suplimentar introdus de catre algoritmul de consens distribuit, prin comunicarea suplimentara.

prin vot, numarul de noduri din sistem influenteaza direct si semnificativ timpul necesar obtinerii unui verdict final al votului. Inseamna ca, daca consideram ca performantele de timp ale infrastructurii Chord clasice sunt bune, avand in vedere faptul ca algoritmul nostru aduce o imbunatatire semnificativa in ceea ce priveste siguranta infrastructurii Chord oferite, putem considera ca fiind bune si performantele infrastructurii noastre Chord, imbunatatite prin algoritmul de consens distribuit, optimizat.

7.1.3 Evaluarea, la nivelul clientului de baze de date, a timpului de raspuns pentru interogările transmise

In aceasta sectiune vom evalua prototipul descris pe larg in sectiunea 4.3. Ne intereseaza in special sa identificam niste corelatii, existente intre factorii externi prototipului elaborat si prototipul in sine, de-a lungul existentei acestuia. Am identificat doua asemenea corelatii, si anume: (1) *numarul de server-e diferite* care ar trebui contactate la un moment dat (in figura 13); si (2) *numarul de interogari identice* care ar trebui executate la un moment dat. Pentru simplitate, vom considera ca interogari efectuate in cadrul testelor noastre doar interogari de DNS, deoarece acestea sunt mai usor de conceput si de lansat si de asemenea este mai usor de masurat durata acestora (in figura 14). Pentru interogari de baze de date, durata depinde mult de gradul de dificultate al interogarii precum si de planificarea interna pentru executia interogarilor primite, de pe server-ul de baze de date, pe langa intarzierile introduse in retea si incarcarea server-ului de baze de date, aceste ultime doua aspecte impactand, dealtfel, si interogarile DNS.

Scenariul de test considerat de noi consta in a avea mai multi clienti, care contin doar componenta de cereri, care a fost descrisa in sectiunea 4.3, precum si un singur worker, reprezentand entitatea secundara de executie a interogarilor. Pentru a demonstra faptul ca pana si pe o masina cu un singur procesor si un singur core prototipul deja prezentat de noi e capabil sa aduca imbunatatiri, am utilizat o asemenea masina ca fiind suportul hardware pentru testele noastre. In primul test, interogarile clientilor au fost adresate mai

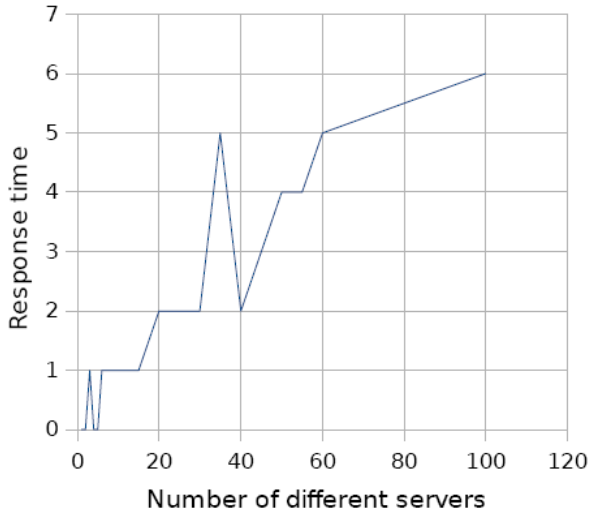

Figura 13: Influenta unui numar crescand de server-e diferite asupra performantelor entitatii secundare de executie a interogarilor.

multor server-e diferite, in timp ce in cel de-al doilea test interogarile tuturor clientilor au fost identice, fiind adresate unui aceluiasi server. In contextul celui de-al doilea test, reamintim faptul ca entitatea secundara de executie a interogarilor a fost optimizata pentru a transmite in exterior un numar minim de interogari, grupand interogarile identice, pentru a minimiza numarul de mesaje transmise in retea, si deci numarul de operatii de intrare-iesire.

Combinand rezultatele obtinute in cazul unui numar variabil de server-e diferite catre care se transmit interogari cu rezultatele obtinute in cazul existentei unui numar variabil de interogari identice, obtinem o estimare suficient de buna a comportamentului sistemului intr-un mediu real, unde, la un moment dat, am putea avea cazul unui numar variabil de interogari, adresate catre mai multe server-e, dintre care unele interogari sunt identice. Este evident faptul ca, din punctul de vedere al server-elor distante, faptul de a li se transmite mai putine interogari, prin gruparea pe care am instituit-o la nivelul prototipului nostru, este si el benefic. Rezulta, din masuratorile descrise in aceasta sectiune, ca doar numarul de server-e diferite ce trebuie contactate influenteaza performantele nivelului de decuplare introdus de noi, pentru cazul in care aplicatia ruleaza pe o masina cu un singur procesor si un singur core. Ne putem imagina ca degradarea care s-ar vedea pe o masina multi-procesor multi-core va fi mult mai mica, poate chiar imperceptibila.

