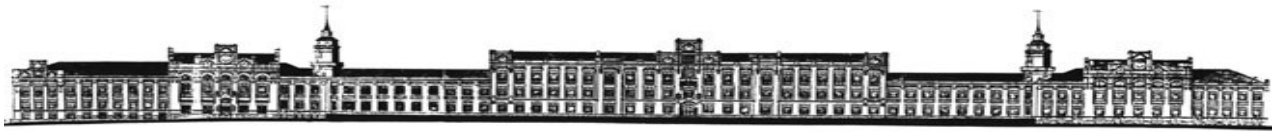


Національний технічний університет України

“Київський Політехнічний Інститут”

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ТЕХНІЧНОЇ КІБЕРНЕТИКИ

ЕЛЕКТРОННИЙ КОНСПЕКТ ЛЕКЦІЙ

З ДИСЦИПЛІНИ

“РОЗПОДІЛЕНІ СИСТЕМИ ОБРОБКИ ІНФОРМАЦІЇ”

НАПРЯМ - 050103 «Програмна інженерія»

Київ - 2016

ЗМІСТ

1 Основні поняття щодо інформаційних систем і технологій

1.1 Інформація, дані, знання

Інформація (від лат. Informatio - «пояснення») - будь-які відомості про якусь подію, сутність, процес тощо, що є об'єктом деяких операцій: сприйняття, передачі, перетворення, зберігання та використання, для яких існує змістовна інтерпретація користувачем (людиною).
Отже, для сприйняття інформації необхідна деяка сприймаюча система, яка може інтерпретувати її, у тому числі перетворювати, визначати відповідність певним правилам і т.п. Інформація використовується у всіх областях людської діяльності, тому будь-який взаємозв'язок і координація дій можливі тільки завдяки інформації.
Інформація з'являється тоді, коли є мета її використання, а мету може сформувати тільки людина.

Дані відносяться до способу подання, зберігання і елементарним операціям обробки інформації. Перш за все, дані - це носій інформації. Образно кажучи, дані - це текст в деякій абетці, а інформація - це розповідь (повідомлення), що має певний семантичний сенс для людини.

Для визначення поняття даних представимо деяку абстрактну ситуацію:

- є деяка система (подія, процес), інформація про яку представляє інтерес;
- є спостерігач, здатний сприймати стани системи і в певній формі фіксувати їх у своїй пам'яті.

Тоді кажуть, що в пам'яті спостерігача знаходяться «дані», що описують стан системи. У загальному випадку таким спостерігачем є інформаційна система.
Таким чином, «дані» можна визначити як контент, зафіксований у певній формі, придатній для подальшої обробки, зберігання та передачі інформаційною системою.
Крім інформації та даних існують також «знання».

Знання - це така інформація, до якої застосовуються алгоритми логічного висновку, що дозволяють отримати нову інформацію.

А далі Ви, робите перший висновок: дівчина не проти зустрітися з Вами. Далі, використовуючи цю інформацію та інформацію з пам'яті, що ще вересень і стоїть відмінна погода, а стипендія вже отримана, але поки не витрачена, генеруєте нову інформацію (робите другий висновок) і пропонуєте пікнік на природі у веселій компанії.
Знання повідомляють, що інформація має прагматичний аспект - тобто існує деяка мета, яка відома системі. В цьому місці з'являється аспект мети. Система, що побудована на знаннях, має момент цілясприямування (телеологічна система). Такі системи менш поширені, тому що несуть деяку відповідальність. Інформаційні системи масові та широкорозповсюджені, тому вважаються менш відповідальними порівняно з експертними системами.
Існують три аспекти роботи ІС з даними:

- визначення даних;

- маніпулювання (обробка) даних;
- керування даними (адміністрування даних).

1.2 Поняття про інформаційні технології

Під інформаційною технологією (ІТ) будемо розуміти певний процес, який схематично (в нотації міжнародного стандарту IDEF0) можна представити так:

Рис. 1.1. Загальна модель інформаційної технології

Тобто як на вході так і на виході ІТ ми маємо не матерію або енергію, а інформацію. Слово «технологія» походить від грецького «techno» - майстерність, мистецтво.

Інформаційна система (ІС) - сукупність організаційних і технічних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.

Автоматизована система (АС) - система, що складається з персоналу та комплексу засобів автоматизації його діяльності, що реалізує інформаційну технологію виконання установлених функцій.

Автоматизована інформаційна система (АІС) - інформаційна система, в якій уявлення, зберігання і обробка інформації здійснюються за допомогою обчислювальної техніки.

2 Загальні засади теорії РСОІ

2.1 Поняття РСОІ

РС містить компоненти, які розподілені з різних комп'ютерів. РС – набір незалежних комп'ютерів, які здавалися їх користувачам єдиної системою. Користувачі і мережеві додатки однаково працюють у РС, незалежно від цього, де й коли відбувається їхня взаємодія. Хост – комп'ютер, у якому розміщені компоненти обчислювальної системи: апаратура й мережева ОС.

Взаємодія РС

Це головна біль для програмістів. РС містить кілька хостів і більше одного компонента. Компоненти мають взаємодіяти один з одним.

Взаємодія компонентів
Хост надає доступ до своїх служб. Компонент може вимагати обслуговування в інших компонентів Для взаємодії використовується той чи інший варіант проміжного рівня, що вирішує проблему неоднорідності і розподілу. Проміжний шар розташовується між компонентами і мережевою ОС.

Проміжний шар
Хост – комп'ютер, у якому виконуються компоненти, складова частина розподіленої системи. Розподілена система – група автономних хостів, з'єднаних за допомогою комп'ютерних мереж. На кожному хості, де виконуються компоненти, функціонує проміжний шар. Компоненти, які координують свої дії, в такий спосіб, створюють таке уявлення, що користувач сприймає систему як єдиний інтегрований обчислювальний засіб.

2.2 Приклади РСОІ

Розподілена система обчислень від Google
PC Nutch побудований з урахуванням Hadoop — це фреймворк , який реалізує ідею MapReduce . Терміни map і reduce надійшли з функціонального програмування. Якщо «reduce» функція без побічні ефекти (тобто вони не змінює нічого поза межами області видимості), то застосовувати яку можна одночасно до кількох елементам вхідного списку. Гугловый фреймворк MapReduce дозволяє прозоро для програми розносити це вираховування за багатьма машинам. Hadoop є open-source реалізацію ж ідеї на Java.

Розподілена система синхронізації коду
Відмінність таких систем - відсутність центрального репозиторія, до якого звертаються клієнтські програми; репозиторіїв можна знайти багато й між ними є можливість синхронізації. Такий механізм роботи дає більше свободи розробникові, замість отримання робочої копії і її відправки знову на центральний репозиторій, розробник може мати простий репозиторій в своє повне володіння - вносити зміни на свій розсуд і лише окремі зміни синхронізувати із головним репозиторієм.

Розподілені системи виявлення спаму.
Інформація розташована у різних репозиторіях (параметри прийдешньої пошти, сигнатури), аналізують інформацію (аналіз технічної інформації повідомлення, аналіз тіла повідомлення (контентний аналіз методами лінгвістики або статистики), виходячи з аналізу у репозиторії, ознак спаму і вимоги синхронізації цієї інформації, корелюється з іншими розподіленими точками збору інформації.

Розподілена система управління мережами.
Забезпечує збір і аналізує інформацію про мережі. Керуюча станція зі спеціальним ПЗ, центральна БД. Агенти в різних пристроях і службах – збирають інформацію, зберігають її на локальних БД, відсилають інформацію у визначені моменти до керуючої станції. Розподілена файлова система (Distributed File System – DFS) для OS Windows Server є ще однією із мережних служб, яка спрощує пошук і освоєння

адміністрування даних в корпоративних мережах. Коли ж ви користуєтеся розподіленою файловою системою, її реальна структура прихована від вас, а насправді може мати особливості динамічного характеру. Так можна створити ієрархічну файлову структуру, кореневий каталог якої буде перебувати на сервері, проте її вузли розподіляться на різних носіях у мережі. DFS є інструмент, що дозволяє користувачам корпоративної інформаційної системи отримати однакове уявлення даних і файлів, розподілених через мережу, оскільки всі вони знаходяться в його машині.

2.3 Нефункціональні вимоги до РСОІ

Вимоги бувають: функціональні і нефункціональні. Функціональні – піддаються локалізації при реалізації. Нефункціональні – відносяться до якості системи – носять глобальний характер, надають значний вплив на вибір загальної архітектури системи на етапі проектування.

Масштабуємість
Здатність системи адаптуватися до майбутнього зростання навантаження на хост. Проблеми - вузькі місця з обслуговування (один сервер для безлічі клієнтів), за даними (один файл із загальною інформацією), по алгоритмам (централізований алгоритм і перевантаження комунікаційної мережі).

Властивості децентралізованих алгоритмів
Ніхто не має повну інформацію про систему. Рішення ухвалюються з урахуванням локальної інформації. Збій щодо одного місця визиває порушення роботи алгоритму, існування єдиного часу непотрібне.

Відкритість
Систему можна легко розширювати й модифікувати (інтеграція нових компонентів, відповідальних новим функціональним вимогам => компоненти повинні мати чітко визначені інтерфейси). Правильний інтерфейс забезпечує можливість правильної співпраці процесу з іншим, які представляють інтерфейс.

Самодостатність і нейтральність
Переносність характеризує, наскільки додаток, зроблений для однієї системи, може працювати у складі інших. Здатність до взаємодії характеризує, наскільки дві різні реалізації системи можуть спільно працювати.

Гнучкість – легкість конфігурування системи, що складається з різних компонентів, і легкість підключення нових компонентів.

Неоднорідність
У розподілених системах, компоненти повинні оголошувати запропоновані послугах. Заявки може бути синхронними/асинхронними. Клієнт і сервер може бути неоднорідними. Причини неоднорідності: Компоненти можуть купуватися в готовому вигляді. Під час створення нового компонента, системою можуть

накладатися вимоги взаємодії з компонентами. Компоненти створюються різними розробниками Використовуються різні технології

Поділ ресурсів
Ресурс – апаратура, ПЗ, дані. Потрібно визначити, кому буде дозволено керувати доступом до ресурсу, потрібно вести облік користувачів. Менеджер ресурсів – компонент, що дає керування доступом до поділюваних ресурсів.

Моделі взаємодії
Клієнт-серверна (сервер надає доступ до ресурсів) Концепція розподілених об'єктів, які надають доступ до наявних в них ресурсів при зверненні інших компонентів.

Відмовостійкість
Якість, коли система може й далі виконувати роботу навіть тоді коли виявлено несправності. Часто використовується надмірність апаратно-програмних засобів та застосування реплікацій (при відмові компонента, починає працювати його копія і обслуговування продовжується).

Прозорість в РСОІ
Має кілька різних аспектів: Прозорість масштабуємості - програміст має знати, як досягається масштабованість розподіленої системи. Прозорість продуктивності – користувач і програміст не знають, як підтримується хороша продуктивність. Прозорість відмови - користувачам і програмістам непотрібен знати, як ОС справляється з відмовами. Прозорість міграції – переміщення компонентів непомітно для користувачів і спеціальних дій із боку розробників цих компонентів Прозорість реплікації – користувачам і розробника непотрібен знати, хто надає послугу – репліка чи основний компонент. Розробники компоненти нічого не винні враховувати можливість його реплікації Репліка – копія, яка залишається синхронізованої з оригіналом Прозорість одночасного виконання. Означає, що програми користувачів не знають, що компоненти запитують послуги одночасно. Кілька компонентів можуть вимагати обслуговування разом з збереженням його якості. Користувачі і розробники бачать, як організується одночасно обслуговування. Прозорість доступу – однаковість інтерфейсів для локальної мережі й віддаленого зв'язку (інтерфейс заявки обслуговування може бути у тому для зв'язку між компонентами одного хоста і різноманітних хостів). Прозорість місцезнаходження – спосіб виклику операції залежить від місцезнаходження компонента (обслуговування об'єкту, що запитує, непотрібно знати про фізичне розташування компонента). Клієнт не повинен знати про місцезнаходження компонента або його репліки.

Віддалений виклик процедури: загальні відомості.
Є машини: A і B. A викликає процедуру, яка виконується на B. count = read(fd , buf , bytes);
Стек при виклику процедури *bytes buf fd* повертає адресу повернення локальних змінних.
Передавати параметри за значенням просто, а по запиту – проблема.

Виклик через клонування відновленням.
Віддалений виклик виглядає як локальний, тобто RPC забезпечує прозорість клієнтові. Сервер також підозрює, що він виконує віддалений виклик. На сервері є аналогічна заглушка; сервер виконує запит, повертає результат. Проблема: передача за адресою. Рішення: можна ігнорувати копію буфера. Послідовність передачі управління при RPC:

Передача параметрів при віддаленому виклику процедур
При такому виклику процедур процес на 1 вузлі викликає процедуру процесу на 2 вузлі. Складність у тому, що згадані процеси працюють у різних адресних просторах (АП). При передачі параметрів за значенням це неважливо, т.ч. значення залежить від АП, а під час передачі по запиту виникають проблеми. Тут можна використовувати інший варіант передачі параметрів – копіювання/відновлення.

Передача параметрів за значенням.
Формується пакет, у якому ім'я процедури і його параметри; Повідомлення приймається заглушкою-сервером. Заглушка на сервері формує виклик процедури (як локальної). Клієнтський процес призупиняє діяти і чекає повернення результату. З отриманням результату він продовжує роботу.

Передача параметрів по запитанню
Приклад: читання віддаленого файлу в масив. Передати копію посилання неможливо, оскільки посилання – це покажчик в АП клієнта, безглуздо передавати її копію. Варіант рішення: можна зробити так - взяти масив, який показує покажчик, і просити передати копію цього масиву. Усе це міститься у повідомленнях, переданих на сервер. На сервері виділяється місце під масив, посилання масиву - в АП серверу. Заглушка передає параметри серверному процесу, і він за всіх правил звертається до ОС й поміщає результат в масив у своїй АП. Заглушка упаковує масив в повідомлення передає його на клієнтську заглушку з допомогою ОС. Клієнтська заглушка розпаковує це повідомлення і поміщає значення у той масив, що й у АП клієнта. Клієнтський процес отримує результат, начебто він звернувся локально. Ця віддаленість прозора для обох сторін. Заглушки повинні мати один протокол, відповідно до якого представлятимуться вбудовані типи. Якщо заглушки працюють за одним й тим протоколом, вони різняться лише інтерфейсами машин. Щоб полегшити напрацювання над створенням заглушок, використовують мову визначення інтерфейсів. IDL – Interface Definition Language.

3 Організація розподілених об'єктів

Розподілений об'єкт (РО) – такий об'єкт, реалізація якого розташована на сервері, а взаємодія клієнтів із нею виконується через певний інтерфейс. У адресний простір клієнта завантажується реалізація цього інтерфейсу – заступник (proxy). Клієнт безпосередньо взаємодіє саме з його заступником. Процес формування інтерфейсу і затримання заступника – прив'язка клієнта до РО. Системи з РО надають посилання об'єктам, що унікальні не більше системи. Такі посилання можуть передаватися між процесами, запущеними на різних машинах.

Види прив'язки

Неявна (автоматична) – клієнт прозоро пов'язують із об'єктом в останній момент дозволу посилання (це коли з імені об'єкта отримуємо посилання нього). Явна – клієнт має викликати спеціальну функцію для прив'язки об'єкта.

Адаптер об'єктів – механізм групування об'єктів відповідно до політики їх активізації. Контролює один чи кілька об'єктів.

Скелетон – образ клієнта на сервері (заглушка серверу). Клієнт через заступника викликає певний метод об'єкту і задає параметри цього. Заступник формує і упаковує повідомлення («маршалінг») і посилає його на сервер. У цьому повідомленні містяться всі посилання на об'єкт, метод і значення параметрів. Сервер отримує повідомлення, і передає його відповідному скелетону . Скелетон як розпаковує повідомлення (демаршалінг), так і безпосередньо викликає певний метод об'єкта із наперед заданими клієнтом параметрами. Якщо передбачається, що метод щось повертає, то скелетон упаковує результат і посилає його заступнику.

Об’єкти, які зберігають і нерезидентні об'єкти

Об’єкт, який зберігають – об'єкт продовжує існувати, навіть перебуваючи постійно в адресному просторі серверного процесу. Тобто, об'єкт можна відтворити з пам'яті незалежно від наявності процесу. Нерезидентний – існує, поки управляє сервер. Коли сервер завершує роботу, цей об'єкт перестає існувати.

Способи визначення місцезнаходження РО

- Іменування – управління просторами імен, котрі представляють собою набори перетинів поміж іменами об'єктів і посиланнями ними.
- Трейдинг – визначення місцезнаходження об'єктів з надання функцій і забезпечення якості обслуговування.

Передача параметрів при зверненні до віддалених об'єктів

Статичне глухе звернення до методів (RMI). При статичному зверненні інтерфейс віддаленого об'єкта описується з допомогою IDL , тобто. інтерфейс відомий на етапі компіляції. Приклад: Опис інтерфейсу футболіста. interface Player : Object { typedef struct Date { short day ; short month ; short year ; } attribute string name ; readonly attribute Date Dob ; }; interface PlayerStore : Object { exception IDNotFound (); short save (in Player); Player load(in short id) raises (IDNotFound); void print(in Player p); } З використанням статичного звернення інтерфейси повинні прагнути бути вже відомі і при зміні інтерфейсу потрібна перекомпіляція .

Динамічне глухе звернення до методів. При динамічному зверненні інтерфейс віддаленого об'єкта заздалегідь ніхто не знає. Параметри звернення до методу збираються для виконання. Заздалегідь невідомо, до якого методу буде звернення. invoke (object , method , input inparam , out outparam);

Передача параметрів. Використовуються посилання на об'єкти як параметри, що передаються при зверненні до віддаленого об'єкту. Об'єкт, посилання на який передається: перебуває у адресному просторі клієнта; причому, перебуває віддалено. Вони реалізуються по-різному. Посилання передається лише до віддалених об'єктів. Якщо об'єкт локальний, то передається копія самого об'єкта. При такому далекому виклику клієнтом серверу здійснюється копіювання об'єкта О1 і передача посилання об'єкта О2 .

3.1 Сервери об'єктів

Сервери об'єктів (СО) – сервери, зорієнтовані на підтримку розподілених об'єктів. СО (на відміну традиційних серверів) НЕ надає конкретні служби,тому що конкретні служби реалізуються об'єктами, розташованими на сервері. СО надає тільки звернення до об'єктів, засновані на запитах від віддалених клієнтів. Активізація об'єкта – переміщення об'єкта в адресне простір серверу (наприклад, десеріалізація).
Правила звернення до об'єктів – політика активізації. Потрібен механізм угруповання об'єктів відповідно з політикою активізації кожного з них. Цим механізмом є адаптер об'єктів. Найчастіше він криється у традиційному наборі побудови СО. Кожен адаптер об'єктів контролює 1 чи кілька об'єктів. З отриманням запиту до жодного з контрольованих об'єктів адаптер перевіряє їх стан й за необхідності активізує відповідно з політикою активізації. Після цього запит передається до заглушки (скелетону) об'єкта, яка виробляє демаршалінг (розпакування параметрів) та здійснює виклик методу. Адаптер не знає про інтерфейсах об'єктів.

Перенесення коду в РСОІ
Перенесення коду необхідний: перерозподілу навантаження між вузлами підвищення продуктивності; зниження трафіку клієнт-серверної взаємодії. У завданні виділяють такі сегменти: сегмент коду – команди; сегмент виконання – контекст завдання; сегмент ресурсів – ресурси.
Моделі перенесення коду
Мінімальні вимоги до перенесення коду пред'являє модель слабкої мобільності – перенесення лише сегмента коду. Програма завжди виконується зі свого вихідного стану. Приклад: Java Applet . У моделі сильної мобільності переноситься сегмент коду і сегмент виконання. Процес припиняється, переноситься і запускається на іншому вузлі. Приклад: мультіагентна платформа.

Типи зв'язку процесу з ресурсом:
Процес передає з точністю той ресурс, на який посилаються аргументи (найсильніший зв'язок); Більше слабкий зв'язок - процесу потрібно лише значення. Найбільш слабка форма зв'язку – процес вказує тільки використання ресурсу певного типу.

Програмний агент.
Автономний процес, здатний реагувати на середовище виконання й викликати у ній зміни, можливе кооперування з іншими агентами (кооперативні агенти) і користувачем. Агент може функціонувати автономно, зокрема виявляти ініціативу. У мультіагентній системі фігурують кооперативні агенти, що вирішують спільне завдання.

3.2 Модель клієнт-сервер та її компоненти

Сервери – процеси, які реалізують служби й надають до них доступ. Клієнти – процеси, що використовують ці служби. Розглянемо варіант прикладу доступу до

БД : При трирівневій організації системи маємо такі логічні рівні: Користувацький інтерфейс. Компоненти обробки даних (безпосередня робота з БД).
Варіанти фізичного поділу рівнів між вузлами
У цьому випадку клієнт «товщає» зліва-направо. Взаємодія клієнта з сервером відбувається у такому порядку:

Загальні зведення про іменовані об'єкти і служби іменування
Іменування – спосіб визначення місцезнаходження розподілених об'єктів, у якому здійснюється управління просторами імен, котрі представляють собою набори перетинів поміж іменами об'єктів і посиланнями, що здійснюються ними. Контекст іменування – послідовність простих імен, що ідентифікує об'єкт. Наприклад “UEFA”, “ England ”, “Premier”, “ Chelsea ”.

Операції:
Зв’язування – реєстрація об’єкта-сервісу на ім'я та об'єктного запиту. Використовується при додаванні об'єкта до системи або за переміщенні / копіюванні існуючого об'єкта. Дозвіл – отримання посилання об'єкт з його імені. Використовується клієнтом щоб одержати доступ до методів об'єкта.

Розміщення мобільних сутностей.
Мобільні сутності – це об'єкти, що потенційно можуть гуляти по різних хостах . І тут простір імен зручно розбити на 3 рівня: Глобальний Адміністративний та Управлінський У перший і другий поміщаються об'єкти, які переміщаються досить рідко. Тут ефективно кешування шляхів. Три рівні для об'єктів використовують часто, і як приклад, використовує наступна система:
На ім'я об'єкта служба іменування визначає її ідентифікатор, потім служба локалізації по ID знаходить його фізичний адресу.

3.3 Засоби синхронізації часу в середовищі розподілених додатків

Загальну інформацію про синхронізацію в PCOI - кожен вузол має свій системний таймер. Виникає проблема синхронізації годин. Нехай у вузла N час Cp(t), тоді: де delta – максимальна швидкість дрейфу часу, t - UTC. Якщо потрібна розсинхронізація трохи більше, то через кожний встановлений період необхідна синхронізація годин.

Алгоритм Крістіана.
Є вузол, який приймає сигнали точного часу – сервер часу. Інші вузли через певні інтервали звертаються до сервера й отримують значення часу. Тут є два моменти: Необхідний час на запит і повернення результату. Рішення: якщо час попросили в останній момент T0 , а отримано в останній момент T1 , то встановити годинник на отриманий час + (T1-T0)/2 . Якщо годинник відстає, ми їх доставляємо. Якщо ж годинник поспішає, і ми їх маємо ставити, що призведе до роз синхронізації, тому це неприпустиме. Тоді ми одномоментно підлаштовуєм годинник, розтягуємо це у часі, просто уповільнюючи їх перебіг.

Алгоритм Берклі.

Один вузол – демон – періодично збирає часи інших вузлів, отримує середнє і розсилає назад.

Логічний годинник. Алгоритм Лампорта.
Є ситуації, коли важливо не точний час виконання процесу, а точна послідовність виконання. Для таких випадків використовують досить часто алгоритм Лампорта для синхронізації логічних годин. Лампорт визначив ставлення: «Відбувається раніше». Воно позначається: **a** і **b**. Це означає, що ці процеси згодні про те, що подія **a** відбувається раніше **b**. Для процесу, якщо **a** раніше **b**, то ставлення виконується. Якщо **a** - посилка повідомлення, а **b** – отримання того самого повідомлення, то ставлення теж виконується. Ставлення транзитивно. У алгоритмі для кожної події **a** ставиться мітка часу C(a). Ця мітка має бути прийнята як достовірно правильна усіма процесами. Тобто, якби справді **a** раніше **b**, то C(a) < C(b). Кожному повідомленню прикріплюється тимчасова мітка. Одержувач порівнює її з своїм часом. Якщо його час менше мітки, воно встановлюється рівним метка+1 . Навіть якщо його 2 повідомлення послані майже одночасно, їх мітки різняться.

3.4 Життєвий цикл розподілених об'єктів

Створення об'єктів.
Потрібно створити об'єкт в адресному просторі (АП) серверу (конструктор створив в АП клієнта). І тому необхідна фабрика. Щоб знайти фабрику, необхідний шукач фабрик. Щоб знайти шукач фабрик, використовується сервер іменування чи трейдинг.
Міграція об'єктів. Об’єктна міграція – копіювання чи переміщення об'єкта серверу із першої машини в іншу. Щоб об'єкт дозволяв міграцію, необхідний інтерфейс, який реалізує операції «copy» і «move» з параметром – шукачем фабрик. Через фабрику створюється об'єкт на новому місці, для нього копіюється стан вихідного об'єкта. Далі у разі переміщення вихідний об'єкт видаляється. При копіюванні: створюється нове посилання на об'єкт (тому що вихідний об'єкт залишається). При переміщенні: Є об'єкт у якомусь адресному просторі. При переміщенні він залишає заступника, а в новому АП формується скелетон («Скелетон програмування» - Skeleton programming) є стилем програмування на основі простих програм на високому рівні структур і так званого фіктивного коду. скелети програми нагадують псевдокод, але дозволяють синтаксичнийаналіз, компіляцію і тестування коду). Така схема прозора для клієнта: він не знає, що відбувалися переміщення. З отриманням запиту відповідь відбувається по прямий, якщо відомо, хто запросив сервіс цього віддаленого об'єкта.
Якщо ланцюжок обірвався (розрив зв'язку, зависання…), дістатись віддаленого об'єкта більше неможливо. Боротися із цим можна з допомогою базової точки. Машина, де створювався об'єкт, називається *базовою точкою*. Потрібна, щоб ця машина мала інформацію про поточне місце розташування створеного нею об'єкта. Тоді то й вдається ланцюжком покажчиків дістатись до об'єкта. Тобто, якщо якийсь проміжний процес зависнув, то відбувається звернення до базової точці.
Копіювання об'єктів. Щоб об'єкт дозволяв копіювання, необхідний інтерфейс, який реалізує операцію «Copy» з параметром – шукачем фабрик. Через фабрику

створюється об'єкт на новому місці, для нього копіюється стан вихідного об'єкта. Складається нове посилання на об'єкт (тому, що вихідний об'єкт залишається).

3.5 Платформа для розробки розподілених додатків корпоративного рівня

Технологія J2EE

Загальні відомості: платформа J2EE (**Java 2 Enterprise Edition)** – комплекс взаємодіючих технологій, які базуються на специфікаціях фірми Sun і які мають стандарт розробки серверних додатків (рівня підприємства). Особливості: Незалежність від операційної платформи. Простота розробки додатків з урахуванням компонентної технології. Переносність і розширюваність. Можливість розробки розподілених додатків. Можливість інтеграції з іншими платформами. Можливість інтеграції з існуючими інформаційними системами. Забезпечення надійний захист інформації. J2EE – набір специфікацій, які визначають правила, які слід дотримуватися постачальникам конкретних реалізацій J2EE , і навіть розробникам додатків. Платформа надає API та виконавче середовище для розробки і виконання корпоративного програмного забезпечення, включаючи мережеві та веб сервіси, та інші масштабовані, розподілені додатки. Java EE розширює стандартну платформу Java (Java SE - Java Standart Edition).
J2EE є промисловою технологією і, в основному, її використовують у високопродуктивних проектах, у яких необхідна надійність, масштабованість, гнучкість. Компанія Oracle, яка придбала Sun (фірму, що створила Java), активно просуває Java EE у поєднанні зі своїми технологіями, зокрема з СУБД Oracle.

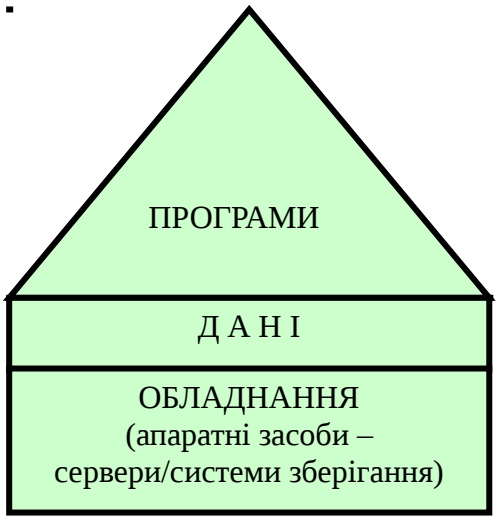
Архітектура J2EE

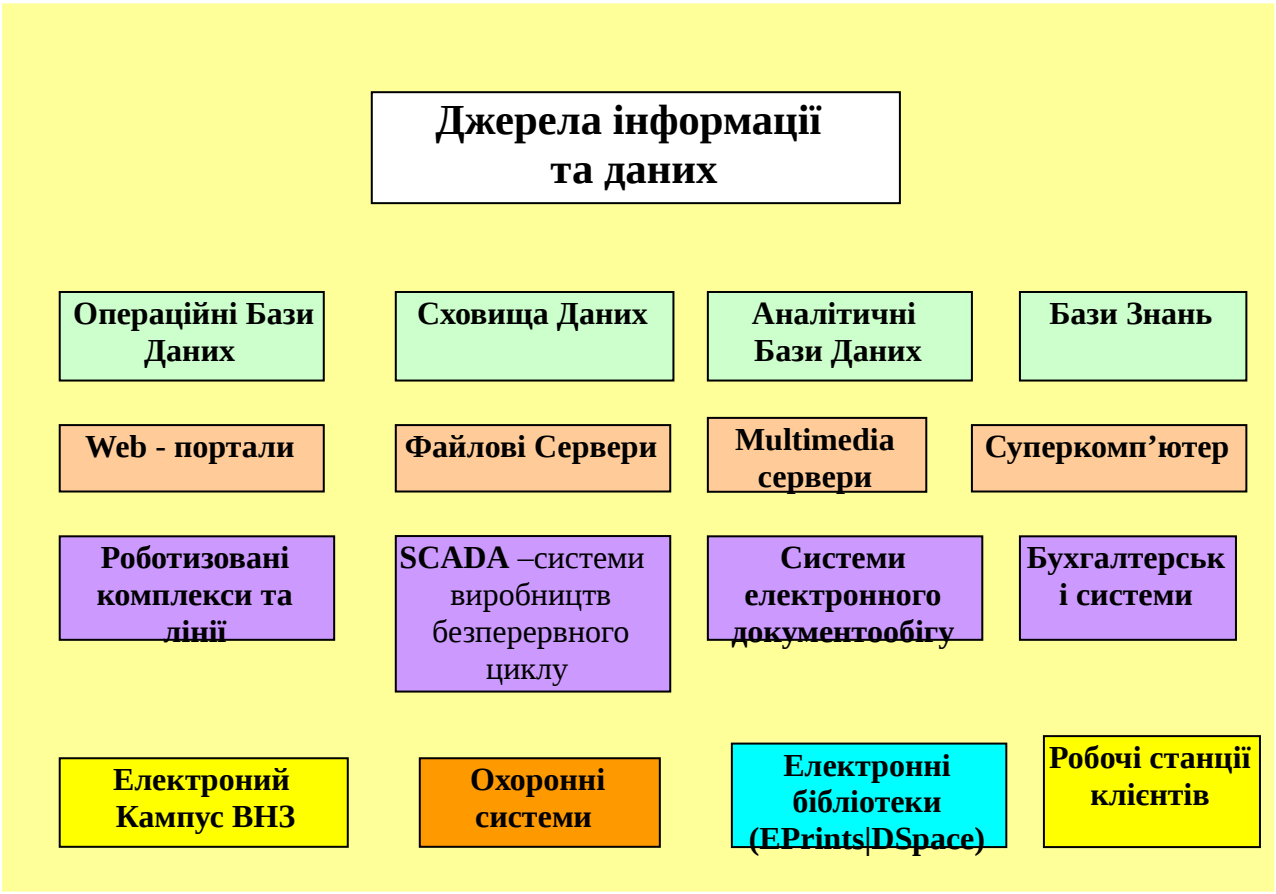
Підтримуються різні типи клієнтів: HTML – браузери, аплети, автономні java-додатки . Рівень уявлення – часто реалізується у вигляді веб-рівня . Рівень бизнес-логики – як рівня EJB (Enterprise Java Beans). Рівень інтеграції – рівень серверу БД – EIS (Enterprise Information Server). Це адаптери ресурсів J2EE. Сервер додатків – містить контейнери компонентів EJB.
Особливості: Доступ до інфраструктурі J2EE. Управління життєвим циклом компонентів EJB. Доступ до БД з допомогою JDBC. Контейнер ізолює компонент від клієнта. Усі запити перехоплюються контейнером. Кожна компонента є об'єктом EJBContext, що є посиланням на контейнер. Контейнер автоматично створює набір з’єднань з БД. Контейнер дозволяє об'єднувати кілька компонент всередині однієї транзакції. Широко використовуються такі абревіатури: JMS – Java Messaging Service; JSP – Java Server Page; JTA – Java Transaction АПІ; JAF – Java Beans Activation; Framework JAXP – Java АПІ for XML Parser; JAAS – Java Authentication and Authorization Service; EJB – Enterprise Java Beans.
EJB – серверна Java технологія, джерело якої можна знайти в транзакціях. Дозволяє швидко і просто розробляти розподілені, транзакційні, безпечні і портируємі Java додатки. Компонентом EJB є Remote – Розширений інтерфейс. Визначає методи компонента. Remote Home – визначає методи життєвого циклу для створення, видалення, пошуку компонент (інтерфейс фабрики класів). Local – цей інтерфейс використовується іншими компонентами, які у тому ж контейнері. Виклик відбувається так. Модулі EJB – об'єднані у групу компоненти EJB , які можуть взаємодіяти.
Типи компонентів EJB :

1) Session – пов'язані з бізнес процесами додатків; мають доступ до БД, але надають обмеження доступу до неї; життєвий цикл – до перезавантаження серверу (виклик сесійних компонентів: сервлети , сторінки JSP, Java додатки). Розділяються на два типи: Stateless – не зберігає інформації про стан, а Statefull – можуть зберігати інформацію про стан (але вони принципово різняться життєвими циклами).
2) Entity – моделюють бізнес дані додатків; надають доступ до БД; часто 1 звертається до 2; t життя = t життя БД (при перезавантажені сервера - автоматично відновлюються); виклик з 1 компонентів WEB;
3) MessageDriven – представляють дії. Їх можна викликати лише відправивши повідомлення цьому компоненту. За допомогою 3 організують доступ до 1; t життя, як у 1. Ланцюжок обертань у J2EE можна визначити так: Java BeansJB це EJB, EJB більш велике поняття. JB – відносять до створення інтерфейсу клієнта, для взаємодії між сторінками. EJB – до створення серверних додатків, тільки візуальні компоненти.

3.6 Фактори розподілу обробки

Основними факторами, які призводять до необхідності розподіленої обробки є розташування в різних вузлах комп’ютерної мережі джерел даних, програмних і апаратних засобів обробки. Причинами які надихають розвиток технологій розподіленої обробки даних та інформації є вимога прискорення обчислень та доступу до даних, підвищення надійності інформаційних сервісів, розширення функціональності прикладних додатків та їх взаємодії з іншими додатками, які написані на різних мовах програмуванння, гетерогенне середовище обробки, інше.





3.7 Типи об'єктів та варіанти обробки

Можна виділити наступні об'єкти, які можуть обробляти розподілені системи:

1. Об'єкти обробки
   Числові ряди, роклад функцій, обчислення матриць
   Інформація:
   a) Електронні документи
   b) Текстові дані
   c) Повідомлення системи обробки чи її компонентів
   d) Фото- та відео ряди чи файли
   e) Оцифрований звуковий ряд (файл)
   f) Компоненти бізнес-процесів
   g) SQL – запити
   h) Дані для Web-додатків (для броузера)
   Програмний код

2. Розпаралелювання обчислень – окремий випадок розподіленої обробки інформації

3. Варіанти обробки

№ п/п	Причина виникнення задач розподілу обробки	Типи даних чи функції	Приклади реалізації
1	Для вирішення задачі розробки паралельної програми знаходження значення f(x) заданої функції f у заданій точці x0 із заданою точністю ε чи інтерування f(x), а також при обчислюванні матриць	-експонента -натуральний логарифм -гРСетричний ряд -тригонометричні функції -стовпи/строки/ блоки матриць	-розклад математичної функцій -програми чисельного інтегрування функції f(x) -обчислення матриць
2	Обробка допускає розбиття обчислень по різних ядрах і процесорах (в межах одного сервера чи робочої станції)	-числові дані -повідомлення -SQL-запити	-віртуальні сервери -SQL-сервери (СУБД, сервери транзакцій)
3	Декілька вузлів мережі повинні оброблювати одну і ту інформацію	-документи -текстові файли -повідомлення	Віддалені філії організації (бухгалтерія, канцелярія, відд.кадрів)
4	Обчислення можна розділити по різних комп’ютерах чи вузлах	-числові дані -повідомлення -SQL-запити	-GRID-мережі -суперкомп’ютери
5	Алгоритм обробки допускає/вимагає паралельну обробку різних функціона-льних частин розподіленого додатку чи сервісу	-числові дані -бізнес-процеси -SQL-запити	-сервери додатків в клієнт-серверній архітектурі -сервіси в SOA і в Хмарі
6	Необхідно забезпечити високу надійність зберігання даних за рахунок їх дублювання та надлишковості	-числові дані -документи -фото-, відео-, звуковий ряд	-RAID – технології (системи зберігання DAS/NAS/SAN) -дублювання ресурсів сервера
7	Треба забезпечити високу надійність обробки обчислювальних процесів за рахунок архітектурних рішень взаємодії серверів	-числові дані -документи -бізнес-процеси	-кластери -багатовузлові катастрофостійкі системи обробки
8	Необхідно забезпечити балансування навантаження для групи серверів у різних вузлах корпоративній мережі чи для віртуальних серверів одного вузла	-числові дані -повідомлення -SQL-запити	-сервери балансування навантажень -сервер транзакцій -пакет ОС

4 Архітектура систем обробки даних

Традиційно обробка даних була централізованою. Робота з даними в архітектурі централізованої обробки даних забезпечувалася одним комп'ютером або кластером комп'ютерів. Як правило, це були крупні комп'ютери, розміщені в центрі обробки даних. Багато задач запускалися безпосередньо з обчислювального центру, результати роботи також опинялися в обчислювальному центрі. Прикладом того є додатки, що здійснюють розрахунок заробітної плати. Рішення інших задач може вимагати інтерактивної участі персоналу, що не знаходиться в центрі обробки даних. Наприклад, ввід даних, необхідний при інвентаризації, може виконуватися персоналом на робочих місцях. В рамках централізованої архітектури кожен користувач забезпечується терміналом, з'єднаним з центральним комп'ютером. Обробка даних є повністю централізованою в наступному сенсі: Централізовані комп'ютери. Один або декілька комп'ютерів розміщуються в центрі обробки даних. У багатьох випадках використовуються мейнфрейми, що вимагають особливих умов, наприклад вентиляція приміщень або знімної підлоги. У організаціях меншого масштабу центральним комп'ютером (або комп'ютерами) є високопродуктивний сервер або середній за можливостями.

