**7 Приложни комуникации**

**Комуникации в разпределените системи**

Осъществяват обмена между паралелните процеси свързвайки ги в единна система за разпределена обработка. Базират се на обмен на съобщения (message passing; РС не поддържат по- удобния за програмиране, но нескалируем модел на общата памет) Ползват се наличните мрежови примитиви за message passing. Организацията на сложни високопаралелни изчисления (стотици, понякога хиляди паралелни процеси) е възможна само с приложението на message

passing модели от високо ниво:

1)мидълуер за клиент-сървер модели:

- RPC (remote procedure call) – специализаран модел на съобщения, подходящ за приложения от типа клиент-сървер

- RMI (remote method invocation) – развитие на RPC за работа с разпределени обекти

- MOM (message-oriented middleware) – управление на транзакциите, приложимо при асинхронни (не клиент-сървер) приложения, работещи на пренципа на e-mail: messageoriented комуникации (в IBM и MS Windows: message queuing)

- поточни данни (streams) – предимно за мултимедийни приложения, данните се интерпретират като поток от аудио и видео информация с ограничения във времето за предаване – “sub real-time”

2)р2р модели

**ТСР клиент-сървер приложения**

стандартен (разширен) и транзакционен (Т/ТСР) вариант - 7.4фиг.

Използва се последовотелно номериране на пакетите за подреждане на съставните съобщения

Състояния на ТСР-процес (клиент или сървер)

LISTEN - ТСР-сървер: изчаква заявка за откриване на съединение (connection) от отдалечен ТСР-порт

SYN-SENT - ТСР-клиент: изчаква отговор ТСР-пакет с установени SYN и

ACK флагове

SYN-RECEIVED - ТСР-сървер: изчаква потвърждение за издаденото

потвърждение за откриване на съединение ESTABLISHED - ТСР-сървер и клиент: преминава в състояние на приемане и предаване на пакети от и към отдалечен порт след установено съединение

TIME-WAIT - ТСР-сървер и клиент: изчакване на времеинтервал (до 4 мин.) преди закриване на съединение

**UDP-базирани клиент-сървер приложения**

При мрежи с надеждни комуник.. За прилож., толериращи загуби. По-висока скорост и намален signaling. Ниска преносимост

**Клиент-сървер със сесийни и**

**представителни протоколи**

Сесийните пр. са специализирани да поддържат обмена в режим на диалог – синхронизация на предаващия и приемащия възел.

Представителните пр. са специализирани в представяне на съдържанието на съобщенията като структурирани записи

**Клиент-сървер с приложни протоколи**

Приложни протоколи (функциите на протокола се реализират от клиент с потребителски интерфейс)-стандартни мрежови приложения за електронна поща, файлов обмен, терминална емулация, достъп до документно съдържание (хипертекст и др.) N.B.: тези протоколи могат да се използват и с различни от първоначално зададените функции – напр. НТТР се ползва в Java RMI за обръщение към отдалечени обектни методи през защитни стени.

**Middleware услуги за клиент-сървер**

Middleware (междинни протоколи между транспортното и приложното ниво) практически е алтернатива на сесията и

представянето (без приложението)

Услуги от високо ниво – проверка за автентичност (authentication), [отдалечен] допуск (authorization) до разпределен ресурс (след успешна проверка за автентичност), разпределено резервиране на ресурси (locking) при множествен достъп, синхронизиране на поточни данни, обмен в реално време (за мултим. appl) Middleware обикновено са настройваеми към изисквания-та на типа appl, запазвайки единен нтерфейс, но ползвайки услугите на различни трансп. протоколи и затова те са необходима надстройка над мрежовите услуги

**RPC**

Отдалечено обръщение към (+ отдалечено изпълнение на) процедура – не специализи-рано, но прилагано главно за клиент-сървер приложения. Начини:

-с локални параметри; - с връщане на резултата; - с отлагане на викащия процес (синхронност); - чрез прозрачен (скрит) обмен на съобщения; - маскиране на разликата в локалните адресни пространства

(N.B.: в С към локалните параметри има обръщение по стойност и обръщение по адрес – специално към масивите)

**RPC операции**

Базират се на клиентски и сърверни стъбове, които емулират RPC като

обръщение към локална процедура.