7.2 Rezultatele evaluării metodelor de exploatare a dinamicităţii peer-to-peer

7.2.1 Evaluarea funcţionalităţii de autoextindere pentru persistenţă a datelor sensibile

In aceasta sectiune vom evalua prototipul descris pe larg in sectiunea 5.1. Ne intereseaza in special cat de mult ajuta prototipul elaborat de noi cresterea gradului de incredere pentru stocarea datelor sensibile, precum si persistenta acestora.

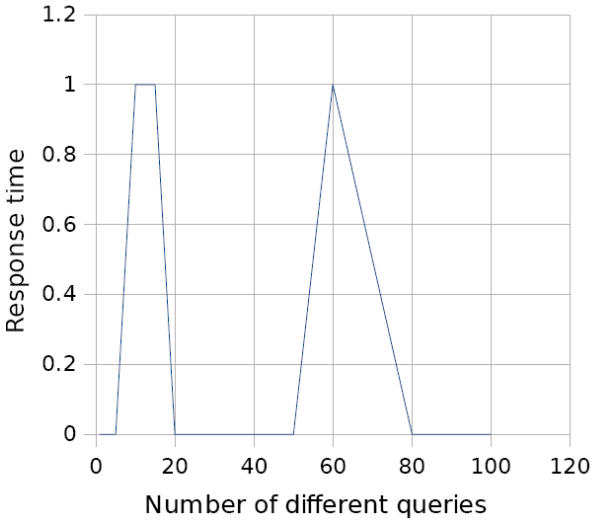

Figura 14: Influenta unui numar crescand de interogari identice asupra performantelor entitatii secundare de executie a interogarilor.

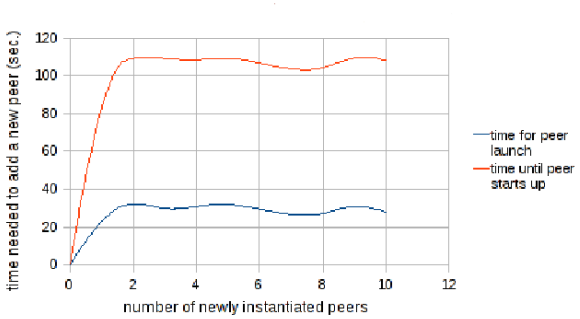

Figura 15: Instantiatierea unui nou nod-fiu la fiecare 5 minute.

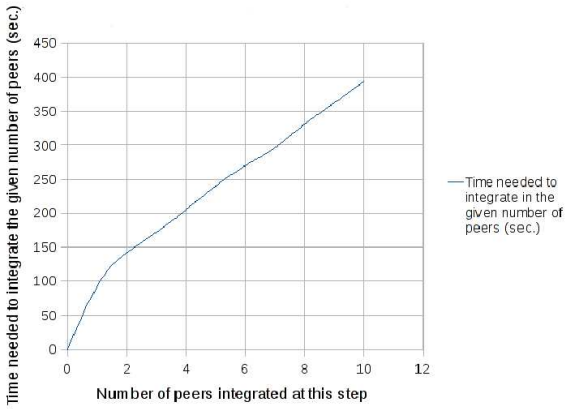

Figura 16: Instantiatierea unui numar crescand de peer-i, la fiecare 5 minute.

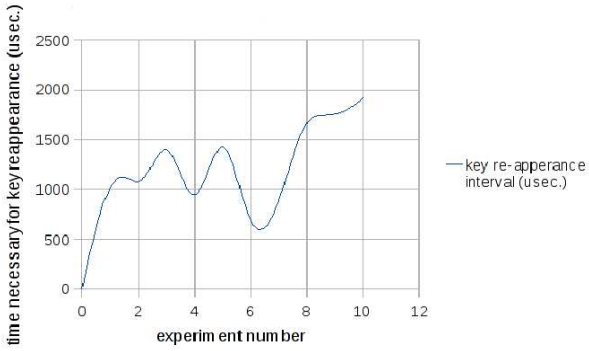

Figura 17: Durata replicarii datelor pe noul lor posesor.

