P2P chat program II

P2P alkalmazások labor, 2. FreePastry mérés

# Pastry

A Pastry egy DHT protokoll, hasonlatos a korábban megismert Chord-hoz. Előnye más DHT megoldásokkal szemben az útválasztás módszere. A Pastry csomópontok útválasztó táblákat tartanak fent, melyek külső forrásokból származó metrikákat is figyelembe vehetnek az útválasztás során (shortest path, hop count…).

Az egyes csomópontok ID alapján vannak gyűrűbe fűzve, minden csomópont ismeri L/2 baloldali, és L/2 jobboldali levelét. E mellett minden csomópont fenntart egy másik szomszédsági táblát, amely a választott metrika szempontjából legközelebbi M szomszédot tartalmazza. És végül minden csomópont tárol egy útválasztó táblát, mely prefix alapon, szintenként osztja fel a címteret. A címek b számrendszerben vannak ábrázolva. Az útválasztó tábla nulladik szintjén (b-1) olyan csomópont szerepel, melynek az első karaktere eltér, az első szinten (b-1) olyan, aminek az első karaktere egyezik, de a második eltér, a másodikon (b-1) olyan, aminek az első két karaktere egyezik, de a többi eltér…

Az útválasztás során minden csomag a cél-azonosítóhoz legközelebbi csomóponthoz fog eljutni. Amennyiben a cím szerepel akár a levelek, akár a szomszédok közt, a csomagot a csomópont közvetlenül küldi. Ha ezek közt a csomópontok közt nem szerepel a célcím, akkor az útválasztó táblában prefix alapján legközelebbi csomóponthoz küldi a csomagot.

# FreeParstry

A FreePastry a Pastry protokoll Java alapú implementációja. A Pastry több kiterjesztését is implementálja, a Scribe-ot, mely egy multicast üzenetet továbbító protokoll, és a Past-ot, mely egy elosztott file-tárolási szolgáltatás Pastry fölött. Ezek mellett megtalálható benne a SplitStream, ami egy elosztott tartalom-megosztó szolgáltatás.

A projekt oldala elérhető itt: <http://www.freepastry.org/FreePastry/>

## Node, NodeFactory, NodeHandle

A Pastry gyűrű csomópontjait a Node objektum testesíti meg. Az egyes csomópontokat NodeId azonosítja. Ez ellentétben az ajánlással egy 160 bites azonosító (20 byte), mely a gyűrűn belül egyedi. A gyűrűhöz csatlakozó csomópont azonosítóját különböző forrásokból lehet megszerezni, a FreePastry tartalmaz egy NodeIdFactory interfacet, melyet az azonosító-generátoroknak implementálniuk kell, és több kész Factory-t is a fejlesztő rendelkezésére bocsájt, melyek a csomópont címéből, a hálózat egy dedikált csomópontjától kérve, vagy véletlenül választva adnak csomóponti azonosítót.

A FreePastry több hálózati protokollhoz is tartalmaz csomópontokat, mi a Pastry csomópontot fogjuk használni. A FreePastry különböző csomópontokat kínál a különböző környezetekbe: a memóriaterületen kommunikáló, szimulációra szánt csomópontoktól az Interneten keresztül kommunikáló, NAT-ot törő csomópontokig. A mérés céljaihoz mi a Socket alapú, közvetlen címzésű csomópontokat választottuk. Ezeket a SocketPastryNodeFactory osztály segítségével hozhatjuk létre.

A kommunikáció megkönnyítésére szolgál a NodeHandle absztrakt osztály. Ez tartalmazza az összes olyan információt, mely a hálózaton belül egy csomópont tényleges eléréséhez szükséges. A mérés során megvalósított Socket alapú hálózatnál ez egy IP cím és egy port. Minden csomópont ismeri a saját NodeHandle-ét. Ha egy új csomópont szeretne belépni a gyűrűbe, akkor elérve az egyik bootstrap csomópontot, annak NodeHandle-ét lekérdezve tud csatlakozni.

## Environment

Az Environment elsősorban szimulációknál fontos, ennek ellenére haasználni kell az éles FreePastry rendszerekben is. Az Environment sok kényelmi szolgáltatással járul hozzá a hatékony teszteléshez. Egy egységes időkezelési mechanizmust vezet be, melynek felhasználásával a csomópontok forrásának módosítása nélkül lehet a valós és a virtuális időkezelés közt váltani. Egy egységes loggert és egy reprodukálható véletlenszám-generátort és egy párhuzamos feladatvégzést könnyítő osztályt is a fejlesztő rendelkezésére bocsájt.