4.1 Централізована обробка

Всі додатки працюють на центральному обчислювальному вузлі. Сюди входять додатки, які за своєю природою вимагають централізації або служать всій організації, наприклад системи розрахунку заробітної плати або додатку для користувачів даного відділу.

Прикладом останніх додатків є графічний пакет системи автоматизованого проектування (САПР), що працює на центральному комп'ютері відділу розробки.

4.1.1 Централізовані дані.

Всі дані зберігаються у файлах і базах даних центру обробки даних. Центральний комп'ютер має до них доступ. Дані включають інформацію, потрібну різним підрозділам, таку як відомості по інвентаризації, а також інформацію для підтримки роботи одного відділу.

4.1.2 Централізоване управління.

Менеджер обробки даних або менеджер інформаційних систем несе відповідальність за засоби обробки даних. Залежно від розміру і важливості обчислювальних засобів контроль здійснюється на середньому або на верхньому керівному рівні.

4.1.3 Приклади централізованого обслуговування.

Служба централізованої обробки даних включає технічний персонал, котрий обслуговує устаткування обробки даних і безпосередньо працює з цим устаткуванням. Крім того, програмування переважно виконується персоналом обчислювального центру.

У такої централізованої організації є ряд переваг. Можна економити на покупці і обслуговуванні апаратного і програмного забезпечення.

У центрі обробки даних можуть працювати професійні програмісти, які вирішують завдання для різних відділів. Керівництво може управляти обробкою даних, вводити стандарти програмування, систематизувати структуру файлів даними, продумати і реалізувати політику безпеки. Як приклад централізованої обробки даних розглянемо засоби обробки даних мережі готелів Holiday Inn. Робота головного офісу готелів, розташованого в Атланті, забезпечується декількома робочими станціями і персональними комп'ютерами, зв'язаними локальною мережею з різними серверами. Ці сервери підтримують більшість файлів, використовуваних в повсякденній роботі і в роботі головного офісу. Центр обробки даних знаходиться за 15 миль від головного офісу і зв'язаний з ним орендованою цифровою лінією з пропускною здатністю 44 Мбіт/с. Центром обробки даних є два мейнфрейми IBM: один служить для обробки транзакцій на замовлення номерів від торгових агентів, готелів, приватних осіб; інший мейнфрейм виконує ділові додатки, наприклад, програми обліку фінансових і людських ресурсів. Центр даних зв'язаний за допомогою супутникових ліній зв'язку з кожним з 1600 готелів мережі Holiday Inn в США, є супутникові канали зв'язку з Європою. Централізована конфігурація відповідає безлічі потреб мережі готелів. Система бронювання номерів - найбільша у своєму роді у всьому світі - обслуговує близько 25 мільйонів запитів на рік.

Єдина централізована система бронювання означає, що найсвіжіша інформація про наявність номерів у всіх готелях зібрана в одному місці. Така система з актуальною інформацією допомагає забезпечити високий відсоток зайнятих номерів в мережі готелів Holiday Inn. Крім того, централізована система збирає і зберігає зведення про поведінку клієнтів та інші подробиці про роботу кожного готелю. Це дозволяє вищому виконавчому керівництву аналізувати інформацію в цілях вироблення рекомендацій по задоволенню вимог клієнтів. Без центральної системи дуже складно було б зібрати і використовувати безліч різних типів даних, необхідних для аналізу попиту.

Засоби обробки даних можуть в тій чи іншій мірі відхилятися від централізованої структури, реалізовуючи стратегію розподіленої обробки даних. Засоби розподіленої обробки даних влаштовані так, що комп'ютери з обробки інформацію (а це звичайно малі комп'ютери), розосереджені по всій організації. Мета такого розосередження - найефективніше обробляти інформацію з погляду функціональних і економічних можливостей або географічного розташування систем. Засоби розподіленої обробки даних можуть включати центральний комп'ютер і додаткові комп'ютери, або така система може складатися з набору однакових обчислювальних засобів. У будь-якому випадку потрібний зв'язок між окремими обчислювальними вузлами. З приведеного тут опису централізованої обробки даних очевидно, що розподілена обробка даних має на увазі розподіл комп'ютерів, обчислювальних завдань і самих даних.

Розподілена обробка даних (Distributed Data Processing, DDP) - це така обробка, при якій все або деякі функції обробки, зберігання і введення- висновку даних разом з функціями управління розосереджені по декількох станціях.

Прикладом систем розподіленої обробки даних служить система обробки інформації ринкового відділення агентства з цінних паперів Дж. П. Моргана (J. P. Morgan). Головним завданням відділення є торгівля цінними паперами, забезпеченими заставами. По таких цінних паперах їх власникам відраховуються відсотки протягом всього часу, поки застави приносять дохід. Визначення цінності вкладення є складним завданням, яке включає як оцінку часу, протягом якого застави приноситимуть відсотки (виплата відсотків припиняється, якщо цінності були продані, рефінансовані або перейшли до іншого власника), так і прогноз змін процентних ставок. Для таких прогнозів використовуються різні алгоритми. У вхідних даних для алгоритмів враховується ряд поточних чинників і історія попередніх транзакцій з величезної бази даних Дж. П. Моргана. Маклери мають доступ до необхідних даних, можуть вибрати алгоритм рішення для даного завдання, у кожного маклера є могутній комп'ютер для розрахунків по вибраному алгоритму, цей комп'ютер може при необхідності дістати доступ до оновленої бази даних.

Окремі маклери мають певну свободу у виборі комп'ютера. Залежно від об'єму робіт, персональних переваг і алгоритмів маклери вибирають найбільш переважну конфігурацію серед IBM-сумісних персональних комп'ютерів, комп'ютерів MAC або високопродуктивних робочих станцій SUN. Ці комп'ютери пов'язані один з одним локальними мережами. Комп'ютери групуються в окремі локальні мережі згідно вимогам маклерів по обміну даними один з одним. Всі комп'ютери маклерів об'єднані

Загальною локальною мережею, підтримуваною поряд високопродуктивних серверів з великими об'ємами пам'яті. Сервери підтримують централізовану базу даних і деякі спеціальні додатки, наприклад додатку для ведення бухгалтерських розрахунків і підготовки звітів, які небажано запускати на комп'ютерах маклерів. Крім того, є засоби зв'язку з іншими відділеннями компанії Моргана і її офісами в Європі і Японії.

4.2 Розподілена обробка даних

4.2.1 Історія розвитку розподіленої обробки даних

До початку 70-х років минулого століття практично повсюдно була поширена централізована обробка даних. З того часу має місце постійна еволюція у бік розподіленої обробки даних. Ми можемо дивитися на цю еволюцію з різних точок зору: з погляду технічних засобів і з погляду мотивів. Спершу розглянемо зміни в індустрії обробки даних, які привели до розвитку засобів розподіленої обробки даних. Потім перейдемо до питання про те, чому все частіше і частіше вибір робиться на користь розподіленої обробки даних.

Ключовим чинником, можливим, що зробив, розподілену обробку даних, є різке і постійне зниження вартості апаратного забезпечення комп'ютерів, що супроводжується зростанням його можливостей. Сьогоднішні персональні комп'ютери володіють швидкостями роботи, наборами інструкцій і об'ємами пам'яті, порівнянними із зовсім недавніми (в межах декількох років) характеристиками мінікомп'ютерів і навіть мейнфреймів. Не менш важливим

чинником є графічний інтерфейс користувача (Graphical User Interface, GUI), що забезпечує безпрецедентні простоту роботи і реактивність системи.

Доступність недорогих, але потужних систем, дозволяє організації не використовувати централізовані засоби обробки даних у поєднанні з терміналами, що забезпечують доступ до них, а розосередити обчислювальні засоби. Проте централізовані засоби можуть як і раніше залишатися доступними. Як і інші типи комп'ютерів, мейнфрейми, що є серцем централізованих засобів, стали дешевшими і могутнішими.

Спершу розглянемо вимоги до обчислювальних засобів корпорацій. У табл. 1 перераховано дев'ять особливих вимог до обчислювальних систем.

Ясно, що частково вимоги можна задовольнити шляхом розподілу дешевих серверів, робочих станцій і персональних комп'ютерів. Повсюдне розповсюдження малих комп'ютерів дозволяє у високому ступені індивідуалізувати обчислення у всіх відділеннях, даючи можливість користувачам автономно вирішувати свої задачі на власному устаткуванні і у власному обчислювальному середовищі, активно використовувати власні обчислювальні засоби, що зрештою веде до збільшення продуктивності всього відділення.

Вимоги до корпоративних обчислювальних засобів - надавати обчислювальні ресурси в будь-якому підрозділі, де це потрібно підтримувати необхідні бюджет і повсякденні витрати, що забезпечують функціонування всіх обчислювальних засобів організації задовольняти особливим вимогам відділень забезпечувати узгодженість всіх операцій, залежних від обчислювальних засобів (наприклад, уникати невідповідностей в роботі різних відділень) задовольняти інформаційним вимогам керівництво, надавати обчислювальні ресурси на надійному, професійному і технічно грамотному рівні, давати підрозділам достатню автономію у виконанні своїх завдань з метою зростання творчих успіхів і продуктивності зберігаючи автономію підрозділів, по можливості збільшувати їх важливість і вплив в рамках всієї організації робити роботу не тільки продуктивною, але і приємною.

Два аспекти вимог користувачів підтверджують сказане - це потреба в нових додатках і малому часі відгуку. Спочатку розглянемо потребу в нових програмах. Щоб організація залишалася конкурентоздатною, кожне відділення повинне постійно прагнути до зростання продуктивності і ефективності праці. Основним джерелом таких поліпшень є опора, що росте, на обробку даних і комп'ютери. Це призводить до того, що в більшості добре керованих організацій потреба в нових додатках виникає частіше, ніж можуть оновлюватися централізовані засоби обробки даних. Тому є безліч причин, включаючи невідповідність методик передачі даних вимогам користувачів і професійних програмістів; крім того, в будь-якій розвиненій системі обробки даних велику частину часу програмісти витрачають на обслуговування програмного забезпечення. Тому росте список необхідних прикладних програм. Звичайний час зміни призначеного для користувача програмного забезпечення в централізованих системах складає від двох до семи років. Спосіб вирішити проблему - використовувати розподілені сервери, робочі станції і персональні комп'ютери. Якщо кінцеві користувачі мають доступ до таких комп'ютерів, то проблема з програмним забезпеченням може бути вирішена двояко:

1.Готові програмні продукти. Є величезний перелік програмного забезпечення для поширених комп'ютерів, таких як UNIX- сервери і робочі станції, комп'ютерів, що працюють під управлінням Windows, комп'ютерів MAC.

2.Програми, створені кінцевими користувачами. Є безліч інструментів для малих систем, що дозволяють користувачам створювати прикладні програми, не вдаючись до традиційних мов програмування. Такі засоби є в електронних таблицях і в програмних засобах управління проектами.

Друга вимога - малий час відгуку. В багатьох додатках, щоб добитися потрібної продуктивності, критично важливо забезпечити малий час відгуку. У мейнфреймах з складною операційною системою, що забезпечує роботу користувачів в режимі розділення часу, як правило, малий час відгуку забезпечити достатньо складно. З іншого боку, для користувача, що працює за персональним комп'ютером або робочою станцією, або що розділяє з невеликим числом інших користувачів ресурси міні-комп'ютера, можна забезпечити дуже короткий час відгуку.

Ми бачимо, що малі розподілені системи фізично розташовані ближче до користувача і більшою мірою призначені для вирішення завдань даного користувача, тому розподілені системи дозволяють підвищити продуктивність роботи користувача і обчислювальну ефективність. Проте при впровадженні розподілених систем керівники повинні дотримуватися обережності. Відсутність централізованих обчислювальних засобів може привести до втрати керованості. Різні відділення можуть використовувати несумісні системи, що утрудняє зв'язки між ними. Рішення по постачанню відділень устаткуванням можуть прийматися без прогнозування вимог і цін, без урахування стандартів на апаратне і програмне забезпечення, на методики програмування. Не менш важливий те, що розробка засобів і заходів щодо обробки даних на рівні підрозділу може утрудняти отримання і використання даних вищим керівництвом. Застосування різних стандартів і засобів роботи з даними в підрозділах ускладнює одноманітний збір даних для складання звітів керівництву.

У табл. 4.1 і 4.2 приводяться деякі ключові достоїнства і недоліки розподілених систем обробки даних.

Таблиця 4.1 Достоїнства розподілених систем

Реактивність	Місцеві обчислювальні засоби можна набудувати так, щоб вони найбільш природним чином відповідали вимогам місцевого керівництва, тоді як централізовані засоби обробки даних повинні відповідати вимогам всієї організації
Доступність	При використанні безлічі сполучених один з одним систем несправність однієї з них приводить до мінімального збитку. Ключові системи і компоненти (наприклад, комп'ютери, важливі операції, що виконують критично, принтери, пристрої зберігання даних) можна дублювати так, щоб резервна система почала працювати відразу ж після поломки основної

Відповідність структурі організації	У багатьох організаціях є децентралізована структура з відповідною політикою і робочими методиками. Вимоги до файлів даних і іншим автоматизованим ресурсам можуть відображати цю структуру
Розділення ресурсів	Дороге апаратне забезпечення, наприклад лазерний принтер, може використовуватися багатьма користувачами. На рівні всієї організації може бути забезпечений доступ до файлів даних. Програмні продукти і бази даних, розроблені на рівні всієї організації, можна поширювати на розподілені засоби
Розширення систем	У централізованих системах зростаюче навантаження і вимоги до нових програмних продуктів приводять до необхідності масової закупівлі устаткування або оновлення програмного забезпечення. Це вимагає значних витрат. Крім того, такі зміни можуть викликати необхідність в перетворенні або модифікації наявних додатків, при цьому існує ризик появи помилок і зниження продуктивності. У розподілених системах можна поступово замінювати програми або апаратні засоби, уникаючи підходу «все або нічого». Крім того, старе устаткування можна зберегти для виконання отдель-ньїх завдань, якщо витрати на переклад додатку на нове устаткування невиправдано високі
Посилення ролі користувача	При роботі з малим легко керованим устаткуванням, що знаходиться поблизу від користувача, у останнього є широкі можливості по зміні параметрів конструк-ции системи і її роботи як за допомогою технічного персоналу, так і безпосередньо
Децентралізована робота і централізований контроль	Децентралізовані засоби і додатки можна набудувати згідно вимогам окремих підрозділів. Такі системи можна оснастити централізованими службами і базами даних для забезпечення контролю на рівні організації

Таблиця 4.2 Якості розподілених систем обробки

Продуктивність кінцевого користувача	У розподілених системах менше час відгуку для користувача, оскільки кожен елемент устаткування виконує менший об' єм робіт. Крім того, додатки і призначений для користувача інтерфейс можна набудувати згідно запитам підрозділу. Керівники подразде-лений можуть оцінювати ефективність роботи цих засобів і вносити необхідні зміни
Незалежність від місцеположення	У розподілених системах є інтерфейси і методики доступу до обчислювальних потужностей. Ці засоби не залежать від місцеположення систем і відстаней між ними. Тому користувач може дістати доступ до обчислювальних засобів всієї організації, пройшовши,

	при необхідності, короткий курс підготовки
Конфіденційність і безпека	При роботі з розподіленими системами легше визначити відповідальність користувачів за безпеку файлів з даними і інших ресурсів. Є фізичні і програмні засоби для запобігання несанкціонованому доступу до даним і ресурсам
Незалежність від постачальників	При правильній реалізації розподілені системи можуть включати устаткування і програмне забезпечення від різних постачальників. Це дає покупцеві обчислювальних засобів відчутну перевагу при укладенні оборудок. Знижується вірогідність того, що організація стане залежати від одного постачальника зі всіма витікаючими наслідками
Гнучкість	Користувачі можуть міняти настройку програмного забезпечення у міру зміни обставин, якщо вони мають повноваження по обслуговуванню програмного забезпечення і щоденному запуску програм. Оскільки з устаткуванням працює тільки один користувач, він може практично без зусиль змінювати конфігурацію устаткування
Складність перевірки і діагностики відмов	Завдяки високому ступеню взаємодії елементів розподілених систем складно з' ясувати причину відмови або зниження продуктивності
Вищий ступінь залежності від засобів зв'язку	Для ефективної роботи розподіленої системи її елементи повинні бути об'єднані комунікаційними і мережевими засобами. Ці засоби критично важливі для щоденної роботи організації
Несумісність устаткування	Можуть виникати складнощі при з' єднанні устаткування від різних постачальників. Для того, щоб гарантовано уникнути таких утруднень, користувачі повинні обмежитися застосуванням стандартизованных програмних і апаратних продуктів
Несумісність типів даних	Аналогічно попередньому пункту, дані, створені в одному додатку, можуть виявитися несумісними з іншими додатками. І знову необхідно обмежитися використанням стандартизованного програмного забезпечення
Управління і керівництво мережами	Оскільки устаткування фізично розосереджене, включає засоби від різних постачальників, може управлятися різними підрозділами, виникають складнощі в забезпеченні загального керівництва, підтримці стандартів на додатки і дані, контролі за інформацією,

	доступною в мережі. Тому засоби обробки даних і послуги можуть виявитися некерованими
Складність в управлінні корпоративними інформаційними ресурсами	Навіть якщо дані і не розосереджені, до них можливий доступ з різних підрозділів. Якщо користувачам розподіленої системи дозволено виконувати оновлення даних (що необхідне в багатьох додатках), виникають складнощі при контролі цілісності і безпеки даних на корпоративному рівні. В деяких випадках можуть виникнути складнощі при зборі необхідної інформації від різних розподілених баз даних
Оптимізація на рівні підрозділів	Керівники підрозділів можуть з легкістю виправдовувати витрати на постачання свого підрозділу апаратним і програмним забезпеченням для розподілених систем. Не дивлячись на те, що кожне постачання може бути обгрунтованим, загальний об' єм постачань у всій організації може опинитися невиправдано високим
Дублювання зусиль	Технічний персонал різних підрозділів може незалежно розробляти аналогічні додатки і файли, це приводить до надмірного і дорогого дублювання

4.2.2 Інтранет

Одним з останніх досягнень в еволюції систем розподіленої обробки даних є Інтранет. Фактично Інтранет надає користувачам в рамках однієї організації ключові можливості і додатки Інтернету. Нижче перераховані основні характеристики інтранету.

Застосовуються стандарти Інтернету, такі як мова гіпертекстової розмітки (hypertext Markup Language, HTML) і простій протокол передачі пошти (Simple Mail Transfer Protocol, SMTP). Використовується набір протоколів TCP/IP для роботи в локальних і глобальних мережах.

У Інтранет циркулює конфіденційна інформація, недоступна через інтернет, хоча при цьому корпорація може мати з'єднання з Інтернетом і власний веб-сервер.

Інтранет, на відміну від Інтернету, керований.

Фактично, Інтранет можна розглядати як одну з форм архітектури клієнт- сервер. Серед переваг Інтранет - легкість в реалізації і використанні.

4.2.3 Екстранет

Іншою недавньою розробкою є Екстранет. Подібно Інтранету в мережах Екстранет використовуються протоколи TCP/IP і додатки, в основному орієнтовані на Всесвітню павутину. Головною рисою Екстранет є те, що ця мережа надає постачальникам і замовникам доступ до ресурсів організації ззовні. Доступ може здійснюватися через Інтернет або інші мережі передачі даних. Проте Екстранет пропонує щось більше, ніж простий доступ через Всесвітню павутину, в якій представлені практично всі компанії. Можливості доступу через Екстранет до корпоративних ресурсів набагато ширші, при цьому, як правило, дотримуються

певні політики безпеки. Як і в Інтранеті, типовою моделлю роботи в Екстранет є архітектура клієнт-сервер.

4.3 Форми розподіленої обробки

Ми визначили систему розподіленої обробки даних як обчислювальні
Засоби, в яких комп'ютери розосереджені в рамках організації і є засоби зв'язку окремих комп'ютерів. За цим загальним визначенням ховається безліч форм розподіленої обробки даних. Для того, щоб вивчити всі ці форми, слідує детальніше зупинитися на функціях і об'єктах, підтримуваних розподіленими системами:

1. Додатки;
2. Контролери пристроїв;
3. Управління;
4. Дані.

Як правило, розподіленими є декілька функцій або об'єктів з приведеного вище переліку. В цілях вивчення конфігурацій розподіленої обробки даних досить розглядати їх окремо. У цьому розділі ми вивчимо перші три пункти, а останній пункт буде розглянутий в наступному розділі.

4.3.1 Розподілені додатки

Розподіл додатків можна розглядати в двох вимірюваннях. По-перше, функції, що виконуються додатками, можуть бути розподілені по-різному:

1.Один додаток розбитий на компоненти, що виконуються на декількох комп'ютерах;

2.Один додаток дублюється на декількох комп'ютерах;

3.Декілька різних додатків розподілено на декількох комп'ютерах.

Розподілене виконання додатків може характеризуватися напрямом розподілу: по вертикалі або по горизонталі. Взагалі кажучи, вертикальне розділення полягає в тому, що один додаток розбивається на компоненти, що виконуються на декількох комп'ютерах. Навпаки, розділення по горизонталі полягає в тому, що один додаток дублюється на декількох комп'ютерах або декілька різних додатків працюють на декількох комп'ютерах. При вертикальному розподілі обробка даних розподілена ієрархічно.

Такий розподіл звичайний або відображає структуру організації, або просто краще всього підходить для даного додатку. Нижче наведені приклади.

Страховка.

Обробка даних зазвичай має два рівні ієрархії. Кожен філіал страхового агентства має комп' ютерну систему для підготовки нових контрактів і обробки заяв. В більшості випадків такі операції виконуються безпосередньо в місцевому офісі. Зведені данні поступають в головний офіс, який на основі контрактів і заяв проводить аналіз ризиків і страхові обчислення.

Залежно від фінансового положення компанії, головний офіс може міняти тарифи і передавати зміни у свої відділення.

Управління процесами.

Функції управління виробничими процесами легко пристосовуються для вертикальних систем розподіленої обробки даних. Кожен крупний напрям робіт

контролюється робочою станцією, одержуючою дані від окремих мікропроцесорних засобів управління процесами. На мікропроцесори покладено завдання автоматизованого опиту датчиків і виконавських елементів на рівні виробничого цеху. Робоча станція прочитує показники датчиків, вивчає відхилення від норми і аналізує тенденції. Робоча станція також може управляти частиною роботи, наприклад, зміною темпу виробництва або зміною типу продукції, що випускається. Розподілені робочі станції забезпечують швидкий відгук на зміну умов виробництва. Всі робочі станції підключені до комп' ютера вищого рівня, на який покладені завдання планування виробництва, оптимізації, збору відомостей для керівництва і обробки загальних даних всієї корпорації.
Як показують ці приклади, системи обробки даних з вертикальним розділенням, як правило, складаються з центральної обчислювальної системи, обладнаної додатковими системами різних рівнів. Структура розподілу обробки даних, як правило, відображає організаційну структуру і/або структуру задач, які вирішуються. Мета такого розподілу - обробляти інформацію на тому рівні ієрархії, де це найвигідніше. Така схема поєднує кращі якості централізованих і розподілених систем обробки даних.
При горизонтальному розподілі обробка даних розділяється між декількома комп'ютерами одного рангу. У такому разі поняття відомий- ведучий відсутні. У разі горизонтального розподілу комп'ютери зазвичай
Працюють автономно, проте в деяких випадках така конфігурація застосовується для рівномірного розподілу навантаження. У багатьох випадках горизонтальний розподіл засобів відображає децентралізацію організації. Ось два приклади.

Система управління повітряним транспортом.
Кожен регіональний центр диспетчерської служби працює незалежно від інших центрів, виконуючи ті ж завдання. В рамках одного центру декілька комп'ютерів служать для обробки даних, що поступають з радара, і відомостей про обстановку в ефірі. Ті ж комп'ютери застосовуються для відображення стану повітряної обстановки.
Як правило, в організації функції по обробці даних розділені і по горизонталі, і по вертикалі. Правління корпорації може мати мейнфрейм з інформаційною системою управління корпорацією і системою підтримки ухвалення рішень. Цей мейнфрейм може також вирішувати інші задачі правління, наприклад зв'язки з громадськістю, стратегічне планування, збір даних про бюджет корпорації і відомостей для фінансової звітності. Вертикальне розділення забезпечується підлеглими комп'ютерними засобами у філіалах. Усередині кожного філіалу завдяки горизонтальному розділенню забезпечується автоматизація рішення офісних задач.

4.3.2 Інші форми розподіленої обробки

На додаток до розподілу додатків або замість нього в системах розподіленої обробки даних може зустрічатися розподіл контролерів пристроїв або мережевого адміністрування. Одним з природних завдань розподіленої обробки даних є підтримка різноманітних пристроїв, керованих з комп'ютера, наприклад банківських автоматів або лабораторного обладнання.

5 Розподілена обробка інформації

5.1 Модель розподіленої обробки інформації

5.1.1 Загальні відомості

Розподілена обробка даних - методика виконання прикладних програм групою систем.
Сутність DDP полягає в тому, що користувач отримує можливість працювати з мережевими службами і прикладними процесами, розташованими в декількох взаємопов'язаних абонентських системах. При цьому можливі кілька видів робіт, які він може виконувати:
• віддалений запит, наприклад, команда, що дозволяє надсилати одиночну заявку на виконання обробки даних;
• віддалена транзакція, що спрямовує групи запитів прикладному процесу;
• розподілена транзакція, що дає можливість використання декількох серверів і прикладних процесів, які виконуються в групі абонентських систем.
Для розподіленої обробки здійснюється сегментація прикладних програм - поділ складної прикладної програми на частини, які можуть бути розподілені по системам локальної мережі.
Сегментація здійснюється за допомогою спеціального інструментального програмного забезпечення, яке автоматизує даний процес. За допомогою технології, що надається об'єктно-орієнтованою архітектурою в результаті виконання зазначеного процесу прикладна програма ділиться на самостійні частини, що завантажуються в різні системи. Завдяки цьому, створюється можливість переміщення програм з однієї системи в іншу і розподіленої обробки даних.
В результаті сегментації кожна виділена частина програми включає управління даними, алгоритм і блок презентації. Завдяки цьому, вона може бути оптимальним чином виконана на основі платформ, що використовуються в мережі.
Передача даних для розподіленої обробки відбувається за допомогою віддаленого виклику процедур або електронної пошти. Перша технологія характеризується високою швидкодією, а друга - низькою вартістю.

5.1.2 Віддалений виклик процедур

Віддалений виклик процедур працює аналогічно місцевому виклику процедур і забезпечує організацію обробки даних. Цій меті служить механізм навігації мережі, пошуку інформації, запуску процесу в декількох системах, передачі отриманих результатів користувачам, хто послав запити. Здійснюваний процес характеризується прозорістю, завдяки якій об'єкти мережі, розташовані між користувачами і програмами невидимі обом партнерам.
Виконання віддаленого виклику процедур є дорогою операцією, бо на весь час її виконання системи, що беруть участь в роботі, повинні по каналам передавати дані один одному. Альтернативою віддаленого виклику є застосування інтелектуальних агентів або виконання розподіленої обробки даних з використанням електронної пошти. Цей метод не вимагає великих витрат, але працює значно повільніше.

5.1.3 Засоби СУРБД

Відомі також програмні засоби Системи Управління розподіленою Базою Даних (СУРБД), вміст якої розташовується в декількох абонентських системах інформаційної мережі.
Завданням СУРБД є забезпечення функціонування розподіленої бази даних. СУРБД повинна діяти так, щоб у користувачів з'явилася ілюзія того, що вони працюють з Базою Даних (БД), що розташована в одній абонентської системі. Використання СУРБД, в порівнянні з групою невзаємопов'язаних баз даних, дозволяє скорочувати витрати на передачу даних в інформаційній мережі. СУРБД так розподіляє файли по мережі, що в кожній системі зберігаються ті дані, які найчастіше використовуються саме в цьому місці.
У СУРБД здійснюється тиражування даних. Його сутність полягає в тому, що зміна, що вноситься в одну частину бази даних, протягом визначеного часу відбивається і в інших частинах бази.
При плануванні обробки даних можуть розглядатися три моделі обробки:
• обробка у тимчасовій локальної мережі;
• централізоване опрацювання;
• обробка в моделі клієнт / сервер.
При будь-якій обробці є три основних рівня маніпулювання даними:
•збереження даних;
• виконання програм, тобто вибірка і обробка даних для потреб прикладної задачі;
• представлення даних і результатів обробки кінцевому користувачеві.
При обробці в тимчасової мережі всі три рівні, як правило, виконуються на одному - персональному - робочому місці. У сучасних технологіях застосування обчислювальної техніки персональної обробки інформації, коли всі дані і засоби їх обробки зосереджені в межах одного робочого місця, і обмін даними між робочими місцями не відбувається або виконується епізодично (наприклад, засобами електронної пошти), поступово відходить у минуле. Сучасні інформаційні, управлінські, офісні системи в більшій чи меншій мірі орієнтуються на обробку для багатьох користувачів, при якій дані доступні (можливо, одночасно доступні) багатьом користувачам з різних робочих місць. Міркування ефективності і надійності вимагають централізації процесів зберігання і обробки даних.
І централізована обробка, і модель клієнт / сервер в рівній мірі використовують переваги централізації. Різниця між цими двома моделями полягає в тому, що при централізованій обробці уявлення інформації кінцевому клієнту також виконується засобами центральної обчислювальної системи - на її терміналах (не інтелектуальних), підключених до обчислювальної системи через порти / канали введення-виведення. У моделі ж клієнт / сервер термінали, що представляють інформацію, є інтелектуальними - самостійними обчислювальними системами (зазвичай персональними комп'ютерами) і пов'язані з сервером через мережеві засоби.
Обчислювальний ресурс (це може бути окрема ЕОМ в мережі або окремий процес в багатозадачній обчислювальній системі), що забезпечує зберігання, адміністрування, надання доступу до даних, називається сервером. Обчислювальні ресурси (окремі ЕОМ або процеси), що забезпечують використання даних і подання їх кінцевому користувачу, називаються клієнтами. Вся модель, яка забезпечує такий розподіл функцій, називається моделлю клієнт / сервер.

При переміщенні більшу частину функцій маніпулювання даними на високопродуктивний сервер можуть бути забезпечені наступні переваги:
• економія обчислювальних ресурсів всієї системи в цілому;
• економія ресурсів засобів комунікацій;
• забезпечення роботи всіх користувачів з однією і тією ж копією даних;
• запобігання фатальних конфліктів між клієнтами при зверненні їх до одних і тих же даних;
• забезпечення надійного адміністрування бази даних, в т.ч. резервного копіювання та розмежування доступу до даних.

5.1.4 Особливості взаємодії клієнтів і серверів

Хоча централізована обробка забезпечує більшу ефективність в супроводі системи і в швидкості обміну, кращою все ж видається модель клієнт / сервер, до достоїнств якої слід віднести перш за все гнучкість - можливість будувати клієнтські робочі місця на різних платформах і в різних операційних середовищах і, таким чином, гнучко пристосовувати можливості інтелектуального терміналу AIPC до завдань, що стоять перед даним робочим місцем.
Розподіл функцій маніпулювання даними між клієнтом і сервером може бути різним, як показано на рис 5.1.
Варіант файлового сервера передбачає, що сервер виконує тільки зберігання даних. При необхідності вся одиниця зберігання даних (файл) пересилається клієнту, і всю подальшу роботу з даними (в тому числі і вибірку) виконує клієнт. Цей варіант вимагає значних ресурсів як обчислювальних (так як на кожному робочому місці повинна розміщуватися копія оброблюваних даних і всі засоби їх обробки), так і комунікаційних (так як по мережі передаються файли повністю). Крім того, цей варіант або обмежує одночасний доступ клієнтів блокуванням файлів, або призводить до неідентичності копій інформації у різних клієнтів. У гранічному випадку варіант файлового сервера зводиться до персональної обробки даних, нехай і в середовищі локальної мережі.
Варіант сервера бази даних передбачає, що на сервер покладається виконання однієї з найбільш трудомістких функцій логіки додатка – вибірки з бази даних тільки тих записів, які необхідні для вирішення конкретного завдання. В цьому випадку економляться ресурси обчислювальної системи і забезпечується дійсно доступ для багатьох користувачів. Разом з тим, в залежності від складності решти логіки додатка, обсяг клієнтської частини ресурсів може все ще залишатися досить великим.

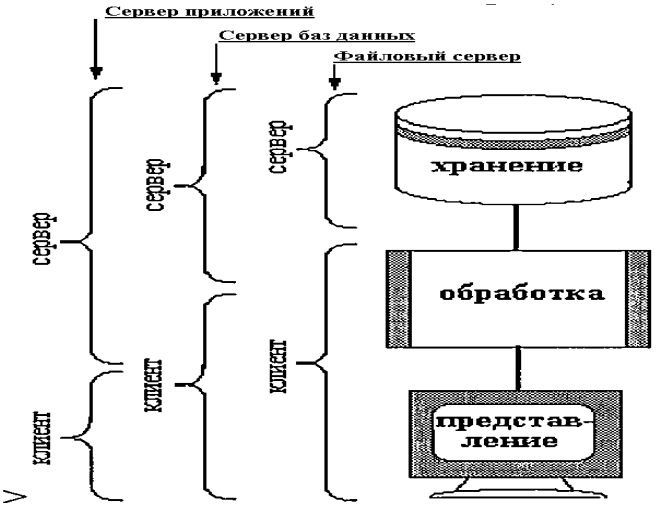


Рис. 5.1 Варіанти розподілу функціональності між сервером та клієнтом

Варіант сервера бази даних особливо ефективний в системах зі спеціалізованими робочими місцями, так як дозволяє підібрати апаратні та операційні середовища робочих місць в найбільш точній відповідності до вирішуваних завдань.
Варіант сервера додатків передбачає, що вся або майже вся логіка додатків виконується на сервері, а в клієнтську частину передаються лише результати обробки. Клієнтська частина, таким чином, відповідальна тільки за представлення результатів кінцевому користувачеві. Такий варіант висуває підвищені вимоги до ресурсів сервера, але обсяги ресурсів клієнтів можуть бути мінімальні. Такий варіант дозволяє також легко організувати гетерогенну систему, в якій різні клієнти будуть працювати в різних операційних середовищах, так як обсяг програмного забезпечення, який потрібно переписати, щоб адаптувати клієнта до нового середовища, відносно невеликий – це тільки частина, залежна від засобів візуалізації на даному робочому місці. У граничному випадку модель сервера додатків, однак, зводиться до централізованої обробки.

5.2 Базові технології розподілених обчислень

Програмне забезпечення (ПЗ) організації розподілених обчислень називають ***програмним забезпеченням проміжного шару*** (*Middleware)*. Новий напрямок організації розподілених обчислень в мережах Internet-Intranet засновано на створенні і використанні програмних засобів, які можуть працювати в різних апаратно-програмних середовищах. Сукупність таких засобів називають ***багатоплатформеним розподіленим середовищем*** – МРС (*crossware*).
Знаходять застосування технології розподілених обчислень RPC (*Remote Procedure Call*), ORB (*Object Request Broker*), MOM (*Message-oriented Middleware*), DCE (*Distributed Computing Environment*), монітори транзакцій, ODBC, ADO та DAO.
RPC – процедурна блокуюча синхронна технологія, запропонована фірмою Sun Microsystems. Виклик віддалених програм подібний виклику функцій в мові С. При пересиланні на основі транспортних протоколів TCP або UDP дані представляються в єдиному форматі обміну XDR. Синхронність і блокування означають, що клієнт, звернувшись до сервера, для продовження роботи чекає відповіді від сервера.
Для систем розподілених обчислень розроблені спеціальні мови програмування, для RPC це мова IDL (*Interface Definition Language*), яка дає користувачеві

можливість оперувати різними об’єктами безвідносно до їх розташування в мережі. На цій мові можна записувати звернення до серверів додатків. Інший приклад мови для систем розподілених обчислень – NewEra в середовищі Informix. RPC входить в багато систем мережевого програмного забезпечення.

ORB – технологія об’єктно-орієнтованого підходу, включає 13 пунктів (служб). Основні служби:

• служба іменування, привласнює об’єктам унікальні імена, в результаті користувач може шукати об’єкт в мережі;

• служба обробки транзакцій, здійснює управління транзакціями з додатків або з операційних систем;

• служба подій, забезпечує асинхронне поширення і обробку повідомлень про події;

• служба забезпечення безпеки – підтримки цілісності даних.

При застосуванні ORB (на відміну від RPC) у вузлі-клієнті зберігати відомості про розташування серверних об’єктів не потрібно, достатньо знати розташування в мережі програми-посередника ORB. Тому доступ користувача до різних об’єктів (програмам, даним, принтерам і т.п.) істотно спрощений. Посередник повинен визначати, в якому місці мережі розміщено запитаний ресурс, направляти запит користувача до відповідного вузла, а після виконання запиту повертати результати користувачеві.

