Розподілені системи

1. Визначення розподіленої системи. Поняття прозорість, відкритість, масштабованість

Розподілена система - це набір незалежних комп'ютерів, представлений користувачеві єдиною об'єднаною системою.

У цьому визначенні є два однаково важливі моменти:

1. Стосовно апаратури - всі машини автономні.
2. Стосовно програмного забезпечення - користувачі думають, що мають справу з єдиною системою.

Основне завдання розподілених систем - полегшити користувачам доступ до віддалених ресурсів і забезпечити їх спільне використання, регулюючи цей процес. Ресурси можуть бути віртуальними, проте традиційно вони включають в себе принтери, комп'ютери, пристрої зберігання даних, файли і дані. Web-сторінки і мережі також входять до цього списку. Існує безліч причин для спільного використання ресурсів. Одна з очевидних - це економічність. Наприклад (візьмемо якусь локальну ІС) набагато дешевше дозволити спільну роботу з принтером декількох користувачів, ніж купувати і обслуговувати окремий принтер для кожного користувача.

Прозорість - полегшує взаємодію розподілених прикладних програм. Цей принцип дозволяє сховати складність реалізації розподілених систем від користувача. Внаслідок цього, розподілена система для користувача стає більше зручною.

Відкрита розподілена система – це система, яка пропонує стандартні засоби й служби доступу до системи широкому колу користувачів, що використовують стандартні синтаксис і семантику всіх протоколів взаємодії.

Масштабування – можливість додавання в розподілену систему нових комп'ютерів для збільшення продуктивності системи, що пов'язане з поняттям балансування навантаження (load balancing) на сервери системи. До масштабування також відносяться питання ефективного розподілу ресурсів серверів, що обслуговують запити клієнтів.

Система може бути масштабованою :

- стосовно її розміру (підключення додаткових користувачів)
- географічно (користувачі й ресурси рознесені в просторі)

2. Концепції апаратних рішень. Мультипроцесори. Гомогенні та гетерогенні мультикомп’ютерні системи

Не дивлячись на те що всі розподілені системи містять по декілька процесорів, існують різні способи їх організації в систему. 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. 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.

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.

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.

У **гомогеннх системах** , відомих під назвою системних мереж (System Area Networks, SAN), вузли вмонтовуються у великій стійці і з'єднуються єдиною, зазвичай високошвидкісною мережею. Дві популярні топології — квадратні грати і гіперкуби. Грати прості для розуміння і зручні для розробки на їх основі друкарських плат. Вони чудово підходять для вирішення двомірних завдань, наприклад завдань теорії графів або комп'ютерного зору (очі робота, аналіз фотографій).

Гетерогенні системи – це ті системи як містять цілий набір незалежних комп'ютерів, з'єднаних різноманітними мережами, тобто системи з неоднорідними характеристиками. Найбільше число існуючих в даний час розподілених систем побудоване по схемі гетерогенних мультикомп’ютерних систем . Це означає, що комп'ютери, що є частинами цієї системи, можуть бути украй різноманітні, наприклад, за типом процесора, розміром пам'яті і продуктивності каналів введення-висновку.

3. Концепції програмних рішень. Розподілені операційні системи. Мережеві операційні системи

Розподілені системи насамперед виконують роль менеджера ресурсів (resource manager) апаратного забезпечення, який допомагає багатьом користувачам і додаткам спільно використовувати такі ресурси як пам"ять, процесори, мережу і різні дані. Інша роль полягає в тому, що вона приховує складність апаратного забезпечення, на базі якого вона побудована.

2 категорії ОС для розподілених комп"ютерів:

- Сильно зв"язані.ОС зазвичай працює з одним, глобальним представленням ресурсів, якими вона керує. Їх зазвичай називають розподіленими операційними системами (Distributed Operating System, DOS).
- Слабко зв"язані. Вони більше нагадують набір окремих ОС, які працюють спільно і надають один одному доступ до якоїсь інформації. Їх називають мережевими операційними системами (Network Operating Systems, NOS).

Розподілені операційні системи

Є два типи ОС - мультипроцесорні та мультикомп"ютерні. Вони відрізняються від звичайних ОС лише тим, що вони підтримують виконання кількох процесів одночасно.

Основна задача ОС - надати різним процесам легкий доступ до спільних ресурсів типу процесора, пам"яті, дисків і тд. Тобто є додатки А і В і кожен з них користується ресурсами так,

ніби ресурси повністю в їх розпорядженні. Але це має відбуватись таким чином, щоб жодна з програм не могла попсувати дані іншої програми.

Мультипроцесорні ОС

Важливо можливістю ОС є можливість підтримки кількох процесорів, які мають доступ до спільної пам"яті. Все, що потрібно - це захистити дані від одночасного доступу до них кількох процесів. Це реалізується за допомогою семафорів і моніторів. Семафор - це ціле число, яке можна збільшити або зменшити. Якщо семафор рівний нулю, то процес продовжується, якщо додатній, то процес блокується. Але семафори часто приводять до помилок. Їм на заміну прийшли монітори. Формально це об"єкт, який надає доступ до даних через якісь свої методи і таким чином має більший контроль над тим, що відбувається. В програмуванні такі конструкції пов"язані з класами mutex.

Мультикомп"ютерні ОС

Вони є набагато складнішими і різноманітнішими, оскільки ресурси тепер не є спільними. Єдиним видом зв"язку тепер є передача повідомлень. Кожен вузол такої системи має ядро для керування локальними ресурсами та модуль для міжпроцесорної взаємодії.

Мережеві ОС

Ці ОС відрізняютсья тим, що їхнє апаратне забезпечення не повинно бути гомогенним і керуватися як одна єдина система. Навпаки, зазвичай вони будуються для набору однопроцесорних систем, кожна з яких має власну ОС. Машини і їх ОС можуть бути різними, але вони просто з"єднані в мережу. Крім того, мережева ОС дозволяє користувачам використовувати служби, які знаходяться на інших машинах.

4. Програмне забезпечення проміжного рівня

Розподілені системи не призначені для керування незалежних комп"ютерів, а мережеві ОС не дають уявлення про єдину злагоджену систему. Вирішення цієї проблеми є у розробці пз, яке в мережевих ОС дозволяє більш менш приховати від користувача різнорідність набору апаратних платформ та підвищити прозорість розподілення. Це і називається ПЗ проміжного рівня.

Багато які розподілені додатки допускають використання зв"язку через сокети та обмін повідомленнями. Крім того додатки часто користуються інтерфейсами локальних файлових систем. Для таких завдань розробляють пз проміжного рівня, щоб забезпечити вищий рівень абстракції при розробці додатків. Тобто це пз стоїть між додатком і мережевою ОС.

Таке пз дуже полегшило роботу програмістам, оскільки тепер додатки можна легко інтегрувати між собою якимось єдиним способом.

Спочатку роль такої системи виконували файли - в них записувалась інформація, яку потім читали янші процеси.

Потім таку систему замінили на Remote Procedure Calls. Тобто додатки отримали можливість передати параметри іншому процесу, на якому виконається якийсь метод, а тоді отримати результат цього методу.

Звичайно, в пз проміжного рівня треба подбати про такі служби як communication facilities - високорівневу передачу повідомлень; іменування - щоб можна було швидко і ефективно знайти потрібний об"єкт; persistence - засоби зберігання даних; distributed transactions - можливість багатьох операції читання та запису в ході виконання однієї атомарної операції; засоби захисту.

5. Модель клієнт-сервер. Клієнти і сервери.Поділ додатків за рівнями. Варіанти архітектури клієнт-сервер

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- **набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;**
- **набір клієнтів, які використовують сервіси, що надаються серверами;**
- **мережа, яка забезпечує взаємодію між клієнтами та серверами.**

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Обов'язки та взаємодія

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- **рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;**

- **прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;**
- **рівень управління даними, який забезпечує зберігання даних та доступ до них.**

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- **модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;**
- **модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.**

Трирівнева архітектура

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка застосунку. Програми проміжного рівня можуть функціювати під управлінням спеціальних серверів застосунків, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення — звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає). Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

6. Рівні протоколів. Низькорівневі протоколи. Транспортні протоколи. Протоколи верхнього рівня

РІВНІ ПРОТОКОЛІВ В МОДЕЛІ OSI (Open Systems Interconnection Reference Model)

Модель OSI складається з 7-ми рівнів, розташованих вертикально один над іншим. Кожен рівень може взаємодіяти тільки зі своїми сусідами й виконувати відведені тільки йому функції.

Модель TCP/IP складається з 4 рівнів, які містять в собі рівні моделі OSI:

1. ***Рівень мережевих інтерфейсів***
2. ***Мережевий рівень***
3. ***Транспортний рівень***
4. ***Прикладний рівень***

Фізичний рівень:

- **передача бітів по фізичних каналах;**
- **формування електричних сигналів;**
- **кодування інформації;**
- **синхронізація;**
- **модуляція.**

Функції фізичного рівня реалізуються у всіх пристроях, підключених до мережі. З боку комп'ютера функції фізичного рівня виконуються мережним адаптером або послідовним портом. Реалізується апаратно.

Канальний рівень

Одним із завдань канального рівня (Data Link layer) є перевірка доступності середовища передачі. Інше завдання канального рівня - реалізація механізмів виявлення й корекції помилок. Для цього на канальному рівні біти групуються в набори, називані кадрами (frames).

Реалізується апаратно.

Мережний рівень (Network layer) служить для утворення єдиної транспортної системи, що поєднує кілька мереж, причому ці мережі можуть використовувати зовсім різні принципи передачі повідомлень між кінцевими вузлами й мати довільну структуру зв'язків. Функції мережного рівня досить різноманітні.

Транспортний рівень (Transport layer) забезпечує додаткам або верхнім рівням стека - прикладному й сеансовому - передачу даних з тим ступенем надійності, що їм потрібно.

Сеансовий рівень (Session layer) забезпечує керування діалогом: фіксує, яка зі сторін є активною в даний момент, надає засоби синхронізації.

Основні завдання сеансового рівня:

1. **Встановлення способу обміну повідомленнями (дуплексний або напівдуплексний);**
2. **Синхронізація обміну повідомленнями**
3. **Організація "контрольних точок" діалогу.**

Представницький рівень має справу з формою подання переданої по мережі інформації, не міняючи при цьому її змісту.

Основні завдання представницького рівня:

1. **перетворення даних із зовнішнього формату у внутрішній;**
2. **шифрування й розшифрування даних.**

Прикладний рівень (Application layer) - це в дійсності просто набір різноманітних протоколів, за допомогою яких користувачі мережі одержують доступ ресурсів, що розділяються.

Основні завдання прикладного рівня:

1. **ідентифікація, перевірка прав доступу;**
2. **принт- і файл-сервіс, пошта, вилучений доступ і т.д.**

НИЗЬКОРІВНЕВІ ПРОТОКОЛИ МОДЕЛІ OSI

Фізичний рівень

Канальний рівень

Мережний рівень

НИЗЬКОРІВНЕВІ ПРОТОКОЛИ МОДЕЛІ OSI

Сеансовий рівень

Представницький рівень

Прикладний рівень

ТРАНСПОРТНИЙ РІВЕНЬ

Транспортний рівень (Transport layer) забезпечує додаткам або верхнім рівням стека - прикладному й сеансовому - передачу даних з тим ступенем надійності, що їм потрібно. Модель OSI визначає п'ять класів сервісу, надаваних транспортним рівнем. Ці види сервісу відрізняються якістю надаваних послуг: терміновістю, можливістю відновлення перерваного зв'язку, наявністю засобів націлити декількох з'єднань між різними прикладними протоколами через загальний транспортний протокол, а головне - здатністю до виявлення й виправлення помилок передачі, таких як перекручування, втрата й дублювання пакетів.

Основні завдання транспортного рівня:

1. **розбивка повідомлення сеансового рівня на пакети, їхня нумерація;**
2. **буферизація прийнятих пакетів;**
3. **впорядочення пакетів, що прибувають;**
4. **адресація прикладних процесів;**
5. **керування потоком.**

Як правило, всі протоколи, починаючи із транспортного рівня й вище, реалізуються програмними засобами кінцевих вузлів мережі - компонентами їх мережних операційних систем. Як приклад транспортних протоколів можна привести протоколи TCP і UDP стека TCP/IP і протокол SPX стека Novell.

Протоколи транспортного рівня TCP/IP надають транспортні послуги прикладним процесам. Основними протоколами транспортного рівня TCP/IP є протокол керування передачею TCP (Transmission Control Protocol) і протокол користувальницьких дейтаграм UDP (User Datagram Protocol). Транспортні послуги цих протоколів суттєво відрізняються. Протокол UDP доставляє дейтаграми без установлення з'єднання. При цьому він не гарантує їхнього доставляння. Протокол TCP забезпечує надійне доставляння байтових потоків (сегментів) із попереднім встановленням транспортного дуплексного з'єднання (віртуального каналу) між модулями TCP мережевих комп'ютерів. Для розв'язання

транспортних завдань протоколи TCP та UDP при передачі даних формують і додають до даних свої заголовки обсягом 20 байт та 8 байт відповідно.

7. Віддалений виклик процедур. Базові операції RPC. Передача параметрів

Основою більшості розподілених систем є явний обмін повідомленнями між процесами. Однак, процедури send і receive не приховують взаємодій, а це не дуже зручно.

Для вирішення цієї проблеми було запропоновано дозволити програмам викликати процедури, які знаходяться на інших машинах. На машині А виконується процес. Він передає машині В аргументи і запускає якусь процедуру, і тим часом призупиняється. Коли на машині В процес закінчується, то він повертає результат машині А і процес на машині А відновлюється. Програміст нічого навіть не помітить. Цей метод називається віддаленим викликом процедур - Remote Procedure Call, RPC.