## Alkalmazás, üzenet

A Pastry csak az üzenetek célba juttatásáért, a gyűrű karbantartásáért felel. A tényleges funkcionalitást az alkalmazások adják. A FreePastry-ban az alkalmazásoknak Application interface-t kell implementálniuk. Ez az interface regisztrálható a csomóponthoz.

Amikor egy alkalmazást regisztrálunk, létrejön egy végpont (Endpoint), ezeket a végpontokat köti össze a különböző csomópontok közt a FreePastry. Az alkalmazást regisztrációkor felruházzuk egy azonosítóval és a FreePastry gondoskodik arról, hogy csak a különböző csomópontokon futó azonos azonosítójú alkalmazások kommunikáljanak egymással. Ezt a kényelmi szolgáltatást érdemes kihasználni oly módon, hogy ha egy alkalmazást több magasabb szintű alkalmazás használ – pld több alkalmazás támaszkodik a Scribe multicastra – akkor azt az alkalmazást több példányban regisztráljuk a csomóponthoz, így nem kell saját kódban eldönteni, hogy az adott csomag melyik magasabb rétegbeli alkalmazáshoz érkezett.

Az alkalmazások üzeneteken keresztül kommunikálnak. Minden üzenet implementálja a Message interface-t. Ez az interface egyetlen metódussal rendelkezik, mely az üzenet prioritását adja vissza. Ezen felül egyetlen tulajdonsága, hogy sorosítható.

## Scribe

A Scribe a Pastry és FreePastry csoportküldést támogató komponense. A Scribe támogatja a multicast (minden csoporttagnak) és az anycast (valamelyik csoporttagnak). A Pastry gyűrű tagjai feliratkozhatnak csoportokra. A csoport tagjai egy multicast fába vannak felfűzve, ez a fa biztosítja a hatékony csomag-továbbítást. A FreePastry-ban a Scribe funkcionalitást a ScribeImpl valósítja meg. Ez egy alkalmazás, így az ott leírtaknak megfelelően kell használni. A csoportokat reprezentáló osztály a Topic. Minden csoportnak van egy azonosítója, mely a nevéből kerül előállításra. Ez az azonosító jelöli ki a gyökércsomópontot is. A Topic egy példányával lehet felíratni a csoportba a ScribeMultiClient interface-t implementáló objektumot.

Az üzenetek küldése a Scribe objektumon keresztül történik. Egy Scribe objektum több Topic-ot is kezelhet és egy csomópont több Scribe alkalmazást is futtathat.

## Past

A Past a FreePasty DHT-je. Hash-fügvényként a FreePastry-ban megszokott azonosító-generátort használja. A FreePastry implementáció lehetőséget biztosít a csomópontokban a perzisztens és a memória-alapú tárolásra, illetőleg támogatja a cachelést. A FreePastry-ban a Past szintén egy alkalmazás, melyet a PastImpl valósít meg.

# Mérési útmutató

A mérések során a korábban elkészített chatalkalmazást fogja kiegészíteni chat-szobákkal és felhasználói nevekkel. Ehhez a FreePastry Scribe és Past funkcióit fogja használni. A chat-szobákba belépő felhasználók megkapnak minden, az adott csoportnak címzett üzenetet, és írhatnak is a csoportnak. A felhasználónevek pedig lehetővé teszik a beszélgető-partnerek azonosítását, amíg az első mérésbe a FreePastry-ban használt azonosítókkal címezte a beszélgetőpartnereket, és csak a szomszédos csomópontokkal tudott beszélgetést kezdeményezni, addig most a Past segítségével el fog készíteni egy elosztott telefonkönyvet, melyben a csomópontok belépéskor felviszik azonosítójukat és felhasználónevüket, és innen kérik le beszélgetőpartnereik nevét.

## Projekt létrehozása

Eclipse alatt fogunk dolgozni. Első lépésként importálni kell a letöltött projektet. A projekthez hozzá kell adni a FreePastry csomagot mely az alábbi linken érhető el: <http://www.freepastry.org/FreePastry/FreePastry-2.1.jar>

A Past funkcionalitásához szükség van egy XPP parserre. Ezt az alábbi linken érheti el:

<http://www.extreme.indiana.edu/xgws/xsoap/xpp/mxp1/index.html>

A letölthető JAR-ok közül töltse le és adja a projekthez a legfrissebb xpp3, xpp3\_min és xpp3\_xpath csomagokat!

## Csoportos küldés

### Scribe tartalom létrehozása

Hozzon létre új osztályt, mely egy csoport-üzenetet reprezentál. Ez az osztály implementálja a ScribeContent interface-t. Két mezője legyen, a feladó azonosítója és az üzenet szövege. Írja felül a toString() metódust, hogy az leírja az üzenetet.