MOM – також об’єктна технологія. Зв’язок з серверами асинхронний. Це одна з найбільш простих технологій, включає команди «послати» і «отримати», які здійснюють обмін повідомленнями. Відрізняється від E-mail реальним масштабом часу. Однак можуть бути варіанти MOM з чергами, тоді режим on-line необов’язковий і при передачі не потрібно підтверджень, тобто опора на протокол IP без встановлення з’єднання.

5.3 Розподілене середовище обробки даних

(*Distributed Computing Environment* (DCE *)) – технологія розподіленої обробки даних, запропонована фондом відкритого програмного забезпечення.

Вона не протиставляється іншим технологіям (RPC, ORB), а є середовищем для їх використання.

Середовище DCE*, розроблене в 1990 р, являє собою набір мережевих служб, призначених для виконання прикладних процесів, розосереджених по групі абонентських систем гетерогенної (неоднорідної) мережі. Основні її компоненти показані в табл.5.1.

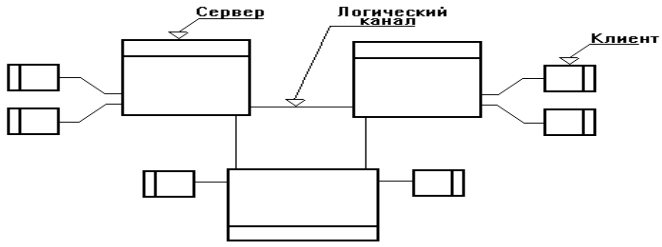
Табл. 5.1 Основні компоненти DCE *.

№ п/п	Служба	Функції, що виконуються
1.	Імена	База Даних (БД) імен клієнтів і засобів, які призначені для доступу користувачів до мережевих служб.

2.	Віддалений доступ	Технологія, що забезпечує взаємодію двох додатків,які розташовані в різних абонентських системах.
3.	Захист даних	Програмне Забезпечення (ПЗ) дозволу на доступ до ресурсів системи чи мережі.
4.	Багатопоточність	Програми, які забезпечують одночасне виконання декількох задач.

Системи, що мають програми розподіленого середовища, відповідно, є серверами і клієнтами. Сервери пов’язані один з одним логічними каналами, по яких передають один одному файли (рис. 5.2)
Кожен сервер має свою групу клієнтів.
Середовище має триступеневу архітектуру: прикладна програма – база даних – клієнт.
Функції, які виконуються середовищем, включають прикладні служби:
• каталогів, що дозволяє клієнтам знаходити потрібні їм сервери;
• інтерфейсу багатопоточної обробки;
• віддаленого виклику процедур;
• обслуговування файлів;
• безпеки даних;
• часу, що синхронізує годинник в абонентських системах.
Програмне Забезпечення (ПЗ) середовища занурюється в Мережну Операційну Систему (МОС). Сервери мають свої, різні, Операційні Системи (ОС). У ролі сервера може також виступати головний комп’ютер зі своєю операційною системою.

Рис. 5.2 Логічна структура середовища



Функціонування розподіленого середовища вимагає виконання ряду адміністративних завдань. До них, в першу чергу, відносяться засоби:
• реєстрації та контролю за ліцензіями користувачів на роботу з прикладними програмами;
• уніфікованих інтерфейсів прикладних програм;
• забезпечення безпеки даних;

• інвентаризації програмного і технічного забезпечення абонентських систем, що працюють в мережі.
З точки зору логічного управління середовище обробки даних ділиться на комірки DCE *. У кожну з них може включатися від декількох одиниць до тисяч абонентських систем. Розміри комірок територіально необмежені. Вхідні в одну і ту ж комірку системи можуть бути розташовані навіть на різних континентах. В комірках виконуються служби:
• контролю права роботи з прикладними програмами і базами даних;
• каталогів, що призначають адреси об’єктів;
• часу, що синхронізує годинники систем;
• ліцензії, що відстежує використання видів сервісу.
Розподіл прикладних процесів по комірках захищає від збоїв, дозволяє пристосувати процеси до конкретних потреб груп користувачів. Останні можуть мати доступ тільки до певних комірок.

5.4 Визначення і завдання розподіленої системи

Визначення розподіленої системи
Розподілена система - це набір незалежних комп'ютерів, що представляється їх користувачам у вигляді єдиної об'єднаної системи.
У цьому визначенні розглядаються два моменти. Перший відноситься до апаратного рівня: всі машини автономні. Другий стосується програмного забезпечення: клієнти вважають, що мають справу з єдиною системою. Розглянемо деякі базові питання, що стосуються як апаратного, так і програмного забезпечення.
Одна з характеристик розподілених систем полягає в тому, що від користувачів приховані відмінності між комп'ютерами і способи зв'язку між ними. Те ж саме відноситься і до зовнішньої організації розподілених систем. Іншою важливою характеристикою розподілених систем є спосіб, за допомогою якого користувачі і додатки одноманітно працюють в розподілених системах, незалежно від того, де і коли відбувається їх взаємодія.
Розподілені системи повинні відносно легко піддаватися розширенню, або масштабуванню. Ця характеристика є прямим наслідком наявності незалежних комп'ютерів, але в той же час не вказує, яким чином ці комп'ютери насправді об'єднуються в єдину систему. Розподілені системи зазвичай існують постійно, однак деякі їх частини можуть тимчасово виходити з ладу. Користувачі і додатки не повинні повідомлятися про те, що ці частини замінені або виправлені або що додані нові частини для підтримки додаткових користувачів або додатків.
Для того щоб підтримати подання різних комп'ютерів і мереж у вигляді єдиної системи, організація розподілених систем часто включає в себе додатковий рівень програмного забезпечення, що знаходиться між верхнім рівнем, на якому знаходяться користувачі і додатки, і нижнім рівнем, що складається з операційних систем, як показано на рис. 5.3. Відповідно, така розподілена система зазвичай називається системою проміжного рівня (middleware).

Рис. 5.3. Приклад розподіленої системи, що організована у вигляді служби проміжного рівня

Відзначимо, що проміжний рівень розподілений серед безлічі комп'ютерів. Розглянемо деякі приклади розподілених систем.

Як перший приклад розглянемо мережу робочих станцій в університеті або відділі компанії. До того ж для персональної робочої станції кожного з користувачів виділяється пул процесорів датацентру, непризначених заздалегідь жодному з користувачів окремо, але динамічно виділяються їм при необхідності. Ця розподілена система може мати архітектуру єдиної файлової системи, в якій всі файли однаково доступні зі всіх машин з використанням постійного шляху доступу. Крім того, коли користувач набирає команду, система може знайти найкраще місце для виконання запитуваної дії, можливо, на власній робочої станції користувача, можливо, на ресурсах робочої станції, що належить комусь іншому іпростоюють, а може бути, і на одному з вільних процесорів датацентру. Якщо система в цілому виглядає і поводиться як класична однопроцесорна система з поділом часу (тобто розрахована на багато користувачів), вона вважається розподіленою системою.

Як другий приклад розглянемо роботу інформаційної системи, яка підтримує автоматичну обробку замовлень. Зазвичай подібні системи використовуються співробітниками декількох відділів, можливо в різних місцях. Так, співробітники відділу продажів можуть бути розкидані по великому регіону або навіть по всій країні. Замовлення передаються з переносних комп'ютерів, що з'єднуються з системою за допомогою корпоративної мережі, а можливо, і за допомогою стільникових телефонів. Замовлення, що надходять, автоматично передаються до відділу планування, перетворюючись там у внутрішні замовлення на поставку, які надходять до відділу доставки, і в заявки на оплату, що надходять до бухгалтерії. Система автоматично пересилає ці документи наявним на місці співробітникам, які відповідають за їх обробку. Користувачі залишаються в повному незнанні про те, як замовлення насправді курсують усередині системи, для них все це уявляється так, ніби вся робота відбувається в централізованій базі даних.

В якості третьої прикладу розглянемо World Wide Web. Web надає просту, цілісну і одноманітну модель розподілених документів. Щоб побачити документ, користувачеві досить активізувати посилання. Після цього документ з'являється на екрані його PC. Немає необхідності знати, з якого сервера доставляється документ, достатньо лише інформації про те, де він розташований. Публікація документа дуже проста: потрібно задати йому унікальне ім'я в формі уніфікованого

покажчика ресурсу (Uniform Resource Locator, URL), яке посилається на локальний файл із вмістом документа. Якби Всесвітня павутина представлялася своїм користувачам гігантською централізованою системою документообігу, вона також могла б вважатися розподіленою системою. На жаль, користувачі усвідомлюють, що документи знаходяться в різних місцях і розподілені по різних серверів.

---

6 Базові концепції побудови розподілених систем

6.1 Апаратна побудова розподілених систем

Не дивлячись на те що всі розподілені системи містять по декілька процесорів, існують різні способи їх організації в систему. Особливо це відноситься до варіантів їх з'єднання і організації взаємного обміну. У цьому розділі ми стисло розглянемо апаратне забезпечення розподілених систем, зокрема варіанти з'єднання машин між собою. Предметом нашого обговорення в наступному розділі буде програмне забезпечення розподілених систем.

За минулі роки було запропоновано безліч різних схем класифікації комп'ютерних систем з декількома процесорами, але жодна з них не стала дійсно популярною і широко поширеною. Нас цікавлять виключно системи, побудовані з набору незалежних комп'ютерів. Ми підрозділяємо всі комп'ютери на дві групи. Системи, в яких комп'ютери використовують пам'ять спільно, зазвичай називаються **мультипроцесорами** (multiprocessors), а що працюють кожен з своєю пам'яттю —**мультикомп'ютерами** (multicomputer). Основна різниця між ними полягає в тому, що мультипроцесори мають єдиний адресний простір, спільно використовуваний всіма процесорами.

6.2 Програмна побудова розподілених ситем

Апаратура важлива для розподілених систем, проте від програмного забезпечення значно сильніше залежить, як така система виглядатиме насправді. Розподілені системи дуже схожі на традиційні операційні системи. Перш за все, вони працюють як менеджери ресурсів (resource managers) існуючого апаратного забезпечення, які допомагають безлічі користувачів і додатків спільно використовувати такі ресурси, як процесори, пам'ять, периферійні пристрої, мережу і дані всіх видів. По-друге, що, ймовірно, більш важливо, розподілена система приховує *складність і гетерогенну природу* апаратного забезпечення, на базі якого вона побудована, надаючи віртуальну машину для виконання додатків.

6.2.1 Комп'ютерні мережі, як окремий випадок розподілених систем

Комп'ютерні мережі належать до розподілених (або децентралізованих) обчислювальних систем. Оскільки основною ознакою розподіленої обчислювальної системи є наявність декількох центрів обробки даних, то поряд із комп'ютерними мережами до розподілених систем відносять також мультипроцесорні комп'ютери і багатомашинні обчислювальні комплекси.

6.2.2 Мультипроцесорні комп'ютери

У мультипроцесорних комп'ютерах є декілька процесорів, кожен із який може відносно незалежно від інших виконувати свою програму. У мультипроцесорі існує загальна для всіх процесорів операційна система, що оперативно розподіляє обчислювальне навантаження між процесорами. Взаємодія між окремими процесорами організована найпростішим способом – через загальну оперативну пам'ять.

Процесорний блок не є закінченим комп'ютером і тому не може виконувати програми без інших блоків мультипроцесорного комп'ютера – пам'яті і периферійних пристроїв. Всі периферійні пристрої є спільними для всіх процесорів мультипроцесорної системи. Територіальну розподіленість мультипроцесор не підтримує – усі його блоки розташовуються в одному або декількох близько розташованих конструктивах, як і в звичайному комп'ютері.

Основна перевага мультипроцесора – його висока продуктивність, що досягається за рахунок паралельної роботи кількох процесорів. Завдяки існуванню спільної пам'яті взаємодія процесорів відбувається дуже швидко, мультипроцесори можуть ефективно виконувати навіть додатки з високим ступенем зв'язку за даними.

Ще однією важливою властивістю мультипроцесорних систем є стійкість до відмов, тобто спроможність продовжувати роботу при відмові деяких елементів, наприклад процесорів або блоків пам'яті. При цьому продуктивність, природно, знижується, але не повністю, як у звичайних системах, у яких відсутня надмірність.

6.2.3 Багатомашинні системи

Багатомашинна система – це обчислювальний комплекс, що включає в себе декілька комп'ютерів (кожен з яких працює під керуванням власної операційної системи), а також програмні й апаратні засоби зв'язку комп'ютерів, що забезпечують роботу всіх комп'ютерів комплексу як єдиного цілого.

6.2.4 Обчислювальні мережі

У обчислювальних мережах програмні й апаратні зв'язки є ще слабшими, а автономність обробних блоків простежується найбільшою мірою – основними елементами мережі є стандартні комп'ютери, що не мають ні спільних блоків пам'яті, ні спільних периферійних пристроїв. Зв'язок між комп'ютерами здійснюється за допомогою спеціальних периферійних пристроїв – мережевих адаптерів, сполучених відносно протяжними каналами зв'язку. Кожний комп'ютер працює під керуванням власної операційної системи, а якась «спільна» операційна система, що розподіляє роботу між комп'ютерами мережі, відсутня. Взаємодія між комп'ютерами мережі відбувається за рахунок передачі повідомлень через мережеві адаптери і канали зв'язку. За допомогою цих повідомлень один комп'ютер звичайно запитує доступ до локальних ресурсів іншого комп'ютера. Такими

ресурсами можуть бути як дані, що зберігаються на диску, так і різноманітні периферійні пристрої – принтери, модеми, факси-апарати і т.д. Поділ локальних ресурсів кожного комп'ютера між усіма користувачами мережі – основна мета створення обчислювальної мережі.

6.3 Завдання та функції розподілених систем обробки

Обговоримо чотири важливих завдання. Розподілені системи можуть легко з'єднувати користувачів з обчислювальними ресурсами і успішно приховувати той факт, що ресурси розкидані по мережі і можуть бути відкритими і масштабованими.

6.3.1 З'єднання користувачів з ресурсами

Основне завдання розподілених систем - полегшити користувачам доступ до віддалених ресурсів і забезпечити їх спільне використання, регулюючи цей процес. Ресурси можуть бути віртуальними, проте зазвичай вони включають в себе принтери, комп'ютери, пристрої зберігання даних, файли і дані. Web-сторінки і мережі також входять в цей список. Існує безліч причин для спільного використання ресурсів. Одна з очевидних - це економічність. Набагато дешевше дозволити спільну роботу з принтером кількох користувачів, ніж купувати і обслуговувати окремий принтер для кожного користувача. Точно також має сенс спільно використовувати дорогі ресурси, такі як суперкомп'ютери або високопродуктивні сховища даних.
З'єднання користувачів з ресурсом також полегшує кооперацію і обмін інформацією, що ілюструється Інтернетом з його простими протоколами для обміну файлами, поштою, документами, аудіо-та відео. Зв'язок через Інтернет призвів до появи численних віртуальних організацій, в яких географічно віддалені один від одного групи співробітників працюють разом за допомогою систем групової роботи (groupware) - програм для спільного редагування документів, проведення телеконференцій і т.п. Так само підключення до Інтернету призвело до появі електронної комерції, що дозволяє купувати і продавати будь-які види товарів, без реального відвідування магазину.
Однак у міру зростання числа підключень і ступеня спільного використання ресурсів все більш і більш важливими стають питання безпеки. В даний час системи мають слабкий захист від підслуховування або вторгнення по лініях зв'язку. Паролі та інша особливо важлива інформація часто пересилаються по мережах відкритим текстом (незашифрованими) або зберігаються на ненадійних серверах. Є багато можливостей для покращення безпеки. Так для замовлення товарів необхідно просто повідомити номер своєї кредитної картки. Рідко це потрібне підтвердження того, що покупець дійсно володіє цією картою. В кращому випадку, замовлення товару таким чином повинно бути можливе тільки в тому випадку, якщо можна фізично підтвердити факт володіння картою за допомогою зчитувача карт. І це не дає 100% гарантії безпеки.
Інша проблема безпеки полягає в тому, що простеження комунікацій дозволяє побудувати профіль переваг конкретного користувача. Подібне відстеження порушує права особистості, особливо якщо проводиться без відома користувача.

Пов'язана з цим проблема полягає в тому, що зростання підключень веде до зростання небажаного спілкування, такого як одержувані по електронній пошті безглузді листи чи нав’язана проти його бажання реклама. Можна захиститися від цього, використовуючи спеціальні інформаційні фільтри, які сортують вхідні повідомлення на підставі їх вмісту.

6.3.2 Прозорість

Важливе завдання розподілених систем полягає в приховуванні факту, що процеси і ресурси фізично розподілені по безлічі комп'ютерів. Розподілені системи, які представляються користувачам і додаткам у вигляді єдиної комп'ютерної системи, називаються прозорими (transparent). Розглянемо реалізацію прозорості в розподілених системах.
Концепція прозорості може бути застосована до різних аспектів розподілених систем, що показано в табл. 6.1.

Табл. 6.1 Характеристики прозорості розподілених систем

Прозорість	Опис
Доступ	Ховається різниця в поданні даних та доступ до ресурсів
Місцезнаходження	Ховається місцезнаходження ресурсу
Переміщення	Ховається факт переміщення ресурсу в інше місце
Зміна місцезнаходження	Ховається факт переміщення ресурсу в процесі обробки в інше місце
Реплікація	Ховається факт реплікації ресурсу
Паралельний доступ	Ховається факт можливого спільного використання ресурсу декількома конкуруючими користувачами
Відмова	Ховається відмова і відновлення ресурсу
Зберігання	Ховається, де зберігається ресурс (програмний) на диску або знаходиться в оперативній пам’яті

6.3.3 Ступінь прозорості

Існують ситуації, коли спроби повністю приховати від користувача розподіленість не розумні. Це відноситься, наприклад, до вимоги надсилати свіжу електронну газету до 7 ранку за місцевим часом, якщо адресат живе в іншому часовому поясі.
Так само в глобальній розподіленій системі, яка з'єднує процес в Сан-Франциско з процесом в Амстердамі, не вдасться приховати факт, що Ви не зможете надсилати повідомлення від одного процесу до іншого швидше, ніж за кілька сотень мс. Швидкість передачі сигналу обмежується не тільки швидкістю світла, скільки швидкістю роботи проміжних пристроїв.
Існує рівновага між високим ступенем прозорості та продуктивністю системи. Багато додатків, призначені для Інтернету, багаторазово намагаються встановити зв'язок з сервером, поки не відмовляться від цього. Спроби замаскувати збій на проміжному сервері, замість спроби працювати через інший сервер, уповільнюють

роботу системи. Було б ефективніше припинити ці спроби або дозволити користувачеві перервати спроби встановлення контакту.
Необхідно, щоб репліки, що знаходяться на різних континентах, були в будь-який момент гарантовано ідентичні. Іншими словами, якщо одна копія змінилася, зміни повинні поширитися на всі системи до того, як вони виконають будь-яку операцію. Одиночна операція оновлення може в цьому випадку займати кілька секунд і її неможливо виконати непомітно для користувачів.
Таким чином, досягнення прозорості розподілу - це важлива мета при проектуванні і розробці розподілених систем, але вона не повинна розглядатися у відриві від інших характеристик системи, наприклад продуктивності.

6.3.4 Відкритість

Інша важлива характеристика розподілених систем - це **відкритість**. Відкрита розподілена система (open distributed system) - це система, яка пропонує служби, виклик яких вимагає стандартний синтаксис і семантику. Наприклад, в комп'ютерних мережах формат, вміст і сенс посилаються і в прийнятих повідомленнях підкоряються типовим правилам. Ці правила формалізовані в протоколах. У розподілених системах служби зазвичай визначаються через інтерфейси (interfaces), які часто описуються за допомогою мови визначення інтерфейсів (Interface Definition Language, IDL). Опис інтерфейсу на IDL майже виключно стосується синтаксису служб. Вони точно відображають імена доступних функцій, типи параметрів, значень, що повертаються, виняткові ситуації, які можуть бути порушені службою і т.п. Складно точно описати, що робить ця служба, тобто семантику інтерфейсів. На практиці подібні специфікації задаються неформально, за допомогою природної мови.
Визначення інтерфейсу допускає можливість спільної роботи довільного процесу, що потребує такого інтерфейсу, з іншим довільним процесом, який надає цей інтерфейс. Визначення інтерфейсу також дозволяє двом незалежним групам створити абсолютно різні реалізації цього інтерфейсу для двох різних розподілених систем, які будуть працювати абсолютно однаково. Правильне визначення самодостатньо і нейтрально. «Самодостатність» означає, що в ньому є все необхідне для реалізації інтерфейсу. Важливо відзначити, що специфікація не визначає зовнішній вигляд реалізації, вона повинна бути нейтральною. **Самодостатність** і **нейтральність** необхідні для забезпечення переносимості і здатності до взаємодії. Здатність до взаємодії (interoperability) характеризує, наскільки дві реалізації систем або компонентів від різних виробників можуть спільно працювати, покладаючись тільки на те, що служби кожної з них відповідають загальному стандарту. **Переносимість** (portability) характеризує те, наскільки додаток, розроблений для розподіленої системи A, може без змін виконуватися в розподіленій системі B, реалізуючи ті ж, що і в A інтерфейси.
Важлива характеристика відкритих розподілених систем - це **гнучкість**. Під гнучкістю розуміється легкість конфігурування системи, що складається з різних компонентів, можливо від різних виробників.
Не повинні викликати труднощів додавання до системи нових компонентів або заміна існуючих, при цьому інші компоненти, з якими не проводилося ніяких дій, повинні залишатися незмінними. Іншими словами, відкрита розподілена система повинна бути **розширюваною**. До гнучкій системі нескладно додати частини, що

працюють під керуванням іншої операційної системи, або замінити всю файлову систему цілком.

6.3.5 Відділення правил від механізмів

У побудові гнучких відкритих розподілених систем вирішальним фактором виявляється організація цих систем у вигляді наборів відносно невеликих і легко замінних або адаптуючихся компонентів. Це передбачає необхідність визначення не тільки інтерфейсів верхнього рівня, з якими працюють користувачі і додатки, але також й інтерфейсів внутрішніх модулів системи і опису взаємодії цих модулів. Це дуже продуктивний підхід. Безліч старих і сучасних систем створювалися цільними так, що компоненти однієї гігантської програми поділялися тільки логічно.
Необхідність змін в розподілених системах часто пов'язана з тим, що компоненти не оптимальним чином відповідають потребам конкретного користувача або додатку. Розглянемо кешування в World Wide Web. Браузери зазвичай дозволяють користувачам адаптувати правила кешування під їхні потреби шляхом визначення розміру кеша, а також того, чи повинен кешований документ перевірятися на відповідність постійно або тільки один раз за сеанс. Однак користувач не може впливати на інші параметри кешування, такі як тривалість збереження документа в кеші або черговість видалення документів з кешу при його переповненні. Також неможливо створювати правила кешування на основі вмісту документа. Так, наприклад, користувач може побажати кешувати дані залізничного розкладу, які рідко змінюються, але не інформацію про пробки на вулицях міста.
Необхідно відокремити правила від механізму. У разі кешування, Web браузер повинен надавати тільки можливості для збереження документів в кеші і одночасно давати користувачам можливість вирішувати, які документи і наскільки довго там зберігати. На практиці це може бути реалізовано наданням великого списку параметрів, значення яких користувач зможе (динамічно) задавати. Ще краще, якщо користувач отримає можливість сам встановлювати правила у вигляді компонентів, які підключаються до браузера. Зрозуміло, браузер повинен розуміти інтерфейс цих компонентів, оскільки йому потрібно буде, використовуючи цей інтерфейс, викликати процедури, що містяться в компонентах.

6.3.6 Масштабованість

Масштабованість системи може вимірюватися за трьома різними показниками. По-перше, система може бути масштабованоїю по відношенню до її розміру, що означає легкість підключення до неї додаткових користувачів і ресурсів. По-друге, система може масштабоватися географічно, тобто користувачі і ресурси можуть бути рознесені в просторі. По-третє, система може бути масштабованою в адміністративному сенсі, тобто бути проста в управлінні при роботі в безлічі адміністративно незалежних організаціях. Однак, система, що володіє масштабованістю по одному або декільком з цих параметрів, при масштабуванні часто дає втрату продуктивності вцілому.

Проблеми масштабованості

Якщо система потребує масштабування, необхідно вирішити безліч різноманітних проблем. Розглянемо процес масшабування. Якщо виникає необхідність збільшити число користувачів або ресурсів, стикаються з обмеженнями, пов'язаними з *централізацією служб*, даних і алгоритмів (табл.6.2). Багато служб централізуються тому, що при їх реалізації передбачалася наявність в розподіленої системі тільки одного сервера, запущеного на конкретній машині. При збільшенні числа користувачів сервер легко може стати критичним місцем системи. Навіть якщо є фактично необмежений запас по потужності обробки даних, ресурси зв'язку з цим сервером в кінці кінців будуть вичерпані.

Таблиця 6.2. Приклади обмежень масштабованості

Концепція	**Приклад**
Централізовані служби	Один сервер на всіх клієнтів
Централізовані дані	Єдиний телефоний довідник, який доступний у режимі підключення
Централізовані алгоритми	Організація маршрутизації на основі повної інформації

На жаль, використання єдиного сервера час від часу неминуче, наприклад, для служб управління особливо конфіденційною інформацією, таких як історії хвороби, банківські рахунки, кредити і т.п. У подібних випадках необхідно реалізовувати служби на одному сервері в окремій добре захищеній кімнаті і відокремлювати їх від інших частин розподіленої системи за допомогою спеціальних мережевих пристроїв. Копіювання інформації, що міститься на сервері, в інші місця для підвищення продуктивності не допускається.

Централізація даних так само шкідлива, як і централізація служб. Неможливо відстежувати телефонні номери і адреси 50 мільйонів чоловік. Припустимо, що кожен запис вкладається в 50 символів. Необхідної ємністю володіє один 2,5-гігабайтний диск. Але і в цьому випадку наявність єдиної бази даних викличе перевантаження вхідних і вихідних ліній зв'язку. Припустимо що в Інтернет служба доменних імен (DNS) реалізована у вигляді однієї таблиці. DNS обробляє інформацію з мільйонів комп'ютерів у всьому світі і надає службу, необхідну для визначення місця розташування web-серверів. Якби кожен запит на інтерпретацію URL передавався на єдиний DNS-сервер, скористатися Web не зміг би ніхто.

Централізація алгоритмів так само не витримує критики. У великих розподілених системах гігантське число повідомлень необхідно направляти по безлічі каналів. Теоретично для обчислення оптимального шляху необхідно отримати повну інформацію про завантаженість всіх машин і ліній і по алгоритмам з теорії графів обчислити всі оптимальні маршрути. Ця інформація потім повинна бути роздана по системі для поліпшення маршрутизації.

Проблема полягає в тому, що збір і транспортування всієї інформації може перевантажити частину мережі. Слід уникати алгоритму, який вимагає передачі інформації, яка збирається з усієї мережі, на одну з її машин для обробки з наступною роздачею результатів. Використовувати слід тільки децентралізовані алгоритми. Ці алгоритми зазвичай мають такі властивості, що відрізняють їх від централізованих алгоритмів:

– жодна з машин не володіє повною інформацією про стан системи;

– машини приймають рішення на основі локальної інформації;

– збій на одній машині не викликає порушення алгоритму;

– не вимагається припущення про існування єдиного часу.

Перші три властивості пояснюють раніше сказане. Останнє, менш очевидно, але не менш важливо. Алгоритм, який реалізує твердження: «Рівно о 12:00:00 всі машини повинні визначити розмір своїх вхідних черг», працювати не буде, оскільки неможливо синхронізувати всі годинники в світі. Алгоритми повинні брати до уваги відсутність повної синхронізації таймерів. Чим більше система, тим більшим буде і неузгодженість. В одній локальній мережі шляхом певних зусиль можна домогтися, щоб розсинхронізація всіх годинників не перевищувала декількох мілісекунд, але неможливо зробити це в масштабі країни або багатьох країн.

У географічній масштабованості є свої проблеми. Одна з основних причин складності масштабування існуючих розподілених систем, розроблених для локальних мереж, полягає в тому, що в їх основі лежить принцип синхронної зв'язку (synchronous communication). У цьому виді зв'язку запитуючий службу агент, якого прийнято називати клієнтом (client), блокується до отримання відповіді. Цей підхід зазвичай успішно працює в локальних мережах, коли зв'язок між двома машинами триває максимум сотні мікросекунд. Однак в глобальних системах необхідно прийняти до уваги факт, що зв'язок між процесами може тривати сотні мілісекунд, тобто на три порядки довше.

Інша проблема, яка перешкоджає географічному масштабуванні, полягає в тому, що зв'язок в глобальних мережах фактично завжди організовується від точки до точки і тому ненадійний. На противагу глобальним, локальні мережі зазвичай дають високонадійний зв'язок, заснований на широкомовної розсилці, що робить розробку розподілених систем для них значно простіше. Для прикладу розглянемо проблему локалізації служби. У локальній мережі система просто розсилає повідомлення всім машинам, опитуючи їх на предмет надання потрібної служби. Машини, що надають службу, відповідають на це повідомлення, вказуючи у відповідному повідомленні свої мережеві адреси. Неможливо реалізувати подібну схему визначення місцеположення в глобальній мережі. Замість цього необхідно забезпечити спеціальні місця для розташування служб, які можуть знадобитися масштабувати на весь світ і забезпечити їх потужністю для обслуговування мільйонів користувачів.

Географічна масштабованість жорстко пов'язана з проблемами централізованих рішень, які заважають масштабуванню за розміром. Якщо є система з безліччю централізованих компонентів, то географічна масштабованість буде обмежуватися проблемами продуктивності і надійності, пов'язаними з глобальним зв'язком. Крім того, централізовані компоненти здатні викликати перевантаження мережі.

У багатьох випадках залишається відкритим питання, як забезпечити масштабування розподіленої системи на безліч адміністративно незалежних областей. Основна проблема, яку потрібно при цьому вирішити, полягає в конфліктах правил, що відносяться до використання ресурсів (і плати за них), управлінню і безпеки.

Так, безліч компонентів розподілених систем, що знаходяться в одній області, зазвичай може бути довірено користувачам, які працюють в цій галузі. В цьому випадку системний адміністратор може тестувати і сертифікувати додатки, використовуючи спеціальні інструменти. Користувачі довіряють своєму системному адміністратору. Однак ця довіра не поширюється за межі області.

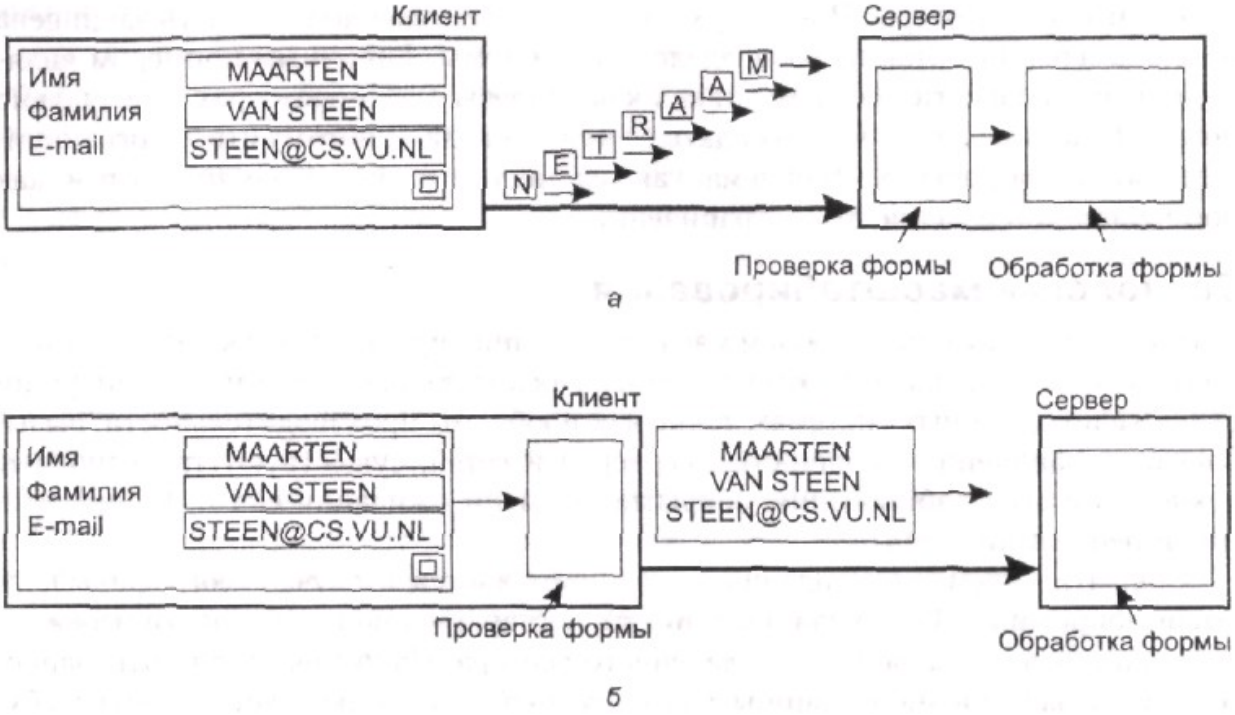
Якщо розподілені системи поширюються на іншу область, можуть знадобитися два типи перевірок безпеки. По-перше, розподілена система повинна протистояти атакам з нової області. Так, наприклад, користувачі нової області можуть отримати обмежені права доступу до файлової служби системи у вихідній області, скажімо, тільки на читання. Точно так же може бути закритий доступ чужих користувачів і до апаратури, такий як дорогі пристрої друку або високопродуктивні комп'ютери. По-друге, нова область сама повинна бути захищена від зловмисних атак з розподіленої системи. Типовим прикладом є завантаження по мережі програм, таких як аплети в web-браузерах. Спочатку нова область не знає, чого очікувати від чужого коду, і тому строго обмежує йому права доступу.

Технології масштабування

Оскільки проблеми масштабованості в розподілених системах, пов'язані з продуктивністю, викликаються обмеженою потужністю серверів і мереж, існують три основні технології масштабування - приховування часу очікування зв'язку, розподіл і реплікація.

Приховування часу очікування зв'язку застосовується і випадку географічного масштабування. Основна ідея: по можливості уникнути очікування відповіді на запит від віддаленого сервера. Наприклад, якщо була запрошена служба віддаленої машини, альтернативою очікуванню відповіді від сервера буде здійснення на запитуючій стороні інших можливих дій. Це означає розробку запитувачем додатку в розрахунку на використання виключно асинхронного зв'язку (*в-sinchronous communication*). Коли буде отримана відповідь, додаток перерве свою роботу і викличе спеціальний обробник для завершення відправленого раніше запиту. Асинхронний зв'язок часто використовується в системах пакетної обробки і паралельних програмах, в яких під час очікування одним завданням завершення зв'язку передбачається виконання інших більш-менш незалежних завдань. Для здійснення запиту може бути запущений новий керуючий потік виконання. Хоча він буде блокований на час очікування відповіді, інші потоки процесу продовжать своє виконання.

Однак багато додатків не в змозі ефективно використовувати асинхронний зв'язок. Наприклад, коли в інтерактивному додатку користувач надсилає запит, він зазвичай не в змозі робити нічого крім, очікування відповіді. У цих випадках найкращим рішенням буде скоротити необхідний обсяг взаємодії, наприклад, перемістивши частину обчислень, зазвичай виконуваних на сервері, на клієнта, процес якого запитує службу. Стандартний випадок застосування цього підходу - доступ до баз даних з використанням форм. Зазвичай заповнення форми супроводжується посилкою окремого повідомлення на кожне поле і очікуванням підтвердження про отримання сервером, як показано па рис. 6.1а. Сервер може перед прийомом введеного значення перевірити його на синтаксичні помилки. Більш успішне вирішення полягає в тому, щоб перенести код для заповнення форми і, можливо, перевірки введених даних на клієнта, щоб він міг послати до сервера цілком заповнену форму (рис.6.1б). Такий підхід – перенесення коду на клієнта - в даний час широко підтримується в Web за допомогою Java-аплетів.



Мал. 6.1. Різниця між перевіркою форми у міру заповнення на сервері (а) і на клієнті (б)

Наступна важлива технологія масштабування - ***розподіл*** (distribution). Розподіл передбачає розбиття компонентів на дрібні частини і подальше рознесення цих частин по системі. Прикладом розподілу є система доменних імен Інтернету (DNS). Простір DNS-імен організовано ієрархічно, у вигляді дерева доменів (domains), які розбиті на зони (zones), що не неперекриваються, як показано на рис. 6.2. Імена кожної зони обробляються одним сервером імен. Можна вважати, що кожне доменне ім'я є ім'ям хоста в Інтернеті і асоціюється з мережевою адресою цього хоста. В основному інтерпретація імені означає отримання мережевої адреси відповідного хоста. Розглянемо ім'я nl.vu.cs.flits. Для інтерпретації цього імені воно спочатку передається на сервер зони Z1 (рис.6.4), який повертає адресу сервера зони Z2, який, ймовірно, зможе обробити залишок імені, vu.cs.flits. Сервер зони Z2 поверне адресу сервера зони Z3, який здатний обробити останню частину імені і повернути адресу відповідного хоста.

Ці приклади демонструють, як служба іменування, що надається DNS, розподілена по кількох машинам і це дозволяє уникнути обробки всіх запитів на інтерпретацію імен одним сервером.

Як інший приклад розглянемо World Wide Web. Для більшості користувачів Web представляється у вигляді гігантської інформаційної системи документообігу, в якій кожен документ має своє унікальне ім'я - URL. Концептуально можна припустити навіть, що всі документи розміщуються на одному сервері. Однак середовище Web фізично рознесене по безлічі серверів, кожен з яких містить певну кількість документів. Ім'я сервера, що містить конкретний документ, визначається по URL-адресою документа. Тільки завдяки подібному розподілу документів Всесвітня павутина змогла вирости до її сучасних розмірів.