Базова ідея проста та елегантна, складнощі виникають вже при реалізації. Проте програмісти з цим справились і цей метод є дуже поширеним зараз.

Ідея RPC полягає в тому, щоб віддалений виклик методу виглядав так само, як і локальний. Для цього в бібліотеці є заглушки для таких методів.

На клієнті ця заглушка виглядає так, як звичайна функція, але вона не виконує код, а запаковує параметри в повідомлення, надсилає його на сервер(send), блокується, поки не отримає від сервера відповідь(receive), а тоді повертає потрібне значення або виконує потрібні присвоєння параметрам, які були передані за посиланням.

На сервері теж є заглушка заглушка цього метода і вона працює подібно. Отримує дані від клієнта, виконує метод, запаковує дані і надсилає клієнту.

Цей підхід гарний тим, що клієнт абсолютно нічого не знає про пересилання повідомлень. Для клієнта це виглядає як звичайний виклик функції.

Передача параметрів по значенню

Розглянемо метод int add(int i, int j). Клієнтська заглушка отримує два параметри і запаковує їх у повідомлення. В повідомлення вона також поміщає номер процедури,яку потрібно викликати. Коли повідомлення приходить на сервер, то серверна заглушка шукає в повідомленні інформацію про те, який метод викликати, а тоді робить відповідний виклик методу. Після закінчення методу серверна заглушка отримує результат методу - суму двох чисел. Її запаковують в повідомлення і надсилають назад до клієнта.

Така схема гарно працює,коли це дві однакові машини, а типи лише скалярні. Але коли машини різні, то виникають проблеми з різними кодовими таблицями(ASCII,

EBCDIC і тд), різними порядками байтів (big/small endian). Тоді треба придумувати якусь крутішу схему.

Передача параметрів по посиланню

Тут треба трошки серйозніше підійти до проблеми. Просто передати вказівники чи референси ми не можем, бо на іншій машині це не матиме значення. Треба спочатку скопіювати в якийсь об"єкт дані, на які вказує посилання чи референс. Цей об"єкт треба переслати серверу і вже на сервері при виклику метода посилатись на той скопійований об"єкт. Таким же чином їх слід повернути назад до клієнта. І вже після отримання результату, клієнтська заглушка мусить скопіювати дані туди,куди вказує посилання чи референс.

Звичайно, це не вирішує проблеми у випадках, коли нам треба передати не лінійний масив, а якесь дерево. Тому тут таокж потрібен крутіший підхід.

8. Зв'язок за допомогою повідомлень. Збереження і синхронність у взаємодіях.

На жаль, жоден із механізмів звернення до віддалених об"єктів не є ідеальним, зокрема через свою синхронність (коли сервер не працює, то клієнт отримує якусь невизначені поведінку, крім того блокування клієнта, поки не отримаємо відповдіь із сервера також зазвичай не є дуже бажаним). Тому чатсо використовують обмін повідомленнями.

Persis-tent communication (збереження зв"язку) - це зв"язок між клієнтом і сервером, при якому повідомлення, яке має надіслатися, зберігається в пам"яті доти, доки не вдасться його таки передати отримувачу.

На противагу йому є transient communication - зв"язок, при якому повідомлення зберігається в системі лише впродовж роботи додатків, які надсилають на отримують це повідомлення. Тобто якщо немає можливості передати повідомлення, то його видаляють.

Також зв"язок поділяють на синхронний та асинхронний.

Asynchronous communication - зв"язок, при якому відправник продовжує роботу після надсилання повідомлення, а не блокується.

Synchronous communication- зв"язок, при якому відправник блокується, поки повідомлення не надішлеться або навіть доки ми не отримаємо відповідь.

9. Зв'язок на основі повідомлень.

Щоб розібратися в безлічі альтернатив в комунікаційних системах, що працюють на основі повідомлень, припустимо, що система організована за принципом комп'ютерної мережі. Додатки завжди виконуються на хостах, а кожен хост надає інтерфейс з комунікаційною системою, через який повідомлення можуть передаватися. Хости з'єднані мережею комунікаційних серверів, які відповідають за передачу (або маршрутизацію) сполучення між хостами.

Як приклад розглянемо розроблену в подібному стилі систему електронної пошти. Хости працюють як призначені для користувача агенти - це користувальницькі додатки, які можуть створювати, надсилати, приймати і читати повідомлення. Кожен хост з'єднується тільки з одним поштовим сервером, кото-рий є по відношенню до нього комунікаційним сервером. Інтерфейс користувача хоста дозволяє призначеному для користувача агенту посилати повідомлення за конкретними адресами. Коли користувальницький агент представляє повідомлення для передачі на хост, хост зазвичай пересилає це повідомлення на свій локальний поштовий сервер, у вихідному буфері якого воно зберігається до пори до часу.

інтерфейс повідомлень

Поштовий сервер видаляє повідомлення зі свого вихідного буфера і шукає, куди його потрібно доставити. Пошуки місця призначення призводять до отримання адреси (транспортного рівня) поштового сервера, для якого призначене повідомлення. Потім поштовий сервер встановлює з'єднання і передає повідомлення на інший, обраний поштовий сервер. Останній зберігає повідомлення у вхідному буфері наміченого одержувача, також званому поштовою скринькою одержувача. Якщо шуканий поштову скриньку тимчасово недоступний, наприклад, відключений, то зберігати повідомлення продовжує локальний поштовий сервер.

Інтерфейс приймає хоста надає призначеним для користувача агентам служби, за допомогою яких вони можуть регулярно перевіряти наявність пошти, що прийшла. Призначений для користувача агент може працювати безпосередньо з поштовою скринькою користувача на локальному поштовому сервері або копіювати нові повідомлення в локальний буфер свого хоста. Таким чином, повідомлення зазвичай зберігаються на комунікаційних серверах, але іноді і на приймаючому хості.

Система електронної пошти - це типовий приклад збереженій зв'язку (persistent communication). При збереженій зв'язку повідомлення, призначене для відсилання, зберігається в комунікаційній системі до тих пір, поки його не вдасться передати одержувачу.

10. Зв'язок на основі потоків даних. Підтримка безперервних середовищ. Потоки даних і якість обслуговування. Синхронізація потоків даних

Зв'язок на основі потоків даних

Для передачі повідомлення одним менеджером черг іншому (можливо, віддаленого) необхідно, щоб кожне повідомлення несло в собі адресу призначення. Для цього використовується заголовок повідомлення. Адреса в MQSeries утворена з двох частин. Перша частина складається з імені менеджера черг, якому це повідомлення повинна бути доставлена, а друга частина - це ім'я черги призначення, яке повідомить цього менеджера, до якої черги додавати повідомлення.

Крім адреси одержувача необхідно також визначити маршрут, яким має слідувати повідомлення.

У більшості випадків маршрути явно зберігаються в таблицях маршрутизації всередині менеджерів черг. Запис в таблиці маршрутизації є пару (destQM, sendQ), де destQM - ім'я менеджера черг, який отримує повідомлення, а sendQ - ім'я локальної черзі відправки, до якої слід додавати повідомлення для цього менеджера. Запис в таблиці маршрутизації в MQSeries називається псевдонімом (alias).

Підтримка безперервних середовищ

Для обміну критичною до часу передачі інформацією розподілені системи зазвичай надають підтримку потоків даних (data streams, або просто streams). Потік даних є не що інше, як послідовність елементів даних. Потоки даних застосовні як для дискретної, так і для безперервного середовища уявлення. Так, канали UNIX або з'єднання TCP / IP є типові приклади дискретних потоків даних (байт-орієнтованих). Відтворення звукового файлу зазвичай вимагає безперервного потоку даних між файлом і пристроєм відтворення.

Тимчасові характеристики важливі для безперервних потоків даних. Для підтримки тимчасових характеристик часто доводиться вибирати між різними режимами передачі. В асинхронному режимі передачі (asynchronous transmission mode) елементи даних передаються в потік один за іншим, але на їх подальшу передачу ніяких обмежень в частині тимчасових характеристик не вводиться. Це традиційний варіант для дискретних потоків даних. Так, файл можна перетворити в потік даних, але з'ясовувати точний момент закінчення передачі кожного елемента даних найчастіше безглуздо.

У синхронному режимі передачі (synchronous transmission mode) для кожного елемента в потоці даних визначається максимальна затримка наскрізний передачі. Якщо елемент даних був переданий значно швидше максимально допустимої затримки, це не важливо.

Потоки даних і якість обслуговування

Тимчасові залежності та інші нефункціональні вимоги зазвичай виражаються у вигляді вимог до якості обслуговування (Quality of Service, QoS). Ці вимоги описують, що повинні зробити базова розподілена система і мережу для того, щоб гарантувати, наприклад, збереження в потоці даних тимчасових співвідношень. Вимоги QoS для безперервних потоків даних в основному характеризуються тимчасовими діаграмами, об'ємом і надійністю. У цьому пункті ми коротко розглянемо вимоги QoS і їх вплив на створення потоку даних.

специфіка QoS

Вимоги QoS можуть бути виражені по-різному. Один з підходів - надати точну специфікацію передачі (flow specification), що містить вимоги щодо пропускної здатності, швидкості передачі, затримки і т. П.

Синхронізація потоків даних

У мультимедійних системах важливе значення має взаємна синхронізація різних потоків даних, можливо, зібраних в комплексний потік. Синхронізація потоків даних передбачає підтримку тимчасових співвідношень між ними. Існує два типи синхронізації.

Найпростіша форма синхронізації має місце між дискретними і безперервними потоками даних. Розглянемо, наприклад, показ слайдів через Web, доповнений звуковим рядом. Кожен слайд передається з сервера на клієнт у вигляді дискретного потоку даних. У той же самий час клієнт відтворює якийсь аудиопоток (або його частина), який відповідає поточному слайду і також надходить з сервера. В даному випадку аудиопоток синхронізується з показом слайдів.

Більш складний тип синхронізації спостерігається між безперервними потоками даних. Черговий приклад відтворення фільму, при якому відеопотік повинен бути синхронізований зі звуком, відомий як синхронізація артикуляції.

11. Потоки виконання в розподілених системах

Важливим властивістю потоків виконання є зручна реалізація блокуючих системних викликів, які відбуваються без блокування всього процесу на час виконання потоку. Ця властивість потоків виконання особливо привабливо в розподілених системах, оскільки воно значно спрощує уявлення взаємодії як одночасне підтримання значної кількості логічних з'єднань.

Хорошими прикладами потоків виконання розподілених системах є **багатопотокові клієнти** , де web-браузер виконує кілька завдань одночасно. Як тільки ми отримуємо основний файл HTML, активізуються окремі потоки виконання, що відповідають за дозагрузку інших частин сторінки. Кожен з потоків виконання створює окреме з'єднання з сервером і отримує від нього дані.

Також, хорошим прикладом є **багатопотокові сервери** , в яких багатопоточність не тільки істотно спрощує код сервера, але і робить набагато простіше розробку тих серверів, в яких для досягнення високої продуктивності потрібно паралельне виконання декількох додатків. У число таких входять і мультипроцесорні системи.

12. Клієнти. Призначені для користувача інтерфейси. Клієнтське програмне забезпечення, що забезпечує прозорість розподілу.

Основне завдання більшості клієнтів - служити передавальною ланкою між користувачем і віддаленим сервером. Підтримка користувальницького інтерфейсу - основна функція більшості клієнтів. У багатьох випадках інтерфейс між користувачем і віддаленим сервером відносно простий і вбудований в апаратуру клієнта. Важливий клас інтерфейсів складають графічні інтерфейси.

Ключова ідея, що стоїть за інтерфейсами, - це поняття про складений документ (compound document). Його можна визначити як набір документів, можливо різних типів (наприклад, текст, малюнки, електронні таблиці і т. Д.), Які інтегруються в одне ціле на рівні користувача інтерфейсу. Інтерфейс, в якому представляється цей документ, приховує той факт, що з різними частинами документа працюють різні додатки. З точки зору користувача, всі частини з'єднані в одне ціле. Якщо зміна однієї з частин тягне за собою зміни інших, призначений для користувача інтерфейс може виробляти необхідні виміри. Хорощим прикладом користувацьким інтерфейсом є система X-Windows, або просто X, яке використовується для управління растровими терміналами, до складу яких входять монітор, клавіатура і координатний пристрій, таке як мишка.

Крім призначеного для користувача інтерфейсу і іншого пов'язаного з додатком програмного забезпечення, клієнтське програмне забезпечення містить компоненти, що забезпечують прозорість розподілу. В ідеалі клієнт не повинен бути обізнаний про свою взаємодію з віддаленими процесами. Прозорість доступу зазвичай забезпечується шляхом генерації (у вигляді заглушки клієнта) з визначення інтерфейсу того, що повинен робити сервер. Заглушка надає такий же інтерфейс, як і сервер, приховуючи при цьому різницю архітектур і реальна взаємодія.

Існують різні способи реалізації прозорості розміщення, перенесення і переміщення. У багатьох випадках важлива кооперація з програмним забезпеченням клієнтської сторони. Наприклад, коли клієнт вже прив'язаний до сервера, він може безпосередньо бути сповіщені про зміну місцезнаходження сервера. В цьому випадку проміжний рівень клієнта може приховувати справжнє місце розташування сервера від користувача і при необхідності непомітно повторити прив'язку до цього сервера. Найбільше, що може помітити додаток клієнта, - це тимчасове падіння продуктивності.Аналогічним чином більшість розподілених систем реалізують прозорість реплікації на стороні клієнта.