Клиентския стъб се извиква като локална процедура, пакетира параметрит(marshaling) в съобщение и с обръщение към локалната ОС ги препраща към отдалечения сървер – операция send, след което извиква операция receive и изчаква резултата от сървера. Сърверния стъб извлича данните от съобщ. и извиква локалната процедура на сървера.

Когато сърверната процедура върне резултата, параметрите са разположени в локалното адресно пространство на сървера, откъдето сърверния стъб ги пакетира в съобщение и ги връща към клиентския стъб (след което стартира операция за изчакване на ново съобщение). За отдалечено обръщение към повече от една процедура се проектират стъбове, които кодират с номер локалните процедури и извършват съответната дешифрация при обръщение

**Последователни стъпки на RPC – 7.12**

- локално обръщение към клиентския стъб - клиентския стъб съставя съобщение (marshaling – представително ниво) и

извиква ОС

- кл ОС изпраща съобщението на сърв. ОС

- сърв. ОС предава съобщението на с. стъб

- с. стъб разпакетира параметрите от съобщението и извиква локалната сърверна процедура

- с. процедура се изпълнява и връща резултата на стъба

- с. стъб съставя съобщение и извиква с. ОС - с. ОС изпраща съобщението на кл. ОС

- кл. ОС предава съобщението на кл. Стъб - стъбът разпакетира съобщението с резултата и го предава на кл.приложение, което възстановява изпълнението си

**Асинхронно, косвено и еднопосочно RPC**

* Асинхронно RPC се прилага когато отдалечената процедура не връща резултат – кл. процес изчаква само потвърждение от сърв. стъб, че заявката е приета, 7.15фиг.
* Косвено (deferred synchronous) RPC се прилага при бавно изпълняващи се заявки с отложено обръщение на клиентската процедура към резултата. За връщане на резултата от косвено RPC сърверът ползва еднопосочно RPC

**Разпределени обекти**

Отдалеченото обръщение към обекти (и методи) разширява възможностите на RPC за прозрачни разпределени ОО приложения.

Разпределените (distributed, remote) обекти поддържат изпълним интерфейс на отдалечена машина т.е. достъпа до техните методи може да се реализира чрез отдалечения интерфейс – 7.16фиг Свързването на клиент с отдалечен обект става чрез зареждане на неговия интерфейс – proxy – в адресното пространдство на клиента (аналогично на кл. стъб при RPC) с единствена задача да пакетира обръщението към отдалечения обектен метод.Сърверът, на който е разположен обекта, изпълнява сърверен стъб (skeleton), който разпекетира обръщението и активира викания метод; след изпълнение на заявката, сърверният стъб пакетира отговора в съобщение до клиента.

N.B.: контекста на разпределените обекти също може да бъде разпределен между няколко сървера, което също остава прозрачно за клиентските програми през отдалечения интерфейс

**Версии на РО**

статични (compile-time; language-level –

Java, C++) и динамични (runtime)

Обекти устойчиви (persistent) и преходни (transient) обекти

**Статични разпределени обекти**

статични: compile-time; language-level – Java, C++; Статичните обекти позволяват компилирането и свързването на ифейсите в кл. и с. стъб. Интерфейсът[ите] към статичния отдалечен обект е публичен и може да се компилира в приложенията. *public interface SomeInterface { ... // method signatures }* Отдалеченият обект имплементира този (и други) интерфейси. Приложението се обръща към този интерфейс на отдалечен обект: *SomeInterface si =(SomeInterface) TransparentItemProxy.getItem( "//someHost:1198/someName", new Class[] { SomeInterface.class } );*

Отдалечените обръщения са с еднакъв синтаксис и семантика като локалните, но методите се изпълняват от отдалеченият сървер

**Динмични разпределени обекти**

динамичните (runtime) обекти преодоляват езиковата зависимост като приложението може да бъде разработено от обекти на различни езици. Интерфейсът към отдалечения обект се зарежда от приложението по време на обръщение:

*// obtain reference*

*Object object = Remote.getItem("//someHost:1198/someName");*

*// typically obtained at runtime*

*String someMethod = "someMethod";*

*// also obtained at runtime*

*Object someArg =new Object[] { someArgs, ... };*

*Object result = Remote.invoke(object, someMethod, someArgs);*

Често се използват се общи потоколи за обмен вкл. групов обмен (UDP/IP multicast), което позволява абониране на appl. за обяви на сървери (announcements) и се маскира адреса на сървера (прозрачна адресация)

**Устойчиви и преходни обекти**

Tекущия контекст на устойчивите обекти се съхранява във вторичната памет и изпълне-нието им може да бъде преустановявано и подновявано от различни сърверни процеси. Преходните обекти терминират щом се терминира и процеса (приложението), който е създал обръщениато към съответния обект. Устойчивите обекти могат да бъдат реферирани и след терминиране на породилия ги процес

**Явно и неявно свързване клиент-обект**

Извикването на обектен метод се извършва само след свързване на обекта. Свързването на обект представлява разполагане на обектния интерфейс под формата на негово прокси в адресното пространство на викащия процес; Свързването може да е неявно (автоматично) стига да се локализира сървера, който изпълнява съответния обект.

Неявното свързване е синтаксис за обръщение към методи по указател (reference) към обекта

Явното свързване изпълнява същата задача чрез обръщение към специална функция bind, с която присвоява обектния указател на локален адрес – прокси

Указателят към разпределен обект съдържа информация за адреса на

изпълняващия го сървер (машинен адрес + локален сърверен порт)

**Статично и динамично RMI**

След свързване на обекта обръщението към неговите методи – RMI – става през лок. обектно прокси.

Статичното обръщение използва готови интрефейсни дефиниции (изготвени напр. с IDL или с помощта на автоматичната стъб-генерация на Java).

Динамичното обръщение генерира интерфейсните дефиниции по време на изпълнение на кода, което се задава със следния примерен синтаксис

invoke(object, method, input\_params, output\_params); // например

invoke(file\_obj, id(append), int);

//id(append) returns method’s id

Динамично обръщение е по-удобно при програмиране напр. на браузери на разпределени обекти, при което обектните интерфейси трябва да се проверяват по време на изпълнение

**Предаване на обектни параметри при RMI**

- Предаването на переметри при RMI е по-прозрачно отколкото при RPC поради глобалната идентифик. на разпределените обекти =>ако параметъра е разпределен / отдалечен обект, то неговата стойност (която е указател!) може да се ползва в операцията свързване – явно или неявно.

- Тъй като се ползват не само отдалечени, а и локални обекти (напр. за ефективност) – локалните обекти-параметри се предават към отдалечения обектен сървер не като указатели, а като стойността на самите локални обекти – 7.23фиг.

**peer-to-peer мрежи**

Р2Р мрежите ползват предимно разпределе-ните общи ресурси на множество възли за протоколна обработка и за комуникации вместо централизираните ресурси на малък брой сървери и на опорните мрежи (backbone networks) – 7.24фиг.

Поради отсъствието на специализирани (dedicated) сървери р2р мрежите и appl. за тях имат инцидентен (ad hoc) характер и имплементират специфичен не-йерархичен модел; всеки от процесите е равнопоставен – с клиентски и сърверни функции.

Tипични приложения: споделяне на данни (файлове), пренос на поточни данни (мултимедия - streaming, телефонни (VOIP)), текстови лични комуникации – форуми, чатове.

Предимства:

- по-пълно използване на ресурсите

- неограничена скалируемост

- отказоустойчивост (проблематично поради ad hoc обслужването)

Проблеми:

- надеждност и качество на услугите - приложения в реално време

- защита (нерегламентиран достъп; leeching – егоистична употреба; скрито зареждане на вредносен код; спам нежелана информация)

- правни проблеми – авторски права на съдържанието; нерегламентирана употреба на системни ресурси.

**BitTorrent протокол**

- портокол, технология и среда за споделяне на файлове в ад-хок р2р мрежи

- 1/3 от Интернет трафика и 1/5 от високо-скоростния трафик (понеже се поддържа основно от PCs и потребителски мрежи, а не от високоскоростните опорни мрежи).