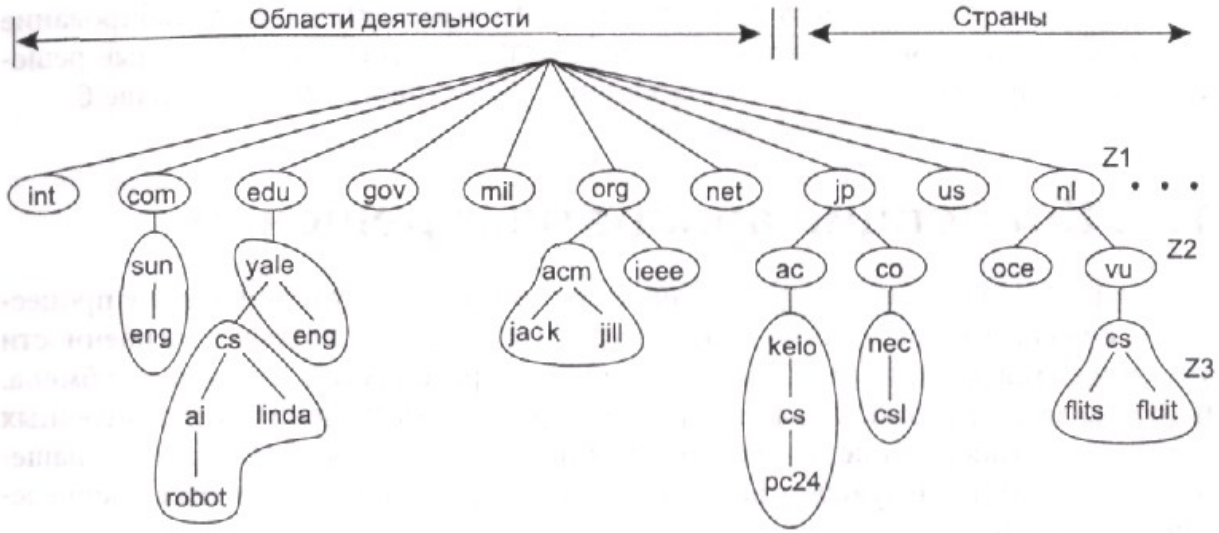


Рис. 6.2. Приклад поділу простору DNS-імен на зони

При розгляді проблем масштабування, які часто виявляються у вигляді падіння продуктивності, нерідко гарною ідеєю є ***реплікація*** (*replication*) компонентів розподіленої системи. Реплікація - це створення копії ресурсу його власником для збільшення продуктивності його спільної обробки для даного ресурсу. Реплікація не тільки підвищує доступність, але і допомагає вирівняти завантаження компонентів, що веде до підвищення продуктивності. Крім того, в сильно географічно розосереджених системах наявність близько лежачої копії дозволяє знизити гостроту здебільшого раніше обговорюваних проблем очікування завершення зв'язку.

Кешування (*caching*) являє собою особливу форму реплікації, причому відмінності між ними нерідко малопомітні або взагалі штучні. Як і в разі реплікації, результатом кешування є створення копії ресурсу, зазвичай в безпосередній близькості від клієнта, що використовує цей ресурс. Однак на противагу реплікації кешування - це дії, що робляться споживачем ресурсу, а не його власником.

На масштабованість може погано вплинути один істотний недолік кешування і реплікації. Оскільки ми отримуємо безліч копій ресурсу, модифікація однієї копії робить її відмінною від інших. Відповідно, кешування і реплікація викликають проблеми ***несуперечності*** (*consistency*).

Допустима ступінь суперечливості залежить від ступеня завантаження ресурсів. Так, безліч користувачів Web вважають допустимим роботу з кешованим документом через кілька хвилин після його приміщення в кеш без додаткової перевірки. Однак існує безліч випадків, коли необхідно гарантувати сувору несуперечливість, наприклад, при грі на електронній біржі. Проблема суворої несуперечності полягає в тому, що зміна в одній з копій повинно негайно поширюватися на всі інші. Крім того, якщо дві зміни відбуваються одночасно, часто буває необхідно, щоб ці зміни вносилися в одному і тому ж порядку в усі копії. Для обробки ситуацій такого типу зазвичай потрібно механізм глобальної синхронізації. На жаль, реалізувати масштабування подібних механізмів вкрай важко, а може бути і неможливо. Це означає, що масштабування шляхом реплікації може включати в себе окремі немасштабуємі рішення.

6.4 **Апаратні рішення**

Незважаючи на те, що всі розподілені системи містять по кілька процесорів, існують різні способи їх організації в систему. Особливо це відноситься до варіантів їх з'єднання і організації взаємного обміну. Розглянемо апаратне забезпечення розподілених систем, зокрема варіанти з'єднання машин між собою
Будемо розглядати системи, побудовані з набору незалежних комп'ютерів. На рис. 4 всі комп'ютери розділені на дві групи. Системи, в яких комп'ютери використовують пам'ять спільно, зазвичай називаються ***Мультіпроцесори*** (*multiprocessors*), а працюють кожен зі своєю пам'яттю - ***мультікомпьютерами*** (*mullicomputers*). Основна різниця між ними полягає в тому, що мультипроцесори мають єдиний адресний простір, який спільно використовується всіма процесорами. Якщо один з процесорів записує, наприклад, значення 44 за адресою 1000, будь-який інший процесор, який після цього прочитає значення, що лежить за адресою 1000, отримає 44. Всі машини задіюють одну і ту ж пам'ять.
На відміну від таких машин в мультікомп’ютерах кожна машина використовує свою власну пам'ять. Після того як один процесор запише значення 44 за адресою тисячі, інший процесор, прочитавши значення, що лежить за адресою тисячі, отримає те значення, яке зберігалося там раніше. Запис за цією адресою значення 44 іншим процесором ніяк не позначиться на вмісті його пам'яті. Типовий приклад мультікомпьютера - кілька персональних комп'ютерів, об'єднаних в мережу.
Кожна з цих категорій може бути підрозділена на додаткові категорії на основі архітектури, що з'єднує їх мережі. На рис. 6.3 ці дві архітектури позначені як ***шинна*** (bus) і комутуєма (switched) архітектури. Під шиною розуміється одиночна мережа, плата, шина, кабель або інше середовище, що з'єднує її машини між собою. Подібну схему використовує кабельне телебачення: кабельна компанія простягає уздовж вулиці кабель, а всім передплатникам робляться відведення від основного кабелю до їх телевізорів.
Комутовані системи, на відміну від шинних, не мають єдиної магістралі, такої як у кабельного телебачення. Замість неї від машини до машини тягнуться окремі канали, виконані із застосуванням різних технологій зв'язку. Повідомлення передаються по каналах з прийняттям явного рішення про комутацію з конкретним вихідним каналом для кожного з них. Так організована глобальна телефонна мережа.

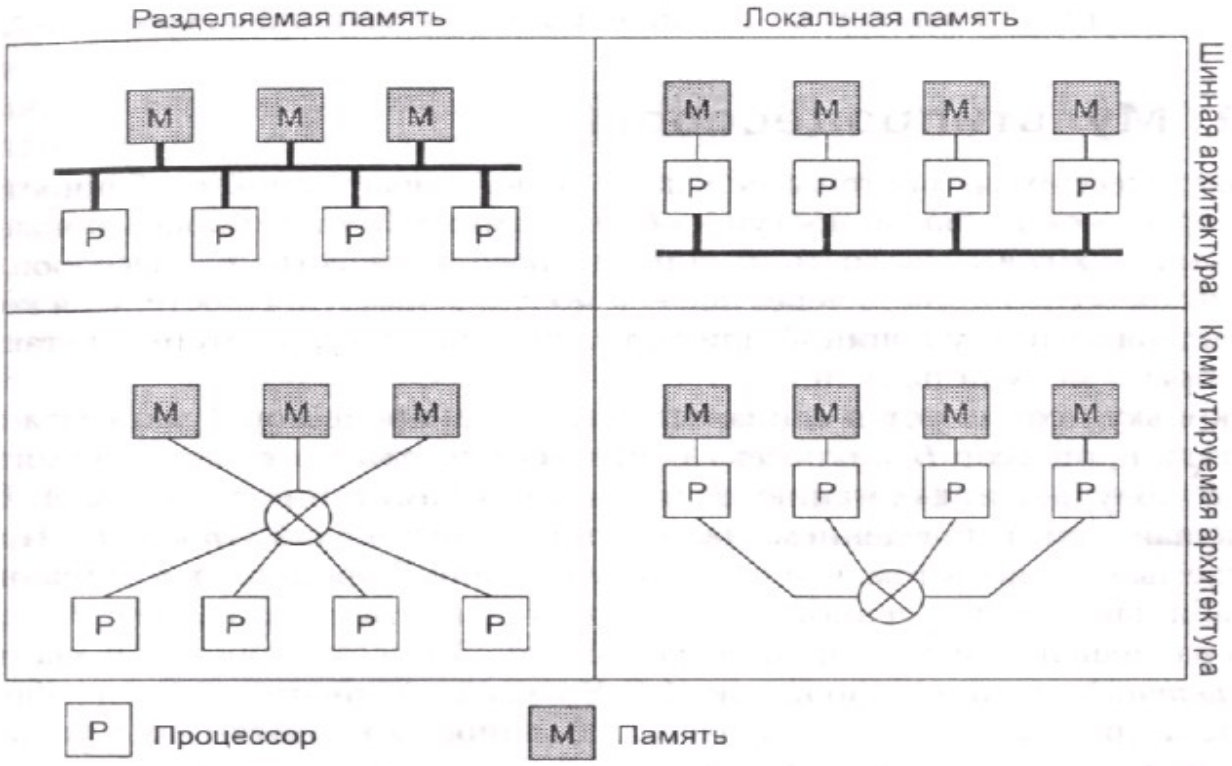


Рис. 6.3. Різні базові архітектури процесорів і пам'яті розподілених комп'ютерних систем

Ми проведемо також поділ розподілених комп'ютерних систем на ***гомогенні*** (*homogeneous*) і ***гетерогенні*** (*heterogeneous*). Це поділ застосовується виключно до мультікомпьютерних систем. Для гомогенних мультікомпьютерних систем характерна одна мережа, яка з'єднує комп'ютери, що використовує єдину технологію. Однакові також і всі процесори, які в основному мають доступ до однакових обсягів власної пам'яті. Гомогенні мультікомп’ютерні системи нерідко використовуються в якості паралельних (працюють з одним завданням), в точності як мультипроцесорні.

На відміну від них гетерогенні мультікомп’ютерні системи можуть містити цілу гаму незалежних комп'ютерів, з'єднаних різноманітними мережами. Так, наприклад, розподілена комп'ютерна система може бути побудована з кількох локальних комп'ютерних мереж, з'єднаних комутованою магістраллю FDDI (Fiber Distributed Data Interface - розподілений інтерфейс передачі даних по оптоволокну) або ATM (Asynchronous Transfer Mode - Технологія асинхронного режиму передачі комірок).

У наступних трьох пунктах розглянемо мультипроцесорні, а також гомогенні і гетерогенні мультікомп’ютерні системи. Ці питання не пов'язані безпосередньо з розподіленими системами, проте вони допомагають краще їх зрозуміти, оскільки організація розподілених систем часто залежить від вхідної до їх складу апаратури.

6.4.1 Мультипроцесори

Мультипроцесорні системи володіють характерною особливістю: всі процесори мають прямий доступ до загальної пам'яті. Мультипроцесорні системи шинної архітектури складаються з деякої кількості процесорів, приєднаних до загальної шині, а через неї - до модулів пам'яті. Найпростіша конфігурація містить плату з шиною або материнську плату, в яку вставляються процесори і модулі пам'яті.

Оскільки використовується єдина пам'ять, коли процесор А записує слово в пам'ять, а процесор У мікросекунди пізніше зчитує слово з пам'яті, процесор У отримує інформацію, записану в пам'ять процесором А. Пам'ять, що володіє такою поведінкою, називається ***узгодженою*** (*coherent*). Проблема даної схеми полягає в тому, що в разі 4 або 5 процесорів шина виявляється стабільно перевантаженою і продуктивність різко падає. Рішення полягає в розміщенні між процесором і шиною високошвидкісної кеш-пам'яті (cache memory), як показано на рис. 6.4. У кеші зберігаються дані, звернення до яких відбувається найбільш часто. Всі запити до пам'яті відбуваються через кеш. Якщо запитані дані містяться в кеш-пам'яті, то на запит процесора реагує вона і звернення до шини не виконуються. Якщо розмір кеш-пам'яті досить великий, ймовірність успіху, звана також коефіцієнтом кеш-влучень (hit rate), велика і шинний трафік в розрахунку на один процесор різко зменшується, дозволяючи включити в систему значно більше процесорів. Загальноприйнятими є розміри кеша від 512 Кбайт до 1 Мбайт, коефіцієнт кеш-влучень при цьому зазвичай складає 90% і більше.

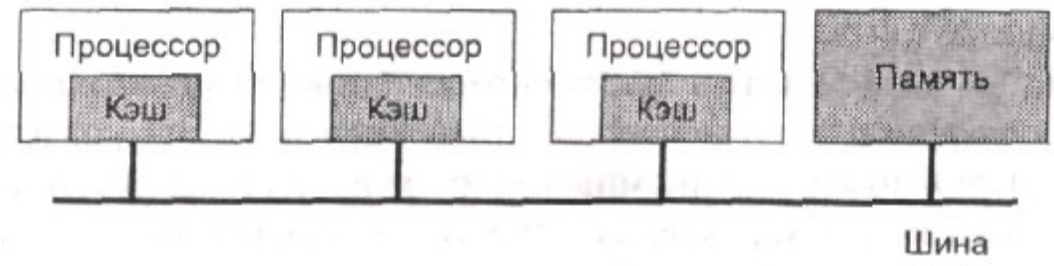


Рис. 6.4. Мультипроцессорная система з шинною архітектурою

Однак введення кеша створює серйозні проблеми саме по собі. Нехай два процесори, А і В, читають одне і те ж слово в свій внутрішній кеш. Потім А перезаписує це слово. Коли процесор У захоче скористатися цим словом, він вважає старе значення зі свого кеша, а не нове значення, записане процесором А. Пам'ять стала неузгодженою, і програмування системи ускладнилося. Кешування проте активно використовується в розподілених системах, тут також стикаються з проблемами неузгодженості пам'яті

Проблема мультипроцесорних систем шинної архітектури полягає в їх обмеженій масштабованості, навіть в разі використання кеша. Для побудови мультипроцессорної системи з більш ніж 256 процесорами для з'єднання процесорів з пам'яттю необхідні інші методи. Один з варіантів - розділити загальну пам'ять на модулі і зв'язати їх з процесорами через комутуючу решітку (crossbar switch), як показано на рис. 6а. як видно з малюнка, з її допомогою кожен процесор може бути пов'язаний з будь-яким модулем пам'яті. Кожен перетин являє собою маленький електронний вузловий комутатор (crosspoint switch), який може відкриватися і закриватися апаратно. Коли процесор бажає отримати доступ до конкретного модулю пам'яті, що з'єднує, їх вузлові комутатори відкриваються, організовуючи запитаний доступ. Гідність вузлових комутаторів в тому, що до пам'яті можуть одночасно звертатися декілька процесорів, хоча якщо два процесори одночасно хочуть отримати доступ до однієї і тої ж ділянки пам'яті, то одному з них доведеться почекати.

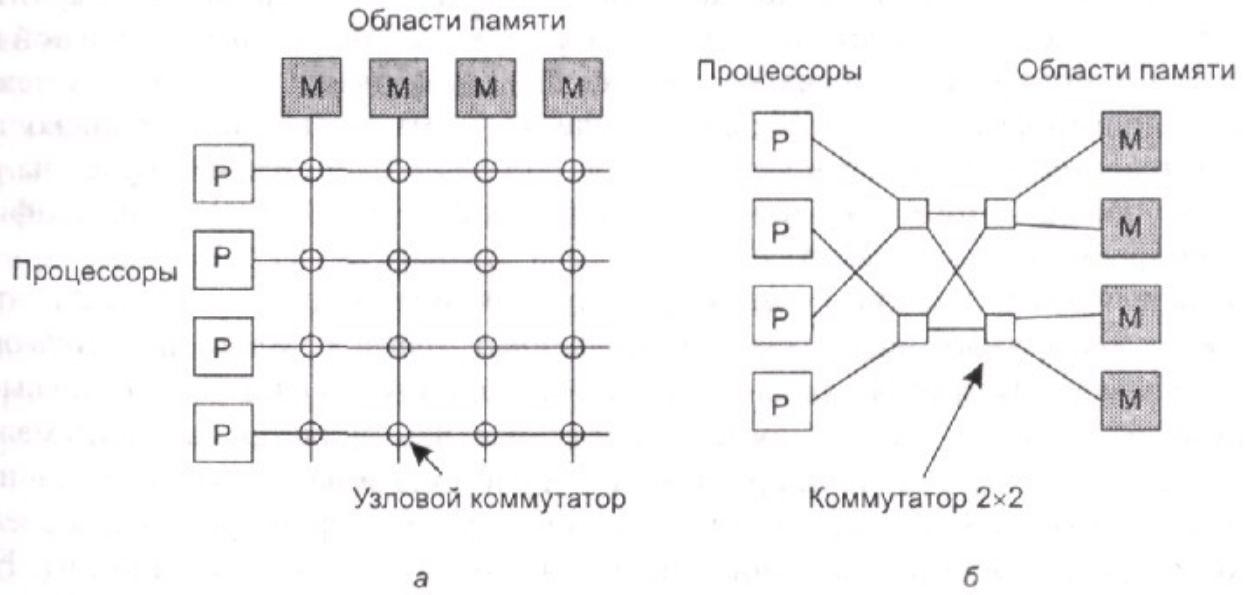


Рис. 6.5 Комутуюча решітка (а). Комутуюча омега-мережа (б)

Недоліком комутуючої решітки є те, що при наявності N процесорів і P модулів пам'яті буде потрібно N^2 вузлових комутаторів. Для великих значень N це число може перевищити можливості пам’яті. Тому були знайдені альтернативні комутуючі мережі, що вимагають меншої кількості комутаторів. Один із прикладів таких мереж - омега-мережа (omega network), представлена на рис. 6.5. Ця мережа містить чотири комутатори 2x2, тобто кожен з них має по два входи і два виходи. Кожен комутатор може з'єднувати будь-який вхід з будь-яким виходом. Якщо уважно вивчити можливі положення комутаторів, стає ясно, що будь-який процесор може отримати доступ до будь-якого блоку пам'яті. Недолік комутуючих мереж полягає в тому, що сигнал, який йде від процесора до пам'яті або назад, змушений проходити через кілька комутаторів. Тому, щоб знизити затримки між процесором і пам'яттю, комутатори повинні мати дуже високу швидкодію.
Намагаються зменшити витрати на комутацію шляхом переходу до ієрархічних систем. В цьому випадку з кожним процесором асоціюється деяка область пам'яті. Кожен процесор може швидко отримати доступ до своєї області пам'яті. Доступ до іншої області пам'яті відбувається значно повільніше. Ця ідея була реалізована в машині з неуніфікованим доступом до пам'яті (NonUniform Memory Access, NUMA). Хоча машини NUMA мають кращий середній час доступу до пам'яті, ніж машини на базі омега-мереж, у них є свої проблеми, пов'язані з тим, що розміщення програм і даних необхідно проводити так, щоб велика частина звернень йшла до локальної пам'яті.

6.4.2 Гомогенні мультикомп’ютерні системи

На відміну від мультипроцессоров побудувати мультікомп’ютерну систему відносно нескладно. Кожен процесор в ній безпосередньо пов'язаний зі своєю локальною пам'яттю. Єдина залишилася проблема - це спілкування процесорів між собою. Необхідна схема з'єднання, але оскільки цікавий тільки зв'язок між процесорами, обсяг трафіку буде на кілька порядків нижче, ніж при використанні мережі для підтримки трафіку між процесорами і пам'яттю.

Розглянемо гомогенні мультікомпьютерние системи. У цих системах, відомих під назвою системних мереж (System Area Networks, SAN), вузли монтуються в великий стійці і з'єднуються єдиною, зазвичай високошвидкісною мережею.
Як і в попередньому випадку, необхідно вибирати між системами на основі шинної архітектури і системами на основі комутації.
У мультікомпьютерних системах з шинною архітектурою процесори з'єднуються за допомогою мережі множинного доступу, наприклад Fast Ethernet. Швидкість передачі даних в мережі зазвичай дорівнює 100 Мбіт / с. Як і в разі мультипроцесорів з шинною архітектурою, мультікомпьютерні системи з шинною архітектурою мають обмежену масштабованість. Залежно від того, скільки вузлів насправді потребують обміну даними, зазвичай не слід очікувати високої продуктивності при перевищенні системою межі в 25-100 вузлів.
У комутованих мультікомпьютерних системах повідомлення, що передаються від процесора до процесора, маршрутизируются в сполучній мережі на відміну від прийнятих в шинній архітектурі широкомовних розсилок. Було запропоновано і побудовано безліч різних топологій. Дві популярні топології - квадратні решітки і гіперкуби - представлені на рис. 6.6. Решіткі прості для розуміння і зручні для розробки на їх основі друкованих плат. Вони прекрасно підходять для вирішення двомірних завдань, наприклад завдань теорії графів або комп'ютерного зору (очі робота, аналіз фотографій).

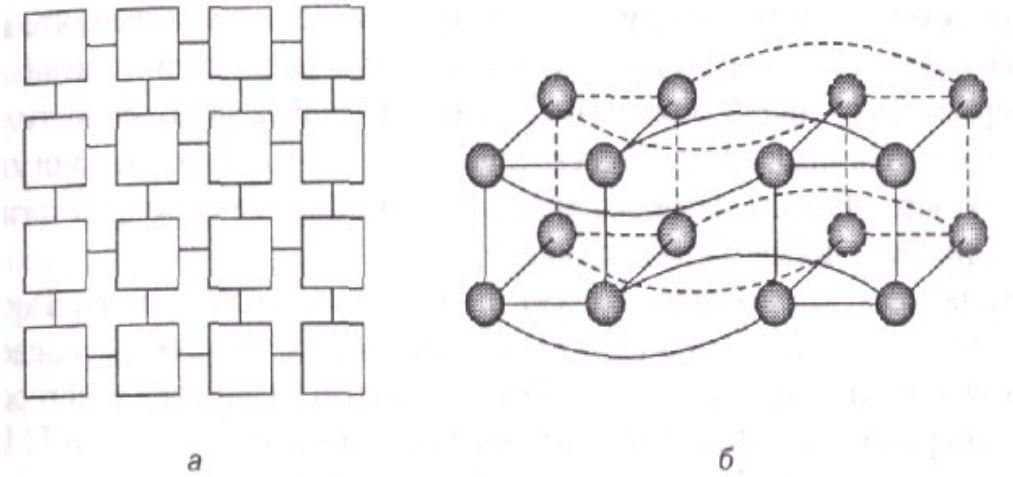


Рис. 6.6. Топології Решітка (а) та Гиперкуб (б)

Гиперкуб (hypercube) являє собою куб розмірності N. Гіперкуб, показаний на рис. 1.7б, четирmjxmsрний. Його можна представити у вигляді двох звичайних кубів, з 8 вершинами і 12 ребрами кожен. Кожна вершина - це процесор. Кожне ребро - це зв'язок між двома процесорами. Відповідні вершини обох кубів з'єднані між собою. Для розширення гіперкуба в п'ятий вимір ми повинні додати до цієї конструкції ще один комплект з двох зв'язаних кубів, з'єднавши відповідні вершини двох половинок фігури. Таким же чином можна створити шестімірний куб, семімірний і т. п.
Комутовані мультікомпьютерні системи можуть бути дуже різноманітні. На одному кінці спектра лежать процесори з масовим паралелізмом (Massively Parallel Processors, MPP), гігантські суперкомп'ютери вартістю в багато мільйонів доларів, що містять тисячі процесорів. Нерідко вони збираються з тих же процесорів, які використовуються в робочих станціях або персональних комп'ютерах. Від інших мультікомпьютерних систем їх відрізняє наявність патентованих високошвидкісних з’єднуючих мереж. Ці мережі проектуються в розрахунку на

малий час затримки і високу пропускну здатність. Крім того, вживаються спеціальні заходи для захисту системи від збоїв. При наявності тисяч процесорів щотижня кілька виходитимуть з ладу. Не можна допустити, щоб поломка одного з них приводила до виведення з ладу всієї машини.
На іншому кінці спектру ми виявляємо популярний тип комутованих мікрокомп'ютерів, відомих як кластери робочих станцій (Clusters Of Work-Nations, COW), основу яких складають стандартні персональні комп'ютери або робочі станції, з'єднані за допомогою комерційних комунікаційних компонентів. Сполучні мережі це те, що відрізняє COW від MPP. Крім того, зазвичай не робиться ніяких особливих заходів для підвищення швидкості введення-виведення або захисту від збоїв в системі. Подібний підхід робить COW простіше і дешевше.

6.4.3 Гетерогенні мультикомп’ютерні системи

Найбільше число існуючих в даний час розподілених систем побудовано за схемою гетерогенних мультікомпьютерних. Це означає, що комп'ютери, які є частинами цієї системи, можуть бути дуже різноманітні, наприклад, за типом процесора, обсягом пам'яті і продуктивності каналів введення-виведення. На практиці роль деяких з цих комп'ютерів можуть виконувати високопродуктивні паралельні системи, наприклад мультипроцесорні або гомогенні мультікомпьютерні. З'єднання їх у мережу також може бути дуже неоднорідним.
Іншим прикладом гетерогенності є створення великих мультікомп’ютерних систем з використанням існуючих мереж і каналів. Так, наприклад, не є чимось незвичайним існування кампусних університетських розподілених систем, що складаються з локальних мереж різних факультетів, з'єднаних між собою високошвидкісними каналами. У глобальних системах різні станції можуть, в свою чергу, з'єднуватися загальнодоступними мережами, наприклад, мережевими службами, запропонованими комерційними операторами зв'язку, такими, як SMDS або Frame relay.
На відміну від систем, що обговорювалися в попередніх пунктах, багатовеликомасштабні гетерогенні мультікомп’ютерні системи потребують глобального підходу. Це означає, що програма не може припускати, що їй постійно буде доступна тільки певна продуктивність або певні служби.
Переходячи до питань масштабування, властивих гетерогенним системам, і з огляду на необхідність глобального підходу, властиву більшості з них, слід зауважити, що створення додатків для гетерогенних мультікомпьютерних систем вимагає спеціалізованого програмного забезпечення.

7 Концепції програмних рішень

Розподілені системи схожі на традиційні операційні системи. Вони працюють як ***менеджери ресурсів*** (*resource managers*) існуючого апаратного забезпечення, які допомагають безлічі користувачів і додатків спільно використовувати такі ресурси, як процесори, пам'ять, периферійні пристрої, мережу і дані всіх видів. Більш важливо, що розподілена система приховує складність і гетерогенну природу апаратного забезпечення, на базі якого вона побудована, надаючи віртуальну машину для виконання додатків.

Розглянемо операційні системи з точки зору розподіленості. Операційні системи для розподілених комп'ютерів можна розділити на дві категорії - сильно зв'язані і слабко зв'язані системи. У сильно пов'язаних системах операційна система в основному працює з одним, глобальним поданням ресурсів, якими вона управляє. Слабко зв'язані системи можуть представлятися набором операційних систем, кожна з яких працює на власному комп'ютері. Однак ці операційні системи функціонують спільно, роблячи власні служби доступними іншим.

Цей поділ на сильно і слабко пов'язані системи пов'язано з класифікацією апаратного забезпечення, яка наведена в попередньому розділі. Сильно зв'язані операційні системи зазвичай називаються ***розподіленими операційними системами*** (*Distributed Operating System*, DOS) і використовуються для управління мультіпроцесорними і гомогенними мультікомпьютернимі системами. Як і у традиційних однопроцесорних операційних систем, основна мета розподіленої операційної системи полягає в приховуванні тонкощів управління апаратним забезпеченням, яке одночасно використовується безліччю процесів.

Слабко зв'язані ***мережеві операційні системи*** (*Network Operating Systems*) використовуються для управління гетерогенними мультікомпьютерними системами. Хоча управління апаратним забезпеченням і є основним завданням мережевих операційних систем, вони відрізняються від традиційних. Ця відмінність випливає з того факту, що локальні служби повинні бути доступними для віддалених клієнтів.

Щоб дійсно скласти розподілену систему, служб мережевої операційної системи недостатньо. Необхідно додати до них додаткові компоненти, щоб організувати кращу підтримку прозорості розподілу. Цими додатковими компонентами будуть програми, відомі як системи проміжного рівня (middleware), які і лежать в основі сучасних розподілених систем. У табл. 7.1 представлені основні дані по розподіленим і мережевим операційним системам, а також засобам проміжного рівня.

Таблиця 7.1. Скорочений опис розподілених і мережевих операційних систем, а також засобів проміжного рівня

Система	Опис	Основне призначення
Розподілені операційні системи	Сильно зв'язані операційні системи для мультипроце-сорів і гомогенних мультікомпьютерних систем	Приховування і управління апаратним забезпеченням
Мережеві операційні системи	Слабко зв'язані операційні системи для гетерогенних мультікомпьютерних систем (локальних і глобальних мереж)	Надання місцевих служб віддаленим клієнтам
Засоби проміжного рівня	Додатковий рівень поверх мережевих операційних систем, який реалізує служби загального призначення	Забезпечення прозорості розподілу

7.1 Розподілені операційні системи

Існує два типи розподілених операційних систем. ***Мультипроцесорна операційна система*** (*multiprocessor operating system*) управляє ресурсами мультипроцесора. ***Мультикомп'ютерна операційна система*** (*multicomputer operating system*) призначена для гомогенних мультікомп'ютеров. Функціональність розподілених операційних систем в основному не відрізняється від функціональності традиційних операційних систем, призначених для комп'ютерів з одним процесором за винятком того, що вона підтримує функціонування декількох процесорів.

7.1.1 Операційні системи для однопроцесорних комп'ютерів

Операційні системи традиційно будувалися для управління комп'ютерами з одним процесором. Основним завданням цих систем була організація легкого доступу користувачів і додатків до поділюваних пристроїв, таким як процесор, пам'ять, диски і периферійні пристрої. Говорячи про поділ ресурсів, ми маємо на увазі можливість використання одного і того ж апаратного забезпечення різними додатками ізольовано один від одного. Для додаткиу це виглядає так, немов ці ресурси перебувають у його повному розпорядженні, при цьому в одній системі може виконуватися одночасно кілька додатків, кожне зі своїм власним набором ресурсів. У цьому сенсі говорять, що операційна система реалізує віртуальну машину (*virtual machine*), надаючи додаткам засоби мультизадачності.
Важливим аспектом спільного використання ресурсів у такий віртуальній машині є те, що додатки відокремлені один від одного. Так, неможлива ситуація, коли при одночасному виконанні двох додатків, А і В, додаток А може змінити дані додатку В, просто працюючи з тією частиною загальної пам'яті, де ці дані зберігаються. Також потрібно гарантувати, що додатки зможуть використовувати надані їм ресурси тільки так, як наказано операційною системою. Наприклад, додаткам зазвичай заборонено копіювати повідомлення прямо в мережевий інтерфейс. Натомість операційна система надає первинні операції зв'язку, які можна використовувати для пересилання повідомлень між додатками на різних машинах.
Отже, операційна система повинна повністю контролювати використання та розподіл апаратних ресурсів. Тому більшість процесорів підтримують як мінімум два режими роботи. У режимі ядра (*kernel mode*) виконуються всі дозволені інструкції, а в ході виконання доступна вся наявна пам'ять і будь-які регістри. Навпаки, в призначеному для користувача режимі (*user mode*) доступ до регістрів і пам'яті обмежений. Так, додатку заборонено працювати з пам'яттю за межами набору адрес, встановленого для нього операційною системою, або звертатися безпосередньо до регістрів пристроїв. На час виконання коду операційної системи процесор перемикається в режим ядра. Однак єдиний спосіб перейти з режиму користувача в режим ядра - це зробити системний виклик, який реалізується через операційну систему. Оскільки системні виклики - це лише базові служби, що надаються операційною системою, і оскільки обмеження доступу до пам'яті і регістрів нерідко реалізується апаратно, операційна система в змозі повністю їх контролювати.

Існування двох режимів роботи призвело до такої організації операційних систем, при якій практично весь їх код виконується в режимі ядра. Результатом часто стають гігантські монолітні програми, що працюють в єдиному адресному просторі. Зворотний бік такого підходу полягає в тому, що переналаштувати систему часто буває нелегко. Іншими словами, замінити або адаптувати компоненти операційної системи без повного перезавантаження, а можливо і повної перекомпіляції і нової установки дуже важко. З точки зору відкритості, проектування програм, надійності або легкості обслуговування монолітні операційні системи - це не найкраща з ідей.
Більш зручний варіант з організацією операційної системи у вигляді двох частин. Одна частина містить набір модулів для управління апаратним забезпеченням, які можуть виконуватися в режимі користувача. Наприклад, управління пам'яттю складається в основному з відстеження, які блоки пам'яті виділені під процеси, а які вільні. Єдиний момент, коли необхідна зупинка в роботі в режимі ядра, - це установка регістрів блоку управління пам'яттю.
Друга частина операційної системи містить невелике мікроядро (*microkernel*), що містить виключно код, який виконується в режимі ядра. На практиці мікроядро повинно містити тільки код для установки регістрів пристроїв, перемикання процесора з процесу на процес, роботу з блоком управління пам'яттю і перехоплення апаратних переривань. Крім того, в ньому зазвичай міститься код, що перетворює виклики відповідних модулів користувацького рівня операційної системи в системні виклики і повертає результати. Такий підхід призводить до організації, показаної на рис. 7.1.

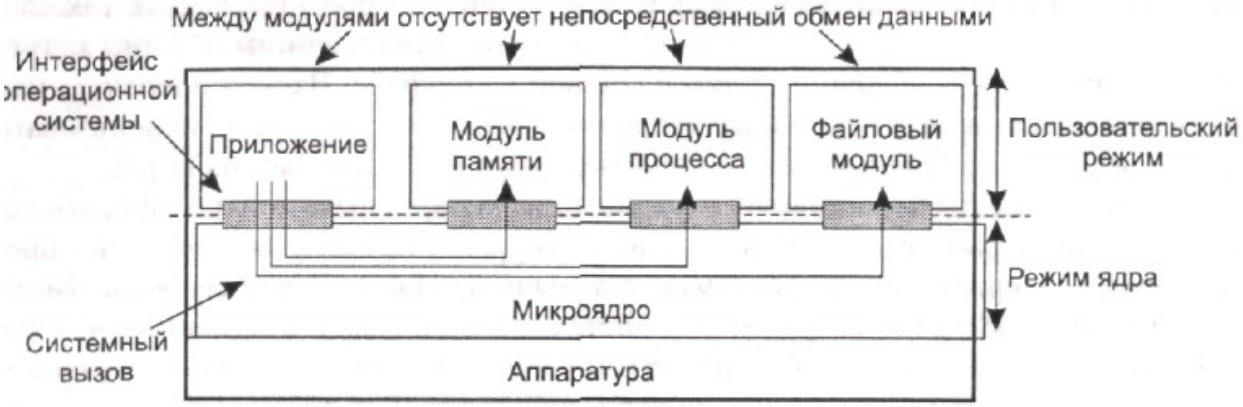


Рис. 7.1 Поділ додатків в операційній системі за допомогою мікроядра

Використання мікроядра дає різноманітні переваги. Найбільш важливе з них полягає в гнучкості: оскільки велика частина операційної системи виповнюється в призначеному для користувача режимі, відносно нескладно замінити один з модулів без повторної компіляції або повторної установки всієї системи. Інша перевага полягає в тому, що модулі користувальницького рівня можуть розміщуватися на різних машинах. Можна встановити модуль управління файлами не на тій машині, на якій він керує службою каталогів. Іншими словами, підхід з використанням мікроядра відмінно підходить для перенесення однопроцесорних операційних систем на розподілені комп'ютери.

У мікроядер є два суттєвих недоліки. По-перше, вони працюють інакше, ніж існуючі операційні системи. По-друге, микроядро вимагає додаткового обміну, що

злегка знижує продуктивність. Однак, зниження продуктивності в 20% навряд чи можна вважати фатальним.