Pentru evaluarea acestui prototip, am folosit un mediu de testare format din mai multe masini virtuale, care ruleaza peste cateva masini fizice, spre a putea avea un numar cat mai mare de peer-i, chiar daca nu am avut la dispozitie un numar atat de mare de noduri fizice. In aceasta sectiune ne vom ocupa in special de evaluatea mecanismului de instantiere de noi peer-i pe noduri suplimentare, adaugate la nevoie infrastructurii peer-to-peer. Astfel, dorim sa masuram: (1) cat de mult dureaza lansarea unui numar de noduri, situate fiecare la distante similare in reteaua fizica fata de nodul parinte, care le lanseaza (in figurile 15 si 16); si (2) cat de mult dureaza "restaurarea" datelor de pe un nod fiu pe noul nod parinte, reprezentand responsabilul si proprietarul de drept al datelor corespunzatoare cheii in cauza (in figura 17).

Se poate observa, din figura 17, ca timpul necesar schimbarii mesajelor, care readuc cheia si datele pe nodul care este de fapt responsabil de ele, este de aproximativ 2ms. Remarcam faptul ca acest timp nu include: (1) latenta de retea ce ar exista intr-un caz real, cand nodurile pot avea o larga distributie geografica; (2) intervalul de timp definit pe nodul fiu pentru a-si contacta periodic parintele prin mesaje de tip "ping", interval care trebuie sa nu fie considerat nici prea mare, dar nici prea mic; si (3) timpul necesar download-ului efectiv al

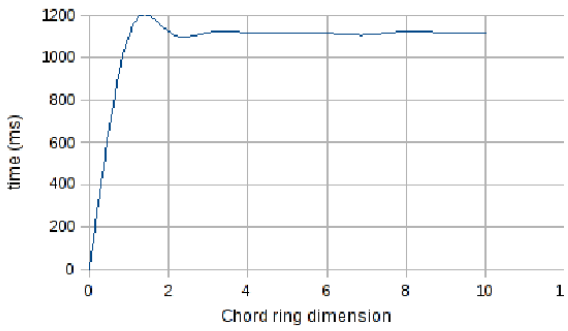

Figura 18: Adaugarea unui nou peer la infrastructura Chord, spre a prelua o parte din incarcarea unui peer supra-incarcat.

datelor aferente cheii pe noul parinte, timp care ar putea fi foarte mare, daca datele sunt mari. Aceste intervale de timp nu au fost considerate la realizarea figurii 17 pentru ca ele pot varia foarte mult, in functie de factori externi prototipului, si, de asemenea, pentru ca oricum insumate ar fi mult mai mari decat timpul necesar procesarii mesajelor schimbate intre peer-ul fiu si peer-ul parinte. Pe viitor ne dorim sa luam in considerare si o valoare medie pentru latenta in retea dintr-un sistem Chord mai larg distribuit geografic, astfel incat rezultatele obtinute de noi sa aiba o relevanta mai mare pentru cazul de utilizare reala. Cu toate acestea, chiar si avand in vedere timpii suplimentari introdusi prin aceste intarzieri, prin utilizarea prototipului elaborat de noi, impreuna cu mecanismele propuse de crestere a redundantei si deci a tolerantei la defecte, solutia noastra se comporta mult mai bine decat o retea Chord obisnuita, in a proteja anumite date foarte valoroase pentru aplicatia de nivel inalt care stocheaza date utilizand Chord.

7.2.2 Evaluarea funcţionalităţii de echilibrare a încărcării, pentru o largă disponibilitate a sistemului

In aceasta sectiune vom evalua prototipul descris pe larg in sectiunea 5.2. Ne intereseaza in special sa masuram timpul suplimentar introdus pentru a spori scalabilitatea si echilibrarea incarcarii intre noduri, prin utilizarea de mecanisme de auto-extindere similare cu cele evaluate in sectiunea 7.2.1.