Маскування збоїв у взаємодії з серверами зазвичай виконується за допомогою клієнтського програмного забезпечення проміжного рівня. Так, клієнтське програмне забезпечення проміжного рівня можна конфігурувати таким чином, щоб воно багаторазово намагалося зв'язатися з сервером або обирало після кількох спроб інший сервер. Можлива також ситуація, коли програмне забезпечення клієнта, в разі якщо web-браузер не в змозі зв'язатися з сервером, повертало б дані, збережені в кеші під час попереднього сеансу зв'язку.

Прозорість паралельного виконання може забезпечуватися спеціальними проміжними серверами, так званими моніторами транзакцій, і вимагає меншої підтримки з боку клієнтського програмного забезпечення .

13. Сервери. Загальні питання розробки. Сервери об'єктів.

Сервер як комп'ютер — це комп'ютер у локальній чи глобальній мережі, який надає користувачам свої обчислювальні і дискові ресурси, а також доступ до встановлених сервісів; найчастіше працює цілодобово, чи у час роботи групи його користувачів.

Сервер як програма — програма, що надає деякі послуги іншим програмам (клієнтам). Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу.

Головною ознакою сервера є здатність машини чи програми переважну кількість часу працювати автономно, без втручання людини, реагуючи на зовнішні події відповідно до встановленого програмного забезпечення.

Види серверів:

-сервер терміналів - сервер, який надає клієнтам обчислювальні ресурси
-файл сервер - сервер, що надає клієнтам доступ до файлових ресурсів
-сервер бд - сервер, що надає клієнтам доступ до бази даних
-сервер додатків - сервер, що надає клієнтам доступ до якоїсь програми, додатку

Сервер об"єктів

Якщо використовується сервер об"єктів, то клієнт-серверну програму пишуть як набіть об"єктів, які між собою спілкуються. Об"єкт клієнта спілкується з сервером використовуючи Object Request Broker (ORB) (це посередник, який дозволяє комп"ютерам спілкуватися між собою через мережу). Клієнт викликає метод віддаленого об"єкта. ORB шукає об"єкт класу сервера і викликає метод, а тоді повертає результат виконання метода назад до клієнта. Об"єкта сервера мусить надавати підтримку багатопоточності та поділу.

14. Перенесення коду. Підходи до переносу коду.

Зазвичай у розподілених системах взаємодія обмежується передачею даних. Але деколи нам потрібно передати програму. І навіть деколи нам це потрібно під час її виконання. Це дозволяє спростити розробку розподілених систем.

Причини переносу коду

Зазвичай це відбувається у формі переносу процесів, тобто коли процес повністю переноситься з однієї машини на іншу. Це дуже складно і затратно, але деколи це дуже потрібно.

Зазвичай причиною є продуктивність. Тобто з дуже загруженої машини процеси переносяться на менш загружену.

Інша причина це гнучкість. Розподілені програми будують так, що програму ділять на частини і вже потім визначають, що де буде виконуватись. А якщо код можна перенести з машини на машину, то визначати це можна навіть динамічно.

Отримання коду клієнтом.

Перевага тут в тому, що на початку клієнту не треба мати в себе все ПЗ. Замість цього при необхідності програми скачуються на клієнті потім видаляються, коли вони вже не потрібні.

Недолік цього підходу пов"язаний із безпекою, адже нема гарантії, що завантажені програми не нашкодить комп"ютеру клієнта.

Підходи до переносу

Процес складається з трьох сегментів. Сегмент коду - це частина коду, яка містить набір інструкцій. Сегмент ресурсів - це посилання на зовнішні ресурси (файли, принтери, інші процеси і тд). Сегмент виконання - сегмент, в якому зберігається поточний стан програми (закриті дані, стек і тд).

Модель слабкої мобільності

Переноситься лише сегмент коду. Тоді програма завжди запускається в початковому стані.

Модель сильної мобільності

Переноситься також сегмент виконання. Тут важливо те, що процес, який зараз виконується, можливо треба буде зупинити, перенести на іншу машину, а тоді відновити виконання. Цей підхід набагато складніший і набагато потужніший.

Також переноси розділяють на ті, які ініціалізовує відправник, і ті, які ініціалізовує отримувач.

15. Перенос і локальні ресурси. Перенесення коду в гетерогенних системах.

Перенос і локальні ресурси.

Перенесення коду нерідко сильно ускладнює те, що сегмент ресурсів не завжди можна перенести з такою ж легкістю без змін, як інші сегменти. При перенесенні процесу на іншу машину процес повинен звільнити зайнятий їм порт і запросити іншого - на тій машині, на яку він був переміщений. В інших випадках перенесення посилання проблем не створює. Щоб зрозуміти, який вплив надає перенесення коду на сегмент ресурсів, було виділено три типи зв'язку процесу з ресурсами . Найбільш сильний зв'язок спостерігається, коли процес посилається на ресурс за його ідентифікатором. Більш слабкий зв'язок процесу з ресурсами матиме місце в тому випадку, якщо процесу необхідно тільки значення ресурсу. І, нарешті, найбільш слабка форма зв'язку має місце в тому випадку, коли процес вказує на необхідність використання ресурсу певного типу.

Локальний ресурс, або загальний мережевий ресурс - в інформатиці, це пристрій або частина інформації, до якої може бути здійснений віддалений доступ з іншого комп'ютера, зазвичай через локальну комп'ютерну мережу або за допомогою корпоративного інтернету, як якби ресурс перебував на локальній машині.

Прикладами такого можуть служити загальний доступ до файлів (також відомий як загальний доступ до диска і загальний доступ до папок), загальний доступ до принтера (спільний доступ до принтера), сканера і т. п. Загальним ресурсом називається «спільний доступ до диску» (також відомим як підключений диск, «загальний тому диска», «загальна папка», «загальний файл», «загальний документ», «загальний принтер».

Перенесення коду в гетерогенних системах.

Розподілені системи створюються з набору гетерогенних платформ, кожна з яких має свою власну машинну архітектуру і операційну систему. Перенесення в подібних системах вимагає, щоб підтримувалися всі ці платформи, тобто сегмент коду повинен виконуватися на всіх цих платформах без перекомпіляції тексту програми. Крім того, ми повинні бути впевнені, що сегмент виконання на кожній з цих платформ буде представлений правильно.

Проблеми можуть бути частково усунені в тому випадку, якщо ми обмежимося слабкою мобільністю. В цьому випадку зазвичай не існує такої інформації часу виконання, яку треба було б передавати від машини до машини. У разі сильної мобільності основною проблемою, яку треба буде рішити, є перенесення сегмента виконання. Проблема полягає в тому, що цей сегмент в значній мірі залежить від платформи, на якій кинонується завдання. Насправді перенести сегмент виконання, не вносячи в нього ніяких змін, можна тільки в тому випадку, якщо машина-приймач має ту ж архітектуру і працює під управлінням тієї ж операційної системи.

16. Програмні агенти. Програмні агенти в розподілених системах. Технологія агентів.

Ці більш-менш незалежні уявлення процесів об'єднаються в те, що нерідко називають програмними агентами - автономні одиниці, здатні виконувати завдання в кооперації з іншими, можливо, віддаленими агентами. Ми визначаємо програмний агент (software agent) як автономний процес, здатний реагувати на середовище виконання і викликати зміни в середовищі виконання, можливо, в кооперації з користувачами або з іншими агентами. Властивість, яке робить агента чимось більшим, ніж процес, - це здатність функціонувати автономно і, зокрема, проявляти при необхідності ініціативу.

Крім автономності найважливішої якості агентів - можливість кооперуватися з іншими агентами. Поєднання автономності та кооперації призводить нас до класу кооперативних агентів. Кооперативний агент (collaborative agent) - це агент, який є частиною мультиагентної системи, тобто системи, в якій агенти, Співпраця, виконують якісь спільні завдання. Типове додаток, що використовує кооперативні агенти, - це електронна конфе-ренція.

Мобільний агент (mobile agent) - це просто агент, у якого є здатність переміщатися з машини на машину. У термінах, які ми використовували при обговоренні перенесення коду в попередньому розділі, мобільні агенти часто вимагають підтримки сильної мобільності, хоча це і не є абсолютно необхідним. Вимога сильної мобільності випливає з того факту, що агенти автономні і активно взаємодіють зі своїм середовищем. Перенесення агента на іншу машину без урахування його стану буде сильно утруднений.

Інтерфейсний агент (interface agent) - це агент, що допомагає кінцевому користувачеві працювати з одним або декількома додатками. Серед традиційно наявних у интерфейсного агента властивостей можна вважати здатність до навчання. Найчастіше вони взаємодіють з користувачами, забезпечуючи їм підтримку.

Дуже близький до багатофункціонального агенту інформаційний агент (information agent). Основна функція подібних агентів - управління інформацією з безлічі різних джерел. Управління інформацією включає в себе впорядкування, фільтрацію, порівняння і т. П.

Технологія агентів.

Компонент управління агентами відстежує агентів на відповідній платформі. Він надає механізми створення і знищення агентів, а також для перегляду поточної кінцевої точки на предмет наявності конкретногоагента.

Крім того, існує і окрема локальна служба каталогів,за допомоги якої агенти можуть дізнатися, які ще агенти є на цій платформі. Служба каталогів в моделі FIPA (Foundation for Intelligent Physical Agents) заснована на використанні атрибутів. Це означає, що агент надає опису своїх служб в поняттях імен атрибутів разом з їх значеннями для даного агента.

Важливий компонент платформи агента - капав зв'язку між агентами (Agent Communication Channel, ACC). У більшості моделей мультиагентних систем агенти зв'язуються один з одним, пересилаючи повідомлення. Модель FIPA - не виключення, вона покладає на ACC відповідальність за все взаємодія між різними платформами агентів. Зокрема, ACC відповідає за надійну і спрямовану зв'язок точка-точка з іншими платформами.

Зв'язок між агентами відбувається за допомогою комунікаційного протоколу прикладного рівня, відомого під назвою мови взаємодії агентів (Agent Communication Language, ACL). У ACL присутній жорстке розділення між метою повідомлення і його змістом. Повідомлення може мати тільки обмежений набір цілей. Наприклад, метою повідомлення може бути запит на надання отримувачем певної служби.

17. Іменовані сутності. Імена, ідентифікатори і адреси. Розширення імен. Реалізація просторів імен.

Іменовані сутності.

Імена грають важливу роль у всіх комп'ютерних системах. Вони необхідні для спільного використання ресурсів, визначення унікальних сутностей, посилань на місця розташування і т. д.

Імена, ідентифікатори і адреси

Ім'я в розподіленій системі являє собою рядок бітів, або символів, який використовується для посилання на сутність. Сутністю в розподіленої системі є практично все. Типовими прикладами є ресурси, включаючи хости, принтери, диски, файли. Інші добре відомі приклади сутностей, часто отримують імена, - це процеси, користувачі, поштові скриньки, групи новин, web-сторінки, графічні вікна, повідомлення, мережеві з'єднання і т. д.

Щоб працювати з сутністю, необхідно мати до неї доступ, для якого ми використовуємо точку доступу (access point). Точка доступу - це ще один спеціальний вид сутності в розподілених системах. Ім'я точки доступу називається адресою (address). Адресу точки доступу сутності часто називають просто адресою сутності.

Сутність може мати більш ніж одну точку доступу, а точка доступу сутності може з часом змінюватися. Адреса - це спеціальний тип імені, який вказує на точку доступу до сутності. Оскільки точка доступу тісно пов'язана з сутністю, зручно використовувати адресу в якості постійного імені відповідної сутності. Однак робити це можна не завжди.

Крім адрес існують і інші типи імен, що вимагають особливого розгляду, наприклад імена, використовувані для однозначної ідентифікації сутності. Правильний ідентифікатор (true identifier) - це ім'я з наступними властивостями:

♦ ідентифікатор посилається не більше ніж на одну сутність;

♦ на кожну сутність посилається не більше одного ідентифікатора;

♦ ідентифікатор завжди посилається на одну і ту ж сутність (тобто не може бути використаний повторно).

Використання ідентифікаторів значно спрощує створення однозначних посилань на сутність.

Розширення імен

Простору імен надають зручний спосіб збереження і вилучення інформації про сутності за їхніми іменами. У загальному вигляді, знаючи ім'я шляху, можна витягти всю інформацію, яка зберігається в вузлі, відповідному цьому імені. Процес пошуку інформації називається розширенням імені (name resolution).

Реалізація просторів імен

Простір імен формує серце служби іменування, служби, яка дозволяє користувачам і процесам додавати, видаляти і знаходити імена. Служба іменування реалізується за допомогою серверів імен. Якщо розподілена система скорочена до розмірів локальної мережі, вона цілком в змозі реалізувати службу іменування за допомогою всього лише одного сервера імен. Однак у великих розподілених системах з великою кількістю сутностей необхідно рознести реалізацію простору імен по декількох серверах імен.

Для ефективної реалізації простору імен його розділяють на три рівні:
+ Глобальний рівень;
+ Адміністративний рівень;
+ Управлінський рівень.
Глобальний рівень (global layer) формується вузлами верхнього рівня, тобто кореневими вузлами та іншими напрямними вузлами, які логічно пов'язані з кореневими, тобто їх дочірніми вузлами. Ці вузли можуть представляти організації або групи організацій, імена яких зберігаються в просторі імен.
Адміністративний рівень (administrational layer) формується з напрямних вузлів, які разом представляють одну організацію. Характерною рисою напрямних вузлів адміністративного рівня є те, що вони представляють групи сутностей, що відносяться до однієї і тієї ж організації або адміністративній одиниці.
Управлінський рівень (managerial layer) складається з вузлів, які зазвичай регулярно змінюються. Наприклад, в цей рівень входять вузли, що представляють хости локальної мережі. З тієї ж причини цей рівень включає вузли, що надають спільно використовувані файли, такі як бібліотеки або об'єктний код.