7.1.2 Мультипроцесорні операційні системи

Важливим, але часто не дуже очевидним розширенням однопроцесорних операційних систем є можливість підтримки декількох процесорів, що мають доступ до спільно використовуваної пам'яті. Концептуально це розширення нескладно. Всі структури даних, необхідні операційній системі для підтримки апаратури, включаючи підтримку декількох процесорів, розміщуються в пам'яті. Основна відмінність полягає в тому, що дані доступні декільком процесорам і повинні бути захищені від паралельного доступу для забезпечення цілісності.
Однак багато операційних систем, особливо призначені для персональних комп'ютерів і робочих станцій, не можуть легко підтримувати кілька процесорів. Основна причина такої поведінки полягає в тому, що вони були розроблені як монолітні програми, які можуть виконуватися тільки в одному потоці управління. Адаптація таких операційних систем під мультипроцесорні означає повторне проектування і нову реалізацію його ядра. Сучасні операційні системи спочатку розробляються з урахуванням можливості роботи в мультипроцесорних системах.
Багатопроцесорні операційні системи націлені на підтримку високої продуктивності конфігурацій з декількома процесорами. Основне їхнє завдання - забезпечити прозорість числа процесорів для додатка. Зробити це досить легко, оскільки сполучення між різними додатками або їх частинами вимагає тих же примітивів, що і в багатозадачних однопроцесорних операційних системах. Повідомлення відбувається шляхом роботи з даними в спеціальній спільно використовуваної області даних, і все що потрібно - це захистити дані від одночасного доступу до них. Захист здійснюється за допомогою примітивів синхронізації. Два найбільш важливих (і еквівалентних) примітива - це семафори і монітори.
Семафор (*semaphore*) може бути представлений у вигляді цілого числа, що підтримує дві операції: up (збільшити) і down (зменшити). При зменшенні спочатку перевіряється, чи перевищує значення семафора 0. Якщо це так, його значення зменшується і виконання процесу триває. Якщо ж значення семафора нульове, що викликаємий процес блокується. Оператор збільшення робить протилежну дію. Спочатку він перевіряє всі заблоковані в даний час процеси, які були нездатні завершитися в ході попередньої операції зменшення. Якщо такі є, він розблокує один з них і продовжує роботу. В протилежному випадку він просто збільшує лічильник семафора. Розблокований процес виконується до виклику операції зменшення. Важливою властивістю операцій з семафорами є те, що вони атомарні (atomic), тобто в разі запуску операції зменшення або збільшення до моменту її завершення (або до моменту блокування процесу) ніякий інший процес не може отримати доступ до семафора.
Відомо, що програмування з використанням семафорів для синхронізації процесу викликає безліч помилок, крім випадків простого захисту поділюваних даних. Основна проблема полягає в тому, що наявність семафорів призводить до неструктурованого коду. Схожа ситуація виникає при частому використанні інструкції *goto*. В якості альтернативи семафора багато сучасних систем, що

підтримують паралельне програмування, надають бібліотеки для реалізації моніторів.
Формально ***монітор*** (*monitor*) являє собою конструкцію мови програмування, таку ж, як об'єкт в об'єктно-орієнтованому програмуванні. Монітор може розглядатися як модуль, в якому знаходяться і процедури. Доступ до змінних можна отримати тільки шляхом виклику однієї з процедур монітора. У цьому сенсі монітор дуже схожий на об'єкт. Об'єкт також має свої захищені параметри, доступ до яких можна отримати тільки через методи, реалізовані в цьому об'єкті. Різниця між моніторами і об'єктами полягає в тому, що монітор дозволяє виконання процедури тільки одному процесу в кожен момент часу. Іншими словами, якщо процедура, що міститься в моніторі, виконується процесом А (кажуть, що А увійшов в монітор) і процес В також викликає одну з процедур монітора, В буде блокований до завершення виконання А (тобто до тих пір, поки А не покине монітор).
Як приклад розглянемо простий монітор для захисту цілої змінної (лістинг 1). Монітор містить одну закриту (*private*) змінну count, доступ до якої можна отримати тільки через три відкритих (*public*) процедури - читання поточного значення, збільшення на одиницю і зменшення. Конструкція монітора гарантує, що будь-який процес, який викликає одну з цих процедур, отримає атомарний доступ до внутрішніх даних монітора.

```
monitor Counter { private:
int count-0:
int blocked_procs = 0; condition unblocked: public:
int valueO { return count;}
void inert) { if (blockecLprocs == 0) count - count + 1:
else
signal( unblocked );
}
void decrC) { If (count — 0) {
blocked_procs = blocked_procs + 1;
waitC unblocked );
blocked_procs = blocked_procs - 1: i / else
count = count - 1;
} )
```

Лістинг 1. Монітор, що оберігає ціле число від паралельного доступу

Монітори придатні для простого захисту спільно використовуваних даних. Однак для умовного блокування процесу необхідно більше.
Припустимо, що потрібно заблокувати процес при виклику операції зменшення, якщо виявляється, що значення *count* дорівнює нулю. Для цієї мети в моніторах використовуються умовні змінні (*condition variables*). Це спеціальні змінні з двома доступними операціями: *wait* (чекати) і *signal* (сигналізувати). Коли процес А знаходиться в моніторі і викликає для умовної змінної, що зберігається в моніторі, операцію wait, процес А буде заблокований і відмовиться від свого виняткового доступу до монітора. Відповідно, процес В, який чекав отримання виняткового доступу до монітора, зможе продовжити свою роботу. У певний момент часу В може розблокувати процес А викликом операції signal для умовної змінної, якого

очікує А. Щоб запобігти наявність двох активних процесів всередині монітора, слід доповнити схему так, щоб процес, який подав сигнал, залишав монітор. Монітор з лістингу 2 - це нова реалізація, яка обговорювалася раніше семафора.

```
monitor Counter {private:
int count-0:
int blocked_procs = 0; condition unblocked: public:
int valueO {return count;}
void inert) {if (blockecLprocs == 0) count - count + 1:
else
signal (unblocked);
}
void decrC) {If (count - 0) {
blocked_procs = blocked_procs + 1;
waitC unblocked);
blocked_procs = blocked_procs - 1: i / else
count = count - 1;
})
```

Лістинг 2. Монітор, що оберігає ціле число від паралельного доступу і блокуючий процес

Монітори є конструкціями мови програмування. Java підтримує монітори, дозволяючи кожному об'єкту охороняти себе від паралельного доступу шляхом використання в ньому інструкції *synchronized* і операцій wait і notify. Бібліотечна підтримка моніторів зазвичай реалізується на базі простих семафорів, які можуть приймати тільки значення 0 і 1. Такі семафори часто називаються змінними-мьютексами (*mutex variables*), або просто м'ютексів. З м'ютексами асоціюються операції *lock* (блокувати) і *unlock* (розблокувати). Захоплення мьютекса можливе тільки в тому випадку, якщо його значення дорівнює одиниці, в іншому випадку викликаємий процес буде блокований. Відповідно, звільнення мьютекса означає установку його значення в 1, якщо немає необхідності розблокувати який-небудь з процесів що чекали на виконання. Умовні змінні і відповідні їм операції також поставляються у вигляді бібліотечних процедур.

7.1.3 Мультікомпьютерні операційні системи

Мультікомпьютерні операційні системи мають набагато більш різноманітних структур і значно складніше, ніж мультипроцесорні. Ця різниця виникає з того, що структури даних, необхідні для управління системними ресурсами, не повинні відповідати умові легкості спільного використання, оскільки їх не потрібно поміщати в фізично загальну пам'ять, що показано на Рис.7.2.

Кожен вузол має своє ядро, яке містить модулі для управління локальними ресурсами - пам'яттю, локальним процесором, локальними дисками.

Крім того, кожен вузол має окремий модуль для міжпроцесорного впливу, тобто посилки повідомлень на інші вузли і прийому повідомлення від них.

Рис. 7.2. Загальна структура мультікомпьютерних операційних систем

Поверх кожного локального ядра лежить рівень програмного забезпечення загального призначення, який реалізує операційну систему у вигляді віртуальної машини, що підтримує паралельну роботу над різними завданнями. Цей рівень може надавати абстракцію мультипроцесорній машині. Іншими словами, він надає повну програмну реалізацію спільно використовуваної пам'яті. Додаткові засоби, що зазвичай реалізуються на цьому рівні, призначені, наприклад, для призначення завдань процесорам, маскування збоїв апаратури, забезпечення прозорості збереження і загального обміну між процесами. Ці засоби є типовими для операційних систем.

Мультікомпьютерні операційні системи, які не надають засоби для спільного використання пам'яті, можуть запропонувати для додатків тільки засобико для обміну повідомленнями. На жаль, семантика примітивів обміну повідомленнями в значній мірі різна для різних систем. Зрозуміти ці відмінності простіше, якщо відзначати, буферизуються повідомлення чи ні. Крім того, необхідно враховувати, чи блокується посилаємий або приймаємий процес. На рис. 7.3 продемонстрований варіант з буферизацією і блокуванням.

Рис. 7.3. Можливості блокування і буферизації при пересиланні повідомлень

Існує два можливих місця буферизації повідомлень - на стороні відправника або на стороні одержувача. Це призводить до чотирьох можливих точок синхронізації, тобто точок можливого блокування відправника або одержувача. Якщо

буферизація відбувається на стороні відправника, це дає можливість заблокувати відправника, тільки якщо його буфер повний, що показано точкою синхронізації S1 на малюнку. З іншого боку, процедура переміщення повідомлення в буфер може повертати стан, що показує, що операція успішно виконана. Це дозволяє відправнику уникнути блокування через переповнення буфера. Якщо ж відправник не має буфера, існує три альтернативних точки блокування відправника: відправлення повідомлення (точка S2), надходження повідомлення до одержувача (точка S3), прийняття повідомлення одержувачем (точка S4), якщо блокування відбувається в точці S2, S3 або S4, наявність або відсутність буфера на стороні відправника не має значення.

Блокування одержувача має сенс тільки в точці синхронізації S3 і може проводитися, тільки якщо у одержувача немає буфера або якщо буфер порожній. Альтернативою може бути опитування одержувачем наявності вхідних повідомлень. Однак ці дії часто ведуть до марної трати процесорного часу або занадто запізнілої реакції на те, яке прийшло повідомлення, що, в свою чергу, призводить до переповнення буфера вхідними повідомленнями і їх втрати.

Інший момент, важливий для розуміння семантики обміну повідомленнями, - надійність зв'язку. Відмінною рисою надійного зв'язку є отримання відправником гарантії прийому повідомлення. На рис. 7.3 надійність зв'язку означає, що всі повідомлення гарантовано досягають точки синхронізації S4. При ненадійному зв'язку всякі гарантії відсутні. Якщо буферизація проводиться на стороні відправника, про надійність зв'язку нічого певного сказати не можна. Також операційна система не потребує гарантовано надійного зв'язку в разі блокування відправника в точці S2.

З іншого боку, якщо операційна система блокує відправника до досягнення повідомленням точки S3 або S4, вона повинна мати гарантовано надійний зв'язок, В іншому випадку можна опинитися в ситуації, коли відправник чекає підтвердження отримання, а повідомлення було втрачено під час передачі. Відношення між блокуванням, буферизацией і гарантіями щодо надійності зв'язку підсумовані в табл. 7.2.

Таблиця 7.2. Співвідношення між блокуванням, буферизацией і надійністю зв'язку

Точка синхронізації	Буферизація відправника	Гарантія надійного зв'язку
Блокування відправника до наявності вільного місця в буфері	Так	Немає необхідності
Блокування відправника до посилки повідомлення	Немає	Немає необхідності
Блокування відправника до посилки повідомлення	Немає	Необхідна
Блокування відправника до обробки повідомлення	Немає	Необхідна

Безліч аспектів проектування мультікомпьютерних операційних систем однаково важливі для будь-якої розподіленої системи. Основна різниця між мультікомпьютерними операційними системами і розподіленими системами

полягає в тому, що в першому випадку зазвичай мається на увазі, що апаратне забезпечення гомогенно і повністю керовано. Безліч розподілених систем, однак, будується на базі існуючих операційних систем.

7.1.4 Системи з розподіленою пам'яттю

Практика показує, що програмувати мультікомпьютерні системи значно складніше, ніж мультипроцесорні. Різниця пояснюється тим, що зв'язок за допомогою процесів, що мають доступ до пам'яті, що спільно використовується, і простих примітивів синхронізації, таких як семафори і монітори, значно простіше, ніж робота з одним лише механізмом обміну повідомленнями. Такі питання, як буферизація, блокування і надійність зв'язку, тільки ускладнюють становище.
З цієї причини проводилися дослідження з питання емуляції пам'яті, що спільно використовується, на мультікомпьютерних системах. Їх метою було створення віртуальних машин з пам'яттю, що працюють на мультікомпьютерних системах, для яких можна було б писати додатки, розраховані на модель пам'яті, що спільно використовується, навіть якщо фізично вона відсутня. Головну роль в цьому відіграє мультікомпьютерна операційна система.
Один з поширених підходів - задіяти віртуальну пам'ять кожного окремого вузла для підтримки загального віртуального адресного простору. Це призводить нас до розподіленої пам'яті (Distributed Shared Memory - DSM) зі сторінковою організацією. DSM - віртуальний адресний простір, який поділяється всіма вузлами (процесорами) розподіленої системи. Програми отримують доступ до даних в DSM приблизно так само, як вони працюють з даними в віртуальної пам'яті традиційних комп’ютерів. У системах з DSM дані переміщаються між локальними пам'ятями різних комп'ютерів аналогічно тому, як вони переміщаються між оперативною та зовнішньою пам'яттю одного комп'ютера. Принцип роботи цієї пам'яті наступний. В системі з DSM адресний простір поділений на сторінки (зазвичай по 4 або по 8 Кбайт), розподілено по всім процесорам системи. Коли процесор адресується до пам'яті, яка не є локальною, відбувається внутрішнє переривання, операційна система зчитує в локальну пам'ять сторінку, яка містить вказану адресу, і перезапускає виконання викликала переривання інструкції, яка тепер успішно виконується. Цей підхід продемонстрований на рис. 7.4а для адресного простору з 16 сторінок і чотирьох процесорів. Це цілком нормальна сторінкова організація, якщо не брати до уваги того, що в якості тимчасового сховища інформації використовується не диск, а віддалена оперативна пам'ять.
У цьому прикладі при зверненні процесора 1 до коду або даних зі сторінки 0, 2, 5 або 9 звернення відбувається локально. Посилання на інші сторінки викликають внутрішнє переривання. Так, наприклад, посилання на адресу зі сторінки 10 викликає внутрішнє переривання операційної системи, і вона переміщує сторінку 10 з машини 2 на машину 1, як показано на рис. 7.4б.
Одне з поліпшень базової системи, що часто дозволяє значно підвищити її продуктивність, - це реплікація сторінок, які оголошуються закритими на запис, наприклад, сайти можуть містити матеріали тексту програми, констант, «тіпьки для читання» або інші закриті на запис структури. Наприклад, якщо сторінка 10 - це секція тексту програми, її використання процесором 1 призведе до пересилання процесору 1 її копії, а оригінал в пам'яті процесора 2 буде продовжувати спокійно зберігатися, як показано на рис. 7.4в. В цьому випадку процесори 1 і 2 обидва

зможуть звертатися до пошуку сторінки 10, не викликаючи при цьому ніяких внутрішніх переривань для вибірки пам'яті.
Сторінки адресного простору розподілені по чотирьох машинах, що показано на 7.4(а).
Ситуація після звернення процесора 1 до пошуку 10 демонструється на 7.4(б). Ситуація, коли запис в сторінку 10 неможливий і необхідна реплікація показана на 7.4(в).
Інша можливість - це реплікація також і не закритих на запис будь-яких сторінок. Поки проводиться тільки читання, ніякої різниці між реплікацією закритих і незакритих на запис сторінок немає. Однак у випадку операції реплікації коли, сторінка раптово змінюється, необхідно вживати спеціальні дії для запобігання появи безлічі несумісних копій. Зазвичай всі копії, крім однієї, перед проведенням запису оголошуються невірними.

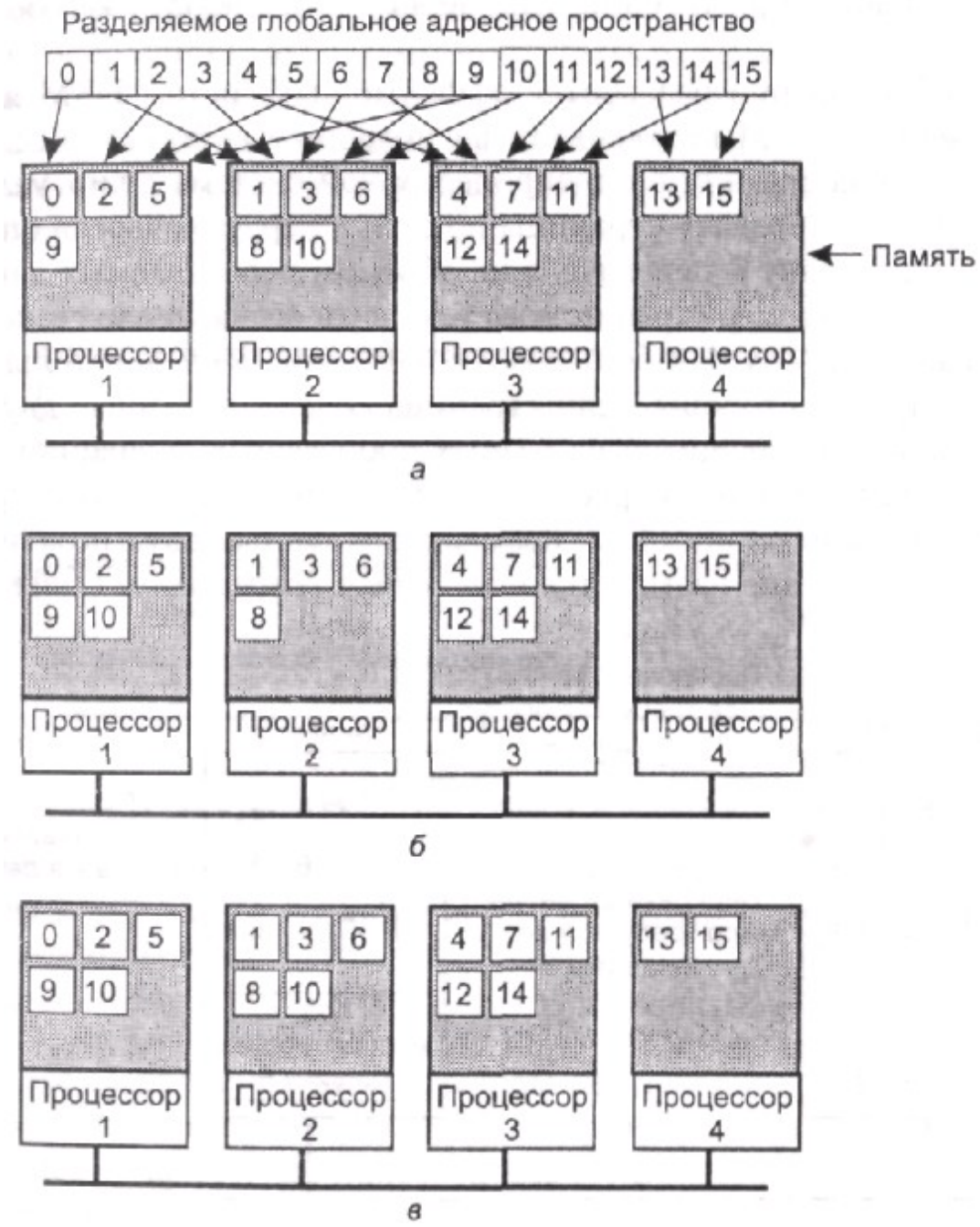


Рис. 7.4. Приклад організації розподілу пам’яті з сторінковою організацією (DSM)

Додаткового збільшення продуктивності можна домогтися шляхом відходу від строгої відповідності між сторінками, які реплікуються. Іншими словами, дозволяють окремій копії тимчасово відрізнятися від інших. Практика показує, що цей підхід дійсно може допомогти, але, може зажадати відстежувати можливу несумісність. Оскільки основною причиною розробки DSM була простота програмування, ослаблення відповідності не знаходить реального застосування.
Іншою проблемою при розробці ефективних систем DSM є питання про розмір сторінок. Витрати на передачу сторінки по мережі, в першу чергу, визначаються витратами на підготовку до передачі, а не залежністю від кількості надісланих даних. Відповідно, великий розмір сторінок може зменшити загальну кількість сеансів передачі при необхідності доступу до великої кількості послідовних елементів даних. З іншого боку, якщо сторінка містить дані двох незалежних процесів, що виконуються на різних процесорах, операційна система буде змушена постійно пересилати цю сторінку від одного процесора до іншого, як показано на рис. 7.5. Розміщення даних двох незалежних процесів на одній сторінці називається помилковим поділом (*false sharing*).

Рис. 7.5. Помилковий поділ сторінки двома незалежними процесами

Для досягнення високої продуктивності великомасштабних мультікомпьютерних систем вдаються до пересилання повідомлень, незважаючи на її високу, у порівнянні з програмуванням систем (віртуальної) пам'яті спільного використання, складність. Це дозволяє зробити висновок про те, що DSM не виправдовує очікувань, вимагаючи високопродуктивного паралельного програмування.

7.2 Мережеві операційні системи

На противагу розподіленим операційним системам мережеві операційні системи не потребують того, щоб апаратне забезпечення, на якому вони функціонують, було гомогенно і управлялося як єдина система. Зазвичай вони будуються для набору однопроцесорних систем, кожна з яких має власну операційну систему, як показано на рис. 7.5. Машини і їх операційні системи можуть бути різними, але всі вони з'єднані в мережу. Крім того, мережева операційна система дозволяє користувачам використовувати служби, розташовані на конкретній машині.

Можливо, буде простіше описати мережеву операційну систему, коротко розглянувши служби, які вона зазвичай надає.

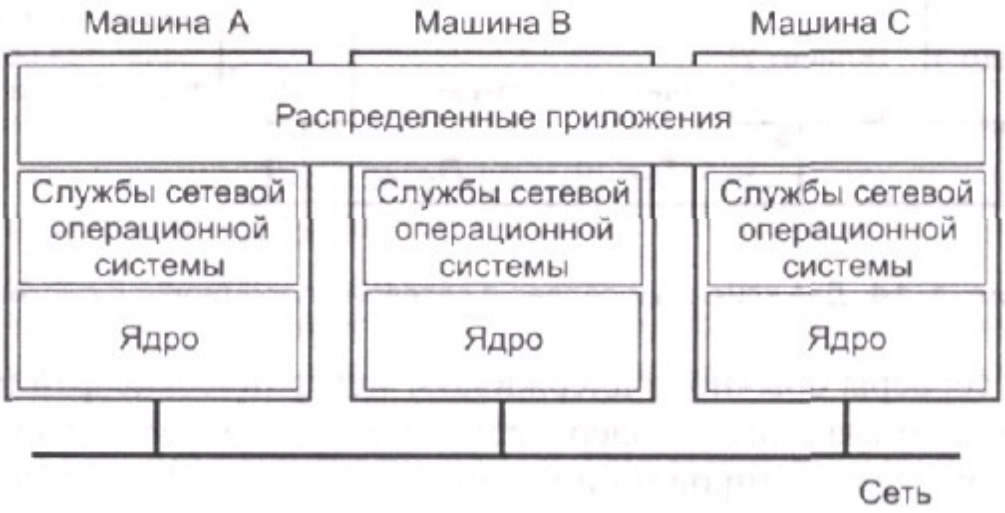


Рис. 7.5 Загальна структура мережевої операційної системи

Служба, зазвичай надається мережевими операційними системами, повинна забезпечувати віддалене з'єднання користувача з іншою машиною шляхом застосування команди.

В результаті виконання цієї команди відбувається перемикання робочої станції користувача в режим віддаленого терміналу, підключеного до віддаленої машині. Це означає, що користувач сидить у графічній робочої станції, набираючи команди на клавіатурі. Команди передаються на віддалену машину, результати з віддаленої машини відображаються у вікні на екрані користувача.

Для того щоб переключитися на іншу віддалену машину, необхідно відкрити нове вікно і скористатися командою rlogin для з'єднання з іншою машиною. Вибір віддаленої машини виробляється вручну.

Мережеві операційні системи також мають в своєму складі команду віддаленого копіювання для копіювання файлів з однієї машини на іншу. При цьому переміщення файла задається в явному вигляді, і користувачеві необхідно точно знати, де знаходяться файли і як виконуються команди.

Така форма зв'язку примітивна. Це змусило проектувальників систем шукати більш зручні варіанти зв'язку та спільного використання такої інформації. Один з підходів передбачає створення глобальної загальної файлової системи, доступної з усіх робочих станцій. Файлова система підтримується однією або декількома машинами, які називаються файловими серверами (file servers). Файлові сервери приймають запити від програм користувачів, що запускаються на інших машинах (не на серверах), які називаються клієнтами (clients), на читання і запис файлів. Кожен запит, що прийшов, перевіряється і виконується, а результат пересилається назад, як показано на рис. 7.6.

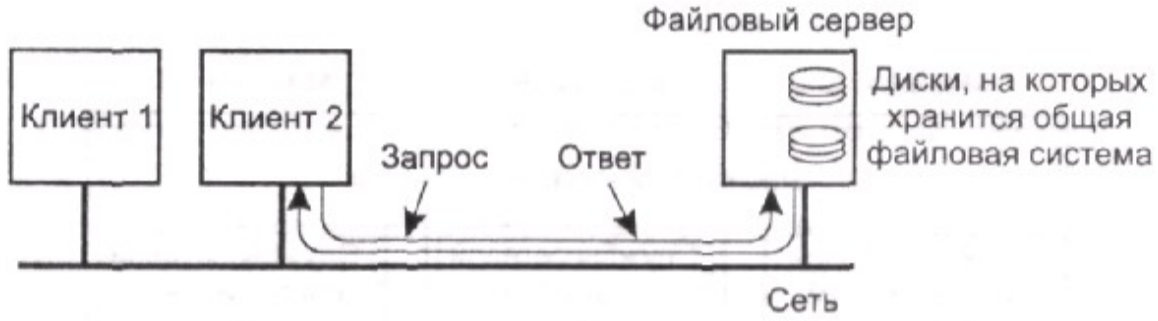


Рис. 7.6. Схема взаємодії двох клієнтів і сервера в мережевий операційній системі

Файлові сервери зазвичай підтримують ієрархічні файлові системи, кожна з кореневим каталогом, що містить вкладені каталоги і файли. Робочі станції можуть імпортувати або монтувати ці файлові системи, збільшуючи свою локальну файлову систему за рахунок файлової системи сервера. На рис. 7.7 показані два файлових сервери. На одному з них є каталог під назвою games, а на іншому - каталог під назвою work (імена каталогів виділені жирним шрифтом). Кожен з цих каталогів містить деякі файли. На обох клієнтах змонтовані файлові системи обох серверів, але в різних місцях файлових систем клієнтів. Клієнт 1 змонтував їх в свій кореневий каталог і має до них доступ по шляхах / games і / work відповідно. Клієнт 2, подібно Клієнту 1, змонтував каталог work в свій кореневої каталог, але вирішив, що ігри (games) повинні бути суто приватною справою. Тому він створив каталог, який назвав / private, і змонтував каталог games туди. Відповідно, він отримає доступ до файлу packwoman через шлях / private / games / packwoman, а не / games / packwoman.
Хоча зазвичай не має значення, в яке місце своєї ієрархії каталогів клієнт змонтував сервер, важливо пам'ятати, що різні клієнти можуть мати різне уявлення файлової системи. Файл залежить від того, як організовується доступ до нього і як виглядає файлова система на самій машині, Оскільки кожна клієнтська машина працює відносно незалежно від інших, неможливо дати гарантії, що вони мають однакову ієрархією каталогів для своїх програм.

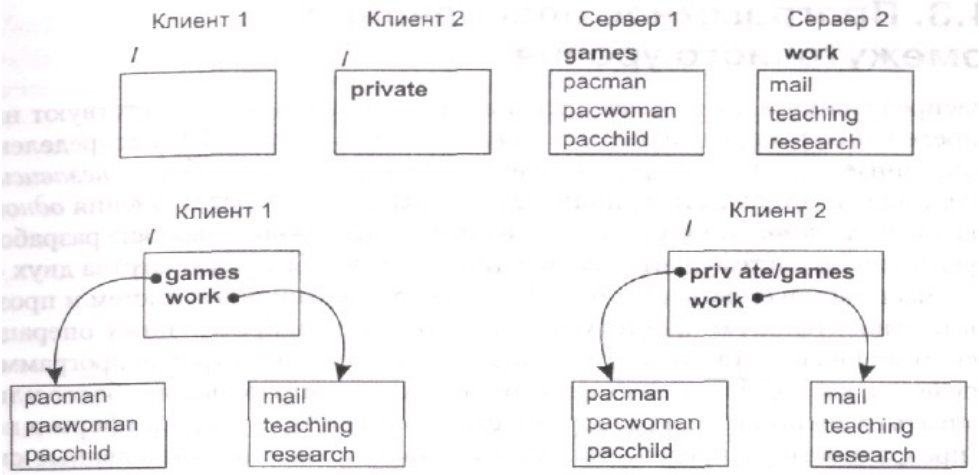


Рис. 7.7. Приклад варіантів незалежного монтування файлових систем клієнтів

Різні клієнти можуть монтувати файлові системи серверів по-різному
Мережеві операційні системи виглядають примітивніше розподілених. Основна різниця між цими двома типами операційних систем полягає в тому, що в розподілених операційних системах робиться серйозна спроба добитися повної

прозорості, тобто створити уявлення єдиної системи. «Брак» прозорості в мережевих операційних системах має деякі очевидні зворотні сторони. Наприклад, з ними часто складно працювати, оскільки користувач змушений явно під'єднуватися до віддалених машин або копіювати файли з однієї машини па іншу. Це викликає також проблеми з управлінням. Оскільки всі машини під управлінням мережевої операційної системи незалежні, часто і управляти ними можна виключно незалежно. В результаті користувач може отримати віддалене з'єднання з машиною X, тільки маючи на ній реєстрацію. Таким чином, якщо користувач хоче використовувати один пароль на «всі випадки життя», то для зміни пароля він змушений буде явно змінити його на кожній машині. В основному всі права доступу відносяться до конкретної машини. Немає простого методу змінити права доступу, оскільки скрізь вони свої. Такий децентралізований підхід до безпеки ускладнює захист мережевої операційної системи від атак зловмисників.
Є також і переваги в порівнянні з розподіленими операціями системами. Оскільки вузли мережевих операційних систем в значній мірі незалежні один від одного, додати або видалити машину дуже легко. У деяких випадках все, що треба зробити, щоб додати вузол, - це під'єднати відповідну машину до загальної мережі і довести до відома про її існування інші машини мережі. В Інтернеті, наприклад, додавання нового комп'ютера відбувається саме так. Щоб відомості про машину потрапили в Інтернет, необхідно дати їй мережеву адресу, а краще символічне ім'я, яке буде внесено в DNS разом з її мережевою адресою.

8 Програмне забезпечення проміжного рівня

Ні розподілені, і мережеві операційні системи не відповідають визначенню розподілених систем, яке дано у розділі 1. Розподілені операційні системи не призначені для управління набором незалежних комп'ютерів, а мережеві операційні системи не дають уявлення однієї узгодженої системи. Виникає питання: а чи можливо взагалі розробити розподілену систему, яка об'єднувала б у собі переваги двох «світів» - масштабованість і відкритість мережевих операційних систем, і прозорість і відносну простоту в використанні розподілених операційних систем? Рішення було знайдено у вигляді додаткового рівня програмного забезпечення, який в мережевих операційних системах дозволяє більш-менш приховати від користувача різнорідність набору апаратних платформ і підвищити прозорість розподілу. Багато сучасні розподілені системи побудовано з розрахунком на цей додатковий рівень, який отримав назву програмного забезпечення проміжного рівня.

8.1 Позиціонування програмного забезпечення проміжного рівня

Багато розподілених додатків допускають безпосереднє використання програмного інтерфейсу, запропонованого мережевими операційними системами. Так, зв'язок часто реалізується через операції з ***сокетами***, які дозволяють процесам на різних машинах обмінюватися повідомленнями. Крім того, додатки часто користуються інтерфейсами локальної файлової системи. Проблема такого підходу полягає в тому, що наявність розподілу занадто очевидно. Рішення полягає в тому, щоб помістити між додатком і мережевою операційною системою проміжний рівень програмної підтримки, що забезпечує додаткове абстрагування. Цей рівень

називається проміжним. Він знаходиться посередині між додатком і мережевою операційною системою, як показано на рис. 8.1.
Кожна локальна система, складова частина базової мережевої операційної системи, надає управління локальними ресурсами і найпростіші комунікаційні засоби для зв'язку з іншими комп'ютерами. Іншими словами, програмне забезпечення проміжного рівня не керує кожним вузлом, ця робота, як і раніше покладається на локальні операційні системи.
Основне наше завдання - приховати різноманітність базових платформ від додатків. Для вирішення цього завдання багато систем проміжного рівня надають більш-менш повні набори служб і «не дозволяють використовувати інші засоби для доступу до цих служб, крім своїх інтерфейсів. Іншими словами, обхід проміжного рівня і безпосередній виклик служб однієї з базових операційних систем вважається неприйнятним.
Після появи і широкого поширення мережевих операційних систем багато організацій виявили, що у них накопичилася маса мережевих додатків, які неможливо легко інтегрувати в єдину систему. Тоді виробники почали створювати незалежні від додатків служби верхнього рівня для цих систем. Типові приклади забезпечували підтримку розподіленої взаємодії і поліпшені комунікаційні можливості.
Була створена організація, покликана визначити загальний стандарт для рішень на базі проміжного рівня. На цей час існує безліч таких стандартів. Стандарти в основному несумісні один з одним, і продукти різних виробників, що реалізують один і той же стандарт, рідко здатні працювати разом. Ймовірно, що це ненадовго.

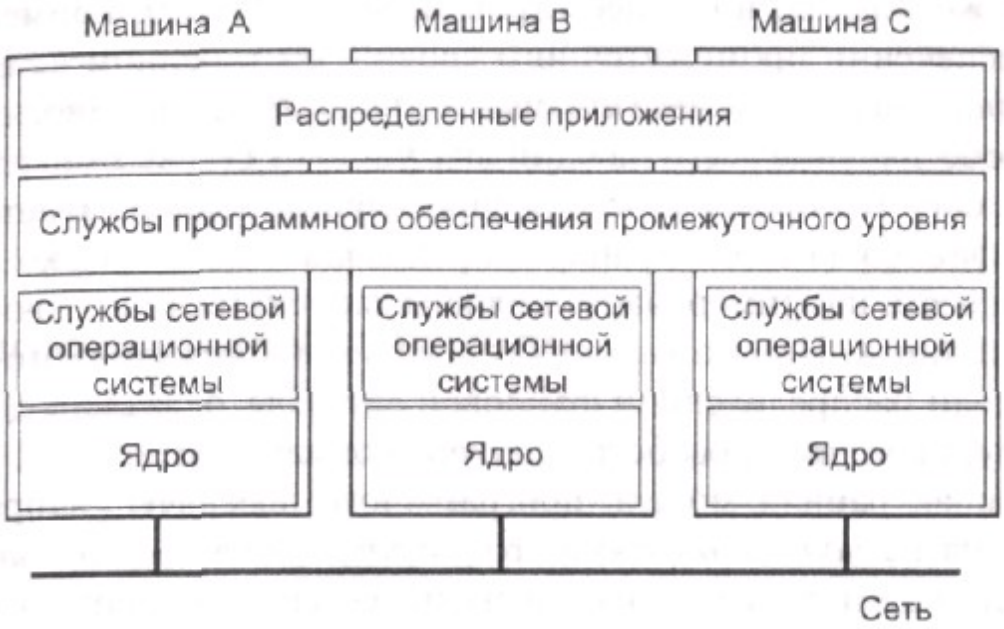


Рис. 8.1. Загальна структура розподілених систем з проміжним рівнем

8.2 Моделі проміжного рівня

Щоб зробити розробку і інтеграцію розподілених додатків якомога простішою, основна частина програмного забезпечення проміжного рівня базується на деякій моделі, або парадигмі, яка визначає розподіл і зв'язок. Щодо простою моделлю є представлення всіх спостережуваних об'єктів у вигляді файлів. Цей підхід був

спочатку введений в UNIX. Подібний же підхід застосовується в програмному забезпеченні проміжного рівня, побудованому за принципом ***розподіленої файлової системи*** (*distributed file system*). У багатьох випадках це програмне забезпечення недалеко пішло від мережевих операційних систем в тому сенсі, що прозорість розподілу можлива лише за допомогою стандартних файлів (тобто файлів, призначених тільки для Великих даних). Процеси часто повинні запускатися виключно на певних машинах. Програмне забезпечення проміжного рівня, засноване на моделі розподіленої файлової системи, виявилося досить легко масштабуємим, що сприяло його популярності.

Інша важлива рання модель програмного забезпечення проміжного рівня заснована на ***віддалених викликах процедур*** (*Remote Procedure Calls*, RPC). У цій моделі акцент робиться на приховуванні мережевого обміну за рахунок того, що процесу дозволяється здійснювати дзвінки для процедури, реалізація яких знаходиться на віддаленому сервері. При виклику такої процедури параметри прозоро передаються на віддалену машину, де, власне, і виконується процедура, після чого результат виконання повертається в точку виклику процедури. За винятком, мабуть, деякої втрати продуктивності, все це виглядає як локальне виконання викликаної процедури: викликає, не повідомляється про те, що відбувся факт мережевого обміну.

Якщо виклик процедури проходить через кордони окремих машин, він може бути представлений у вигляді прозорого звернення до об'єкту, що знаходиться на віддаленому сервері. Це призвело до появи різноманітних систем проміжного рівня, що реалізують уявлення про ***розподілені об'єкти*** (*distributed objects*). Ідея розподілених об'єктів полягає в тому, що кожен об'єкт реалізує інтерфейс, який приховує всі внутрішні деталі об'єкта від його користувача. Інтерфейс містить методи, реалізовані об'єктом. Процес бачить інтерфейс.

Розподілені об'єкти часто реалізуються шляхом розміщення об'єкта на одній з машин і відкриття доступу до його інтерфейсу з безлічі інших. Коли процес викликає метод, реалізація інтерфейсу на машині з процесом перетворює виклик методу в повідомлення, що пересилається об'єкту. Об'єкт виконує запитуваний метод і відправляє назад результати. Потім реалізація інтерфейсу перетворює відповідь, яку було надіслане у відповідний адрес, у значення, яке передається процесу, що викликав. Як і у випадку з RPC, процес може виявитися не поінформованим про ц обміни.

Моделі можуть спростити використання мережевих систем. Найкращим чином це видно на прикладі World Wide Web. Успіх середовища Web в основному залежить від того, що вона побудована на базі простої, але високоефективної ***моделі розподілених документів*** (*distributed documents*). У моделі, прийнятої в Web, інформація організована у вигляді документів, кожний з яких розміщений на машині, розташування якої абсолютно прозоро. Документи містять посилання, що зв'язують поточний документ з іншими. Якщо слідувати за посиланням, то документ, з яким пов'язане це посилання, буде вилучено з місця його зберігання та виведений на екран користувача. Концепція документа не обмежується виключно текстовою інформацією. У Web підтримуються аудіо- та відеодокументи, а також різні види документів на основі інтерактивної графіки.

8.3 Служби проміжного рівня

Існує кілька стандартних служб для систем проміжного рівня. Все програмне забезпечення проміжного рівня повинно реалізовувати прозорість доступу шляхом надання ***високорівневих засобів зв'язку*** (*communication facilities*), що приховують низкоуровневую пересилання повідомлень по комп'ютерній мережі. Інтерфейс програмування транспортного рівня, що надається мережевою операційною системою, повністю замінюється іншими засобами. Спосіб, яким підтримується зв'язок, в значній мірі залежить від моделі розподілу, запропонованої програмним забезпеченням проміжного рівня користувачам і додаткам. Так само як віддалений виклик процедур і звернення до розподілених об'єктів. Крім того, багато систем проміжного рівня надають засоби для прозорого доступу до віддалених даних, такі як розподілені файлові системи або розподілені бази даних. Прозора доставка документів, що реалізується в Web, - це ще один приклад комунікацій високого рівня (односпрямованих).