Intrucat metoda de auto-extindere este comuna cu solutia pentru sporirea tolerantei la defecte si a scalabilitatii, prezentata in sectiunea 5.1, si pentru evaluarea prototipului de echilibrare a incarcarii, am folosit un mediu de testare format din mai multe masini virtuale, care ruleaza peste cateva masini fizice, spre a putea avea un numar cat mai mare de peer-i, chiar daca nu am avut la dispozitie un numar atat de mare de noduri fizice. In aceasta sectiune ne vom ocupa in special de evaluatea mecanismului de instantiere de noi peer-i pe noduri suplimentare, adaugate la nevoie infrastructurii peer-to-peer. Astfel, dorim sa masuram: (1) timpul scurs intre momentul cand un peer isi da seama ca este supraincarcat

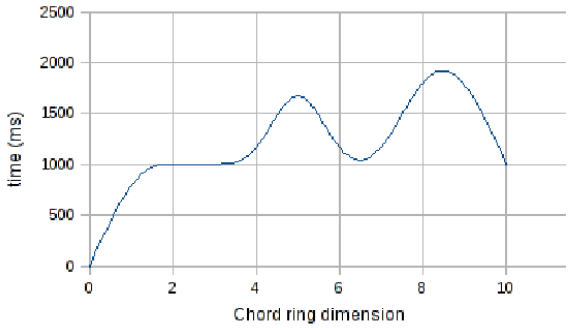

Figura 19: Un peer extrem de incarcat trebuie sa paraseasca inelul Chord, toate indatoririle fiindu-i preluate de catre succesorul sau.

si momentul in care supra-incarcarea sa este preluata de catre un peer nou-instantiat (in figura 18); si (2) timpul scurs intre momentul cand un peer isi da seama ca este extrem de incarcat, observatie in urma careia va decide sa paraseasca reteaua, pentru ca nu-si mai poate indeplini, oricum, corect indatoririle, si momentul in care toata incarcarea sa este preluata de catre nodul sau de rezerva, adica succesorul sau (figura 19).

Inainte de a descrie vreunul din experimentele realizate, am dori sa mentionam faptul ca, pe langa nodurile virtuale care ruleaza peste noduri fizice, in cadrul *infrastructurii fizice* pe care am utilizat-o, conexiunile de retea dintre noduri s-au realizat prin intermediul unei retele locale de mare viteza, avand o latenta de aproximativ 2ms. Pe de alta parte, pentru *infrastructura logica* utilizata, adica inelul Chord, am definit un interval de stabilizare al topologiei egal cu 1 secunda. Am considerat important sa mentionam aceste valori de timp, pentru ca ele prezinta un impact semnificativ asupra comportamentului sistemului de echilibrare a incarcarii, precum si asupra capacitatii acestuia de a face fata dinamicitatii.

Concluzia acestor doua tipuri de experimente, luate in ansamblu, realizate pentru a evalua capacitatea de auto-echilibrare si de scalabilitate a prototipului nostru, este faptul ca, intr-o maniera foarte simpla si transparenta, acestuia i se poate adauga capacitate suplimentara de stocare, materializata atat prin adaugarea de spatiu fizic suplimentar de stocare pentru peer-ii individuali existenti, cat si prin adaugarea de noi peer-i Chord, avand o capacitate proprie de stocare. In acest mod, unui operator uman sau unei aplicatii externe li se pot pune la dispozitie in mod eficient, din punct de vedere al consumului de energie si al nevoilor clientilor, toate tipurile de resurse de stocare avute la indemana.

7.3 Rezultatele evaluării unui serviciu generic, descentralizat pentru sporirea disponibilităţii sale

In aceasta sectiune dorim sa realizam o evaluare preliminara a modelului de distribuire a unui serviciu generic, prezentat in capitolul 3. Pentru a evalua acest prototip s-a utilizat un

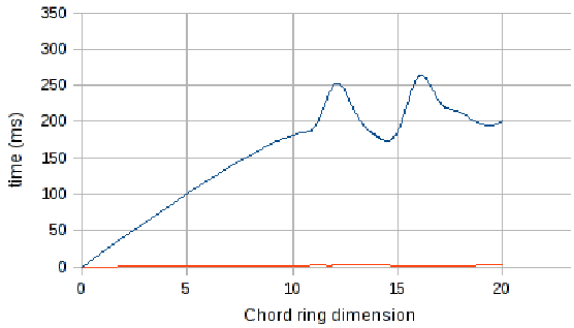

Figura 20: Timpul suplimentar introdus de catre sistemul de descentralizare in contextul transmiterii cererilor si a raspunsurilor intre client si serviciu.

load balancer, distribuit pe 3 noduri, pentru fiecare existand cate o pondere egala in cadrul inregistrarilor DNS de tip NAPTR, respectiv SRV. In spatele acestui load balancer exista o retea Chord, formata din 0 pana la 20 de peer-i, fiecare peer aflandu-se in stransa legatura cu un server ce ruleaza pe acelasi nod, toate server-ele care ruleaza pe peer-i fiind de acelasi tip. Reteaua care interconecteaza atat peer-ii intre ei, cat si peer-ii de nodurile de load balancing, ce se afla in fata serviciului propriu-zis, este o retea LAN de mare viteza. Mentionam faptul ca reteaua Chord utilizata este o retea Chord imbunatatita conform contributiilor descrise in sectiunile 4.1, 4.2, 5.1, si 5.2, si care au fost evaluate si validate in sectiunile anterioare ale acestui capitol, pentru a oferi o mai mare scalabilitate, toleranta la defecte, persistenta a datelor si echilibrare a incarcarii peer-ilor.