18. Розміщення мобільних сутностей. Іменування і локалізація сутностей. Прості рішення. Підходи на основі базової точки. Ієрархічні підходи.

Іменування і локалізація сутностей

Сутності іменуються для того, щоб мати можливість їх знайти і отримати до них доступ. Виділяють три типи імен: імена, зручні для сприйняття, ідентифікатори і адреси. Оскільки розподілені системи будуються для людей і для доступу до сутності, необхідно знати її адресу, фактично всі системи іменування підтримують відображення імен, зручних для сприйняття, в адреси.
Для ефективної реалізації повномасштабного простору імен зручно розбити простір імен на три рівні. Глобальний та адміністративний рівні характеризуються тим, що імена змінюються нечасто. Точніше, вміст вузлів цих частин простору імен відносно постійний. Внаслідок цього реплікація і кешування здатні підвищити ефективність реалізації.
Вміст вузлів управлінського рівня часто змінюється. Тому продуктивність операцій пошуку і поновлення на цьому рівні стає особливо важливою. На практиці вимоги по продуктивності можна задовольнити шляхом реалізації вузлів на локальних високопродуктивних серверах імен.

Прості рішення

Широкомовне і групова розсилки
Розглянемо розподілену систему, побудовану на основі комп'ютерної мережі, що надає ефективні механізми широкомовної розсилки. Зазвичай подібні механізми надаються в локальній мережі, в якій всі машини приєднані до одного кабелю. Бездротові локальні мережі також потрапляють в цю категорію.
Локалізація сутності в такому середовищі проста: повідомлення, що містить ідентифікатор сутності, широкомовної розсилкою доноситься до кожної машини і кожна з машин відгукується на цей запит перевіркою, чи не на ній чи розміщена ця сутність. Ті машини, які можуть надати точку входу до шуканої суті, посилають Вам відповідь, що містить адресу точки входу.
Широкомовлення з ростом мережі втрачає ефективність. Втрати пропускної здатності мережі на пересилку повідомлень - не єдина проблема, більш важливо те, що безліч хостів змушені переривати свою роботу через запит, на який вони не відповідатимуть. Одне з можливих рішень цієї проблеми - перехід до групової розсилки, при якій запит отримує лише обмежена група хостів.
Групова розсилка може також використовуватися для локалізації сутностей в некомутованих мережах. Так, наприклад, в Інтернеті підтримується групова розсилка мережевого рівня, при якій хостам дозволено приєднуватися до конкретної групи розсилки. Ця група визначається адресою групової розсилки. Коли

хост посилає повідомлення на адресу групового розсилання, мережевий рівень надає зручну службу з доставки цього повідомлення кожному з членів групи.
Передача вказівників
Інший популярний підхід до локалізації мобільних сутностей заснований на передачі вказівників. Принцип простий. Коли сутність переміщається з A в B, вона зберігає посилання на своє нове місце розташування в A. Перевага цього підходу - в його простоті: як тільки сутність локалізується за допомогою, наприклад, традиційної служби іменування, клієнт може знайти її поточну адресу, пройшовши по ланцюжку переданих вказівників.
Є в наявності і комплект недоліків. По-перше, якщо не вживати спеціальних заходів, ланцюжок може стати настільки довгим, що локалізація сутності стане занадто дорогим задоволенням. По-друге, всі проміжні розташування в ланцюжку повинні підтримувати свою частину ланцюжка так довго, як це буде необхідно. Третій і найсерйозніший недолік - вразливість до втрати посилань. Як тільки вказівник, що пересилається, з якихось причин буде втрачено, сутність виявиться неможливим локалізувати. Таким чином, важливо зберегти ланцюжок коротким і гарантувати збереження пересилаються покажчиків.

Підходи на основі базової точки

Використання широкомовної розсилки та передачі покажчиків створює проблеми масштабованості. Широкомовні і групові розсилки важко ефективно реалізувати в великомасштабних мережах, а довгі ланцюжки пересилаються покажчиків створюють проблеми з продуктивністю і чутливі до обривів зв'язків.
Популярний підхід до підтримки мобільних сутностей в повномасштабних мережах полягає у введенні поняття базової точки (home location), з якої відстежується поточне місце розташування об'єкта. Для захисту базової точки від збоїв в мережі або процесах можна застосовувати спеціальні методики. На практиці базовою точкою зазвичай вибирається те місце, де була створена сутність.
Підхід на основі базової точки використовується в якості аварійного методу служб локалізації, заснованих на передачі покажчиків. Іншим прикладом подібного підходу є схема мобільної IP-адреси.Кожен мобільний хост має фіксовану IP-адресу. Будь-який зв'язок з цим IP-адресом спочатку перенаправляється агенту бази (home agent) мобільного хоста. Цей агент знаходиться в локальній мережі, відповідної мережевою адресою, містить IP-адреса мобільного хоста. Кожен раз, коли мобільний хост переміщується в іншу мережу, він запитує тимчасову адресу для зв'язку. Ця електронна адреса, яка називається контрольною адресою (care-of address), реєструється агентом бази.
Коли агент бази отримує від мобільного хоста пакет, він перевіряє місцезнаходження хоста. Якщо хост знаходиться в поточній локальній мережі, пакет просто пересилається йому. В іншому випадку він передається туди, де в даний час знаходиться хост, тобто поміщається у вигляді даних в IP-пакет і пересилається на контрольну адресу. Одночасно відправник пакета повідомляється про поточне місцезнаходження хоста.

Ієрархічні підходи

В ієрархічній схемі мережа ділиться на домени. Це сильно нагадує ієрархічну організацію DNS. Домен верхнього рівня охоплює мережу цілком. Кожен домен ділиться на безліч піддоменів. Домен самого нижнього рівня, званий листовим доменом (leaf domain), зазвичай відповідає локальній мережі в комп'ютерних мережах.
Також аналогічно DNS і іншим ієрархічним системам іменування. Кожен домен D має асоційований з ним направляючий вузол dir (D), який відстежує сутності домену. Таким чином, ми отримуємо дерево напрямляючих вузлів. Направляючий вузол домену верхнього рівня, іменований кореневих напрямних вузлом (root directory node), містить відомості про всі сутності.

Щоб відстежувати місцезнаходження сутностей, кожна сутність, яка перебуває в домені D, представлена локалізуючої записом (location record) в направляючому вузлі dir (D). Локалізуючий запис для сутності E в направляючому вузлі N листового домену D містить поточну адресу сутності в цьому домені. З іншого боку, спрямовуючий вузол N' в домені наступного рівня D', в який входить D, містить в локалізуючому записі для E тільки покажчик на N. Відповідно, на кореневому вузлі знаходяться локалізуючі записи для всіх сутностей, і кожен локалізуючий запис містить вказівник на направляючий вузол в домені нижнього рівня, в якому знаходиться сутність, що ідентифікується цим записом.

19. Логічний годинник. Відмітки часу Лампорта. Векторні позначки часу

У багатьох випадках необхідно, щоб всі машини домовилися про використання одного і того ж часу. Не настільки вже й важливо, щоб цей час збігався з істинним часом, який кожну годину оголошують по радіо. Для роботи програми, достатньо, щоб всі машини вважали, що зараз 10:00, навіть якщо насправді зараз 10:02. Так, для деякого класу алгоритмів подібна внутрішня несуперечливість має набагато більше значення, ніж те, наскільки їх час близько до реального. Для таких алгоритмів прийнято говорити про логічний годиник.

У своїй статті Лампорт показав, що хоча синхронізація годин можлива, вона не обов'язково повинна бути абсолютною. Якщо два процеси не взаємодіють, немає необхідності в тому, щоб їх годинники були синхронізовані, оскільки відсутність синхронізації залишиться непоміченим і не створить проблем. Крім того, він вказав, що зазвичай має значення не точний час виконання процесів, а його порядок.

Проте підхід Лампорта дозволяє лише встановити послідовність подій, а не зв"язок між ними. Причинно-наслідковий зв"язок між процесами може бути дотриманий, якшо використовувати векторні відмітки часу. Векторна відмітка часу VT(a), яку присвоюють події a, має таку властивість: якщо VT(a) < VT(b) для події b, то a причинно-передує події b. Векторні відмітки часу створюються шляхом приписування кожному процесу P вектора Vi, який має такі властивості:

- Vi[i] - це число подій, які сталися до поточного часу (включно з процесом Pi)
- Якщо Vi\j] = k, то процем Pr знає, що разом з процесом P сталось k подій. (???)

20. Розподілені транзакції. Модель транзакцій.

Концепція транзакцій тісно пов'язана з концепцією взаємних виключень. Алгоритми взаємного виключення забезпечують одночасний доступ не більш, ніж одного процесу до спільно використовуваних ресурсів , таким як файл, принтер і т. п. Транзакції загалом також захищають загальні ресурси від одночасного доступу декількох паралельних процесів. Однак транзакції можуть і багато іншого. Зокрема, вони перетворюють процеси доступу і модифікації великої кількості елементів даних в одну атомарну операцію. Якщо процес під час транзакції вирішує зупинитися на півдорозі і повернути назад, всі дані відновлюються з тими значеннями і в тому стані, в якому вони були до початку транзакції.

Модель транзакцій.

Один процес оголошує, що хоче почати транзакцію з одним чи декількома процесами. Після цього вони можуть узгодити різні умови, створювати і видаляти сутності, виконувати операції. Потім ініціатор оголовує, що він пропонує всім іншим підтвердити, що роботу зроблено. Якщо

всі підтверджують, то результати зберігаються і стають постійними. Якщо хоча б один процес не підтверджує (через збої чи інші причини), то стан процесів повертається до такого вигляду, який вони мали до початку транзакції, враховуючи те, що всі зміни файлів, бд і та кдалі скасовуються. Цей принцип "все або нічого" дуже спрощує програмістам роботу.

21. Класифікація транзакцій. Реалізація транзакцій. Управління паралельним виконанням транзакцій

КЛАСИФІКАЦІЯ ТРАНЗАКЦІЙ

Транзакцією ми вважаємо серію операцій, що задовільняють властивості ACID. Цей тип транзакцій також називається **плоскою транзакцією** (flat transaction). Плоскі транзакції –це найбільш простий та найчастіше вживаний тип транзакцій. Але вони мють багато обмежень, які змушують нас до пошуків альтернативних моделей. Основне обмеження в тому, що вони не можуть давати частинного результату у випадку закінчення або переривання.

Вкладені транзакції

Деякі обмеження можуть бути прибрані за рахунок вкладених транзакцій. Транзакції верхнього рівня можуть розкладатися на дочірні, що працюють паралельно на різних машинах для підвищення продуктивності або полегшення програмування.

Розподільні транзакції

Різниця між вкладеними і розподільними транзакціями невелика, але важлива. Вкладені транзакції – ті, що логічно розділюються на ієрархічно організовані дочірні. На противагу їм, роздільні транзакції логічно представляють собою плоскі,неподільні транзакції, що працюють з розподіленими даними.

РЕАЛІЗАЦІЯ ТРАНЗАКЦІЙ

Переважно використовують 2 метода

Закритий робочий простір

Коли процес починає транзакцію, він отримує закриий робочий простір, що містить всі файли, до яких йому потрібен доступ. Поки транзакція не завершиться, або не перерветься, всі операції зчитування та запису будуть відбуватися не в файловій системі, а в закритому робочому просторі. Проблема цього методу в тому, що копіювання всього в простір дуже дороге.

Журнал з випередженим записом

Згідно з цим методом, файли дійсно модифікуються там, де й знаходяться, але перед тим, як якийсь блок буде змінено, в журнал вноситься запис про те, як транзакція проводить зміни, який файл і блок змінюються, попередні і нові значення.

УПРАВЛІННЯ

Ціль управління паралельним виконанням транзакцій полягає в тмоу, щоб дозволити декільком транзакціям виконуватись одночасно, але таким чином, щоб набір елементів даних, що обробляються, був несуперечливим. Це досягається за рахунок того, що доступ транзакцій до елементів даних організовується в певному порядку.

22. Несуперечність і реплікації. Доводи на користь реплікації.

Важливим питанням для розподілених систем є реплікація даних. Дані зазвичай реплікуються для підвищення надійності і збільшення продуктивності. Одна з основних проблем при цьому – збереження несуперечності реплік. Реалізація моделей несуперечності пам’яті для

великомасштабних роздільних систем дуже тяжка задача. Можна використовувати такі прості моделі, як клієнтські моделі несуперечності, які обмежуються несуперечнсітю з точки зору одного (можливо, мобільного) клієнта. На підтримку несуперечності реплік грають два більш-менш незалежних аспекти: розповсюдження оновлень та підтримання несуперечності реплік.

На користь реплікацій є два основних доводи – надійність та продуктивність. Якщо файлову систему було пошкоджено, вона може продовжувати працювати після збою в одній з реплік, просто переключившись на другу. Крім того, маючи декілька копій, легше боротись зі збоями даних.

Реплікація підвищує продуктивність, коли розподільну систему доводиться масштабувати на багато машин і географічних зон. Мінусом реплікацій є та ж наявність багатьох копій, що змушує переносити зміни з однієї на всі інші (що є затратним) .

23. Реплікація об'єктів. Реплікація як метод масштабування.