Важлива служба, загальна для всіх систем проміжного рівня, - це ***іменування*** (*naming*). Служби іменування можна порівняти з телефонними книгами або довідниками типу «Жовтих сторінок». Вони дозволяють спільно використовувати і шукати сутності (як в каталогах). Однак при масштабуванні виникають серйозні труднощі. Проблема полягає в тому, що для ефективного пошуку імені в великій системі розташування шуканої суті повинно вважатися фіксованим. Таке припущення прийнято в середовищі World Wide Web, в якій будь-який документ названий за допомогою URL. URL містить ім'я сервера, на якому знаходиться документ з даними URL-адресою. Таким чином, якщо документ переноситься на інший сервер, його URL змінюється.

Багато систем проміжного рівня надають спеціальні засоби зберігання даних, також іменовані ***засобами безпеки*** (*persistence*). В простій формі збереження забезпечується розподіленими файловими системами, але більш досконале програмне забезпечення проміжного рівня містить інтегровані бази даних або надає засоби для зв'язку додатків з базами даних.

Якщо зберігання даних відіграє важливу роль для оболонки, то зазвичай надаються засоби для ***розподілених транзакцій*** (*distributed transactions*). Застосування транзакцій дає можливість застосування безлічі операцій читання і запису в ході однієї атомарної операції. Під атомарною ми розуміємо те, що транзакція може бути або успішною (коли всі операції запису завершуються успішно), або невдалої, що залишає всі задіяні дані без змін. Розподілені транзакції працюють з даними, які, можливо, розкидані по декількох машинах.

Надання таких служб, як розподілені транзакції, особливо важливо, оскільки маскування збоїв для розподілених систем нерідко утруднена. На жаль, транзакції легше масштабувати на декількох географічно віддалених машинах, ніж на безлічі локальних.

Практично всі системи проміжного рівня надають засоби забезпечення ***захисту*** (*security*). У порівнянні з мережевими операційними системами проблема захисту в системах проміжного рівня полягає в тому, що вони розподілені. Проміжний рівень в принципі не може «сподіватися» на те, що базові локальні операційні системи будуть адекватно забезпечувати захист всієї мережі. Відповідно, захист частково лягає на програмне забезпечення проміжного рівня. У поєднанні з вимогою

розширюваності захист перетворюється в одну з найбільш важко реалізованих в розподілених системах служб.

8.4 Проміжний рівень і відкритість

Сучасні розподілені системи зазвичай створюються у вигляді систем проміжного рівня для декількох платформ. При цьому додатки створюються для конкретної розподіленої системи і не залежать від платформи (операційної системи). На жаль, ця незалежність часто замінюється жорсткою залежністю від конкретної системи проміжного рівня. Проблема полягає в тому, що системи проміжного рівня часто значно менше відкриті, ніж потрібно.

Відкрита розподілена система визначається повнотою (завершеністю) її інтерфейсу. Повнота означає реальну наявність всіх необхідних для створення систем описів. Неповнота опису інтерфейсу призводить до того, що розробники систем змушені додавати свої власні інтерфейси. Різними командами розробників відповідно до одних і тих же стандартів створюються різні системи проміжного рівня і додатки, написані під одну з систем, не можуть бути безпосередньо перенесені під іншу без значних зусиль.

Неповнота призводить до неможливості спільної роботи двох реалізацій, незважаючи на те, що вони підтримують абсолютно однаковий набір інтерфейсів, але різні базові протоколи. Так, якщо дві реалізації засновані на несумісних комунікаційних протоколах, підтримуваних мережевою операційною системою, нелегко домогтися їхньої спільної роботи. Необхідно, щоб і протоколи проміжного рівня, і його інтерфейси були однакові, як показано на рис. 8.2.

Для гарантії спільної роботи різних реалізацій необхідно, щоб до сутностей різних систем можна було однаково звертатися. Якщо до сутностей в одній системі звернення йде через URL, а в іншій системі - через мережеву адресу, перехресні звернення приведуть до проблем. У подібних випадках визначення інтерфейсів повинні точно задавати вид посилань.

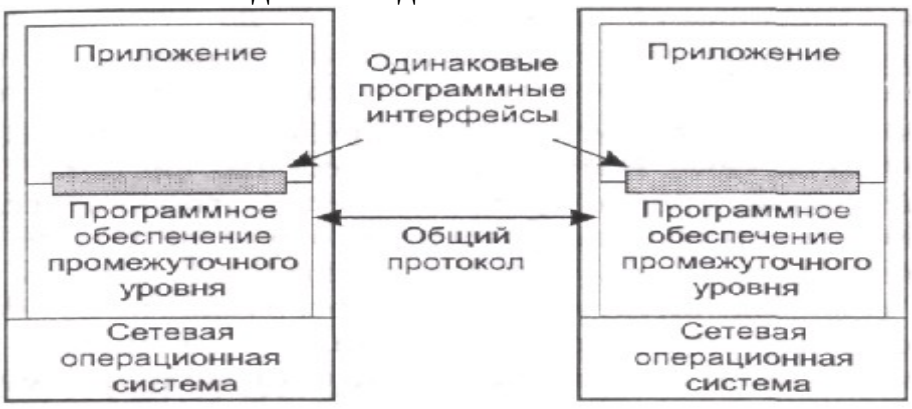


Рис. 8.2. Структура комунікаційних протоколів двох різномовних додатків

У відкритих розподілених системах повинні бути однаковими як протоколи, що використовуються проміжними рівнями кожної з систем, так і інтерфейси, що надаються додаткам.

8.5 3.4 Порівняння систем

Коротке порівняння розподілених операційних систем, мережевих операційних систем і розподілених систем проміжного рівня наведено в табл. 8.1.

Таблиця 8.1. Порівняння розподілених операційних систем, мережевих операційних систем і розподілених систем проміжного рівня

Характеристика	**Розподілена операційна система**		**Мережева операційна система**	**Розподілена система проміжного рівня**
	Мульти-процесорна	мульти-комп'ютерна		
Ступінь прозорості	Дуже висока	Висока	Висока	Низька
Ідентичність операційної системи на всіх вузлах	Підтримується	Підтримується	Не підтримується	Не підтримується
Число копій ОС	1	N	N	N
Комунікаційна на основі	Пам'яті, яка спільно використовується	Повідомлень	Залежно від моделі	Файлів
Управління ресурсами	Глобальне, централізоване	Глобальне, розподілене	Окремо на вузлі	Окремо на вузлі
Масштабованість	Відсутня	Помірна	Різна	Так
Відкритість	Закрита	Закрита	Відкрита	Відкрита

Розподілені операційні системи працюють краще, ніж мережеві. У мультипроцесорних системах потрібно приховувати тільки загальне число процесорів. Складніше приховати фізичний розподіл пам'яті, тому не просто створити мультикомпьютерную операційну систему з повністю прозорим розподілом. Розподілені системи часто підвищують прозорість шляхом використання спеціальних моделей розподілу і зв'язку. Розподілені файлові системи добре приховують місце розташування файлів і доступ до них. Однак вони втрачають в спільності, і користувачі можуть отримати проблеми з деякими додатками.

Розподілені операційні системи гомогенні, тобто кожен вузол має власну операційну систему (ядро). У мультипроцесорних системах немає необхідності копіювати дані - таблиці та ін., оскільки всі вони знаходяться в загальній пам'яті і можуть використовуватися спільно. У цьому випадку весь язв'язок також здійснюється через загальну пам'ять, в той час як в мультикомпьютерних системах потрібні повідомлення. У мережевих операційних системах зв'язок найчастіше базується на файлах. Наприклад, в Інтернеті велика частина обміну здійснюється шляхом передачі файлів. Крім того, проте, інтенсивно використовується обмін

повідомленнями високого рівня у вигляді систем електронної пошти і дощок оголошень. Зв'язок в розподілених системах проміжного рівня залежить від моделі, на якій заснована система.
Ресурси в мережевих операційних системах і розподілених системах проміжного рівня управляються на кожному вузлі, що робить масштабування цих систем відносно простим. Однак практика показує, що реалізація в розподілених системах програмного забезпечення проміжного рівня часто призводить до обмеженості масштабування. Розподілені операційні системи здійснюють глобальне управління ресурсами, що ускладнює їх масштабування. У зв'язку з централізованим підходом (коли всі дані знаходяться в загальній пам'яті) в мультипроцесорних системах вони погано масштабуються.
Мережеві операційні системи та розподілені системи проміжного рівня виграють у відкритості. В основному вузли підтримують стандартний комунікаційний протокол типу TCP / IP, що спрощує організацію їх спільної роботи. Однак можуть зустрітися труднощі з перенесенням додатків під різні платформи. Розподілені операційні системи в основному розраховані не на відкритість, а на максимальну продуктивність.

9 Модель клієнт-сервер

9.1 Клієнти і сервери

У базовій моделі клієнт-сервер всі процеси в розподілених системах діляться на дві групи, які можливо перекриваються. Процеси, що реалізують такі служби, як службу файлової системи або бази даних, називаються ***серверами*** (*servers*). Процеси, що запитують служби у серверів шляхом посилки запиту і подальшого очікування відповіді від сервера, називаються ***клієнтами*** (*clients*). Взаємодія клієнта і сервера, відома також під назвою режим роботи ***запит-відповідь*** (*request-reply behavior*), ілюструє рис. 9.1.

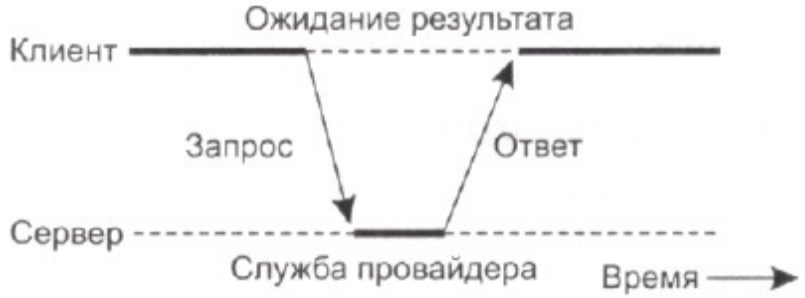


Рис. 9.1. Узагальнена взаємодія між клієнтом і сервером

Якщо базова мережа надійна, як локальні мережі, взаємодію між клієнтом і сервером може бути реалізовано за допомогою простого протоколу, що не потребує встановлення з'єднання. У цьому випадку клієнт, запитуючи сервер, визначає свій запит в формі повідомлення із зазначенням в ньому служби, якою він бажає скористатися, і необхідних для цього вихідних даних. Повідомлення надсилається серверу. Останній, в свою чергу, постійно очікує вихідного повідомлення, отримавши його, обробляє, упаковує результат обробки у відповідь для повідомлення, що було надіслане, і відправляє його клієнту. Використання не потребує з'єднання для протоколу, що дає суттєвий виграш ефективності. До тих пір поки повідомлення не почнуть пропадати або спотворюватися, можна цілком

успішно застосовувати протокол типу запит-відповідь. Створити протокол, який стійкий до випадкових збоїв зв'язку, - нетривіальне завдання. Можна дати клієнту можливість повторно надіслати запит, на який не було отримано відповіді. Проблема, однак, полягає в тому, що клієнт не може визначити, чи дійсно початкове повідомлення із запитом було втрачено або помилка сталася під час передачі відповіді. Якщо загубилася відповідь, повторна посилка запиту може привести до повторного виконання операції.
В якості альтернативи в багатьох системах клієнт-сервер використовується надійний протокол з установленням з'єднання. Хоча це рішення в зв'язку з його відносно низькою продуктивністю не підходить для локальних мереж, воно використовується в глобальних системах, для яких ненадійність є властивістю більшості з’єднань. Практично прикладні протоколи Інтернету засновані на надійних з'єднаннях по протоколу TCP / IP. У цих випадках щоразу, коли клієнт запитує службу, до посилки запиту серверу він повинен встановити з ним з'єднання. Сервер зазвичай використовує для посилки листа у відповідь те ж саме з'єднання, після чого воно розривається. Проблема полягає в тому, що установка повторного сеансу може призвести до втрати з'єднання в сенсі щодо витрачення часу і ресурсів, особливо якщо повідомлення із запитом і відповіддю невеликі.

9.2 4.2 Поділ додатків за рівнями

Одне з головних питань полягає в тому, як розділити клієнт і сервер. Зазвичай чіткої відмінності немає. Наприклад, сервер розподіленої бази даних може постійно виступати клієнтом, передає запити на різні файлові сервери, які відповідають за реалізацію таблиць цієї бази даних. У цьому випадку сервер баз даних сам по собі не робить нічого, крім обробки запитів.
Однак, розглядаючи безліч додатків типу клієнт-сервер, призначених для організації доступу користувачів до баз даних, рекомендується розділяти їх на три рівні:

– рівень інтерфейсу користувача;
– рівень обробки;
– рівень даних.

Рівень інтерфейсу користувача містить все необхідне для безпосереднього спілкування з користувачем, наприклад управління дисплеєм. Рівень обробки зазвичай містить додатки, а рівень даних - власне дані, з якими відбувається обробка.

4.2.1 Рівень призначеного для користувача інтерфейсу

Рівень інтерфейсу користувача зазвичай реалізується на клієнтах. Цей рівень містить програми, за допомогою яких користувач може взаємодіяти з додатком. Складність програм, що входять в призначений для користувача інтерфейс, дуже різна.
Найпростіший варіант програми для користувача інтерфейсу не містить нічого, крім символьного (НЕ графічного) дисплея. Такі інтерфейси зазвичай використовуються при роботі з мейнфреймами. У тому випадку, коли мейнфрейм контролює інтенсивність взаємодії, включаючи роботу з клавіатурою і монітором, ми навряд чи можемо говорити про модель клієнт-сервер. Однак у багатьох випадках термінали користувачів виконують деяку локальну обробку, здійснюючи,

наприклад, сервіс «луна-друк», коли вводяться рядкі тексту або надаючи інтерфейс форм, в якому можна відредагувати введені дані до їх пересилання на головний комп'ютер.
В даний час навіть в середовищі мейнфреймів спостерігаються більш досконалі інтерфейси, призначені для користувача. Зазвичай на клієнтських машинах є як мінімум графічний дисплей, на якому можна задіяти спливаючі меню або випадаючи списки і безліч керуючих елементів, доступних для миші або клавіатури. Типові приклади таких інтерфейсів - надбудова X-Windows, яка використовується в багатьох UNIX-системах, і більш ранні інтерфейси, розроблені для персональних комп'ютерів, що працюють під управлінням MS-DOS і Apple Macintosh.
Сучасні інтерфейси більш функціональні. Вони підтримують спільну роботу додатків через єдине графічне вікно і в ході дій користувача забезпечують через це вікно обмін даними. Наприклад, для видалення файлу часто досить перенести значок, відповідний щодо цього файлу, на значок сміттєвого кошика. Аналогічним чином багатотекстові процесори дозволяють користувачеві переміщувати текст документа в інше місце, користуючись тільки мишею.

4.2.2 Рівень обробки
Багато додатків моделі клієнт-сервер побудовані з трьох різних частин: частини, яка займається взаємодією з користувачем, частини, яка відповідає за роботу з базою даних або файлової системою, і середньої частини, що реалізує основну функціональність програми. Ця середня частина логічно розташовується на рівні обробки. В протилежність призначених для користувача інтерфейсів або баз даних, на рівні обробки важко виділити загальні закономірності. Однак розглянемо кілька прикладів для роз'яснення питань, пов'язаних з цим рівнем.

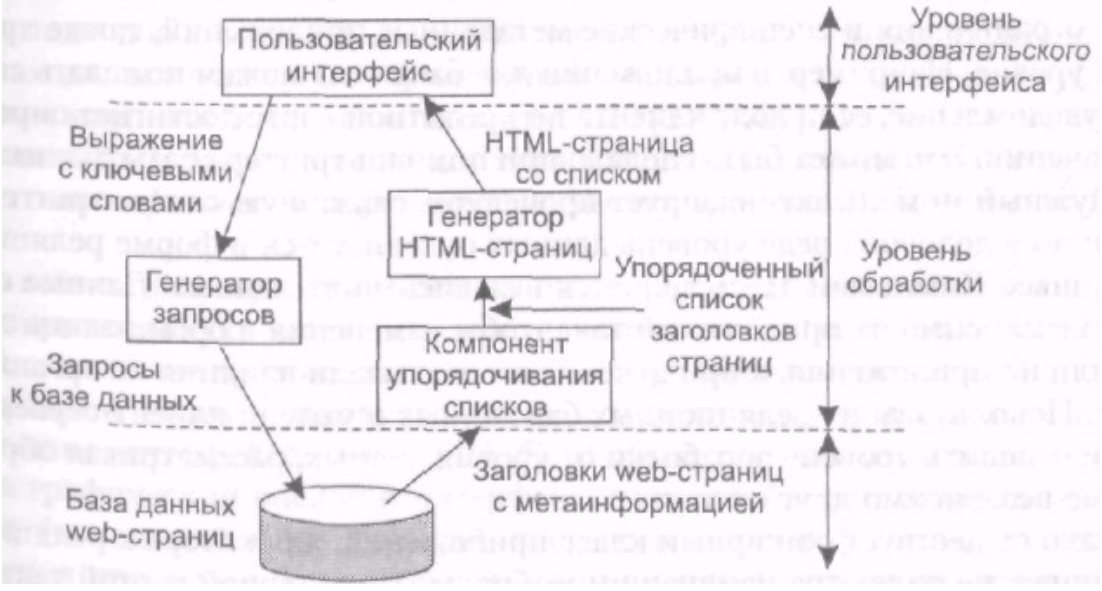


Рис.9.2 Узагальнена організація трирівневої пошукової машини для Інтернету

Як перший приклад розглянемо пошукову машину в Інтернеті. Якщо відкинути всі віконні прикраси, призначений для користувача інтерфейс пошукової машини дуже простий: користувач вводить рядок, що складається з ключових слів, і отримує список заголовків web-сторінок. Результат формується з великої бази переглянутих і проіндексованих web-сторінок. Ядром пошукової машини є програма, яка трансформує введений користувачем рядок в один або кілька запитів до бази

даних. Потім вона поміщає результати запиту в список і перетворює цей список в набір HTML-сторінок. В рамках моделі клієнт-сервер частина, яка відповідає за вибірку інформації, зазвичай знаходиться на рівні обробки. Ця структура показана на рис. 9.2.

Як другий приклад розглянемо систему підтримки прийняття рішень для фондового ринку. Так само як і в пошуковій машині, цю систему можна розділити на зовнішній інтерфейс, який реалізує роботу з користувачем, внутрішню частину, що відповідає за доступ до бази з фінансовою інформацією, і проміжну програму аналізу. Аналіз фінансових даних може зажадати хитромудрих методик і технологій на основі статистичних методів і штучного інтелекту. У деяких випадках для того, щоб забезпечити необхідні продуктивність і час відгуку, ядро системи підтримки фінансових рішень має виконуватися на високопродуктивних комп'ютерах.

Останнім прикладом буде типовий офісний пакет, що складається з текстового процесора, додатків для роботи з електронними таблицями, комунікаційних утиліт і т. п. Подібні офісні пакети зазвичай підтримують узагальнений призначений для користувача інтерфейс, можливість створення складних документів і роботу з файлами в домашньому каталозі користувача. У цьому випадку рівень обробки включатиме в себе відносно великий набір програм, кожна з яких покликана підтримувати якусь із функцій обробки.

4.2.3 Рівень даних

Рівень даних в моделі клієнт-сервер містить програми, які надають дані для оброблюючих їх додатків. Специфічною властивістю цього рівня є вимога ***збереження*** (*persistence*). Це означає, що коли програма не працює, дані повинні зберігатися в певному місці в розрахунку на подальше використання. У найпростішому варіанті рівень даних реалізується файловою системою, але частіше для його реалізації задіюється повномасштабна база даних. У моделі клієнт-сервер рівень даних зазвичай знаходиться на стороні сервера.

Крім простого зберігання інформації рівень даних зазвичай відповідає за підтримку цілісності даних для різних додатків. Для бази даних підтримка цілісності означає, що метадані, такі як опис таблиць, обмеження і специфічні метадані додатків, також зберігаються на цьому рівні. Наприклад, в додатку для банку ми можемо побажати сформувати функцію повідомляти про спроби погасити борг клієнта за допомогою кредитної картки, коли він досягне певного значення. Це може бути зроблено за допомогою тригера бази даних, який в потрібний момент активізує процедуру, пов'язану з цим тригером.

Зазвичай в діловому середовищі рівень даних організується у формі реляційної бази даних. Ключовим тут є незалежність даних. Дані організуються незалежно від додатків так, щоб зміни в організації даних не впливали на додатки, а додатки не чинили впливу на організацію даних. Використання реляційних баз даних в моделі клієнт-сервер допомагає відокремити рівень обробки від рівня даних, розглядаючи обробку і дані незалежно один від одного.

Однак існує великий клас програм, для яких реляційні бази даних не є найкращим вибором. Характерною рисою таких додатків є робота зі складними типами даних, для яких простіше моделі в поняттях об'єктів, а не відносин. Приклади таких типів даних - від простих наборів прямокутників і кіл до проекту літака в разі його автоматизованого проектування. Також і для мультимедійних систем значно

простіше працювати з відео та звуком, використовуючи специфічні для них операції, ніж з моделями цих потоків у вигляді реляційних таблиць.
У тих випадках, коли операції з даними значно простіше висловити в поняттях роботи з об'єктами, має сенс реалізувати рівень даних засобами об'єктно-орієнтованих баз даних. Подібні бази даних не тільки підтримують організацію складних даних в формі об'єктів, але і зберігають реалізації операцій над цими об'єктами. Таким чином, частина функціональності, яка припадала на рівень обробки, мігрує в цьому випадку на рівень даних.

9.3 Варіанти архітектури клієнт-сервер

Найпростіша організація припускає наявність всього двох типів машин:

- Клієнтські машини, на яких є програми, що реалізують тільки інтерфейс користувача або його частину.
- Сервери, що реалізують все інше, тобто рівні обробки даних.

Проблема подібної організації полягає в тому, що насправді система не є розподіленою: все відбувається на сервері, а клієнт являє собою функціонально простий термінал.

Багатоланкові архітектури
Один з підходів до організації клієнтів і серверів - це розподіл програм, які перебувають на рівні додатків. В якості першого кроку розглянемо поділ на два типи машин: на клієнти і на сервери, що призведе до фізично ***<u>двохланкової архітектури</u>*** (*physically two-tiered architecture*). Один з можливих варіантів організації - помістити на клієнтську сторону термінальну частину призначеного для користувача інтерфейсу, як показано на (рис.9.3а), дозволивши з додатком віддалено контролювати представлення даних. Альтернативою цьому підходу буде передача клієнту всієї роботи з призначеним для користувача інтерфейсом (рис.9.3б). В обох випадках відділений від програми графічний зовнішній інтерфейс, є пов'язаний з іншою частиною програми (що знаходиться на сервері) за допомогою специфічного для цього додатка протоколу. У подібній моделі зовнішній інтерфейс робить тільки те, що необхідно для надання інтерфейсу відповідному додатку.

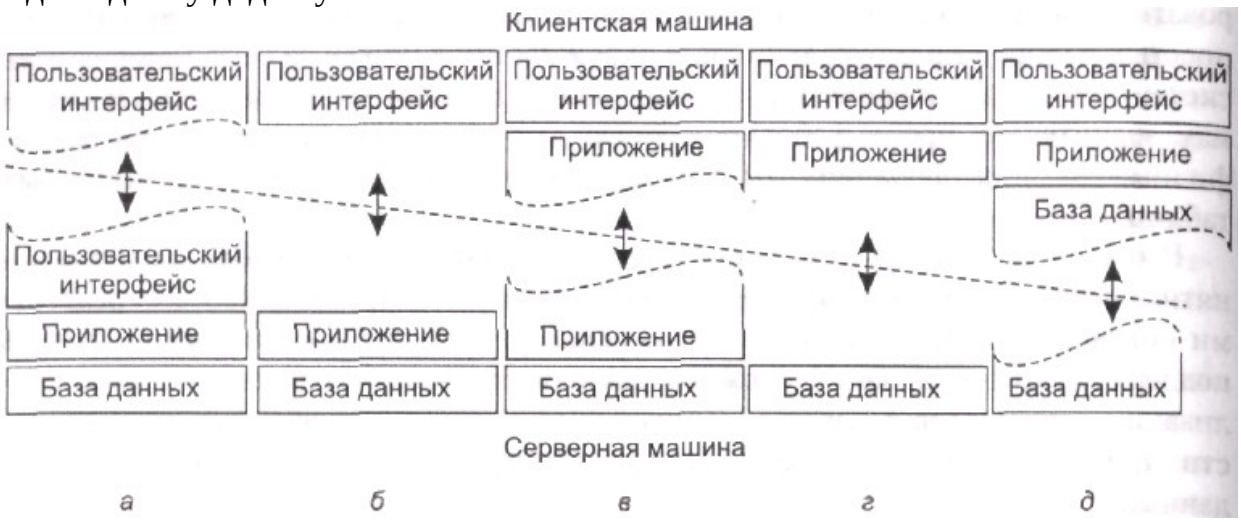


Рис. 9.3 Альтернативні форми організації архітектури клієнт-сервер

Можна перенести у зовнішній інтерфейс частину програми, як показано на рис. 9.3в. Прикладом може бути варіант, коли додаток створює форму безпосередньо перед її заповненням. Зовнішній інтерфейс перевіряє правильність і повноту заповнення форми і при необхідності взаємодіє з користувачем. Іншим прикладом організації системи за зразком, поданим на рис. 9.3в, може служити текстовий процесор, в якому базові функції редагування здійснюються на стороні клієнта з локально кешованими даними або знаходяться з даними в пам'яті, а спеціальна обробка, така як перевірка орфографії або граматики, виконується на стороні сервера.

У багатьох системах клієнт-сервер популярна організація, що представлена на рис. 9.3г і 9.3д. Ці типи організації застосовуються і тому випадку, коли клієнтська машина - персональний комп'ютер або робоча станція з'єднана мережею з розподіленою файловою системою або базою даних. Велика частина програми працює на клієнтській машині, а всі операції, з файлами або базою даних передаються на сервер. Малюнок 9.3д відображає ситуацію, коли частина даних міститься на локальному диску клієнта. Так, наприклад, при роботі в Інтернеті клієнт може поступово створити на локальному диску великий кеш найбільш часто відвідуваних web-сторінок.
Іноді може знадобитися щоб сервер міг працювати в якості клієнта. Така ситуація, яка відображена на рис. 9.4, призводить до фізично ***трирівневої архітектури*** (*physically three-tiered architecture*).
У подібній архітектурі програми, які є частиною рівня обробки, виносяться на окремий сервер, але додатково можуть частково знаходитися і на машинах клієнтів і серверів. Типовий приклад трирівневої архітектури - обробка транзакцій. В цьому випадку окремий процес - монітор транзакцій - координує всі транзакції, можливо, на декількох серверах даних.

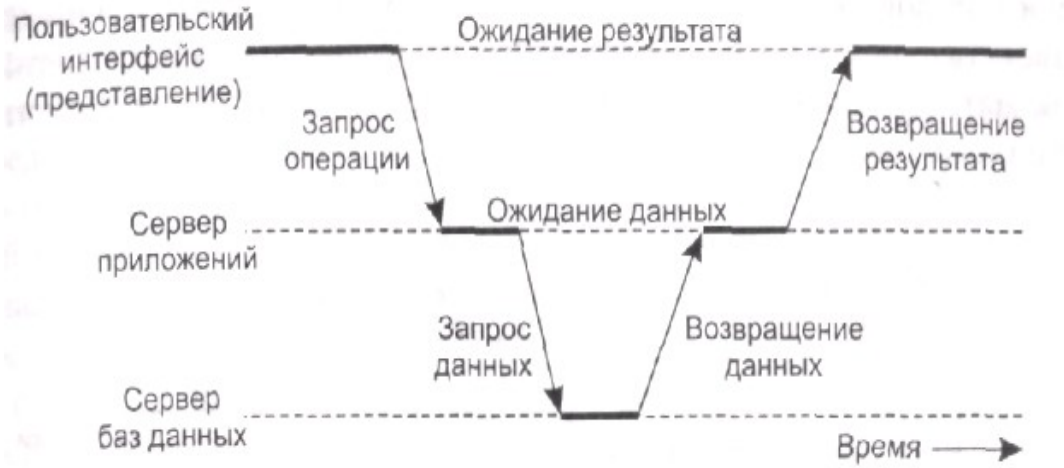


Рис. 9.4. Приклад сервера, що діє як клієнт

4.3.2 Сучасні варіанти архітектури
Багаторівневі архітектури клієнт-сервер є прямим продовженням поділу додатків на рівні користувача інтерфейсу, компонентів обробки і даних. Різні рівні взаємодіють відповідно до логічної організації додатків. У безлічі бізнес-додатків розподілена обробка еквівалентна організації багаторівневої архітектури додатків клієнт-сервер. Ми будемо називати такий тип розподілу - ***вертикальним розподілом*** (*vertical distribution*). Характеристичною особливістю вертикального

розподілу є те, що він досягається розміщенням логічно різних компонентів на різних машинах. Це поняття пов'язане з концепцією ***вертикального розбиття*** (*vertical fragmentation*), яка використовується в розподілених реляційних базах даних, де під цим терміном розуміється розбиття по стовпцях таблиць для їх зберігання на різних машинах.

Однак вертикальний розподіл - це лише один з можливих способів організації додатків клієнт-сервер. У сучасних архітектурах розподіл на клієнти і сервери відбувається способом, відомим як ***горизонтальний розподіл*** (*horizontal distribution*). При такому типі розподілу клієнт або сервер може містити фізично розділені частини логічно однорідного модуля, причому робота з кожною з частин може відбуватися незалежно. Це робиться для вирівнювання завантаження.

Як варіант поширеного прикладу горизонтального розподілу розглянемо web-сервер, який реплікується на декілька машин локальної мережі, як показано на рис. 9.5. На кожному з серверів міститься один і той же набір web-сторінок, і всякий раз, коли одна з web-сторінок оновлюється, її копії негайно розсилаються на всі сервери. Сервер, для якого буде переданий вихідний запит, вибирається за ***правилом «каруселі***» (*round-robin*). Ця форма горизонтального розподілу вельми успішно використовується для вирівнювання навантаження на сервери популярних web-сайтів.

Таким же чином можуть бути розподілені і клієнтські частини для нескладного додатку, призначеного для колективної роботи, коли можна не мати сервера взагалі. У цьому випадку говорять про обраний ***тимчасовий розподіл*** (*peer-to-peer distribution*). Це відбувається, наприклад, якщо користувач хоче зв'язатися з іншим користувачем. Обидва вони повинні запустити один і той же додаток, щоб почати сеанс. Третій клієнт може спілкуватися з одним з них або обома, для чого йому потрібно запустити той самий додаток.

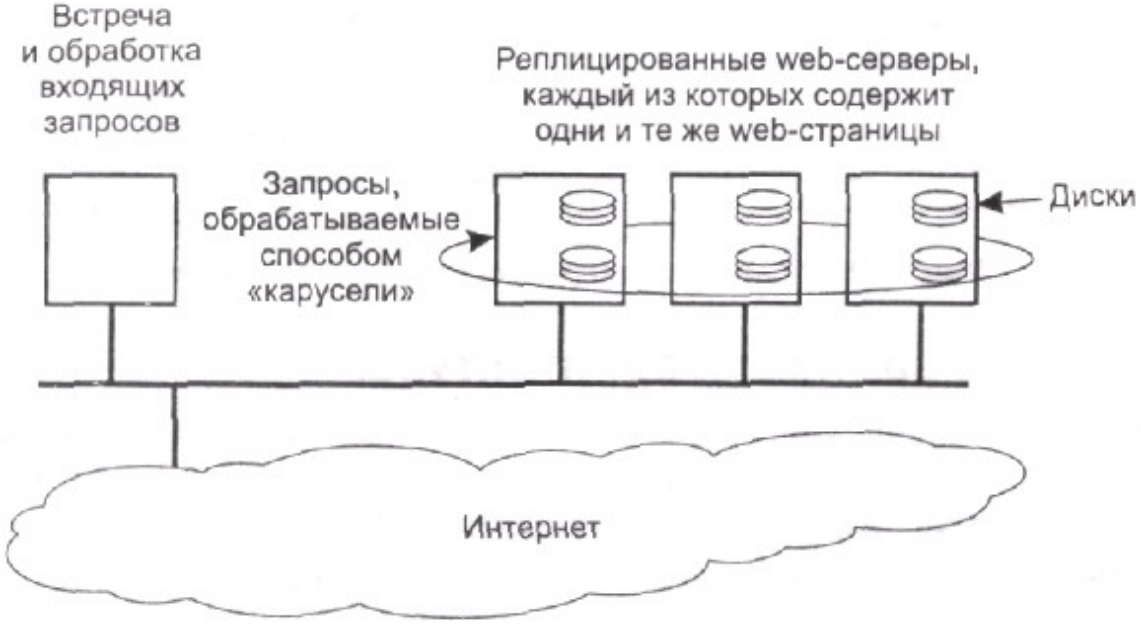


Рис. 9.5. Приклад горизонтального розподілу web-служби

10 Моделі обчислень для системи клієнт сервер

10.1 Історичний розвиток обчислень в автоматизованих системах

Основні визначення

Сервер - будь-яка система, процес, комп'ютер, які володіють довільним обчислювальним ресурсом (пам'яттю, часом тощо).

Клієнт - будь-яка система, процес, комп'ютер, користувач, які запитують у сервера який-небудь ресурс, які користуються будь-яким ресурсом або обслуговуються сервером іншим способом.

Клієнт-сервер - це вид розподіленої системи, в якій є сервер, що виконує запити клієнта, причому сервер і клієнт спілкуються між собою з використанням того або іншого протоколу.

Варіанти побудови інформаційних додатків і етапи комп'ютеризації управління в промисловості

Перші автоматизовані системи організаційного управління з'явилися в промисловості ще в 70-ті роки. Комп'ютеризація промислових підприємств у нас пройшла величезну дорогу перш, ніж з'явилася архітектура клієнт-сервер. Апаратною базою найперших АСУП були лампові та напівпровідникові РС «Урал», «Мінськ», БЕСМ. У 80-ті роки апаратною основою систем управління стали РС серії ЄС, міні-РС VAX і СМ. У 90-ті роки основою комп'ютеризації управління стали персональні РС та їх локальні мережі [2].

Кожному етапу розвитку автоматизованих систем управління підприємством відповідали своя панівна архітектура додатків, свої технології. Першою з'явилася централізована архітектура, архітектура локальних РС, а потім і архітектура файл-сервер. Всі варіанти побудови інформаційних додатків в тій чи іншій мірі до цих пір активно використовуються на практиці. Оскільки всі ці технології не минуле, а сьогодення комп'ютеризації в промисловості, перш ніж перейти до архітектури клієнт-сервер, давайте розглянемо їх трохи докладніше.

Централізована архітектура додатків

Історично першими з'явилися комп'ютерні системи з централізованою архітектурою додатків. При використанні цієї архітектури все програмне забезпечення автоматизованої системи виконується централізовано на одному комп'ютері, який виконує одночасно багато завдань і підтримуючому велику кількість користувачів. На цьому комп'ютері повністю здійснюється процес вводу/виводу інформації, а також її прикладна обробка. В якості центрального комп'ютера для такої системи може застосовуватися або велика РС (названа також мейнфреймом) або так звана міні-РС. До подібного комплексу підключаються периферійні пристрої для введення / виведення інформації від кожного користувача.

Спочатку єдиним способом спілкування користувача з подібною технікою був так званий пакетний режим - введення інформації здійснювалося, наприклад, за допомогою пачки перфокарт. Обробивши отримане завдання, машина видавала роздруківку результатів роботи.

У другій половині 70-х з'явилися пристрої для діалогової взаємодії з комп'ютером - термінали. Однак, вони не виконували жодних функцій, крім виведення символів, отриманих від комп'ютера на дисплей і передачі введених з клавіатури символів в комп'ютер. Тому подібний пристрій отримав назву "dumb terminal" - дослівно «тупий термінал».

Найчастіше у нас в подібних системах на підприємствах (на Заході це зустрічається помітно рідше) взаємодію між співробітниками підприємства і РС здійснювалося не безпосередньо, а через операторів РС, які отримували вихідну інформацію в паперовому вигляді, заносили її в комп'ютер, а потім передавали користувачам результати роботи програми у вигляді роздруківок. Природно, що подібний цикл обробки інформації вимагав численного обслуговуючого персоналу, займав тривалий час, породжував велику кількість помилок і був дуже негнучким і досить дорогим. Крім того, робота в централізованій архітектурі вимагала спеціалізованих знань. А це створювало додатковий бар'єр між управлінцем і системою автоматизації і ускладнювало процес її адаптації до потреб користувачів.

Тому за допомогою подібних систем було доцільно автоматизувати тільки окремі процеси в управлінні підприємством з дуже високою трудомісткістю ручної обробки інформації. До них відносилися, як правило, завдання бухгалтерії, не занадто вимогливі до оперативності обробки, з дуже великим обсягом рутинних обчислень і підготовки вихідних форм. Сюди могли входити розрахунок заробітної плати, бухгалтерський облік матеріальних цінностей, відвантаженої продукції. Користувачі могли також здійснювати введення інформації та отримання результатів обробки самостійно, з використанням алфавітно-цифрових терміналів. У такому варіанті централізовані системи існують і успішно функціонують досі.

У великих зарубіжних компаніях на мейнфреймах досі ведеться значна частина завдань, пов'язаних з комп'ютеризацією управління, насамперед облікових. В СРСР же подібні системи ніколи не були широко поширені через малу кількість і низьку надійність вітчизняних великих РС. Успішно працюючих донині вітчизняних систем з централізованою архітектурою взагалі одиниці. Найбільша з них, це всім відома глобальна система «Експрес» для продажу залізничних квитків через каси.

Є дві головні причини того що системи на великих РС експлуатуються досі, незважаючи на великі витрати на спеціалізовану апаратуру та інформаційні канали для підтримки надійного зв'язку між кожним терміналом і комп'ютером.