La inceput, in primul experiment, reteaua Chord este vida, ceea ce inseamna ca nu exista nici un nod care sa ofere serviciul dorit. Apoi, succesiv, dupa ce a fost instantiat manual un prim peer, impreuna cu serviciul aferent, ce urmeaza a fi distribuit, care a format o *retea Chord elementara*, ce este capabila sa ofere serviciul generic considerat, s-au utilizat metode de auto-extindere a infrastructurii Chord, descrise in capitolul 5, ce au fost evaluate in sectiunile 7.2.1 si 7.2.2. Pentru fiecare dimensiune considerata a retelei Chord s-au realizat masuratori ale: (1) timpului suplimentar, introdus pentru ca o *cerere* sa ajunga de la client la serviciul pe care acesta il contacteaza, de catre infrastructura noastra de distribuire generica a unui serviciu, formata din load balancer-ul distribuit si din inelul Chord de peer-i pe care ruleaza si serviciul generic considerat; si (2) timpul suplimentar, introdus pentru ca *raspunsul* serviciului generic distribuit si descentralizat sa ajunga de la peer-ul Chord pe care cererea a fost procesata pana la client, scurt-circuitand, de aceasta data, nivelul de load balancing.

In figura 20 se pot vedea timpii suplimentari introdusi in receptionarea si procesarea cererii, respectiv in trimiterea raspunsului, conform modelului nostru de descentralizare si distribuire a unui serviciu. Mentionam ca acesti timpi s-au obtinut scazand timpii de procesare si de transmitere a mesajelor prin retea, care se pierd si in mod obisnuit, cand serviciul generic nu este distribuit conform modelului nostru din timpii totali, obtinuti: (1) in cazul receptionarii si procesarii cererii, calculand diferenta de timp dintre momentul cand clientul a

trimis mesajul, determinat prin intermediul unei marci de timp prezente in mesaj, si momentul cand server-ul a terminat de construit raspunsul catre client; si (2) in cazul transmiterii raspunsului, calculand diferenta de timp dintre momentul cand server-ul a trimis raspunsul, determinat tot prin intermediul unei marci de timp prezente in mesaj, si momentul cand clientul receptioneaza raspunsul.

Se observa ca timpul suplimentar, introdus de catre modelul de distribuire in scopul descentralizarii unui serviciu, pentru receptionarea si procesarea cererii este in medie de 200ms. Acest fapt nu constituie o surpriza, intrucat, conform articolului de descriere a functionalitatii si de evaluare a Chord, pentru un inel de aproximativ 10-30 de peer-i, timpul necesar realizarii rout-arii unui mesaj, in conditiile chiar ale unei volatilitati moderate a nodurilor, este de aproximativ 200ms. Pe de alta parte, timpul suplimentar introdus de catre modelul de distribuire in scopul descentralizarii unui serviciu, pentru transmiterea raspunsului catre client la o cerere a acestuia, este aproape inexistent. Rezulta, deci, ca ceilalti timpi suplimentari introdusi de catre distribuirea descentralizata a serviciului, conform modelului introdus in aceasta lucrare, sunt neglijabili.

Asadar, per ansamblu, timpul suplimentar introdus in general de catre serviciul distribuit, pe care il percepe clientul de la momentul trimiterii cererii si pana la momentul primirii raspunsului pentru aceasta din partea server-ului descentralizat, este egal cu timpul necesar rout-arii unui mesaj Chord intr-u inel de 10-20 de peer-i, ceea ce inseamna, de fapt, ca avem un cost pentru beneficiile pe care Chord le introduce, care nu este, insa, mai mare decat insusi costul rout-arii unui mesaj prin intermediul inelului Chord, care este de fapt componenta principala de descentralizare a serviciului, care ii permite acestuia sa devina mai larg disponibil catre clientii sai, prin metode enuntate deja in capitolul 3, cat si in sectiunile 4.1, 4.2, 5.1, si 5.2.