Важливим питанням для розподілених систем є реплікація даних. Дані звичайно реплікуються для підвищення надійності й збільшення продуктивності. Одна з основних проблем при цьому - збереження несуперечності реплік. Якщо в одну з копій вносяться зміни, то необхідно забезпечити, щоб ці зміни були внесені й в інші копії, інакше репліки більше не будуть однаковими.

На користь реплікації є два основних доводи - надійність і продуктивність. По-перше, дані реплікуються для підвищення надійності системи. Якщо файлова система реплікована, вона може продовжувати свою роботу після збою в одній з реплік, просто переходячи на іншу. Підтримуючи кілька копій, легше протистояти збоям даних

Інший довід на користь реплікації даних - продуктивність.Основна ідея складається в переміщенні копії даних ближче до процесу, що ними користується, це веде до скорочення часу доступу.

При маштабуванні є свої плюси та мінуси із використанням реплікацій. Головний плюс - це розвантаження системи при великій к-сті запитів, а тому й збільшенням продуктивності її роботи вцілому. Тобто, замість того щоб усі запити оброблялись одним сервером, вони розприділяються між певною к-стю серверів, кожен з яких має актуальні дані.

Серед недоліків же ж головним є забезпечення несуперечності реплікованих даних. Тобто, щоб внісши зміни у репліку А, а в наступний момент після читання даних із репліки В - це були дані оновлені із змінами на репліці А. Для забезпечення цього після кожної операції запису усі існуючі репліки повинні синхронізуватись із цими оновленнями, що негативно вплине на продуктивність.

Проте, якщо пожертвувати усією строгістю несуперечності до певної прийнятної межі, можна значно зменшити негативний вплив на продуктивність усієї системи. Проте тоді доведеться змиритись із ситуацію, що в деякі моменти "копії" одних і тих же ж даних будуть відрізнятись на протязі певного інтервалу часу, поки не пройде оновлення.

24. Моделі несуперечності, орієнтовані на дані.

За традицією несуперечність завжди обговорюється в контексті операцій читання і запису над спільно використовуваними даними, доступними в розподіленій пам'яті (поділюваній) або у файловій системі (розподіленій). Сховище даних може бути фізично рознесене по декількох машинах. Зокрема, кожний із процесів, що бажає одержати доступ до даних зі сховища, може використовувати доступне локальне сховище із копію даних. Операції запису поширюються й на інші копії,операції читання не вимагають додаткової синхронізації.

Є багато реалізацій несуперечностей, кожна з яких має свої недоліки і переваги. Кожна з них може бути оптимальним рішенням між надійністю та продуктивністю у певних ситуаціях. Вони відрізняються обмеженнями, складністю реалізації, простотою програмування й продуктивністю.

1. **Строга несуперечність - найбільш обмежений варіант, але оскільки її реалізація в розподілених системах, по суті, неможлива, то вона ніколи в них не застосовується.**
2. **Лінеаризуємість - більше слабка модель, заснована на ідеї синхронізованих годин. У ній робити висновок про коректність паралельних програм простіше, але все ще занадто складно для того, щоб її можна було використати для побудови реальних програм.**
3. **Із цього погляду кращою моделлю є послідовна несуперечність, що застосовна, популярна серед програмістів і дійсно широко використовується. Однак у неї є проблема - низька продуктивність. Єдиний спосіб поліпшити показники продуктивності - це послабити модель несуперечності.**
4. **Причинна несуперечність і несуперечність FIFO являють собою ослаблені моделі, у яких відсутній глобальний контроль над тим, які операції в якому порядку виконуються. Різним процесам послідовність операцій здається різною. Ці дві моделі розрізняються в тім, які послідовності вважаються припустимими, а які ні.**

Інший підхід складається у введенні явних змінних синхронізації, як зроблено для слабкої несуперечності, вільної несуперечності й заелементної несуперечності. Коли процес виконує операцію зі звичайним елементом спільно використовуваних даних, відсутні які-небудь гарантії із приводу того, коли його зможуть побачити інші процеси. Зміни поширяться тільки при явній синхронізації. Ці три моделі розрізняються способом синхронізації, але у всіх випадках процес може робити множинні операції читання й запису в критичній області без

реального перенесення даних. Після закінчення критичної області результат передається іншим процесам або зберігається в готовому виді, очікуючи, поки інший процес не зажадає цих даних.

Слабка, вільна й заелементна несуперечності вимагають додаткових програмних конструкцій, які при правильному використанні дозволяють програмістам представити справу так, начебто сховище даних має послідовну несуперечність. У принципі ці три моделі, що використовують явну синхронізацію, можна задіяти для підвищення продуктивності, однак цілком імовірно, що для різних прикладних програм успішність такого підходу буде різною.

25. Порівняння моделей несуперечності

Вони відрізняються обмеженнями, складністю реалізації, простотою програмування і продуктивністю.

Несуперечнісь	Опис
сувора	Абсолютна упорядкованість в часі всіх звернень до спільно використовуваної пам'яті
Лінеарізуемость	Всі процеси спостерігають всі звернення до спільно використовуваної пам'яті в одному і тому ж порядку Звернення впорядковані відповідно до (неунікальні) глобальними відмітками часу
послідовна	Всі процеси спостерігають всі звернення до спільно використовуваної пам'яті в одному і тому ж порядку. Звернення не впорядковані за часом
причинний	Всі процеси спостерігають всі звернення, пов'язані причинно-наслідковим зв'язком, до спільно використовуваної пам'яті в одному і тому ж порядку
FIFO	Всі процеси спостерігають операції записи будь-якого процесу в порядку їх виконання. Операції запису різних процесів можуть спостерігатися різними процесами в різному порядку
Слабка	Спільно використовувані дані можуть вважатися повинні суперечити одна одній тільки після синхронізації
Вільна	Спільно використовувані дані стають не суперечили одна одній після виходу з критичної області
поелементна	Спільно використовувані дані, що відносяться до даної критичної області, стають не суперечили одна одній при вході в цю область

//це основне, якщо маєте багато часу можете дописати ще це

Сувора несуперечливість - найбільш обмежений варіант, але оскільки її реалізація в розподілених системах, по суті, неможлива, то вона ніколи в них не застосовується.
Лінеарізуемость - слабша модель, заснована на ідеї синхронізованих годин. У ній робити висновок про коректність паралельних програм простіше, але все одно дуже складно для того, щоб її можна було використовувати для побудови реальних програм. З цієї точки зору найкращою моделлю є послідовна несуперечливість, яка може бути застосована, популярна серед програмістів і дійсно широко використовується. Однак у неї є проблема - низька продуктивність. Єдиний спосіб поліпшити показники продуктивності - це послабити модель несуперечності. Деякі з можливостей ілюструє табл. 6.5. Моделі перераховані в приблизному порядку зниження обмежень.
Причинний несуперечливість і несуперечливість FIFO є ослаблені моделі, в яких відсутня глобальний контроль над тим, які операції в якому порядку виконуються. Різних процесів послідовність операцій здається різною. Ці дві моделі відрізняються в тому, які послідовності вважаються допустимими, а які ні.
Інший підхід полягає у введенні явних змінних синхронізації, як зроблено для слабкої несуперечності, вільної несуперечності і поелементної несуперечності. Ці три моделі наведені в табл. 6.6. Коли процес виконує операцію зі звичайним елементом спільно використовуваних даних, відсутні будь-які гарантії щодо того, коли його зможуть побачити інші процеси. Зміни поширяться тільки при явній синхронізації. Ці три моделі розрізняються способом синхронізації, але у всіх випадках процес може виробляти множинні операції читання і запису в критичній області без реального перенесення даних. Після закінчення критичної області результат передається іншим процесам або зберігається в готовому вигляді, чекаючи, поки інший процес не зажадає цих даних.
Кажучи коротко, слабка, вільна і поелементно несуперечності вимагають додаткових програмних конструкцій, які при правильному використанні дозволяють програмістам представити справу так, ніби сховище даних має послідовної непротиворечивостью. В принципі ці три моделі, що використовують явну синхронізацію, можна задіяти для підвищення продуктивності, однак цілком імовірно, що для різних додатків успішність такого підходу буде різною.

26. Поняття відмовостійкості. Основні концепції. Моделі відмов. Маскування помилок за допомогою надмірності.

Відмовостійкість тісно пов'язана з поняттям надійних систем (dependable systems). Надійність - це термін, що охоплює безліч важливих вимог до розподілених систем [241], включаючи:

+ Доступність (availability);

♦ безвідмовність (reliability);

♦ безпеку (safety);

♦ ремонтопридатність (maintainability).

Доступність - це властивість системи перебувати в стані готовності до роботи. Зазвичай доступність показує ймовірність того, що система в даний момент часу буде правильно працювати і виявиться в стані виконати свої функції, якщо користувачі того зажадають.

Іншими словами, система з високим ступенем доступності - це така система, яка в довільний момент часу, швидше за все, знаходиться в працездатному стані.

Під безвідмовністю мається на увазі властивість системи працювати без відмов протягом тривалого часу. На противагу доступності безвідмовність визначається в поняттях тимчасового інтервалу, а не моменту часу. Система з високою безвідмовністю - це система, яка, швидше за все, буде безперервно працювати протягом відносно тривалого часу. Між безотказностью і доступністю є невелика, але суттєва різниця. Якщо система відмовляє на одну мілісекунди щогодини, вона має доступність порядку 99,9999%, але вкрай низьку безвідмовність. З іншого боку, система, яка ніколи не відмовляє, але кожен серпень відключається на два тижні, має високу безвідмовність, але її доступність становить всього 96%. Ці дві характеристики - не одне і те ж.

Безпека визначає, наскільки катастрофічна ситуація тимчасової нездатності системи належним чином виконувати свою роботу. Так, багато систем управління процесами, що використовуються, наприклад, на атомних електростанціях або космічних кораблях, повинні мати високий ступінь безпеки. Якщо ці керуючі системи навіть тимчасово, на короткий термін, перестануть працювати, результат може бути жахливий. Безліч прикладів відбувалися в минулому подій показують, як важко побудувати безпечну систему (і може бути ще більше таких прикладів очікують нас в майбутньому).

І, нарешті, ремонтопридатність визначає, наскільки складно виправити неполадки в описуваної системі. Системи з високою ремонтопридатністю можуть також мати високий ступінь доступності, особливо при наявності засобів автоматичного виявлення і виправлення неполадок. Однак, як ми побачимо пізніше в цій главі, говорити про автоматичне виправлення неполадок набагато простіше, ніж створювати здатні на це системи.

Тип	Опис
Несправність	Сервер перестав працювати, хоча до моменту відмови працював правильно
пропуск даних	Сервер неправильно реагує на вхідні запити
пропуск прийому	Сервер неправильно приймає вхідні запити
пропуск передачі	Сервер неправильно відправляє повідомлення
Помилка синхронізації	Реакція сервера відбувається не в певний інтервал часу
Помилка відгуку	Відгук сервера невірний
помилка значення	Сервер повертає неправильне значення
помилка передачі	Сервер відхиляється від вірного потоку управління
довільна помилка	Сервер відправляє випадкові повідомлення в випадкові моменти

Основний метод маскування помилок - використання надмірності (redundancy). Можливе застосування трьох типів надмірності - інформаційної надмірності, тимчасової надмірності і фізичної надмірності [218]. У разі інформаційної надмірності до повідомлення додаються додаткові біти, за якими можна провести виправлення збійних бітів. Так, наприклад, можна додати до передаваемимданним код Хеммінга для відновлення сигналу в разі зашумленного каналу передачі.

При тимчасової надмірності вже виконану дію при необхідності здійснюється ще раз. Як приклад до цього способу розглянемо транзакції (описані в розділі 5). Якщо транзакція

була перервана, її можна без будь-яких побоювань повторити. Тимчасова надмірність особливо корисна, якщо ми маємо справу з прохідним або переміжним відмовою.
У разі фізичної надмірності ми додаємо в систему додаткове обладнання або процеси, які уможливлюють роботу системи при втраті або непрацездатності деяких компонентів. Фізична надмірність, таким чином, може бути як апаратної, так і програмної. Так, наприклад, можна додати до системи додаткові процеси, так що у випадку краху деяких з них система продовжить функціонувати правильно. Іншими словами, за допомогою реплікації досягається високий ступінь відмовостійкості.

27. Відмовостійкість процесів. Питання розробки. Маскування помилок і реплікація.

Питання розробки

Основний підхід до захисту від наслідків відмови процесів - об'єднати кілька ідентичних процесів в групу. Основна властивість всіх подібних груп полягає в тому, що коли повідомлення надсилається групі, його отримують всі члени цієї групи. Таким чином, якщо один з процесів групи перестає працювати, можна сподіватися на те, що його місце займе інший .

Групи процесів можуть бути динамічними. Можуть створюватися нові групи і ліквідуватися старі. В ході системної операції процес може увійти в групу або покинути її. Процес може входити в кілька груп одночасно.

Мета угруповання полягає в тому, щоб перейти від розгляду окремих процесів до розгляду нової абстракції - групи процесів. Так, процес може посилати повідомлення групі серверів, не знаючи нічого про те, скільки їх там і де вони знаходяться, причому склад групи серверів при кожному виклику може бути різним.

Однорангові і ієрархічні групи

Всі групи можна розділити відповідно за їх внутрішньою структурою. У деяких групах всі процеси рівні між собою. Ніяких начальників немає, і всі рішення приймаються колективно. В інших групах існує щось на зразок ієрархії. Так, наприклад, один з процесів - координатор, а все решта - прості виконавці.