По-перше, централізована архітектура при необхідній якості апаратури і програмного забезпечення дуже надійна (особливо в плані цілісності даних). Абсолютна продуктивність мейнфреймів не поступається найбільшим комплексам в архітектурі клієнт-сервер. Велика РС достатньої потужності може забезпечувати стійку роботу величезної кількості користувачів одночасно. Працездатність подібних систем перевірена багаторічним досвідом роботи. Той же «Експрес»

одночасно обслуговує тисячі користувачів, розташованих по всій Україні. Доведення будь-якої нової системи до такого ж рівня зажадає не один рік.

По-друге, витрати на заміну подібної системи, включаючи прямі витрати - на заміну апаратури і програмного забезпечення і непрямі витрати - на перепідготовку користувачів, розбудову інфраструктури і зміну технології обробки інформації складають суму, величезну навіть для великої західної компанії. Оригінальний текст таких систем складає сотні тисяч і навіть мільйони рядків, причому ці рядки писалися протягом десятиліть.

Для розробки додатків в подібних системах часто використовується мова COBOL (COmmon Business Oriented Language, Мова загального призначення, орієнтована на бізнес). Ця мова була розроблена за замовленням Пентагону ще на початку 60-их, і була спочатку призначена для вирішення інформаційних завдань, пов'язаних з комерційною діяльністю. Протягом двох десятків років на Заході (насамперед, у США) вона була основним інструментом розробки систем управління виробництвом. На неї було розроблено безліч інформаційних систем для великих компаній, які експлуатуються і понині. Тому в Америці програмісти на COBOL'е - досі одні з найбільш дефіцитних і високооплачуваних фахівців з інформаційних технологій.

Справедливості заради, варто відмітити, що використання великих PC саме по собі ще не обов'язково означає використання застарілого програмного забезпечення для розробки. З одного боку, багато систем, що використовуються в додатках клієнт-сервер, існують і для централізованих платформ. Наприклад, найвідоміша і поширена у нас система керування базами даних масштабу підприємства - Oracle спочатку з'явилася для міні-PC і мейнфреймів. Програмне забезпечення для цих платформ досі займають дуже значну частку серед загального числа експлуатуємих систем компанії Oracle.

З іншого боку, у міру поширення архітектури клієнт-сервер з'явилися можливості для її інтеграції з централізованою архітектурою. Зараз більшість фірм, що випускають великі PC, стали пропонувати рішення, які дозволяють використовувати їх в якості серверів в архітектурі клієнт-сервер. Подібні гетерогенні (різнорідні) комплекси дозволяють, по-перше, спільно експлуатувати централізовані програми та системи клієнт-сервер, а по-друге, поєднують переваги розподіленої архітектури з надійністю мейнфреймів.

Персональні PC - локальні завдання

Наступним значним етапом розвитку комп'ютеризації організаційного управління стали персональні PC. З'явившись на Заході на початку, а у нас в кінці 80-их, цей інструмент привів до революційного розширення сфери комп'ютеризації управління. Комп'ютер з дорогого і складного в експлуатації пристрою, що застосовується в промисловості вкрай рідко, для вирішення окремих завдань, перетворився на масовий інструмент для автоматизації повсякденних завдань конторської діяльності.

За своїм визначенням, персональний комп'ютер призначений для автоматизації рішення завдань конкретного співробітника. При цьому між працівником та інструментом автоматизації не має ніяких посередників, а це сильно полегшує використання комп'ютера в поточній роботі. Користувацький інтерфейс (спосіб взаємодії з користувачем) для PC набагато зручніше, багатший і, що найголовніше, простіше в засвоєнні, ніж інтерфейс будь-якої централізованої системи.

Ще однією важливою перевагою персональної PC є те, що прийоми взаємодії з різним програмним забезпеченням однотипні або вже в будь-якому випадку схожі. Як правило, кожна з прикладних систем, розроблених в централізованій архітектурі, має свій унікальний інтерфейс. У той же час, при освоєнні нової програми для PC можна використовувати, щонайменше, частину знань, отриманих раніше при роботі з іншим програмним забезпеченням.

PC можна освоїти самостійно (або майже самостійно), і він завжди знаходиться під рукою, безпосередньо на робочому місці. Тому персональні комп'ютери і у нас, і на Заході, стали повсюдно застосовуватися насамперед для вирішення користувачами своїх власних, локальних завдань. Найбільш активно використовуваними застосуваннями стали електронні таблиці та текстові редактори.

Не буде перебільшенням сказати, що ці програми використовуються на PC, в тому числі і у промисловості, більше за всіх інших, разом узятих. Комп'ютеризація підготовки документів, безсумнівно, знімає з конкретних виконавців великий обсяг рутинної роботи, створює у них розуміння можливостей застосування PC як корисного і дуже зручного інструменту автоматизації управління. Однак, при цьому рано чи пізно з'ясовується, що сама по собі поява у співробітників персональних PC, навіть у великій кількості, ще не призводить до помітних якісних змін в управлінні. Особливо сильно це помітно на середніх та великих підприємствах.

Цьому є кілька об'єктивних причин. Одна з них у тому, що настільні додатки, такі як Word і Excel, не призначені для роботи з великим обсягом структурованої інформації, в них досить важко відображати складні взаємозв'язки між даними. Тому персональний комп'ютер може дуже швидко перетворитися на сховище великої кількості слабо пов'язаних між собою розрізнених документів. При цьому в них може міститися величезна кількість інформації, але швидко знайти потрібні дані, а вже тим більше, отримати повну картину на основі цієї інформації часто не здатний навіть її господар. До того ж, зберігання інформації у вигляді окремих, не пов'язаних один з одним документів, може призвести до появи протиріч між ними. Наприклад, відсутні надійні способи гарантувати, що підсумкова сума в одному документі відповідає розшифровці цієї ж суми в іншому.

Звичайно, для вирішення цієї проблеми можна використовувати ще один клас настільних додатків. Це спеціалізовані системи управління базами даних для персональних PC, так звані «настільні» СУБД. Вони краще пристосовані для роботи з великими обсягами структурованої інформації, її пошуку, сортування, здійснення вибірок, підготовки звітів.

Ще одна проблема розглянутої архітектури полягає в тому, що наявність великого числа не пов'язаних між собою PC на окремих робочих місцях часто викликає такий же локальний погляд на автоматизацію. На кожному робочому місці вирішуються вузькі завдання комп'ютеризації діяльності конкретного виконавця. При цьому часто не враховується навіть необхідність інформаційної взаємодії між робочими місцями, не кажучи вже про комп'ютеризацію завдань в рамках єдиної автоматизованої системи управління виробництвом.

Автоматизована система керування, побудована з використанням локальних PC, не надто ефективно використовує обчислювальні ресурси. Персональний комп'ютер за своєю архітектурою нагадує маленький мейнфрейм, який встановлений окремо. Всі операції повністю виконуються на одному окремо взятому комп'ютері, відсутній будь-який поділ ресурсів між PC. При збільшенні обсягів інформації, що обробляється в системі, для того, щоб підтримувати прийнятну продуктивність, доводиться нарощувати потужність комп'ютера, причому якщо зростає навантаження на декілька PC, то необхідно модернізувати кожен з них окремо, а це значні витрати.

Можна, звичайно, використовувати персональні комп'ютери для автоматизації локальних задач, для яких достатньо окремого робочого місця. Однак, більшість завдань не є ізольованими одне від одного, оскільки в системі керування підприємством завжди існує тісні зв'язки між окремими ланками, горизонтальні та вертикальні. Кожен співробітник обмінюється тією чи іншою інформацією зі співробітниками свого підрозділу, з іншими підрозділами, з керівництвом і з підлеглими.

При наявності ізольованих один від одного PC, обмін даними між ними або буде здійснюватися в паперовому вигляді, або на дискетах. Обидва цих варіанти не дуже зручні - перший варіант призведе до повторного введення інформації в комп'ютер. Другий, як ми вже відзначали, незручний і неефективний.

У міру збільшення числа PC та підвищення кваліфікації користувачів, на підприємстві все сильніше усвідомлюється потреба в швидкому, надійному і зручному способі оперативного обміну великими обсягами даних між комп'ютерами, а головне, між користувачами. Це дозволило б, як мінімум, передавати інформацію між окремими робочими місцями, а як максимум, вирішувати завдання більш високого рівня, об'єднуючи окремі автоматизовані місця в інтегровані комплекси. І, природно, попит породив пропозицію - були створені локальні обчислювальні мережі та архітектура файл-сервер.

10.2 Локальні обчислювальні мережі та архітектура файл-сервер

Локальні обчислювальні мережі (ЛОМ) призначені для об'єднання окремих обчислювальних пристроїв. Вони включають в себе те чи інше середовище (дротове або бездротове), що забезпечує високу швидкість передачі даних, приймально-передавальні пристрої, що здійснюють зв'язок між комп'ютером і середовищем, а також набір протоколів, які визначають порядок взаємодії в мережі.

Відстань між пристроями в локальних мережах, як правило, складає десятки-сотні метрів, тобто, ці технології застосовуються в межах однієї будівлі. ЛВС забезпечують високу швидкість інформаційного обміну, але саме поняття «високої швидкості» сильно змінилося. Якщо на початку 90-их високою вважалася швидкість в 1 мегабіт в секунду, то зараз звичайним вважається 100 мегабіт в секунду, а на підході та 1 гігабіт / с, що в 1024 разів більше. Найчастіше, коли в організації з'являється хоча б кілька комп'ютерів, виникає потреба налагодити оперативний обмін інформацією між ними. Тому незабаром після появи персональних комп'ютерів для їх об'єднання були створені локальні обчислювальні мережі.

Цьому сприяли два фактори. По-перше, кількість PC у багатьох організаціях і на підприємствах стало значним, і завдання високопродуктивного оперативного обміну інформацією між робочими стала особливо актуальною.

Другим важливим чинником стало постійне зниження вартості обладнання для мереж. Це призвело до того, що створення локальної мережі перестало бути недозволеною розкішшю, яка була по кишені тільки найбагатшим організаціям. Взагалі, розвиток мережевих технологій з одночасним зниженням вартості йшло приголомшуючими темпами.

Найбільш легкий, дешевий і простий варіант використання програмних технологій з'єднання комп'ютерів за допомогою ЛВС - це однорангова мережа. У такій мережі всі комп'ютери рівноправні один з одним. Кожен комп'ютер при цьому може робити свої ресурси доступними для інших. В якості таких ресурсів виступають перш за все дисковий простір і друкуючі пристрої. При цьому не передбачається наявність спеціалізованого комп'ютера - сервера, виділеного виключно для спільного використання.

Однорангова мережа дозволяє вирішити проблеми передачі файлів між окремими персональними комп'ютерами, а також поділу дефіцитних ресурсів між декількома користувачами. Однак, вона не може бути використана для створення додатків, в яких необхідний одночасний доступ до даних з декількох PC. Крім того, надійність і продуктивність подібної мережі вкрай обмежена.

Розвиток інформаційних систем зажадало створення апаратних і програмних технологій для одночасного доступу до ресурсів великого числа користувачів. Для вирішення цієї проблеми була запропонована і успішно реалізована, архітектура файл-сервер. На певному етапі комп'ютеризації підприємства ця архітектура відповідає потребам управління і дозволяє створювати додатки, які виконують завдання масштабу відділу.

Архітектура файл-сервер припускає наявність трьох основних компонентів: файлового сервера, файлового клієнта і набору локальної мережі для спілкування між ними.

Файловий сервер - це комплекс апаратних і програмних засобів, що забезпечує спільний доступ до файлових ресурсів (а також до принтерів) через локальну мережу багатьом користувачам одночасно. Як правило, при побудові файлового

сервера використовується спеціалізована апаратура (з великим об'ємом оперативної пам'яті, швидкодіючими і більш надійними зовнішніми пристроями). Файловий сервер функціонує під управлінням спеціалізованого програмного забезпечення - мережевої операційної системи. В якості стандарту де-факто на промислових підприємствах застосовується серверна операційна система Netware фірми Novell.

Файловий клієнт - це набір програмного забезпечення, що забезпечує доступ до файлових ресурсів сервера (або серверів) з персонального комп'ютера. Клієнт встановлюється на кожному робочому місці, з якого має здійснюватися доступ до сервера. Як правило, доступ до ресурсів здійснюється прозоро для прикладних програм, тобто для користувача вони представляються аналогами локальних ресурсів його персонального комп'ютера.

І нарешті, для нормального функціонування файл-серверних додатків, природно, необхідна власне локальна мережа, що з'єднує між собою клієнт і сервер. Працездатність файл-серверного додатка безпосередньо залежить від надійності і продуктивності локальної мережі.

Як випливає з самого терміна файл-сервер, весь обмін між клієнтськими робочими місцями і сервером здійснюється на рівні файлів. Типові команди, які передаються серверу в цій архітектурі - це відкрити файл, прочитати певну кількість байт з файлу, записати у файл певне число байт, закрити файл. При цьому сервер не володіє жодною інформацією про вміст файлів, тому всю обробку даних виконує клієнт. Це негативно впливає на продуктивність та на масштабованість системи.

Використання архітектури файл-сервер для будь-якого підприємства є великим кроком вперед у порівнянні з окремими локальними PC. Перш за все, файл-сервер дає саму принципову можливість створення багатокористувацьких додатків, в яких доступ до одних і тих же даних здійснюється одночасно більш ніж з одного робочого місця. По-друге, оскільки поліпшення сервера підвищує продуктивність роботи з будь-якого клієнтського місця, ця архітектура більш ефективна з точки зору модернізації системи. Як правило, файл-сервер має спеціальні засоби, які підвищують надійність зберігання даних на сервері та зменшують вірогідність втрати інформації після програмного або апаратного збою. І нарешті, в архітектурі файл-сервер підвищується безпека системи, оскільки, як правило, сервер дозволяє надавати різний рівень доступу до системи для різних користувачів. Як правило, при підключенні клієнта до сервера запитується логін і пароль, залежно від цього надається відповідний набір доступних ресурсів.

Варто зауважити, що, оскільки в архітектурі файл-сервер вперше з'являються клієнт і сервер, то, строго кажучи, в ній теж використовуються клієнт-серверні технології, якщо брати саме широке значення цього терміна (про це ми будемо говорити трохи пізніше). Однак, через те, що взаємодія клієнта і сервера здійснюється на занадто низькому рівні - рівні файлів, в інформаційних додатках, створених в цій архітектурі, вся обробка даних ведеться на клієнті. Для створення файл-серверних додатків, як правило, використовуються ті ж інструменти, що і для створення локальних додатків.

На певному етапі розвитку інформаційної системи підприємства архітектура файл-сервер є найбільш природною. Вона дозволяє створити на підприємстві автоматизовані комплекси масштабу відділу. Ці комплекси складаються з робочих місць. Дані завдань зберігаються на сервері, як набір файлів, до яких здійснюється доступ з усіх робочих місць відділу. В архітектурі файл-сервер можна частково автоматизувати і обмін даними між різними відділами за допомогою передачі файлів.

Однак, архітектура файл-сервер має цілий ряд принципових обмежень. Це обмеження по числу одночасно працюючих додатків - з їх зростанням різко зростає навантаження на мережу. Це потенційні проблеми зі збереженням даних при одночасному внесення змін з різних місць. Це обмеження по складності операцій по обробці даних, оскільки вся вона здійснюється на клієнті. Це, нарешті, принципова неможливість гарантувати з боку сервера цілісність інформації в базі даних, оскільки їх обробка здійснюється кожним клієнтом окремо.

На певному етапі розвитку інформаційної системи підприємства всі ці обмеження стають принципово важливими. Як правило, цей стрибок відбувається при переході до вирішення в рамках автоматизованої системи завдань масштабу підприємства. При цьому постає завдання переходу до архітектури клієнт-сервер.

Архітектура клієнт-сервер

Детально про те, що таке технології та архітектура клієнт-сервер, і чому вони здатні допомогти при зростанні інформаційних систем на підприємстві, ми поговоримо нижче. Тут же ми коротко опишемо принципові архітектурні відмінності цієї системи від файл-серверної, яка на сьогодняшній день найбільш поширена в промисловості.

Передумови появи клієнт-серверної технології:

- число користувачів більше 10 - 15 осіб;
- висока вартість великих OEM та засобів розробки для них;
- вирішення завдань масштабу підприємства і управління підприємством в цілому;
- онлайн робота з віддаленими користувачами;
- критичне забезпечення цілісності інформації (банки тощо).

В основі клієнт-серверних технологій лежать дві ідеї:

- загальні для всіх користувачів дані на одному або декількох серверах;
- багато користувачів (клієнтів) на різних обчислювальних установках спільно (паралельно і одночасно) оброблюють загальні дані.

Тобто системи, засновані на цих технологіях, розподілені лише щодо користувачів, тому часто їх вважають видом багатокористувацьких систем.

Як і для файл-серверної архітектури, складовими компонентами клієнт-серверної архітектури є сервер, клієнтські місця і мережева інфраструктура. Однак, на

відміну від попереднього випадку, сервер тут є вже не сервером файлів, а сервером баз даних або навіть сервером додатків. Таким чином, на сервер лягає не просто зберігання файлів, а підтримання бази даних в цілісному стані або, в разі сервера додатків, навіть виконання тієї чи іншої частини прикладної задачі. Природно, що вимоги до сервера при цьому можуть зростати в рази.

З іншого боку, те, що сервер володіє інформацією про характер збереженої бази даних, дозволяє набагато збільшити ефективність обробки. Тому для багатьох завдань автоматизації навантаження на сервер за рахунок більш оптимального виконання операцій над даними в порівнянні з аналогічними файл-серверними додатками може навіть зменшитися.

Відповідно, спілкування між клієнтом і сервером відбувається не на рівні файлів, а на рівні обміну запитами. Клієнт передає серверу високорівневі запити на отримання тієї чи іншої інформації або на її зміну, а сервер повертає клієнту результати виконання запитів. При цьому, на відміну від файл-серверної архітектури, доступ до даних не є прозорим для користувача програми. Тому, технологія розробки таких додатків принципово відмінна від локальних і файл-серверних систем.

На сучасному етапі мережеве забезпечення для архітектури клієнт-сервер аналогічно файл-серверному. Клієнт-серверні системи можуть будуватися з використанням тих же мережевих технологій і на тій же мережевій інфраструктурі. Більш того, як правило, на підприємстві мирно співіснують обидві ці архітектури. Це викликано двома основними причинами. По-перше, що на будь-якому підприємстві багато завдань, пов'язаних із зберіганням та обміном документами, які являють собою окремі файли, а для них архітектура файл-сервер оптимальна. По-друге, файл-серверні завдання в тому чи іншому обсязі майже завжди зберігаються в тому чи іншому обсязі та експлуатуються поряд зі створюваними клієнт-серверними додатками.

З точки зору мережевої взаємодії принципово новим у додатках клієнт-сервер є можливість переходу до використання глобальної мережі. Швидкість обміну в такій мережі може бути на порядок нижче, а відстані між робочими місцями можуть досягати кілометрів і навіть десятків кілометрів. І все це працює, оскільки в додатках клієнт-сервер обсяг переданої інформації може бути радикально скорочений за рахунок використання високорівневих запитів до даних. Але головною зміною, яка може бути здійснена промисловим підприємством при переході до клієнт-серверної архітектури - це якісний стрибок у масштабах завдань, що вирішуються при комп'ютеризації, в тому, наскільки комплексно автоматизується керування виробництвом, в рівні цілісності та достовірності даних, що зберігаються в інформаційній системі.

Слід зазначити, що поява клієнт-серверних додатків на підприємстві - процес важкий і часто болючий. Причому, крім вирішення технічних та фінансових проблем, при цьому переході дуже важливо здійснити зміну самого рівня управління, що тягне за собою величезну організаційну роботу.

10.3 Моделі взаємодії клієнт-сервер

У своєму розвитку технології «клієнт-сервер» пройшли кілька етапів, тому є різні моделі технології. Їх реалізація заснована на поділі структури СУБД на три компоненти:

- введення і відображення даних (інтерфейс з користувачем);
- прикладний компонент (запити, події, правила, процедури та функції, які характерні для даної предметної області);
- функції керування ресурсами (файловою системою, базою даних тощо).

Тому, в будь-якому додатку виділяються наступні компоненти:

- компонент подання даних;
- прикладний компонент;
- компонент керування ресурсом.

Зв'язок між компонентами здійснюється за певними правилами, які називають «протокол взаємодії».

Існують різні класифікації, але однією з найпоширеніших є використання чотирьох моделей технології «Клієнт-сервер»:

1. Модель файлового серверу (File Server - FS) (рис. 10.1а).
2. Модель віддаленого доступу до даних (Remote Data Access - RDA) (рис. 10.1б).
3. Модель сервера БД (Data Base Server - DBS) (рис. 10.1в).
4. Модель сервера додатків (Application Server - AS) (рис. 10.1г).

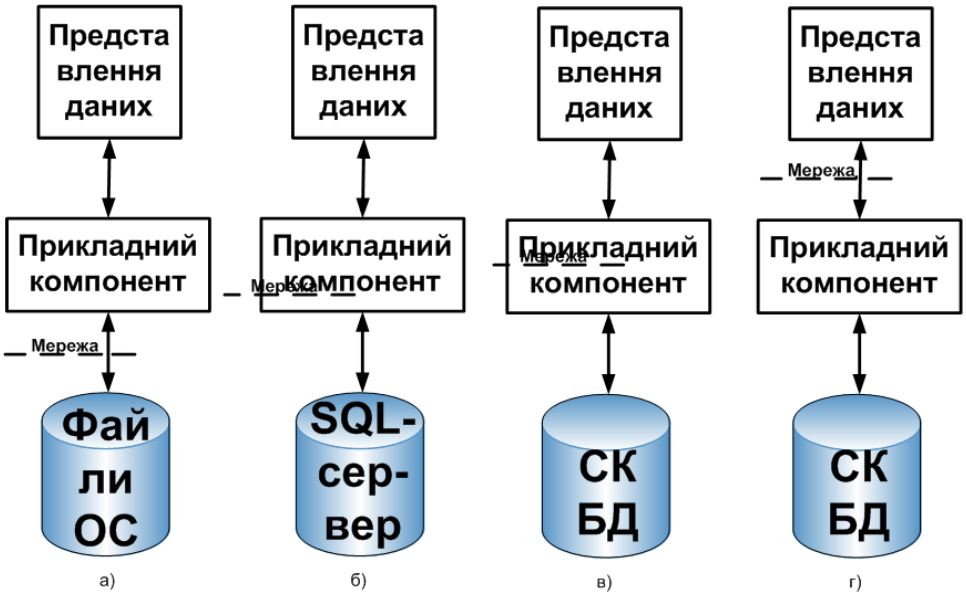


Рис. 10.1. Моделі технології «Клієнт-сервер»: а) FS; б) RDA; в) DBS; г) AS

Історично першою з'явилася модель розподіленого представлення даних, яка реалізовувалася на універсальній обчислювальній машині з підключеними до неї

неінтелектуальними терміналами. Управління даними та взаємодія з користувачем при цьому об'єднувалися в одній програмі, на термінал передавалася тільки «картинка», сформована на центральному комп'ютері.

Потім, з появою персональних комп'ютерів (ПК) і локальних мереж, були реалізовані моделі доступу до віддаленої Бази Даних. Деякий час базової для мереж ПК була архітектура файлового сервера. При цьому один з комп'ютерів є файловим сервером, на клієнтах виконуються додатки, у яких сполучені компонент подання й прикладний компонент (СУБД і прикладна Програма). Протокол обміну при цьому представляє набір низькорівневих викликів операцій файлової системи. Така архітектура, реалізована, як правило, за допомогою персональних СУБД, має очевидні недоліки - високий мережевий трафік і відсутність уніфікованого доступу до ресурсів.

З появою перших спеціалізованих серверів баз даних з'явилася можливість іншої реалізації моделі доступу до віддаленої Бази Даних. У цьому випадку ядро СУБД функціонує на сервері, протокол обміну забезпечується за допомогою мови SQL. Такий підхід у порівнянні з файловим сервером веде до зменшення завантаження мережі й уніфікації інтерфейсу «клієнт-сервер». Однак, мережевий трафік залишається досить високим, крім того, як і раніше неможливо задовільнити адміністрування додатків, оскільки в одній програмі сполучаються різні функції.

Пізніше була розроблена концепція активного сервера, який використовував механізм збережених процедур. Це дозволило частину прикладного компонента перенести на сервер (модель розподіленого додатку). Процедури зберігаються в словнику бази даних, розділяються між декількома клієнтами й виконуються на тому ж комп'ютері, що і SQL-сервер. Переваги такого підходу: можливо централізоване адміністрування прикладних функцій, значно знижується мережевий трафік (оскільки передаються не SQL-запити, а виклики збережених процедур). Недолік - обмеженість засобів розробки збережених процедур у порівнянні з мовами загального призначення (C і Pascal).

На практиці зараз звичайно використовуються змішані підходи:

- найпростіші прикладні функції виконуються збереженими процедурами на сервері;
- більш складні реалізуються на клієнті безпосередньо в прикладній програмі.

Зараз ряд постачальників комерційних СУБД оголосило про плани реалізації механізмів виконання збережених процедур з використанням мови Java. Це відповідає концепції «тонкого клієнта», функцією якого залишається тільки відображення даних (модель віддаленого подання даних).

Останнім часом також спостерігається тенденція до все більшого використання моделі розподіленого додатка. Характерною рисою таких додатків є логічний поділ додатка на дві та більше частини, кожна з яких може виконуватися на окремому комп'ютері. Виділені частини додатка взаємодіють одна з одною, обмінюючись повідомленнями в заздалегідь погодженому форматі. У цьому випадку дволанкова

архітектура клієнт-сервер стає триланкової (рис. 10.2.), а в деяких випадках, вона може включати і більше ланок.

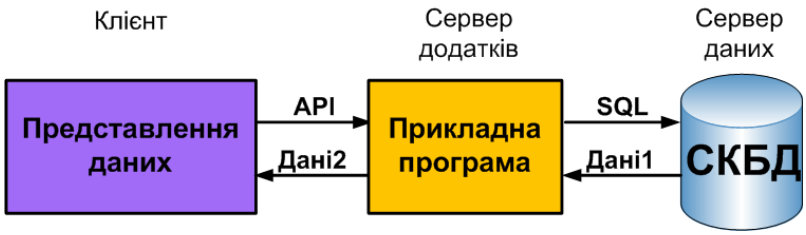


Рис. 10.2. Триланкова архітектура

10.4 Модель обробки з використанням файлового серверу

В задачах обробки інформації, заснованих на системах баз даних, існують два варіанти розташування даних: локальний і віддалений. У першому випадку говорять про доступ до локальних даних, у другому - про доступ до віддалених даних. Локальні дані, як правило, розташовуються на жорсткому диску комп'ютера, на якому працює користувач, і знаходяться в монопольному керуванні цього користувача. Користувач при цьому працює автономно, не залежачи від інших користувачів та жодним чином не впливаючи на їх роботу. Дистанційні дані розташовуються поза комп'ютера користувача (користувачів) - на файловому сервері мережі або на спеціально виділеному для цих цілей комп'ютері.

Модель файлового серверу (File Server - FS) – це природне розширення персональних СУБД для підтримки багатокористувацького режиму і в цьому плані ще довго буде зберігати своє значення (рис. 10.3.).

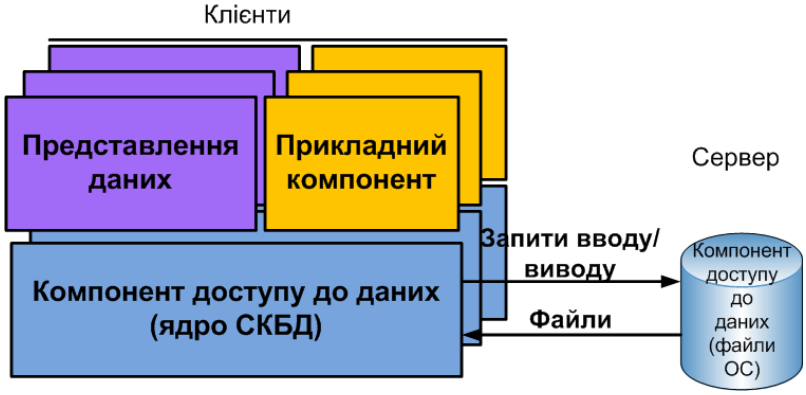


Рис. 10.3. Схема взаємодії FS-моделі

Особливості FS-моделі:

- всі основні компоненти розміщуються на клієнтському комп'ютері;
- модель характеризує не стільки спосіб створення ІС, скільки загальний спосіб взаємодії комп'ютерів в локальній мережі;
- один з комп'ютерів виділяється і визначається файловим сервером, тобто загальним сховищем будь-яких даних;
- сервер виконує чисто пасивну функцію;
- дуже проста та зрозуміла модель.

У стандартній файл-серверній архітектурі дані, розташовуючись на файл-сервері, є, по суті, пасивним джерелом. Вся відповідальність за їх отримання, обробку, а також за підтримку цілісності бази даних лежить на додатку, запущеному з робочої станції. При цьому, оскільки обробка даних здійснюється на робочій станції, по мережі переганяється вся необхідна для цієї обробки інформація, хоча обсяг даних, які цікавлять користувача, може бути менше в десятки разів. Наприклад, якщо користувача цікавлять всі працівники заданого підприємства, які беруть участь у конкретному проекті, його додаток «отримає» спочатку всіх працівників і всі проекти з бази даних, і тільки після цього виконає необхідну вибірку.

Історично на персональних комп'ютерах використовувався саме цей підхід як більш простий в освоєнні. Однак великий обсяг даних, які переганяються по мережі, швидко «забиває» мережу вже при невеликому числі користувачів, істотно обмежуючи можливості зростання (масштабованості). Цей основний і самий істотний недолік змусив шукати способи зменшення навантаження на мережу.

Тут додатки виконуються на робочих станціях. Додаток включає модулі для організації діалогу з користувачем, бізнес-правила (транзакції), що забезпечують необхідну логіку обчислень, і ядро СУБД. Часто ядро СУБД в моделі файлового сервера не є вираженим і являє собою набір функцій, пов'язаних з іншими компонентами додатка. Додаток, включаючи і ядро СУБД, дублюється на різних робочих станціях. На файловому сервері зберігаються тільки файли бази даних (індекси, файли даних тощо) і деякі технологічні файли (оверлейні файли, файли сортування тощо). Оператори SQL-звернення до СУБД, закодовані в прикладній програмі, обробляються ядром СУБД на робочій станції. СУБД організовує доступ до файлів бази даних для виконання оператора. По мережі передаються запити на читання/запис даних, індекси, проміжні та результуючі дані, блоки технологічних файлів.

На основі моделі файлового сервера функціонують такі популярні СУБД як FoxPro (Microsoft), dBase (Borland), CF-Clipper (Computer Associates International), Paradox (Borland) тощо. СУБД розглянутого класу коштують недорого, прості в установці та освоєнні. Також відсутні високі вимоги до продуктивності сервера та програмні компоненти СУБД не розподілені.

Але вони мають ряд істотних недоліків:

- системи, розроблені на базі цих СУБД, мають низьку продуктивність, оскільки всі проміжні дані передаються, як правило, по низькошвидкісний шині мережі, а прикладні програми і ядро СУБД виконуються на малопотужних робочих станціях;
- високий мережевий трафік;
- низька масштабованість;
- відсутність механізмів безпеки БД;
- ці СУБД не підтримують розподілену обробку.

В силу перерахованих недоліків модель файлового сервера практично не використовується в розподілених інформаційних системах.

10.5 Модель обробки з використанням віддаленого доступу до даних

Модель віддаленого доступу до даних (Remote Data Access – RDA) – архітектура, яка заснована на обліку специфіки розміщення і фізичного маніпулювання даними у зовнішній пам'яті для реляційних СУБД (рис. 10.4.).

У RDA-моделі для обробки даних виділяється спеціальне ядро - так званий *SQL-сервер*, який приймає на себе функції обробки запитів користувачів, іменованих тепер клієнтами. Сервер баз даних являє собою програму, яка виконується, як правило, на потужному комп'ютері. Додатки-клієнти посилають з робочих станцій запити на вибірку (вставку, оновлення, видалення) даних. При цьому сервер виконує всю «брудну» роботу з відбору даних, відправляючи клієнтові тільки необхідну «вичавлювання». Якщо наведений вище приклад перебудувати з урахуванням клієнт-серверної архітектури, то додаток-клієнт «отримає» від сервера в якості результату список тільки тих працівників, які беруть участь у заданому проекті, і не більше того!
Такий підхід забезпечує вирішення трьох важливих завдань:

- зменшення навантаження на мережу;
- зменшення вимог до комп'ютерів-клієнтам;
- підвищення надійності та збереження логічної цілісності бази даних.

Тут додатки також виконуються, в основному, на робочих станціях. Додаток включає модулі для організації діалогу з користувачем і бізнес-правила (транзакції). Ядро СУБД є загальним для всіх робочих станцій і функціонує на сервері. Оператори звернення до СУБД (SQL-оператори), закодовані в транзакції, не виконуються на робочій станції, а пересилаються для обробки на сервер. Ядро СУБД транслює запит і виконує його, звертаючись для цього до індексів та інших проміжних даних. Назад на робочу станцію передаються тільки результати обробки оператора.

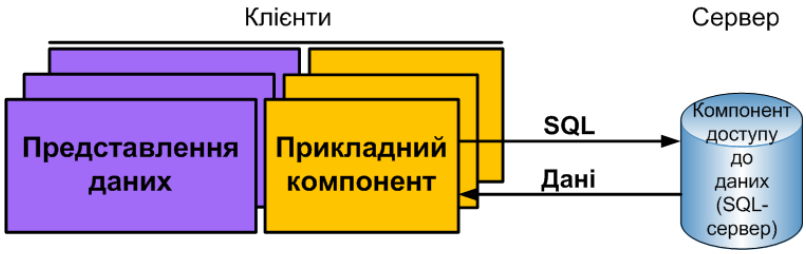


Рис. 10.4. Схема взаємодії RDA-моделі

У файлах БД на сервері знаходиться також і системний каталог БД.

У числі іншого, в каталог БД поміщаються:

- відомості про зареєстрованих користувачів;
- привілеї користувачів;

На клієнтських комп’ютерах встановлюються частини СУБД, які реалізують:

- інтерфейсні функції;
- прикладні функції.

Прикладний компонент включає:

- бібліотеки запитів;
- процедури обробки даних.

Прикладний компонент повністю розміщується і виконується на клієнтській частині - формує SQL-інструкції, що направляються SQL-серверу.

SQL-сервер - спеціальний програмний компонент, який:

- орієнтований на інтерпретацію SQL-інструкцій;
- високошвидкісне виконання низькорівневих операцій з даними;
- приймає і координує SQL-інструкції від різних клієнтів;
- виконує SQL-запити;
- перевіряє та забезпечує виконання обмежень цілісності даних;
- направляє клієнтам результати обробки SQL-інструкцій - набори даних.

До переваг RDA-моделі слід віднести:

- зменшення завантаження мережі;
- SQL-сервер забезпечує виконання обмежень цілісності та безпеки даних;
- уніфікований інтерфейс взаємодії прикладної частини ІС із загальними даними;
- в рамках SQL така взаємодія реалізована через ODBC-протокол, та називається *інтероперабельністю*.

Недоліки RDA-моделі:

- високі вимоги до клієнтських комп'ютерів;
- великий обсяг технологічних змін;
- зміна структури управління та бізнес-процесів;
- значний, хоча і менший, ніж у моделі FS, мережевий трафік.

Наступним великим недоліком є великий обсяг власне технологічних змін, що виникають при спробі впровадження архітектури клієнт-сервер. Клієнт-серверна система вимагає іншого рівня технічної грамотності з боку, як співробітників інформаційних служб, так і кінцевих користувачів. Витрати на перепідготовку користувачів та експлуатаційного персоналу, перебудова структури автоматизації підприємства становлять більшу частину айсберга, ніж ясно видимі прямі витрати на модернізацію апаратури, закупівлю та/або розробку необхідного забезпечення.

І, нарешті, самим великим підводним каменем на шляху створення АС на підприємстві є необхідність міняти структуру управління і пов'язані з цим організаційні витрати. Спроба впровадити нові технологічні рішення нічого не змінюючи в суті автоматизованих бізнес-процесів може закінчитися безрезультатно. При цьому підприємство понесе прямі матеріальні збитки через

великий обсяг дорогого апаратного та програмного забезпечення, лежачого мертвим вантажем, а також через відволікання співробітників від виконання основних службових обов'язків. У кращому випадку будуть впроваджені окремі ділянки клієнт-серверної системи, при цьому фактично нове програмне забезпечення буде використовуватися на старому ідейному рівні.

10.6 Модель обробки з використанням сервера бази даних

Для поліпшення попередньої RDA-моделі в сучасних СУБД використовується ***модель сервера баз даних*** *(DataBase Server - DBS)*, в якій на сервері можуть запускатися так звані збережені процедури і тригери, які разом з ядром СУБД утворюють сервер бази даних (рис. 10.5.).

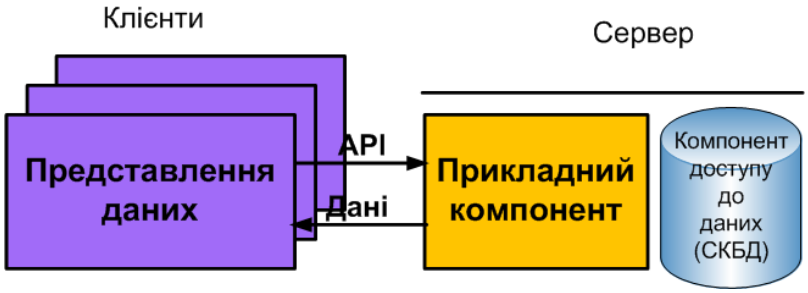


Рис. 10.5. Схема взаємодії DBS-моделі

До збережених процедур можна звертатися з додатків на робочих станціях. Це дозволяє скоротити розмір коду прикладної програми і зменшити потік SQL-операторів з робочої станції, так як групу необхідних SQL-запитів можна закодувати в збереженій процедурі. Тригери - це програми, які виконуються ядром СУБД перед або після оновлення (UPDATE, INSERT, DELETE) таблиці бази даних. Вони дозволяють автоматично підтримувати цілісність бази даних.