7.4 Rezultatele evaluării unui SIP presence server, descentralizat pentru sporirea disponibilităţii sale

In aceasta sectiune dorim sa realizam o evaluare preliminara a modelului de distribuire a unui SIP presence server, prezentat in 6, si care se bazeaza pe modelul generic de distribuire si descentralizare a unui serviciu, care a fost prezentat in 3, fiind evaluat in sectiunea 7.3. Ipotezele privind load balancer-ul si reteaua Chord sunt identice cu cele din sectiunea 7.3, si anume: (1) load balancer-ul este compus din 3 noduri, avand o probabilitate si o pondere egala in cadrul inregistrarilor DNS de tip NAPTR si SRV; (2) dimensiunea retelei Chord, imbunatatite, identice cu cea folosita in sectiunea 7.3, variaza intre 0 si 20 de peer-i, dintre care primul peer este pornit manual, iar urmatorii sunt obtinuti, pe baza unor configurari la initializarea fiecarui peer, conform metodelor de autoextindere descrise in capitolul 5, si evaluate in sectiunile 7.2.1 si 7.2.2; si (3) reteaua care interconecteaza oricare doua noduri din cadrul sistemului distribuit descentralizat este o retea LAN rapida. Mentionam, de asemenea, ca server-ul de presence utilizat, care ruleaza pe fiecare nod din inelul Chord, este server-ul open-source OpenSIPS.

Pentru fiecare dimensiune considerata a retelei Chord s-au realizat masuratori ale impactului de timp asupra unui client de SIP presence, datorat distribuirii server-ului de presence prin intermediul load balancer-ului si a retelei Chord, insumand timpul scurs de la trimi-

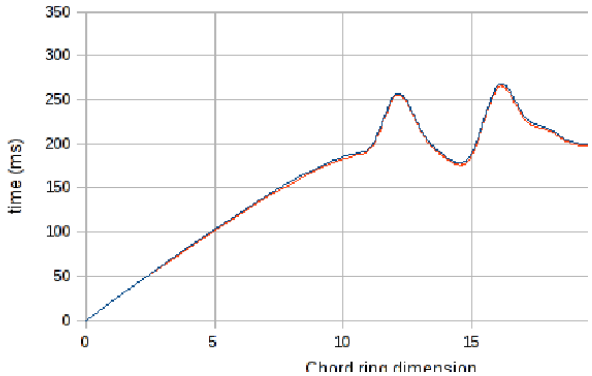

Figura 21: Impactul distribuirii in scopul obtinerii unei mai largi disponibilitati, a unui server de presence SIP, asupra unui client al acestuia.

terea cererii de catre client si pana la primirea raspunsului din partea server-ului, in cazul in care: (1) clientul se afla in spatele unui NAT, si deci raspunsul trebuie sa ii fie trimis de catre server-ul care il trateaza utilizand si load balancer-ul, pentru care exista un port deschis in cadrul NAT-ului; si (2) clientul nu se afla in spatele unui NAT, caz in care server-ul de presence ii poate trimite raspunsul direct, scurt-circuitand astfel load balancer-ul, spre a economisi timp de trimitere in retea si spre a nu-l mai incarca suplimentar pe acesta si cu transmiterea raspunsurilor.

In figura 21 se poat vedea impactul de timp asupra clientului a descentralizarii server-ului de presence, atat in cazul in care clientul se afla in spatele NAT-ului, cat si in cazul in care acesta nu se afla in spatele NAT-ului. Timpii de impact au fost obtinuti, similar cu procedeul din sectiunea 7.3, prin scaderea, din durata totala de timp de cand un client a trimis mesajul si pana cand a primit raspunsul, a duratei de timp necesara in mod obisnuit transmiterii mesajului si a raspunsului si procesarii cererii primite de la client de catre nodul server de presence SIP. Se poate remarca faptul ca impactul descentralizarii server-ului de presence, spre a-l face mai larg disponibil, este foarte apropiat pentru clienti aflati in spatele NAT-ului de cel pentru clienti care nu se afla in spatele NAT-ului, timpul suplimentar introdus fiind in medie de 200ms, adica exact durata rout-arii unui mesaj prin inelul Chord de dimensiune maxim 20 de noduri. Comparand rezultatele obtinute pentru distribuirea in scopul obtinerii unei mai largi disponibilitati a server-ului de presence din cadrul OpenSIPS cu rezultatele obtinute pentru distribuirea generica a unui serviciu, din sectiunea 7.3, putem remarca faptul ca ele sunt foarte apropiate, ceea ce inseamna ca server-ul de presence a fost distribuit corect, conform modelului generic, fapt ce realizeaza o validare preliminara a acestui model de distribuire si descentralizare, in scopul obtinerii unei mai largi disponibilitati.