Будь-яка з цих організацій має свої переваги і недоліки. Однорангова група симетрична і не має одиничної точки відмови. Якщо в одному з процесів знаходять помилку, група просто стає менше, але продовжує існувати. Недолік тимчасових груп полягає в тому, що процес прийняття рішень більш складний. Так, наприклад, для того щоб домовитися про щось, необхідно проводити голосування, що тягне за собою певну затримку і необхідність додаткових дій.

Ієрархічна група володіє протилежними властивостями. Втрата координатора тягне за собою зупинку роботи всієї групи, але поки він знаходиться в робочому стані, приймає рішення сам, нікого при цьому не турбуючи.

Маскування помилок і реплікація.

Групи процесів пропонують рішення частини завдання побудови відмовостійких систем. Зокрема, група ідентичних процесів дозволяє замаскувати наявність в цій групі одного або більше відмовили процесів. Іншими словами, ми можемо реплікувати процеси і організувати їх в

групу, замінюючи одиночний (вразливий) процес відмовостійкою групою. Є два способи проведення подібної реплікації - з використанням протоколів на основі первинної копії або протоколів реплікуючого запису.

У разі відмовостійкості реплікація на основі первинної копії зазвичай застосовується у формі протоколу первинного архівування. В цьому випадку група процесів організується в ієрархію, в якій первинна копія координує всі операції запису. На практиці первинна копія фіксована, хоча при необхідності її роль може взяти на себе одна з архівних копій. Насправді при помилку в первинної копії архівні копії, використовуючи певний алгоритм голосування, вибирають нову первинну копію.

Протоколи використовуються в формі активної реплікації або протоколів на основі кворуму. Ці рішення і застосовуються для організації набору ідентичних процесів в однорангову групу. Її головна перевага полягає в тому, що така група не має єдиної точки відмови, а ціною за цю перевагу є розподілена координація.

Важливе питання використання груп процесів для підвищення відмовостійкості полягає в тому, наскільки значною повинна бути реплікація. Система буде називатися стійкою до k відмов, якщо вона залишиться працездатною після відмови k компонентів. Якщо компоненти, звані також процесами, зупиняються без повідомлення, то наявності k + 1 процесів досить, щоб забезпечити стійкість до k відмов. Якщо k з них просто припинять роботу, буде використаний відповідь від одного залишився

28. Захист. Загальні питання захисту. Загрози, правила і механізми.

Одна з складових частин розподілених систем - це захист. Навряд чи хто-небудь може посперечатися з тим, що це одна з найбільш складних частин, адже захист повинен пронизувати всю систему цілком.

Системи захисту в розподілених системах можна розділити на дві незалежні частини.

Одна з них - це зв'язок між користувачами або процесами, можливо, розташованими на різних машинах.

Інша частина систем захисту - це авторизація, яка дозволяє гарантувати, що процеси отримають тільки ті можливості доступу до ресурсів розподіленої системи, на які мають право. Авторизацію і контроль доступу можна розглядати спільно.

Загрози, правила і механізми

Захист в комп'ютерних системах жорстко пов'язаний з поняттям надійності. Говорячи неформально, надійною комп'ютерною системою вважається система, службам якої ми виправдано довіряємо. Адже надійність передбачає доступність, безвідмовність, безпеку і ремонтопридатність.

Інший спосіб поглянути на захист в комп'ютерних системах - вважати, що ми намагаємося захистити служби та дані від загроз захисту. Ми виділяємо чотири типи загроз захисту [351]:

♦ перехоплення (interception);

+ Переривання (interruption);

♦ модифікація (modification);

+ Підробка (fabrication).

Перехопленням ми називаємо таку ситуацію, коли неавторизований агент отримує доступ до служб або даними. Типовий приклад перехоплення - коли зв'язок між двома агентами підслуховує хтось третій.

Прикладом переривання може служити пошкодження або втрата файлу.

Модифікації включають в себе неавторизовані зміни даних або фальсифікацію служб з тим, щоб вони не відповідали своєму оригінальному призначенню.

Підробці відповідає ситуація, коли створюються додаткові дані або здійснюється діяльність, неможлива в нормальних умовах.

Відзначимо, що переривання, модифікація і підробка можуть розглядатися як форми фальсифікації даних.

Правила захисту (security policy) точно описують дозволені і заборонені дії для системних сутностей. У поняття «системні сутності» входять користувачі, служби, дані, машини і т. П. Найбільш важливі з них:

Шифрування (encryption);

Аутентифікація (authentication);

Авторизація (authorization);

Аудит (auditing).

Шифрування - фундамент комп'ютерної захисту. Іншими словами, шифрування - це засіб реалізації конфіденційності. Крім того, шифрування дозволяє нам перевірити, чи не змінювалися чи дані, даючи можливість контролювати цілісність даних.

Аутентифікація використовується для перевірки заявленого імені користувача, клієнта, сервера тощо. У випадку з клієнтом основна ідея полягає в тому, що до початку роботи служби з клієнтом служба повинна визначити справжність клієнта. Зазвичай користувачі аутентифікують себе за допомогою пароля, однак існують і інші способи аутентифікації клієнта.

Після того як клієнт аутентифікований, необхідно перевірити, чи має він право на проведення запитуваних дій.

Засоби аудиту використовуються для контролю за тим, що робить клієнт і як він це робить. Хоча кошти аудиту не захищають від загроз захисту, журнали аудиту постійно використовуються для аналізу «дірок» в системах захисту з подальшим прийняттям заходів проти порушників.

29. Питання розробки. Криптографія.

30. Захищені канали. Аутентифікація. Цілісність і конфіденційність повідомлень.

31. Контроль доступу. Управління захистом. Управління ключами.

У моделі клієнт-сервер запити включають операції над ресурсами, контрольовані сервером. Запит клієнта зазвичай звертається до методу конкретного об'єкта. Такий запит може бути виконаний тільки в тому випадку, якщо клієнт має достатні права доступу (access rights). Підтвердження прав доступу називається контролем доступу (access control).

Загальноприйнятий підхід до моделювання прав доступу суб'єкта по відношенню до об'єктів полягає в побудові матриці контролю доступу (access control matrix). Кожен суб'єкт представлений рядком цієї матриці, кожен об'єкт - стовпчиком. Якщо суб'єкт S намагається звернутися до методу T об'єкта O, монітор посилань повинен перевірити, чи зазначений метод т в M [s, o]. Якщо T в M [s, o] відсутня, виклик не відбудеться. Оскільки системі може знадобитися підтримувати тисячі користувачів і мільйони потребують захисту об'єктів,

реалізація матриці контролю доступу в вигляді реальної матриці є далеко не найкращим рішенням.

Один з найпопулярніших способів полягає в тому, що кожен з об'єктів підтримує список прав доступу суб'єктів, які мають доступ до цього об'єкта. Це означає, що матриця розбивається на стовпці, які розподіляються по об'єктах, при цьому порожні елементи відкидаються. Такий спосіб реалізації є так званим списком контролю доступу (Access Control List, ACL). Вважається, що кожен об'єкт має власний, асоційований тільки з ним ACL.

Управління захистом та ключами:

Щоб створити захищений канал, по якому дві сторони можуть передавати повідомлення, потрібно створити загальний секретний ключ. Один із найбільш складних елементів управління ключами - це поширення вихідних ключів. У симетричних криптосистемах вихідний загальний секретний ключ повинен поширюватися по безпечних каналах з аутентифікацією і конфіденційно. У разі асиметричні алгоритми шифрування нам необхідно поширити відкритий ключ так, щоб одержувачі були впевнені, що ключ дійсно є парою заявленому закритому ключу. Іншими словами, як показано на рис. 8.31, б, хоча відкритий ключ сам по собі може бути посланий навіть простим текстом, необхідно, щоб канал, по якому він пересилається, забезпечував аутентифікацію. Закритий ключ, зрозуміло, необхідно пересилати тільки по безпечному каналу, з ідентифікацією та конфіденційно.

32. Розподілена система об'єктів C0RBA. Огляд. Зв'язок.

CORBA (англ. *Common Object Request Broker Architecture*) — це запропонований консорціумом OMG технологічний стандарт розробки розподілених застосунків.

Завдання CORBA — інтегрувати розподілені системи, дати можливість програмам, що написані різними мовами та працюють у різних вузлах мережі, взаємодіяти одна з одною так само просто, наче вони знаходяться в адресному просторі одного процесу.

CORBA об'єднує програмний код в об'єкт, що містить інформацію про функціональність та інтерфейси доступу. Готові об'єкти можуть викликатися з інших програм або розташованих у мережі об'єктів CORBA.

CORBA використовує мову опису інтерфейсів OMG IDL для визначення протоколів взаємодії об'єктів із зовнішнім світом. Стандарт CORBA описує правила відображення IDL у мову реалізації об'єкта: Ada, C, C++, Lisp, Smalltalk, Java, COBOL, PI/i і Python. Також існують нестандартні відображення у Perl, Visual Basic, Ruby і Tcl, які реалізовані розробленими для них засобами ORB.

Об'єкти за значенням (ОЗЗ)

Крім віддалених об'єктів в CORBA 3.0 визначено поняття ОЗЗ. Код методів таких об'єктів за замовчуванням виконується локально. Якщо ОЗЗ був отриманий з віддаленого боку, то необхідний код повинен або бути наперед відомий обом сторонам, або бути динамічно завантажений. Щоб це було можливо, запис, що визначає ОЗЗ, містить поле Code Base — список URL, звідки може бути завантажено код. У ОЗЗ можуть також бути і віддалені методи.

Компонентна модель CORBA (CCM) — описує стандартний каркас додатку для компонент CORBA. CCM побудований під сильним впливом Enterprise Javabeans (EJB) і фактично є його незалежним від мови розширенням.

ORB, Object Request Broker (брокер об'єктних запитів) — це об'єктна шина, яка дає змогу об'єктам напряму виробляти і відповідати на запити інших об'єктів, розташованих як локально (на одному комп'ютері, але в різних процесах), так і віддалено. Клієнта не цікавлять комунікаційні та інші механізми, з використанням яких відбувається взаємодія між об'єктами, виклик і збереження серверних компонентів.

33. Розподілена система об'єктів DCOM. Огляд. Зв'язок.

Об'єктна модель розподілених компонент (*Distributed Component Object Model*, скорочено DCOM) — власна технологія Microsoft для організації взаємодії між компонентами програмного забезпечення, розподіленого між комп'ютерами в мережі. DCOM, що спочатку мала назву «Network OLE», розширює інтерфейс Microsoft COM і забезпечує нижні рівні зв'язку з інфраструктурою сервера Microsoft COM+. Наразі, ця технологія застаріла та замінена Microsoft .NET.

Доповнення «D» до COM відбулося завдяки використанню DCE/RPC — більш розширеної версії Microsoft, відомої як MSRPC.

У термінах розширень які були додані до задач COM, DCOM повинен розв'язати проблеми:

1. Маршалінг — серіалізація та десеріалізація аргументів та поворотних значень викликів методів «на дроті».
2. Розподілене прибирання сміття — гарантуючи, що посилання, утримувані клієнтами інтерфейсів звільнені, наприклад, тоді, коли процес клієнта перервався, або зв'язок з мережею втрачено.

Один із ключових факторів у вирішенні цих проблем — використання DCE/RPC як основного механізму RPC поза DCOM. DCE/RPC чітко визначає правила щодо маршалінгу і відповідальності за вивільнення пам'яті. Крім того, технологія DCOM, підтримку якої має будь-яка ОС сімейства Windows, поєднує переваги технологій доступу до даних з клієнт-серверною технологією.

34. Розподілена система об'єктів Globe. Огляд. Зв'язок.

Globe - це розподілена система об'єктів, в якій особливу роль грає масштабованість. Структура Globe визначалася завданнями побудови великих глобальних систем, здатних підтримувати велику кількість користувачів і об'єктів. Основним при такому підході є метод перегляду об'єктів. Як і в разі інших систем об'єктів, об'єкти в Globe розглядаються як інкапсуляція сутностей та операцій над ними.

Важлива відмінність Globe від інших систем об'єктів, особливо від інших масштабованих систем, таких як Legion полягає в тому, що об'єкти можуть також інкапсулювати реалізацію правил розподілу стану об'єктів за кількома машинам.

Взагалі кажучи, об'єкти в Globe відповідають за все, що тільки можна. Так, наприклад, об'єкт визначає, як, коли і куди може переміститися його стан. Крім того, об'єкт вирішує, чи слід робити репліки його стану і, якщо так, як саме має відбуватися реплікація. Об'єкт може також визначати свої правила захисту та реалізацію. Нижче ми опишемо, яким чином досягається така інкапсуляція.

Об'єктна модель

На відміну від більшості інших розподілених систем об'єктів, Globe не працює на віддалених об'єктів. Об'єкти в Globe мають здатність зберігатися в фізично роздробленому стані, тобто стан об'єктів може бути розподілено між декількома процесами.

Інтерфейс

Процес, пов'язаний з розподіленим об'єктом, отримує локальну реалізацію інтерфейсів цього об'єкта. Ця локальна реалізація називається локальним поданням (local representative), або просто локальним об'єктом (local object). В принципі, так чи інакше, локальний об'єкт має стан.Вся реалізація об'єкта прихована за пропонованими процесу інтерфейсами.Всякий локальний об'єкт реалізує стандартний інтерфейс SOInf об'єкта, подібний до інтерфейсу IUnknown в DCOM. Зокрема, подібно Querylnterface в DCOM, SOInf має метод getlnf, який отримує в якості вихідних даних ідентифікатор інтерфейсу і повертає покажчик на цей інтерфейс, дозволяючи іншим процесам отримати доступ до його реалізації в об'єкті. *Є й інші приклади подібності між локальними об'єктами Globe і об'єктами DCOM. Так, наприклад, передбачається, що кожен локальний об'єкт має відповідний об'єкт класу, який може створювати нові локальні об'єкти.*

Локальні об'єкти реалізують бінарні інтерфейси, які складаються в основному з таблиць покажчиків на функції. Інтерфейси описуються на мові визначення інтерфейсів, який в основному схожий на аналогічні мови, які використовуються в CORBA і DCOM, але має деякі приватні відмінності.