- Базира се на споделяне на дискови, компютърни, комуникационни ресурси с минимална сърверна поддръжка

* ВТ-клиента: 1.Поддържа списък на локални работни копия на споделените файлове и прозрачно обслужва заявки (seeder).

 2.Генерира последователност от заявки за зареждане на файл. Заявките са TCP-формат към множество възли поддържащи отделни части от файла – по-сложно управление (signaling), но потенциално уплътнен трафик. Негарантирана скорост и ред на зареждане – неподходящ за изпреварващо зареждане на поточни ММ данни (progressive streaming).

Планът на заявките е rarest-first

3. Зарежда метаданни за заявен файл – torrent – списък с текущите клиенти, поддър-жащи копия на части от файла и адрес на сървера, координиращ процеса (tracker)

* ВТ-сървер – tracker: директория с мета-данни за съдържанието (торенти – на файлово ниво); контролира коефицента на споделяне на клиентите(за избягване на leeching)

**Комуникации, базирани на съобщения**

МОМ (Message Oriented Middleware), messaging – комуникационен модел за

асинхронни разпределени приложения, базиран на обмен на съобщения.

Обменът е прозрачен и индиректен за комуникиращите процеси на приложението (вкл. “леки”, мобилни и преносими приложения). Съобщенията се съхраняват в опашки на сърверната инфраструктура.

Ограничена приложимаст за online и РВ. Подобрява планирането и обслужването в комуникационната инфраструктура – load balancing и т.н.

Моделите на обмен на съобщения са устойчиви и преходни.

Устойчивите съобщ. (обикновено пакети данни) се записват във вторична памет. Неустойчивите съобщ не се съхраняват при рестартиране на системата–напр. мониторинг

Най-разпространени технологии:

MPI, IBM's MQSeries, Microsoft Message Queuing, DECmessageQ

**МОМ – модел с преходни комуникации**

**(Message Passing)**

МОМ (както и приложенията с обмен на съобщения) се базират директно върху

примитивите на транспортното ниво посредством съединения (sockets).

Това са крайни точки нa комуникационните канали достъпни за ОС, които формират системен комуникационен интерфейс и се използват за достъп до специфична примитивна услуга *на транспортния протокол.* Съединителни примитиви в ТСР (7.29): socket откриване на ново съединение,

bind присвояване на локален адрес на съединението, listen деклариране на готовност за откриване на комуникационни канали, connect заявка за откриване на комуникационен канал (обикн. на клиента), accept приемане на заявка за откриване на комуникационен канал с блокиране на заявителя докато се създаде ново съединение (обикн. на сървера), send изпращане на съобщение по канала,

receive приемане на съобщение по канала, close закриване на комуникационен канал.

**МОМ – модел с устойчиви комуникации**

**(Message Queuing) -** подходящ за асинхронни приложения с буфериране на съобщенията в транзит и с време на обмен от порядъка на минути.

Едно или повече приложения от един възел (адрес) имат приемен буфер за съобщения (message queue); адресирането на съобщенията се извършва именно по уникалното име на буфера. При MOM буферите винаги са локални с обслужваните от тях процеси или поне в същата локална мрежа, но наборът от възможни буфери се разпределя между възлите. Буферните имена се транслират до мрежови адреси, за което се поддържа транслираща таблица – като при DNS и e-mail адресите – 7.31фиг.

**MQ инфраструктура**

Буферни мениджъри (queue managers) – 2 типа (7.33.1): - за поддържане на интерфейс към комуникиращите приложения; - маршрутизатори - routing се базира на таблици с двойките маршрутизатор-мрежов адрес; мрежовия адрес се извлича от идентификатора на получаващия процес. Поддържат регистър на предадените съобщения (log) за защита и отказоустойчивост обслужват заявките за групово предаване (multicast) като изпращат репликират изпратеното съобщение

Брокери на съобщения (message brokers)

форматът на обменяните съобщения в разпределените асинхронни приложения не е стандартизиран (както в мрежовите протоколи); Брокерите тук са в ролята на форматни конвертори или шлюзове (gateways, но на приложно ниво!), всъщност са част от разпределеното приложение, а не мрежов компонент – 7.33.2