### Scribe kliens létrehozása

Hozzon létre egy csoportos chat ablakot! Alapul veheti az előző órán elkészített ChatConversation-t. A csoport-kliensnek implementálnia kell a ScribeMultiClient interface-t. Ennek az interface-nek a deliver metódusában fogja megkapni a multicast üzeneteket, ezeket irassa ki a beszélgetésben!

#### Scribe alkalmazás

A ChatPeer konstruktorában, a ChatApplication létrehozása után példányosítson egy ScribeImpl-t is! Használja a „scribe\_instance” nevet!

#### Feliratkozás csoportba

Hozzon létre egy HashMap-et, mely az eddig felvett Topic-okat rendeli a csoportok neveihez! Hozzon létre a ChatPeer-ben egy új metódust, mely a csoport nevét várja paraméterként! A metódusban ellenőrizze, hogy feliratkozott-e a már a felhasználó a csoportra. Ha nem, példányosítson egy új Topic-ot és egy új csoportos chat ablakot majd iratkozzon fel rá a Scribe.subscribe() metódussal. Sikeres feliratkozás esetén adja hozzá a Topic-ot a Topic-ok listájához!

#### Csoportos üzenet küldése

Hozzon létre egy metódust a ChatPeer-ben, mellyel csoprtnak lehet üzenetet küldeni! A metódus várjon két paramétert, egy csoprt-nevet és egy üzenetet! Keresse ki a beszélgetést, és hívja meg vele a Scribe.publish() metódust!

#### Leiratkozás csoportról

Hozzon létre metódust, mellyel le lehet iratkozni egy csoportról! A paraméterül kapott név alapján keresse meg a Topic-ot, és használja paraméterként a Scribe.removeChild() metódushoz. A használandó lokális NodeHandle-t a node objektumon keresztül érheti el.

#### Események bekötése

A MainWindow „Join topic” gombjának megnyomásával ajánlja fel a felhasználónak, hogy válasszon csoportnevet, majd csatlakozzon a csoporthoz!

A csoportos beszélgetés ablak bezárásakor iratkozzon le a csoportról!

A csoportos beszélgetés ablak üzenetküldésével küldjön el üzenetet!

## Felhasználók azonosítása névvel

Ennek a funkciónak a megvalósításához szükség van felhasználó-nevekre. Ezt a mintaalkalmazás elkéri az alkalmazás indulásakor, és elmenti a MainWindow.userName mezőbe. Az értékét csomópont létrehozásakor továbbadja a ChatPeer-nek.

### Past tartalom

Hozza létre a DHT által tárolt tartalmat: adjon hozzá a projekthez egy UserProfile nevű osztályt, mely a ContentHashPastContent-ből származik. A konstruktora fogadjon egy azonosítót, egy nevet és a csomóont azonosítóját. Előbbivel hívja meg az ősosztály konstruktorát, utóbbi kettőt tárolja mezőként.

### Past alkalmazás

A Past alkalmazást megvalósító PastImpl létrehozása kicsit bonyolultabb, mint az eddig megismert alkalmazásoké.

Hozzon létre egy MemoryStorage példányt a cache számára. Hozzon létre egy másikat a tényleges tároláshoz. Hozzon létre egy 512 kByte-os LRUCache-t! Hozzon létre egy új StorageManagerImpl osztályt a megfelelő paraméterekkel. Végül hozzon létre egy PastImpl példányt, melynek neve „past\_instance” és a replikák száma 0.

### Lokális felhasználónév

Adja hozzá a helyi felhasználóból készített UserProfile-t a DHT-hez! Az azonosítót generálja a felhasználó nevéből, a csomópont azonosítóját a node mezőn keresztül érheti el.

### Felhasználó elérése

Hozzon létre egy findPeer metódust, amely egy felhasználónevet és egy Continuation<PastContent, Exception> callbacket fogad. Hívja meg ezekkel a past.lookup() metódust!

### Eseménykezelés

Kösse be a „Chat with peer…” gombot oly módon, hogy az alkalmazás egy dialógusban kérje be a távoli felhasználó nevét, majd indítson el egy keresést. Ha a keresés sikeresen tér vissza, nyisson meg egy beszélgetés-ablakot, ellenkező esetben jelezze a felhasználónak az eredménytelen keresést. Vigyázzon, a keresés nem csak hibával, de sikertelenül is végződhet, ekkor a PastContent értéke null lesz.

Próbálja ki az alkalmazás!