Існує два різновиди локальних об'єктів Globe. Primitive local object - це локальний об'єкт, який не містить інших локальних об'єктів. На противагу йому, composite local object - це об'єкт, який складався з кількох (можливо також складових) локальних об'єктів. Для підтримки складових об'єктів таблиця інтерфейсів локальних об'єктів Globe складається з пар покажчиків (стан, метод). Кожен покажчик на стан відповідає даним, що належить одному конкретному локальному об'єкту. У разі інтерфейсу примітивного локального об'єкта всі покажчики на стан визначають одні і ті ж дані, що відносяться до стану самого об'єкта. У разі складових об'єктів покажчики на стан визначають стану різних об'єктів, що входять в цей складений об'єкт.

У плані відновлення після збоїв сервер об'єктів Globe гарантує, що локальні об'єкти, які слід відновлювати, будуть збережені на диску.

35. Порівняння систем CORBA, DOOM і Globe.

а) філософія

Системи розроблялись з різними цілями. CORBA це фактичний результат спроб створення стандартної платформи проміжного рівня на якій мали б змогу спільно працювати застосунки від різних виробників.

Основною ціллю створення DCOM було розширення функціональних можливостей при збереженні сумісності з існуючими версіями,які були включені в попередні системи Windows. Основна ціль Globe - забезпечення масштабування.

CORBA надає стандартну мову IDL, на якій виконуються визначення інтерфейсів з подальшим перетворенням на тексти програм на вибраній мові програмування. У DCOM використовуються бінарні інтерфейси (табличні). При такому підході інтерфейси об'єктів визначаються незалежно від мови програмування.

б)запити

Розподілені системи об'єктів майже завжди базуються на моделі віддалених об'єктів. Такий підхід часто потребує додаткових настройок, зокрема для підтримки кешування та реплікації. в CORBA для зміни вихідних та вхідних запитів використовуються перехоплювачі, в Globe настроювання виконується за допомогою різних локальних об'єктів, які всі разом утворюють єдиний розподілений об'єкт.

Крім синхронних звернень до методів CORBA і DCOM підтримують асинхронні звернення та події. Globe не допускає таких альтернатив.

в)сервери

Організація серверів об'єктів загалом дуже подібна. У всіх випадках сервер може підтримувати більше ніж один об'єкт. Відмінності полягають в гнучкій настройці сервера. В CORBA гнучкість забезпечується за допомогою адаптерів об'єктів. DCOM пропонує стандартний сервер об'єктів, який може бути перебудований під конкретний застосунок. В Globe сервери об'єктів доволі прості, бо спеціальні властивості реалізовуються всередині розподілених об'єктів.

г) іменування

Відмінність полягає в посиланнях на об'єкти. І в CORBA, і в DCOM посилання залежать від їх місцезнаходження, а в GLOBE ні, але для цієї системи потрібна глобальна служба локалізації для розширення посилань в контактні адреси.(Вони визначають де і як можна знайти даний об'єкт).

д) реплікація

В CORBA реплікація підтримується лише в плані відмовостійкості. DCOM взагалі не допускає реплікації. Розробник застосунку має за необхідності використовувати спеціалізований сервер реплікації або явну програмну реплікацію. В Globe реплікація підтримується кожним об'єктом окремо.

е) захист

CORBA пропонує повністю захищену архітектуру, яка забезпечує захист кожного об'єкта окремо.Захист в DCOM тісно пов'язаний з організацією доступу до вже існуючих служб захисту(напр. KERBEROS). В Globe захист також визначається для кожного об'єкта індивідуально, при цьому об'єктам надаються засоби для прив'язки до існуючих служб захисту.

36. Розподілені файлові системи. Мережева файлова система компанії Sun

Розподілені файлові системи дозволяють декільком процесам протягом тривалого часу спільно працювати зі спільними даними, забезпечуючи їх надійність і захищеність.

Network File System (NFS) — протокол мережевого доступу до файлових систем, спочатку розроблений Sun Microsystems в 1984 році. Заснований на протоколі виклику віддалених процедур Дозволяє підключати (монтувати) віддалені файлові системи через мережу.

NFS абстрагована від типів файлових систем як сервера, так і клієнта, існує безліч реалізацій NFS-серверів і клієнтів для різних операційних систем і апаратних архітектур. У цей час (2007) використовується найзріліша версія NFS v.4, що підтримує різні засоби аутентифікації (зокрема, Kerberos і LIPKEY) і списків контролю доступу.

pNFS (паралельний NFS) — включена в найсвіжішу версію стандарту NFS v4.1 специфікація, яка забезпечує реалізацію загального доступу до файлів, що збільшує швидкість передачі даних пропорційно розмірам і ступеню паралелізму системи.

NFS надає клієнтам прозорий доступ до файлів і файлової системи сервера. На відміну від FTP протокол NFS здійснює доступ тільки до тих частин файлу, до яких звернувся процес, і основна перевага його в тому, що він робить цей доступ прозорим. Це означає, що будь-який застосунок клієнта, що може працювати з локальним файлом, з таким же успіхом може працювати й з NFS файлом, без будь-яких модифікацій самої програми.

NFS-клієнти одержують доступ до файлів на NFS-сервері шляхом відправлення RPC запитів на сервер.

Реалізація NFS складається з кількох компонентів. Деякі з них локалізовані або на сервері, або на клієнті, а деякі використовуються й тим і іншим:

- Протокол NFS визначає набір запитів (операцій), які можуть бути спрямовані клієнтом до сервера, а також набір аргументів і значення, які повертаються, для кожного із цих запитів.
- Протокол віддаленого виклику процедур (RPC) визначає формат всіх взаємодій між клієнтом і сервером. Кожний запит NFS посилається як пакет RPC.
- Зовнішнє подання даних (XDR — External Data Representation) забезпечує машинно-незалежний метод кодування даних для пересилання через мережу.
- Програмний код сервера NFS відповідає за обробку всіх запитів клієнта й забезпечує доступ до експортованих файлових систем.
- Програмний код клієнта NFS реалізує всі звернення клієнтської системи до віддалених файлів шляхом посилки серверу одного або декількох запитів RPC.

- Протокол монтування визначає семантику монтування й розмонтування файлових систем NFS. NFS використовує кілька фонових процесів-демонів. На сервері набір демонів nfsd очікують запити клієнтів NFS і відповідають на них.
- Демон mountd обробляє запити монтування. На клієнті набір демонів biod обробляє асинхронне уведення/вивід блоків файлів NFS.
- Менеджер блокувань мережі (NLM — Network Lock Manager) і монітор стану мережі (NSM — Network Status Monitor) разом забезпечують засоби для блокування файлів у мережі.
-

37. Файлова система Coda

Coda - Розподілена (мережева) файлова система (ФС), розроблена як дослідницький проект в університеті Карнегі - Меллона в 1987 році під керівництвом Махадева Сатьянарайанана (англ. Mahadev Satyanarayanan). Дана файлова система розроблена на основі старої версії AFS (AFS-2) і має безліч схожих можливостей. Поширюється під ліцензією GNU GPL.

Coda все ще знаходиться в розробці, але акцент зміщується від наукових дослідження в бік створення надійного продукту для комерційного використання.
Система Coda в даний час інтегрована в кілька популярних операційних систем на базі UNIX, наприклад в Linux. Coda у багатьох відношеннях відрізняється від NFS, особливо щодо високої доступності. Вимога забезпечити високу доступність змусила розробників застосовувати вдосконалені схеми кешування, які дозволяють клієнту продовжувати операцію навіть у разі відключення від сервера.

Можливості

(необхідні для мережевих (розподілених) файлових систем)

- Coda знаходиться у вільному доступі під ліберальної ліцензією
- Відключені операції для мобільних обчислень
- Висока продуктивність на клієнтській стороні завдяки постійному кешуванню
- Реплікація сервера
- Модель безпеки для аутентифікації, шифрування і управління доступом
- Продовження роботи при збоях в серверній мережі
- Адаптація до пропускної здатності мережі
- Хороша розширюваність
- Хороша семантика обміну, навіть в разі збоїв мережі

Підтримувані платформи

Coda спочатку була розроблена для UNIX-платформ. В даний час, вона включена в ядро Linux 2.6. Також Coda була перенесена на FreeBSD. Існує проект по перенесенню даної ФС на платформи Microsoft Windows, починаючи від ери Windows 95 / Windows 98, до Windows NT [3] і Windows XP, [4] за допомогою проектів з відкритим вихідним кодом на кшталт DJGCC DOS C Compiler і Cygwin.

38. Розподілені системи документів. World Wide Web. Зв'язок. Процеси.

Розподілені системи документів

Розподілена система — це набір незалежних комп’ютерів, що постають перед користувачем єдиною об’єднаною системою.

В цьому визначенні оговорюються два момента. Перший відноситься до апаратури: вся машина автономна. Другий стосується програмного забезпечення: користувачі думають, що мають справу з єдиною системою. Важливі обидва момента.

Однією з найважливіших причин, що сприяли популярності як мержі, так і розподілених систем, стала поява Всесвітньої павутини (World Wide Web). Перевага Web полягає у відносній простоті парадигми: все навколо - це документи.

Web в даний час є найважливішою розподіленою системою документів.

Іншою важливою системою документів, що з'явилася раніше Web, є Lotus Notes. В протилежність Web в основі системи Notes лежать не файли, а бази даних.

World Wide Web

World Wide Web (WWW) можна вважати гігантською розподіленою системою, яка для доступу до пов'язаних документів містить мільйони клієнтів і серверів. Сервери підтримують набори документів, а клієнти надають користувачам простий інтерфейс для доступу і проглядання цих документів.

WWW - це, по суті, гігантська система з архітектурою клієнт-сервер і мільйонами серверів по всьому світу. Кожний сервер підтримує набір документів, кожний документ міститься у файлі (хоча в деяких випадках документи можуть генеруватися за запитом). Сервер приймає запити на видачу документів і передає їх клієнту. Крім того, він може приймати запити на збереження нових документів.

Найпростіший спосіб посилання на документ - уніфікований покажчик ресурсу (Uniform Resource Locator, URL). URL-адреса подібна міжопераційному посиланню на об'єкт (IOR) в CORBA або контактній адресі в Globe. Вона визначає, де знаходиться цільовий документ. Звичайно це робиться шляхом вбудовування в початковий документ DNS-імені відповідного серверу разом з ім'ям файлу, за допомогою якого сервер може знайти цільовий документ в своїй локальній файловій системі. Крім того, URL визначає протокол прикладного рівня, що використовується для пересилки документа по мережі. Протоколів існує декілька.

Клієнт взаємодіє з web-серверами за допомогою спеціальної програми, званої браузером (browser). Браузер відповідає за правильне відображення документа. Крім того, браузер обробляє операції введення від користувача, основна з яких - вибір посилання на інший документ, який потім витягується з сховища і виводиться на екран.

Зв'язок

Взаємодія в Web відбувається по спеціальному протоколу HTTP, який визначає всі допустимі операції з документами, що в основному обмежуються отриманням документа з серверу і переміщенням його на сервер.

Взаємодія в Lotus Notes підтримується за допомогою традиційної підсистеми Remote Procedure Call (RPC) – віддаленого виклику процедур. Крім того, в Notes робиться нахил на механізми зв'язку верхнього рівня, в основному електронну пошту, для чого в цій системі передбачені різноманітні засоби автоматичного прийому, обробки і відправки пошти.

Слід зазначити, що взаємодія між процесами на одній машині також здійснюється по-різному. В Web поведінку клієнтів і серверів в цьому відношенні жорстко визначається базовою операційною системою в тому значенні, що будь-який зв'язок між процесами реалізується механізмами операційної системи. З іншого боку, в Notes для маскування різниці між операційними системами клієнтів і серверів використовується спеціальний рівень, NOS. Такий підхід покращує переносимість багатьох прикладних програм Notes.

Процеси

Порівнюючи Web і Lotus Notes через процесі, видно, що багато задач розв'язуються в них схожим чином. Найважливішими клієнтами для Web є браузери, що надають користувачам графічний інтерфейс для отримання і проглядання документів.

У Lotus Notes клієнту також надається графічний інтерфейс для проглядання записів, що зберігаються у віддаленій базі даних. Крім того, клієнти Notes звичайно мають спеціальні програми для редагування записів за допомогою форм, представлень і т.п. Оскільки клієнт Notes призначений для роботи зі всіма типами записів (наперед визначеними), необхідності в наданні йому додаткової функціональності не виникає.

Організація серверів цих двох систем також має схожі риси за винятком того, що традиційний web-сервер звичайно більш пристосований для роботи з файловою системою. На противагу цьому сервер Notes Domino ближчий до традиційних серверів баз даних.

Гнучкість серверної частини Web реалізується за допомогою програм CGI. Ці програми звичайно запускаються у вигляді окремого процесу і можуть взаємодіяти з базами даних. Таким чином, можливості web-серверу наближаються до можливостей серверу Domino. Крім того, web-сервери можуть підтримувати модулі, які динамічно завантажуються, і відомі як сервлети.

39. Розподілені системи узгодження. Огляд. Зв'язок. Процеси.