Модель сервера бази даних (БД) підтримують наступні СУБД: Oracle, DB2 (IBM), MS SQL Server (Microsoft), MySQL (Oracle), FireBird, PostgreSQL, Sybase (SAP), тощо. Причому на перші чотири СУБД припадають понад 85% ринку. СУБД розглянутого класу мають наступні переваги:

- системи, мають високу продуктивність, тому що запити виконуються на високошвидкісних серверах;
- зниження мережевого трафіку, тому що по шині передаються тільки SQL-запити і результати з виконання;
- СУБД підтримують розподілену обробку;
- більш гнучка «настройка» на предметну область;
- забезпечення узгодженого стану даних;
- надійність зберігання і обробки даних;
- ефективна координація колективної роботи користувачів із загальними даними;
- в рамках цих СУБД пропонується велика кількість сервісних продуктів, що полегшують розробку додатків і створення розподіленої системи.

Дійсно, тепер сервер БД (які інколи мають назву SQL-сервер) повертає клієнтському додатку тільки «вичавлювання» того, що він переглянув в базі, а воно, в загальному випадку, дійсно становить малу частину від загального обсягу. Тому в мережі не спостерігається різкого збільшення навантаження при збільшенні кількості клієнтів. Клієнтські ж додатки можуть виконуватися на менш потужних (в порівнянні з сервером) комп'ютерах завдяки тому, що їм практично не потрібно виконувати ніякої додаткової обробки отриманих від сервера результатів запиту (хоча, звичайно ж, це не заборонено). Побічним ефектом зменшення навантаження на мережу є підвищення швидкості виконання додатків клієнтів. Крім того, система легше масштабується - легше і дешевше замінити один сервер на більш потужний, ніж десятки робочих станцій.
Ці СУБД мають і недоліки:

- вони набагато дорожче СУБД попереднього класу, складні в освоєнні;
- для ефективної роботи цих СУБД потрібні високошвидкісні (а тому й дорогі) сервери та мережі.

Найбільш важливим результатом переходу в архітектуру клієнт-сервер є гарантоване збереження логічної цілісності бази даних, тобто система стає більш стійкою і більш захищеною. Досягається це завдяки можливості перекласти турботу про збереження цілісності бази на сервер. Для цього «хороші» сервери мають великий набір вбудованих механізмів, що захищають систему від невірних дій клієнтів. Серед цих механізмів можна назвати такі як обмеження цілісності, декларативна цілісність посилань, тригери, віртуальні таблиці (подання), авторизація користувачів тощо.

10.7 Модель обробки з використанням сервера додатків

Щоб рознести вимоги до обчислювальних ресурсів сервера у відношенні швидкодії і пам'яті за різними машинам, використовується *модель сервера додатків (Application Server AS)* (рис.10.6.). AS-модель зберігає сильні сторони DBS-моделі.

Особливості AS-моделі:

- перенесення прикладного компонента АІС на спеціалізований сервер;
- на клієнтських машинах - тільки інтерфейсна частина системи;
- виклики функцій обробки даних спрямовуються на сервер додатків;
- низькорівневі операції з даними виконує SQL-сервер.

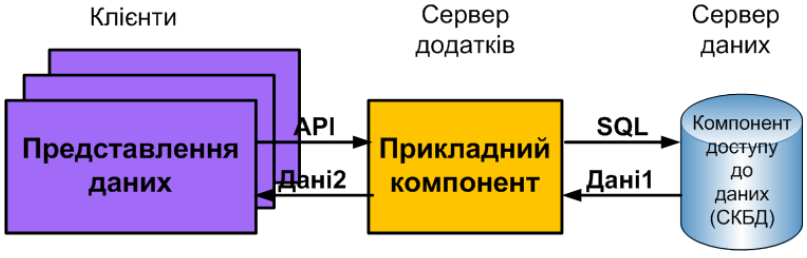


Рис. 10.6. Схема взаємодії AS-моделі

Використання цієї моделі дозволяє розвантажити робочі станції, тобто перейти до «тонких» клієнтів. Звичайно, сервер додатків можна організувати і за допомогою збережених процедур. Але для реалізації збережених процедур використовують мови високого рівня (наприклад, в Oracle - мову PL/SQL), тому програми виходять ресурсоємними. Причому можливості цих мов дуже широкі: від циклів до обробки даних на рівні бітів. Збережені процедури також не підтримують розподілені додатки, тобто вони не забезпечують автоматичний запуск необхідної програми на іншому сервері.

Сучасні СУБД використовують наступні процедурні мови:

- Transact-SQL (MS SQL Server, Sybase);
- PL/SQL (Oracle);
- PSQL (FireBird);
- SQL PL (IBM DB2);
- JetSQL (MS Access);
- PL/pgSQL (PostgreSQL).

У тому випадку, коли інформаційна система об'єднує досить велику кількість різних інформаційних ресурсів і серверів додатків, постає питання про оптимальне управління всіма її компонентами. Для цього використовують програмні засоби, які часто називають *менеджерами транзакцій*, *моніторами транзакцій (Transaction Processing Monitor - TPM)*, і які розміщуються на сервері додатків. Також менеджер транзакцій і додатки можуть запускаютися на одному комп'ютері (хоча це не є обов'язковим), щоб зменшити потік SQL-запитів по мережі.

Транзакція - це послідовна сукупність операцій над даними (SQL-інструкцій), що має окреме смислове значення. Але часто в моніторах транзакцій поняття транзакції розширюється - в даному випадку це не атомарна дія над базою даних, а будь-яка дія в системі – видача повідомлення, запис в індексний файл, друк звіту тощо.

Для спілкування прикладної програми з монітором транзакцій використовується спеціалізований API (Application Program Interface - інтерфейс прикладного програмування), який реалізується у вигляді бібліотеки, що містить виклики основних функцій (встановити з'єднання, викликати певний сервіс тощо). Сервери додатків (сервіси) також створюються за допомогою цього API, кожному сервісу присвоюється унікальна назва. Монітор транзакцій, отримавши запит від прикладної програми, передає її виклик відповідному сервісу (якщо той не запущений, породжується необхідний процес), після обробки запиту сервером додатків повертає результати клієнту. Для взаємодії моніторів транзакцій з серверами баз даних розроблений протокол XA. Наявність такого уніфікованого інтерфейсу дозволяє використовувати в рамках однієї програми кілька різних СУБД.

Монітор транзакцій усуває такі витрати спільної обробки:

- *втрачені зміни* - дві транзакції одночасно змінюють один об'єкт БД;
- *брудні дані* - одна транзакція змінює об'єкт, а друга читає дані з нього;

- *неповторювані читання* - одна читає об'єкт, а друга транзакція змінює його.

Для ізоляції транзакцій і подолання ситуацій неузгодженої обробки даних використовують *серіалізацію транзакцій* - виконання транзакцій таким чином, щоб результат їхньої спільного виконання був еквівалентний результату їх послідовного виконання.

Використання моніторів транзакцій у великих системах дає наступні переваги:

- концентрація всіх прикладних функцій на сервері додатків забезпечує значну незалежність як від реалізації інтерфейсу з користувачем, так і від конкретного способу керування ресурсами. При цьому також забезпечується централізоване адміністрування додатків, оскільки всі додатки знаходяться в одному місці, а не «розмазані» по мережі на клієнтських робочих місцях;
- монітор транзакцій в змозі сам запускати і зупиняти сервери додатків. В залежності від завантаження мережі та обчислювальних ресурсів він може перенести або скопіювати частину серверних процесів на інші вузли. Це забезпечує балансування навантаженням серверів;
- забезпечується динамічна конфігурація системи, тобто без її зупинки може бути доданий новий сервер ресурсів або сервер додатків;
- підвищується надійність системи, тому в разі збоїв сервер додатків може бути переміщений на резервний комп'ютер;
- з'являється можливість керування розподіленими базами даних.

До недоліків AS-моделі слід віднести:

- необхідні додаткові потужності (може навіть окремий додатковий сервер);
- підвищує трафік в мережі.
-

10.8 Використання моделі відкритих систем

Перші інформаційні системи, з'явилися в середині XX-го століття. Вони ґрунтувалися на мейнфреймах компанії IBM, файлової системі OC/360, а згодом на ранніх СУБД типу IMS і IDMS (в СРСР на клонах відповідно ЄС РС, ОКА і КАМА). Ці системи прожили довге і корисне життя, багато хто з них до цих пір експлуатуються. Але з іншого боку, повна орієнтація на апаратні засоби і програмне забезпечення IBM породила серйозну проблему «***успадкованих систем***» (*legacy systems*). Проблема полягає в тому, що виробничий процес не дозволяє припинити або навіть призупинити використання морально застарілих систем, щоб перевести їх на нову технологію. Багато серйозних дослідників сьогодні зайняті намаганнями вирішити цю проблему. Серйозність проблеми успадкованих систем з очевидністю підтверджує, що інформаційні системи (ІС) і БД, які лежать в їх основі, є занадто відповідальними і дорогими продуктами, щоб можна було дозволити собі їх переробку при зміні апаратної платформи або навіть системного програмного забезпечення (головним чином ОС і СУБД). Для цього програмний продукт повинен мати властивості легкої переносимості з одної апаратно-програмної платформи на іншу. Це, однак, не означає, що при переносі не

можуть знадобитися якісь зміни у вихідних текстах; головне, щоб такі зміни не означали корінної переробки системи.

Іншою природною вимогою до інформаційних систем є можливість їх розвитку за рахунок включення додаткових програмних та інформаційних компонентів.

Яким же чином можна одночасно задовольнити обидві ці вимоги вже на стадії проектування та розробки інформаційної системи? Відповіддю, яка здається правильною, є наступна: необхідно слідувати принципам «Відкритих Систем» і відповідних міжнародних або фактичних загальновизнаних стандартів у всіх необхідних видах забезпечення.

Основною ідеєю підходу відкритих систем є спрощення комплексування інформаційно-обчислювальних систем за рахунок міжнародної та національної стандартизації апаратних і програмних інтерфейсів. Головною спонукальною причиною розвитку концепції відкритих систем з'явився повсюдний перехід до використання комп'ютерних мереж і ті проблеми комплексування апаратно-програмних засобів, які викликав цей перехід. У зв'язку з бурхливим розвитком технологій глобальних комунікацій відкриті системи набувають ще більшого значення і масштабності.Прихильність концепції відкритих систем повинна забезпечити власникам ІС незалежність від конкретного постачальника. Орієнтуючись на продукцію компаній, які дотримуються стандартів відкритих систем, споживач, який набуває будь-який продукт такої компанії, не потрапляє до неї в рабство. Він може продовжити нарощування потужності своєї системи шляхом придбання продуктів будь-якої іншої компанії, що дотримується цих стандартів. Причому це стосується як апаратних, так і програмних засобів.

Практичною опорою системних і прикладних програмних засобів відкритих систем є стандартизована операційна система. В даний час такою системою є UNIX. Фірмам-постачальникам різних варіантів ОС UNIX в результаті тривалої роботи вдалося прийти до угоди про основні стандарти цієї операційної системи. Зараз всі поширені версії UNIX в основному сумісні по частині інтерфейсів, що надаються прикладним (а в більшості випадків і системним) програмістам. Незважаючи на появу нового претендента на стандарт системи Windows, саме UNIX залишиться основою відкритих систем у найближчі роки.

Технології та стандарти відкритих систем забезпечують реальну і перевірену практикою (в тому числі і в СНД) можливість виробництва системних і прикладних програмних засобів з властивостями ***мобільності*** (*portability*) та ***інтероперабельності*** (*interoperability*). Властивість мобільності означає порівняльну простоту перенесення програмної системи в широкому спектрі апаратно-програмних засобів, відповідних стандартам. Інтероперабельність означає спрощення комплексування нових програмних систем на основі використання готових компонентів зі стандартними інтерфейсами. Під цим розуміється дотримання певних правил або залучення додаткових програмних засобів, що забезпечують можливість взаємодії незалежно розроблених програмних модулів, підсистем або навіть функціонально завершених програмних систем.

Використання підходу відкритих систем вигідно і виробникам, і користувачам. Перш за все, відкриті системи забезпечують природне рішення проблеми поколінь апаратних і програмних засобів. Виробники таких засобів не змушують вирішувати всі проблеми заново; вони можуть, принаймні, тимчасово продовжувати кооперування, використовуючи існуючі компоненти.

Зауважимо, що при цьому виникає новий рівень конкуренції. Всі виробники зобов'язані забезпечити деяке стандартне середовище, але змушені добиватися його якомога кращої реалізації. Звичайно, стандарти не панацея: через якийсь час вони починають грати роль стримування прогресу, і тоді їх необхідно переглядати.

Перевагою для користувачів є те, що вони можуть поступово замінювати компоненти системи на більш досконалі, не втрачаючи працездатності системи. Зокрема, в цьому криється рішення проблеми поступового нарощування обчислювальних, інформаційних та інших потужностей комп'ютерної системи.

10.8.1 Міжнародні стандарти SQL

Розглянемо трохи докладніше, які стандарти можна використовувати при розробці інформаційної системи в даний час. Сьогодні програмне забезпечення ІС пишеться на деякій мові програмування, в неї вбудовуються оператори мови SQL і виклики бібліотечних функцій операційної системи.

Відповідно, перш за все, слід звертати увагу на ступінь стандартизованості використовуваної мови програмування. На сьогодняшній день прийняті міжнародні стандарти мов Fortran, Pascal, Ada, C, C++. Fortran, навіть у своєму найбільш розвиненому вигляді стандарту Fortran-95, не є мовою, відповідним для програмування інформаційних систем. У стандарт мови Pascal не включені засоби роздільної компіляції програм. Звичайно, в принципі можна оформити повний вихідний текст ІС у вигляді одного файлу, але навряд чи це розумно і практично. Мова Ada, взагалі кажучи, придатна для будь-яких цілей. На неї можна писати і інформаційні системи (що, до речі і роблять американські і деякі вітчизняні військові). Але аж надто вона громіздка.

На думку багатьох програмістів, найбільш зручний стандарт на сьогоднішній день існує для мови С. Досвід показує, що при грамотному використанні стандарту С ANSI/ISO проблеми переносу програм, пов'язані з особливостями апаратури або компіляторів, практично не виникають (якщо враховувати наявні в самому стандарті рекомендації по створенню переносних програм). Декількох років виявилося достатньо, щоб забезпечити повну відповідність стандарту практично всіх індустріальних компіляторів мови С.

Дуже важливо, що в стандарті ANSI С специфіковані не тільки мова, але й базові бібліотеки стандартної системи програмування (зокрема, основні компоненти бібліотеки вводу/виводу). Наявність стандарту на бібліотечні функції та його суворе дотримання в реалізаціях в ряді випадків дозволяє створювати не дуже складні додатки, що переносяться не тільки між різними апаратними платформами, але і між різними операційними середовищами.

Був, нарешті, прийнятий стандарт C++. Мабуть, це означає (принаймні, на це можна сподіватися), що через кілька років можна буде говорити про мобільне програмування на C++ в тому ж сенсі, в якому можна говорити про це сьогодні по відношенню до C. Маючи на увазі взаємодії з базами даних, ми говоримо майже виключно про мову SQL. SQL з самого свого зародження був складною мовою зі змішаною декларативно-процедурною семантикою і не ідеальним синтаксисом. Тим не менш, саме SQL став єдиною практично використовуваною мовою реляційних баз даних, хоча були й інші кандидати, зокрема, мова системи Ingres Quel (QBE).

У результаті тривалого процесу стандартизації мови до теперішнього часу прийнято багато міжнародних стандартів SQL, які наведені в таблиці 10.1.

Таблиця 10.1. Стандарти SQL

Рік	**Назва**	**Зміни**
1986	SQL-86	Перший варіант стандарту, прийнятий інститутом ANSI і схвалений ISO
1989	SQL-89	Невеликі зміни
1992	SQL-92	Значні зміни (ISO 9075); рівень Entry Level стандарту SQL-92 був прийнятий як стандарт FIPS 127-2
1999	SQL:1999	Додана підтримка регулярних виразів, рекурсивних запитів, підтримка тригерів, базові процедурні розширення, нескалярні типи даних і деякі об'єктно-орієнтовані можливості
2003	SQL:2003	Введені розширення для роботи з XML-даними, віконні функції (застосовувані для роботи з OLAP-базами даних), генератори послідовностей і засновані на них типи даних
2006	SQL:2006	Функціональність роботи з XML-даними значно розширена. З'явилася можливість спільно використовувати в запитах SQL і XQuery
2008	SQL:2008	Поліпшено можливості віконних функцій, усунуті деякі неоднозначності стандарту SQL:2003

Якщо прагнути до досягнення переносимості додатку, то слід намагатися обмежити себе мінімально достатнім набором стандартних засобів. У разі, коли нестандартне рішення деякої операційної системи дозволяє істотно оптимізувати роботу інформаційної системи, розумно гранично локалізувати ті місця програми, в яких це рішення використовується.

Те, про що ми говорили досі, відноситься здебільшого до порівняно невеликих і централізованих систем (звичайно, навіть централізовані системи сьогодні, як правило, ґрунтуються на архітектурі «клієнт-сервер»). У деякому роді системи баз даних, засновані на архітектурі «клієнт-сервер», є наближенням до розподілених ІС, звичайно, істотно спрощеним наближенням, але зате не вимагають рішення основного набору проблем дійсно розподілених баз даних. Основною проблемою систем, заснованих на архітектурі «клієнт-сервер», є те, що відповідно до концепції відкритих систем від них потрібна мобільність в якомога ширшому класі апаратно-програмних рішень відкритих систем. Навіть якщо обмежитися UNIX-орієнтованими локальними мережами, в різних мережах застосовується різна апаратура та протоколи зв'язку. Спроби створення систем, що

підтримують всі можливі протоколи, призводить до їх перевантаження мережевими деталями на шкоду функціональності.

Однак, якщо мати на увазі більш масштабні проекти, реалізовані не в локальних, а як мінімум корпоративних мережах, то проблема забезпечення інтероперабельності окремо спроектованих і розроблених програмних компонентів істотно ускладнюється. При створенні великих розподілених програмних систем практично неможливо зобов'язати численних розробників слідувати загальній технології (може бути й тому, що загальноприйнята технологія сьогодні відсутня). У кінцевому ж рахунку система повинна бути інтегрована і в міру можливості мати достатню ефективність. Ще більш складний аспект цієї проблеми пов'язаний з можливістю використання різних представлень даних в різних вузлах неоднорідною мережі. У різних комп'ютерах може існувати різна адресація, подання чисел, кодування символів тощо.

10.8.2 **Міжнародні стандарти протоколів для розподілених систем обробки**

Простежимо найбільш важливі віхи в процесі забезпечення інтероперабельності розподілених програмних компонентів. Можливо, це приватна точка зору, але здається, що першим кроком був механізм «програмних гнізд» (sockets), розроблений в університеті Берклі. Основним недоліком програмних гнізд є те, що це чисто транспортний механізм. Зокрема, подання переданих по мережі даних є машинно-залежною.

Наступним важливим кроком була поява протоколу віддаленого виклику процедур (Remote Procedure Calls - RPC) і його реалізація. Ідея цього механізму дуже проста. У більшості випадків взаємодія програмних компонентів є асиметричною. Практично завжди можна виділити клієнта, якому потрібна деяка послуга, і сервер, який таку послугу здатний надати. Більш того, як правило, клієнт не може продовжувати своє виконання, поки сервер не виконає дію, яку від нього вимагають. Іншими словами, типовою взаємодією клієнта та сервера є процедурна взаємодія. Спільним рішенням проблеми мобільності систем, заснованих на архітектурі «клієнт-сервер» є опора на програмні пакети, що реалізують протоколи RPC. При використанні цих засобів, звернення до сервісу в віддаленому вузлі виглядає як звичайний виклик процедури. Засоби RPC, в яких, природно, міститься вся інформація про специфіку апаратури локальної мережі та мережевих протоколів, переводять виклик в послідовність мережевих взаємодій. Тим самим, специфіка мережного середовища і протоколів прихована від прикладного програміста.

При виклику віддаленої процедури програми RPC виконується перетворення форматів даних клієнта в проміжні машинно-незалежні формати і потім перетворення у формати даних сервера. При передачі відповідних параметрів виконуються аналогічні перетворення.

Якщо система реалізована на основі стандартного пакета RPC, вона може бути легко перенесена в будь-яке відкрите середовище.

З точки зору клієнтської програми звертання до сервера нічим не відрізняється від виклику локальної процедури. При реалізації механізму виклику віддалених процедур на підставі специфікації інтерфейсу процедури на стороні клієнта генерується локальний представник віддаленої процедури - stub, а на стороні сервера - спеціалізований перехідник - proxy.

Протокол RPC та сервери баз даних

Термін «сервер баз даних» зазвичай використовують для позначення всієї СУБД, заснованої на архітектурі «клієнт-сервер», включаючи і серверну, і клієнтську частини. Такі системи призначені для зберігання і забезпечення доступу до баз даних.

Хоча зазвичай одна база даних цілком зберігається в одному вузлі мережі та підтримується одним сервером, сервери баз даних являють собою просте і дешеве наближення до розподілених баз даних, оскільки загальна база даних доступна для всіх користувачів локальної мережі.

Принципи взаємодії між клієнтськими і серверними частинами для доступу до БД

Доступ до бази даних від прикладної програми або користувача проводиться шляхом звернення до клієнтської частини системи. В якості основного інтерфейсу між клієнтською і серверною частинами виступає мова баз даних SQL.

Це мова по суті справи представляє собою поточний стандарт інтерфейсу СУБД у відкритих системах. Збірна назва SQL-сервер відноситься до всіх серверів баз даних, заснованих на SQL. Дотримуючись обережності при програмуванні можна створювати прикладні інформаційні системи, мобільні в класі SQL-серверів.

Сервери баз даних, інтерфейс яких базується виключно на мові SQL, мають переваги та недоліки. Очевидна перевага - стандартність інтерфейсу. В ідеалі, хоча поки це не зовсім так, клієнтські частини будь-якої SQL-орієнтованої СУБД могли б працювати з будь-яким SQL-сервером незалежно від того, хто його зробив.

Недолік теж досить очевидний. При такому високому рівні інтерфейсу між клієнтською і серверною частинами системи на стороні клієнта працює занадто мало програм СУБД. Це нормально, якщо на стороні клієнта використовується малопотужна робоча станція. Але якщо клієнтський комп'ютер має достатню потужність, то часто виникає бажання покласти на нього більше функцій керування базами даних, розвантаживши сервер, який є вузьким місцем всієї системи.

Одним з перспективних напрямків СУБД є гнучке конфігурування системи, при якому розподіл функцій між клієнтською і користувацькою частинами СУБД визначається при установці системи.

10.8.3 **Технології об’єктного зв’язку даних**

Уніфікація взаємодії прикладних компонентів з ядром ІС для клієнт-серверних систем дозволила виробити аналогічні рішення і для інтеграції розрізнених локальних БД під керуванням «настільних» СУБД в складні децентралізовані гетерогенні розподілені системи - об'єктне зв'язування даних.

З вузької точки зору технологія об'єктного зв'язування вирішує завдання забезпечення доступу з однієї локальної БД, відкритої одним користувачем, до даних в іншій локальної БД, можливо знаходиться на іншій машині та відкритої іншим користувачем.

Об'єктний доступ до даних DOA:

- сучасні «настільні» СУБД підтримують технологію «об'єктного доступу до даних» - DAO;
- технічно DAO заснована на протоколі ODBC (рис. 10.7.), який прийнятий як стандарт для доступу до даних на SQL-серверах і до будь-яких даних під керуванням реляційних СУБД - ODBC-драйвера;
- сучасні «настільні» СУБД забезпечують прямий доступ до об'єктів зовнішніх БД «своїх» форматів - зв'язані об'єкти;
- для доступу до БД найбільш поширених форматів і файлів електронних таблиць - ISAM (Indexes Sequential Access Method)-драйвера.

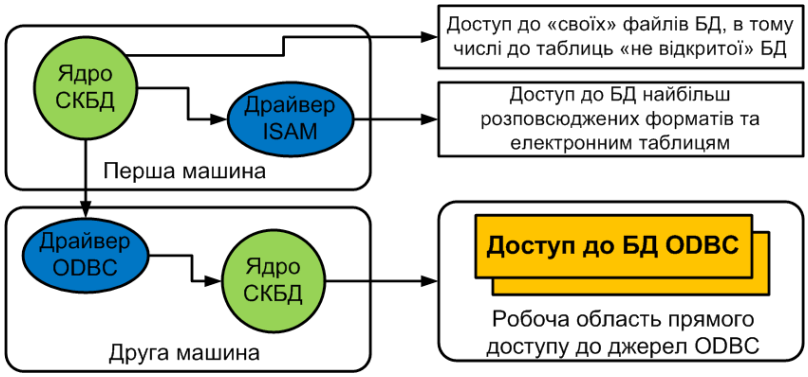


Рис. 10.7. Принцип доступу до даних через ODBC

Недоліки зв’язку даних:

- при великих обсягах даних у зв'язаних таблицях мережевий трафік істотно збільшується;
- відсутність надійних механізмів безпеки даних;
- відсутність надійних механізмів забезпечення обмежень цілісності.

Переваги протоколів віддаленого виклику процедур

Згадувані вище протоколи віддаленого виклику процедур особливо важливі в системах керування базами даних, заснованих на архітектурі «клієнт-сервер».

По-перше, використання механізму віддалених процедур дозволяє дійсно перерозподіляти функції між клієнтською і серверною частинами системи, оскільки теоретично будь-який компонент системи може розташовуватися і на стороні сервера, і на стороні клієнта.

По-друге, механізм віддаленого виклику приховує відмінності між взаємодіючими комп'ютерами. Фізично неоднорідна локальна мережа комп'ютерів приводиться до логічно однорідної мережі взаємодіючих програмних компонентів. В результаті користувачі не зобов'язані серйозно піклуватися про разову закупівлю сумісних серверів і робочих станцій.

Типове розділення функцій між клієнтами і серверами

У типовому на сьогодняшній день випадку на стороні клієнта СУБД працює тільки таке програмне забезпечення, яке не має безпосереднього доступу до баз даних, а звертається для цього до сервера з використанням мови SQL.

У деяких випадках хотілося б включити до складу клієнтської частини системи деякі функції для роботи з «локальним кешем» бази даних, тобто з тією її частиною, яка інтенсивно використовується клієнтською прикладною програмою. У сучасній технології це можна зробити тільки шляхом формального створення на стороні клієнта локальної копії сервера бази даних і розгляду всієї системи як набору взаємодіючих серверів.

З іншого боку, іноді хотілося б перенести більшу частину прикладної системи на сторону сервера, якщо різниця в потужності клієнтських робочих станцій і сервера надто велика. Взагалі, при використанні RPC це зробити неважко. Але потрібно, щоб базове програмне забезпечення сервера дійсно дозволяло це. Зокрема, при використанні ОС UNIX проблеми практично не виникають.

Вимоги до апаратних можливостей і базовому програмному забезпеченню клієнтів і серверів

З попередніх міркувань видно, що вимоги до апаратури та програмного забезпечення клієнтських і серверних комп'ютерів розрізняються залежно від виду використання системи.

Якщо поділ між клієнтом і сервером досить жорсткий (як у більшості сучасних СУБД), то користувачам, працюючим на робочих станціях або персональних комп'ютерах, абсолютно все одно, яка апаратура та операційна система працюють на сервері, лише б він справлявся з виникаючим потоком запитів.

Але можуть виникнути потреби перерозподілу функцій між клієнтом і сервером, тоді вже зовсім не все одно, які операційні системи використовуються.

Протокол зовнішнього представлення даних XDR

Дуже корисною складовою (щодо незалежної від інших складових) сімейства протоколів RPC є протокол зовнішнього подання даних (External Data

Representation - XDR). У цьому протоколі специфікована машинно-незалежне представлення даних, що розуміються механізмом RPC. Ідея полягає в тому, що при передачі параметрів виклику віддаленої процедури і при отриманні її відповідних параметрів відбувається подвійне перетворення даних спочатку з вихідного машинно-залежного формату в формат XDR, а потім з цього формату в машинно-залежний цільовий формат. В результаті взаємодіючі програми позбавлені потреби знання специфіки машинно-залежних уявлень даних. Можливо, саме протокол XDR має визначальним значенням в зв'язці протоколів виклику віддалених процедур (хоча, звичайно, можливості XDR досить обмежені).

10.8.4 **Брокерна архітектура CORBA**

Нарешті, мабуть найбільш перспективним на сьогодняшній день є підхід, запропонований та специфікований міжнародним консорціумом Object Management Group (OMG). Знову-таки, основна ідея проста. Як би ми не прагнули виробити єдині мовні засоби для розробки розподілених інформаційних систем, схоже реально уніфікація неможлива. Є багато різних підходів, кожен з яких в чомусь виграє у інших. Не можна сказати, що якийсь підхід є визначальним. Мабуть, єдине, на чому сходиться переважна більшість дослідників і розробників розподілених програмних систем, це схильність до використання парадигми об'єктної орієнтації.

Переваги цієї парадигми в тому, що об'єктно-орієнтований стиль проектування та розробки програмних систем (і в особливості інформаційних систем) стимулює повторне використання програмних компонентів (за рахунок наявності механізму спадкування класів), забезпечує незалежність використання інформаційних ресурсів від особливостей їх реалізації (за рахунок застосування принципів інкапсуляції), нарешті, при проектуванні та розробці інформаційних систем дозволяє виробляти комплексну розробку структур даних і керуючих ними процедур (із застосуванням технології об'єктно-орієнтованих систем баз даних).

Але проблема полягає в тому, що відсутня загальна об'єктна модель. У різних об'єктно-орієнтованих системах програмування і системах баз даних набагато більше спільного, ніж різного, але їх відмінності часто принципові настільки, що об'єкти, розроблені та створені в різних системах, не здатні взаємодіяти. Вони не інтероперабельні. Це, звичайно, дуже погано. І особливо погано з тієї причини, що постійне нарощування можливостей Internet робить принципово можливим використання інформаційних ресурсів незалежно від їх реального розташування.

Група OMG пропонує обмежене, але практично досяжне вирішення цієї проблеми. Це загальна архітектура Object Management Architecture (OMA) та її конкретне втілення Common Object Request Broker Architecture (CORBA). Перш за все, оскільки на сьогодні відсутнє (і навряд чи з'явиться в найближчому майбутньому) об'єктна модель, яка могла б служити «спільним дахом» для існуючих об'єктно-орієнтованих мов і систем програмування, то єдиним практично можливим виходом було вироблення мінімальної об'єктної моделі, що володіє обмеженими можливостями, але має явні аналоги в найбільш поширених об'єктних системах. В архітектурі CORBA така мінімальна модель називається Core Object Model, і їй

відповідає мова специфікації (не програмування!) Інтерфейсів об'єктів Interface Definition Language (IDL).

Щоб забезпечити можливість взаємодії об'єкта, існуючого в одній системі програмування, з деяким об'єктом з іншої системи програмування, у вихідний текст першого об'єкта (правильніше сказати, її класу) повинна бути поміщена специфікація на мові IDL інтерфейсу того об'єкта, метод якого повинен бути викликаний. Аналогічна специфікація повинна бути поміщена на стороні об'єкта. Далі вихідні тексти повинні бути оброблені відповідним прекомпілятором IDL (таких прекомпіляторов повинно матися рівно стільки, скільки підтримується інтероперабельних об'єктних середовищ; специфіковані правила вбудовування IDL в програми на мовах C, C++ і Smalltalk). Специфікація інтерфейсу об'єкта на мові IDL являє собою головним чином набір сигнатур методів об'єкта. У кожній сигнатурі визначається назва методу, список імен і типів даних вхідних параметрів, а також список назв і типів даних результатів виконання методу. Крім того, IDL дозволяє визначати структурні типи даних (звичайно, прекомпілятор IDL на основі визначень таких типів виробляє специфікації типів цільової системи програмування).

Що ж є результатом роботи прекомпілятора? Все дуже схоже на RPC. На стороні клієнта генерується текст об'єкта-stub, який грає роль локального представника віддаленого об'єкта. Робота з цим об'єктом-посередником відбувається за правилами відповідної системи програмування: в середовищі C++ stub - це об'єкт C++, в середовищі Smalltalk stub - це об'єкт Smalltalk тощо. З іншого боку, в коді, що генеруються в stub, закладені знання про те, що потрібно зробити для звернення до методу віддаленого об'єкту.

На стороні сервера генерується текст об'єкта-sceleton, свого роду посередника при доступі до віддаленого об'єкту. З одного боку, sceleton «знає», що звернення є віддаленим. З іншого боку такий посередник вміє звертатися до віддаленого об'єкту за правилами його системи програмування.

Звичайно, для доставки повідомлення від клієнта до сервера і отримання відповідних результатів повинен існувати системний (вірніше, напівсистемний - middle-ware) компонент, що відповідає саме за виконання таких функцій. В архітектурі CORBA такий компонент називається брокером об'єктних заявок (Object Request Broker - ORB). У функції ORB головним чином входить передача даних в машинно-незалежному форматі від клієнта серверу й від сервера клієнтові. Крім того, ORB відповідає за правильні вказівки мережевої адреси об'єкта-сервера.

Інтерфейс між stub, sceleton і ORB заснований на специфікаціях інтерфейсу прикладного рівня (Application Programmer Interface - API) архітектури CORBA. Цей інтерфейс відкритий для програмістів. На ньому ґрунтується метод динамічного виклику об'єктів без потреби специфікації їх інтерфейсів мовою IDL. Природно, програмування на рівні API ORB є більш обтяжуючою справою, ніж, при використанні IDL. Однак іноді інтерфейси віддалених об'єктів, які викликаються, просто невідомі в статиці, їх можна отримати тільки на стадії виконання програми, і тоді не обійтися без використання API .

OMG не володіє правами міжнародної організації по стандартизації. Тим не менш, прийняті в цій організації документи мають виключно високий міжнародний авторитет і стають фактичними стандартами. Проблемою стандарту OMG CORBA було те, що була відсутня стандартизація взаємодій рівня ORB-ORB. В результаті ORB-и, які вироблені різними компаніями, не могли спільно працювати. Вже існуючі ORB-и не розуміли один одного. У 1997 році OMG прийняла новий стандарт CORBA-2, в якому специфікований протокол взаємодій між ORB-ами. Реалізації цього протоколу вже з'явилися, і є надія на реальне використання інформаційних ресурсів Internet. Зараз широко використовується стандарт CORBA 3.0, який вийшов у 1999 році. І вже сьогодні існує багато реалізацій брокерів (Borland Enterprise Server, VisiBroker Ed, MICO, omniORB, ORBit2, JacORB, TAO, Orbacus, Orbix, PolyORB).

Звичайно, CORBA - це далеко не все, що потрібно. На сьогодні це складна і далеко незрозуміла проблема - семантична інтероперабельність об'єктних систем.

ПЕРЕЛІК ПОСИЛАНЬ

Основна література

1. Струбицький П.Р., Бондар О.В. Програмна інженерія – Навч. посібник –Тернопіль: ТАНГ, 2004 -78с.
2. Липаев В.В. Программная инженерия.Методологические основы: Учебник.- ТЕИС, 2006-608с.
3. Програмна інженерія.-/Лавріщева К. М.-Підручник.-К.: Академперіодика, 2008.- 319 с.
4. Э. Танненбаум, М. Ван Стеен. Распределенные системы. Принципы и парадигмы. СПб: Питер 2003 г.
5. Д. Саймино. Сети интранет: внутреннее движение. М. "Book Media Publisher". 1997 г.
6. Дж. Вакка. Безопасность интранет. М. "Book Media Publisher". 1997 г.
7. Создание intranet. Официальное руководство Microsoft. “BHV – Санкт Петербург”. 1998 г.
8. Скотт Хилайер, Дэниел Мизик Программирование Active Server Pages. Москва. Русская Редакция, 1999 г.
9. Спирин Н.А., Лавров В.В. Методы планирования и обработки результатов инженерного эксперимента: конспект лекции Н.А. Спирина – Екатеринбург: ГОУ ВПО УГТУ – УПИ, 2004.- 257 с.
10. Тимоти Бадд Объектно-ориентированное программирование в действии = An Introduction to Object-Oriented Programming. — СПб.: «Питер», 1997. — 464 с. — (В действии). — ISBN 5-88782-270-8
11. Петрушин В.Н, Ульянов М.В. Информационная чувствительность компьютерных алгоритмов. — М.: ФИЗМАТЛИТ, 2010. — 224 с. [стр. 138–163].
12. Айвазян С. А., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: Основы моделирования и первичная обработка данных. Справочное издание. – М.: Финансы и статистика, 1983. – 471 с.
13. Основы инженерии качества программных систем.-Андон Ф. И., Коваль Г. И., Коротун Т. М., Лаврищева Е. М., Суслов В. Ю.-Киев:Академпериодика,2007.-680с.
14. Баженова, И. Ю. Язык программирования Java / И.Ю.Баженова. - М.: Диалог-МИФИ, 2008. - 254 с.
15. Бумфрей Ф, Диренцо О, Дакетт Й. XML. Новые перспективы WWW. - Издательство: "ДМК Пресс", 2006. - 688 с.
16. Кингсли, Х.Э. JavaScript в примерах / Х.Э.Кингсли. - Издательство: "ДМК Пресс", 2009. - 272 с.

Додаткова література

1. http://bin-login.name/pba/book.pdf та http://hpcc.kpi.ua/hpc-book
2. http://www.simulation.kiev.ua/dbis/lection19.html
3. Таненбаум, Э. Распределенные системы: Принципы и парадигмы / Э. Таненбаум, М. Стеен. - СПб: Питер, 2003 - 877с.
4. Хабибуллин, И.Ш Самоучитель Java 2 / И.Ш. Хабибуллин. - СПб.: БХВ - Петербург, 2005 - 720с.
5. Фуфаев, Э.В. Разработка и эксплуатация удаленных баз данных : Учебник / Э.В. Фуфаев, Д.Э. Фуфаев. - 2-е изд.стереотип.- М.: , 2009 - 256с.
6. Цимбал, А Технология создания распределенных систем / А. Цимбал, М. Аншина. - СПб: Питер, 2003 - 576с.