7.5 Concluzii ale evaluarii rezultatelor experimentale

In acest capitol am realizat evaluarea rezultatelor stiintifice originale introduse de catre aceasta teza. Subliniem si cu aceasta ocazie faptul ca noi consideram ca esential in cresterea

disponibilitatii serviciului este DHT-ul. In sine, orice DHT este un sistem scalabil, echilibrat si tolerant la defecte, permitand totodata si multe imbunatatiri si extensii, dintre care am prezentat cateva ca fiind contributii originale ale acestei teze, pe care le-am evaluat in acest capitol. Nu pretindem ca evaluarea ar fi fost una exhaustiva, asa cum mentionam, dealtfel, in fiecare sectiune a acestui capitol. In masuratorile efectuate am urmarit doua directii mari: (1) imbunatatirea timpului dintre 2 esecuri consecutive ale sistemului (*M*ean *T*ime *B*etween *F*ailures – MTBF), in special in sectiunea 7.2, si (2) minimizarea impactului de timp al componentelor şi algoritmilor pentru mentinerea unei infrastructuri disponibile, pe parcursul intregii evaluari a rezultatelor. Consideram ca evaluarea facuta aici a fost suficienta pentru a sublinia viabilitatea directiei urmate in cercetarea noastra, viabilitate sustinuta si prin cele 7 lucrarile publicate in conferinte si reviste de specialitate, elaborate de-a lungul acestui doctorat.

8 Contribuţii şi concluzii

In capitolele anterioare am identificat si analizat problemele care pot sa impacteze disponibilitatea unui serviciu distribuit. Problemele identificate si analizate sunt: (1) persistenta datelor; (2) toleranta la defecte; (3) scalabilitatea; (4) echilibrarea incarcarii; si (5) securitatea sistemului. Pentru acestea, pe parcursul lucrarii de fata, oferim solutii propri generice, bazate pe o arhitectura descentralizata de tip peer-to-peer. In capitolele 3 si 6 am propus o maniera proprie de a combina toate contributiile originale prezentate in capitolele 4 si 5, pentru a putea intr-adevar sa obtinem serviciul distribuit larg-disponibil, pe care il dorim. Platformei generice de descentralizare a unui serviciu i-am conferit o forma unitara si completa, care reuseste sa combine toate avantajele solutiilor prezentate in capitolele 4 si 5. De asemenea, in capitolul 7, am evaluat, una cate una, performantele pe care le aduc fiecare dintre contributiile noastre, fiecare in parte. Am masurat independent fiecare contributie, caci consideram ca solutiile noastre la problemele identificate in studiul bibliografic sunt generice si independente. Rezultatele obtinute sunt valabile si pot fi, deci, aplicate si utilizate in orice context ar fi nevoie de ele, nu doar in cel al descentralizarii unui serviciu, spre a-i creste disponibilitatea.

Vom prezenta, pe scurt, principalele contributii originale aduse prin intermediul tezei de fata:

1. In capitolul 4.3: o solutie pentru decuplarea executiei interogarilor, astfel incat un server care are nevoie sa interogheze un alt server sa nu astepte blocant raspunsul. El poate, deci, prelua si deservi alti clienti pana la primirea raspunsului de la server-ul distant. Aceasta contributie este utila si necesara intrucat orice serviciu am dori sa furnizam, server-ul proiectat va trebui sa transmita un minim de interogari catre alte server-e. De aceea, se poate ca performantele serviciului pe care dorim sa il oferim sa poata fi impactate, prin dependenta de un server distant, de entitati al caror comportament nu il putem controla.

2. In capitolul 4.1: o solutie pentru stocarea persistenta a datelor, utilizand o baza de date distribuita si descentralizata, bazata pe arhitectura peer-to-peer Chord. Peer-ii Chord controleaza noduri independente, pe care ruleaza server-e obisnuite de baze de date.

Acesta solutie este utila si necesara deoarece orice serviciu, si cu atat mai mult unul care se doreste larg disponibil, are nevoie sa stocheze pe o durata cat mai lunga datele clientilor pe care ii deserveste.

3. In capitolul 4.2: o solutie pentru asigurarea unui grad de siguranta si incredere pentru stocare cat mai mare, chiar si in prezenta unor noduri malitioase in cadrul sistemului distribuit. Aceasta solutie este bazata tot pe arhitectura Chord si defineste un protocol de consens distribuit adaptat acestei arhitecturi. Acesta solutie este utila si necesara, deoarece orice serviciu, si cu atat mai mult unul care se doreste larg-disponibil, are nevoie *sa stocheze cu un grad mare de incredere* date, pe o durata cat mai lunga, indiferent daca in sistem exista sau nu noduri malitioase.

4. In capitolul 5.1: o solutie pentru asigurarea tolerantei la defectele individuale ale nodurilor din sistem, si intr-o oarecare masura si pentru sporirea scalabilitatii sistemului. Toleranta la defecte este crescuta in special pentru datele critice pe care serviciul le stocheaza. Solutia prezentata este bazata si ea tot pe arhitectura DHT-ului Chord, prin definirea unui mecanism de autoextindere, ce este invocat pentru a spori gradul de redundanta cu care sunt stocate datele critice. Acesta solutie este utila si necesara deoarece orice serviciu, si cu atat mai mult unul care se doreste larg-disponibil, are nevoie sa asigure toleranta la defecte, spre a proteja datele clientilor sai, pe care le stocheaza, si in special datele critice ale acestora.

5. In capitolul 5.2: o solutie pentru asigurarea echilibrarii incarcarii nodurilor, si, in consecinta, si pentru sporirea scalabilitatii sistemului. Solutia noastra se bazeaza tot pe DHT-ul Chord, utilizand la maxim avantajele dispersiei uniforme oferite de functia de hashing a DHT-ului. Datorita acestei functii de dispersie, solutia noastra reuseste sa integreze, la nevoie, tot prin autoextindere, noi resurse, exact acolo unde ele sunt necesare, adica acolo unde incarcarea este prea mare. Aceasta solutie este utila si necesara deoarece orice serviciu, si cu atat mai mult unul care se doreste larg-disponibil, are nevoie sa asigure echilibrarea incarcarii nodurilor si, deci, scalabilitate in ceea ce priveste numarul de clienti pe care ii poate deservi.

6. In capitolul 3: o solutie pentru combinarea avantajelor de persistenta a datelor, de incredere, de toleranta la defecte, de echilibrare a incarcarii si de scalabilitate. Propunem, deci, la final, o metoda generica, bazata pe DNS si pe DHT-uri, de a avea un unic punct de acces la un serviciu care beneficiaza de toate avantajele mentionate mai sus. In acest mod, descentralizarea serviciului, utilizand arhitectura Chord, in scopul largirii disponibilitatii acestuia, devine transparenta pentru clientii serviciului. Aceasta solutie este utila intrucat ofera o "reteta" generica, ce se poate aplica oricarui serviciu distribuit, in scopul largirii disponibilitatii acestuia. Mai mult, aceasta solutie este validata nu doar prin masuratori, ci si prin aplicarea ei pentru un server SIP de prezente.

7. In capitolul 6: O solutie pentru a obtine un serviciu SIP larg disponibil, prin aplicarea solutiei generice de descentralizare a unui serviciu pentru un server de prezenta SIP. La randul sau, server-ul de prezenta descentralizat a fost evaluat, spre a confirma ca i s-a sporit disponibilitatea fata de clienti, in urma modificarilor aduse.

Pe baza rezultatelor obtinute, evaluand experimental contributiile originale ale tezei, putem conchide ca obiectivul tezei principal al tezei a fost indeplinit. Asadar, am obtinut o

reteta pentru realizarea unui serviciu larg disponibil cu impact de performanta rezonabil. Impactul de performanta in vederea disponibilitatii, observat in urma interpretarii datelor experimentale, este in general comparabil cu perioada de stabilizare a DHT-ului Chord. Cu alte cuvinte, daca mediul academic si mediul de productie considera Chord, si DHT-urile in general, ca fiind o solutie viabila pentru servicii utilizate *la scara larga* (large scale), atunci si solutia originala propusa de noi poate fi considerata viabila. Un argument in favoarea faptului ca utilizand Chord putem obtine solutii practice viabile este faptul ca acest DHT este utilizat in implementari legate de P2P SIP, pentru care in prezent se pregateste un RFC. Acesta, desi nu unic, este si un argument important in favoarea *actualitatii* subiectului abordat de teza in cauza.

Pe viitor ne propunem sa imbunatatim unele contributii prezentate, in directiile descrise in momentul prezentarii si evaluarii performantelor lor. De asemenea, ne propunem sa investigam si alte metode de a spori disponibilitatea unui serviciu, pe care mai apoi sa le aplicam unor servicii distribuite particulare, critice, asa cum este si SIP.