Моделі узгодження.

Основним підходом, який використовується в системах узгодження, є відділення обчислювальних процесів від механізмів їх узгодження.В тому випадку, коли процеси володіють зв'язністю посилань і часу та узгодження здійснюється безпосередньо, назвемо його прямим узгодженням , зв'язність посилань зазвичай має вид явної ідентифікації співбесідника в процесі взаємодії. Так,

наприклад, процес може взаємодіяти з іншим процесом тільки в тому випадку, якщо він знає ідентифікатор процесу, з яким хоче обмінятися інформацією. Тимчасова зв'язність означає, що обидва що взаємодіють один з одним процесу активні одночасно.

Інший тип узгодження спостерігається в тому випадку, якщо процеси не зв'язані за часом, але зв'язані по посиланнях. Називається узгодженням через поштову скриньку . В цьому випадку для взаємодії зовсім не потрібно, щоб два що взаємодіють один з одним процесу виконувалися одночасно. Замість цього взаємодія відбувається шляхом посилки повідомлень в поштову скриньку, можливо, використовуваний спільно.

Комбінація зв'язності за часом і незв'язності по посиланнях утворює групу моделей узгодження на зустрічі . У незв'язній по посиланнях системі процеси не мають повної інформації один про одного. Іншими словами, коли процес хоче погоджувати свою діяльність з іншими процесами, він не може звернутися до них безпосередньо. Натомість використовується метафора зустрічі, на якій збираються процеси, щоб скоординувати свою діяльність. Модель припускає, що процеси, що беруть участь в зустрічі, виконуються одночасно.

Найбільш широко відомий варіант узгодження — це поєднання не зв'язкових за часом і по посиланнях процесів, представлений генеративним зв'язком. Основна ідея генеративного зв'язку полягає в тому, що набір незалежних процесів може використовувати підлягаючий зберіганню простір даних, що розділяється, організовуваний за допомогою кортежів. Кортежі — це іменовані записи, що містять декілька типізованих полів. Процес може поміщати в простір даних запису будь-якого типу, що розділяється,

TIB/Rendezvous

Система TIB/Rendezvous спочатку була описана в поняттях інформаційної шини (information bus), мінімальної комунікаційної системи групи процесів, заснованої на наступних принципах. Ступінь залежності комунікаційної системи від прикладних програм ядра дуже низка. Так, наприклад, з ядра повністю виключена складна семантика впорядкування повідомлень, оскільки передбачається, що ці питання вирішуються на прикладному рівні.

Другий принцип побудови системи полягає в тому, що повідомлення описують себе самі. На практиці це означає, що додаток може перевірити вхідне повідомлення, щоб визначити, яка його структура і які дані воно містить. Відмітимо, що в протилежність цьому правилу в більшості комунікаційних систем передбачається, що процес вже обізнаний про формат вхідних повідомлень і йому залишається лише вірно інтерпретувати їх зміст.

Третій принцип побудови полягає в тому, що процеси не мають посилальної зв'язності. Причина введення цього принципу викликана тим, що обслуговування працюючої системи не повинне вести до її зупинки, а також необхідністю спростити додавання нових процесів «на льоту». Ці вимоги простіше виконати в тому випадку, коли процеси не посилаються явним чином один на одного. Посилальна незв'язність забезпечується шляхом використання адресації по темі.

Jini

Наступним прикладом системи узгодження, є система Jini компанії Sun Microsystems. Віднесення Jini до систем узгодження засноване в першу чергу на тому, що ця система здатна підтримувати генеративний зв'язок за допомогою Linda-подобной служби під назвою JavaSpace, на якій і буде зосереджено нашу увагу. Існує багато служб і засобів, які роблять Jini більше, ніж просто системою узгодження. Проте все це тільки надає належні ваги вибору цієї системи як приклад.

Jini — це розподілена система, що складається з різних, але взаємозв'язаних елементів. Вона жорстко прив'язана до мови програмування Java, хоча багато з її принципів рівно можуть бути реалізовані і за допомогою інших мов. Важливою частиною системи є модель узгодження генеративного зв'язку. Перш ніж говорити про загальну архітектуру системи Jini, давайте обговоримо цю модель.

Порівняння TIB/Rendezvous і Jini.

Огляд

Дві системи узгодження, є характерними представниками розподілених систем узгодження.

Як TJB/Rendezvous, так і Jini націлені на організацію посилальної незв'язності процесів, тобто на надання засобів, за допомогою яких взаємодіючі процеси могли б за бажання залишатися більш менш безіменними. TIB/Rendezvous посилальну незв'язність забезпечує за допомогою механізму публікації/підписки, а Jini — генеративним зв'язком через JavaSpace. Крім того, Jini забезпечує ще і тимчасову незв'язність процесів.

Інша відмінність між цими системами полягає в тому, що TIB/Rendezvous обслуговує велику частину взаємодії між процесами. У Jini же головна ідея полягає в тому, що система повинна представити процеси один одному, але відразу після цього взаємодія між ними забезпечується зверненнями RMJ мови Java.

Зв'язок

Взаємодія в TIB/Rendezvous здійснюється в першу чергу за допомогою базового механізму публікації/підписки. В результаті в цій системі найважливішу роль грає групова розсилка. Робляться спеціальні заходи, щоб гарантувати успішність групової розсилки і в глобальних мережах. Щоб зв'язок не залежав від прикладних програм, повідомлення зроблені такими, що самовизначаються.

В протилежність цій системі Jini, по суті, використовує групову розсилку тільки для того, щоб відразу знайти служби пошуку, які потім допомагають клієнтові шукати інші процеси. Будь-яка інша взаємодія в основному реалізується зверненнями RMI мови Java, у тому числі і взаємодія з серверами JavaSpace.

У обох системах події грають важливу, але різну роль. У TIB/Rendezvous механізм подій обслуговує всі взаємодії. В принципі вхідне повідомлення може бути отримане тільки в тому випадку, якщо безпосередньо у одержувача встановлений обробник події для цього повідомлення. З погляду прикладної програми це означає, що для обробки вхідних повідомлень не потрібні блокуючі операції.

Події в Jini мають іншу природу. Кожен процес може запропонувати службу подій, завдяки якій з'являється можливість зареєструвати інший процес для подальшої передачі повідомлень. Коли відбувається певна подія, зареєстрований процес робить зворотний виклик, який може бути відповідним чином оброблений. Механізм подій Jini — це, по суті, служба зворотного виклику, яка організовується між парами процесів.

Процеси

Оскільки TIB/Rendezvous і Jini, по суті, містять тільки засоби взаємодії між процесами, в способах організації самих процесів немає нічого особливого. Крім того, хоча обидві системи підтримують безліч специфічних процесів, які реалізують, наприклад, транзакції, не застосовують ніяких спеціальних мір, щоб зробити ці процеси хоч би трохи відмінними від процесів призначених для користувача застосувань.

40. Технологія CUDA. Платформа паралельних обчислень CUDA. Переваги CUDA.

CUDA - програмно-апаратна архітектура паралельних обчислень, яка дозволяє істотно збільшити обчислювальну продуктивність завдяки використанню графічних процесорів фірми Nvidia. CUDA SDK дозволяє програмістам реалізовувати на спеціальному спрощеному діалекті мови програмування С алгоритми, здійснювані на графічних процесорах Nvidia, і включати спеціальні функції в текст програми на С. Архітектура CUDA дає розробнику можливість на свій розсуд організовувати доступ до набору інструкцій графічного прискорювача і управляти його пам'яттю.

Паралельні обчислення з CUDA

Напрямок обчислень еволюціонує від «централізованої обробки даних» на центральному процесорі до «спільної обробки» на CPU і GPU. Для реалізації нової обчислювальної парадигми компанія NVIDIA винайшла архітектуру паралельних обчислень CUDA, на даний момент представлену в графічних процесорах GeForce, ION, Quadro і Tesla і забезпечує необхідну базу розробникам ПЗ. Говорячи про споживчому ринку, варто зазначити, що майже всі основні програми для роботи з відео вже обладнані, або будуть оснащені підтримкою CUDA-прискорення, включаючи продукти від Elemental Technologies, MotionDSP і LoiLo. Область наукових досліджень з великим ентузіазмом зустріла технологію CUDA. Приміром, зараз CUDA прискорює AMBER, програму для моделювання молекулярної динаміки, використовувану більше 60000 дослідниками в академічному середовищі і фармацевтичними компаніями по всьому світу для скорочення термінів створення лікарських препаратів.

На фінансовому ринку компанії Numerix і CompatibL анонсували підтримку CUDA в новому додатку аналізу ризику контрагентів і досягли прискорення роботи в 18 разів. Numerix використовується майже 400 фінансовими інститутами. Показником зростання застосування CUDA є також зростання використання графічних процесорів Tesla в GPU обчисленнях. На даний момент більше 700 GPU кластерів встановлені по всьому світу в компаніях зі списку Fortune 500, таких як Schlumberger і Chevron в енергетичному секторі, а також BNP Paribas в секторі банківських послуг. Завдяки нещодавно випущеним системам Microsoft Windows 7 і Apple Snow Leopard, обчислення на GPU займуть свої позиції в секторі масових рішень. У цих нових операційних системах GPU постане не тільки графічним процесором, але також і універсальним процесором для паралельних обчислень, що працюють з будь-яким додатком.

Програмна архітектура

В основі інтерфейсу програмування додатків CUDA лежить мова С з деякими розширеннями. Для успішної трансляції коду на цій мові до складу CUDA SDK входить власний С-компілятор командного рядка nvcc компанії Nvidia. Компілятор nvcc створений на основі відкритого компілятора Open64 і призначений для трансляції host-коду (головного, керуючого коду) і device-коду (апаратного коду) (файлів з розширенням .cu) в об'єктні файли, придатні в процесі складання кінцевої програми або бібліотеки у середовищі програмування, наприклад, в NetBeans. В архітектурі CUDA використовується модель пам'яті GRID, кластерне моделювання потоків і SIMD-інструкції. Застосовна не тільки для високопродуктивних графічних обчислень, але і для різних наукових обчислень з використанням відеокарт nVidia. Учені і дослідники широко

використовують CUDA в різних областях, включаючи астрофізику, обчислювальну біологію та хімію, моделювання динаміки рідин, електромагнітних взаємодій, комп'ютерну томографію, сейсмічний аналіз і багато іншого. У CUDA є можливість підключення до додатків, що використовують OpenGL і Direct3D. CUDA - кросплатформленість для таких операційних систем як Linux, Mac OS X і Windows. 22 березня 2010 nVidia випустила CUDA Toolkit 3.0, який містив підтримку OpenCL.

Платформа паралельних обчислень CUDA

Платформа паралельних обчислень CUDA® забезпечує набір розширень для мов C і C++, що дозволяють висловлювати як паралелізм даних, так і паралелізм завдань на рівні дрібних і великих структурних одиниць. Програміст може вибрати засоби розробки: мови високого рівня, такі як C, C++, Fortran або ж відкриті стандарти, такі як директиви OpenACC. Платформа паралельних обчислень CUDA використовується на сьогоднішній день в тисячах GPU-прискорених додатків і тисячах опублікованих наукових статтях. Повний список засобів розробки і екосистема рішень CUDA доступний розробникам.

Переваги CUDA

У порівнянні з традиційним підходом до організації обчислень загального призначення допомогою можливостей графічних API, у архітектури CUDA відзначають наступні переваги в цій області:

1. Інтерфейс програмування додатків CUDA (CUDA API) заснований на стандартній мові програмування Сі з деякими обмеженнями. На думку розробників, це повинно спростити і згладити процес вивчення архітектури CUDA
2. Колективна між потоками пам'ять (shared memory) розміром в 16 Кб може бути використана під організований користувачем кеш з більш широкою смугою пропускання, ніж при вибірці зі звичайних текстур
3. Більш ефективні транзакції між пам'яттю центрального процесора і відеопам'яттю
4. Повна апаратна підтримка цілочисельних і побітових операцій
5. Підтримка компіляції GPU коду засобами відкритого LLVM.

41. Ієрархічна модель CUDA.

Концепція CUDA відводить GPU роль масивно-паралельного співпроцесора. У літературі про CUDA основна система, до якої підключений GPU, коротко називається терміном хост (host). Аналогічно сам GPU по відношенню до хосту часто називається просто пристроєм (device). CUDA програма задіює як CPU, так і GPU. На CPU виконується послідовна частина коду і підготовчі стадії для GPU-обчислень. Паралельні ділянки коду можуть бути перенесені на GPU, де одночасно виконуватимуться великою кількістю ниток (потоків). Важливо відзначити ряд принципових відмінностей між звичайними потоками CPU і потоками GPU:

1. Потік (thread) GPU надзвичайно легкий, його контекст мінімальний, регістри розподіленні заздалегідь;

2. Для ефективного використання ресурсів GPU програмі необхідно задіяти тисячі окремих потоків, тоді як на багатоядерному CPU максимальна ефективність, зазвичай, досягається при числі потоків, рівному або в кілька разів більшому кількості ядер.

Робота потоків на GPU відповідає принципу SIMD. Проте є суттєва відмінність. Тільки потоки в межах однієї групи (для GPU архітектури Fermi - 32 потоки), так званого варпу (warp) виконуються фізично одночасно. Потоки різних варпів можуть знаходитися на різних стадіях виконання програми. Такий метод обробки даних позначається терміном SIMT (Single Instruction – Multiple Theads). Управління роботою варпів виконується на апаратному рівні. По ряду можливостей нових версій CUDA простежується тенденція до поступового перетворення GPU в самодостатній пристрій, повністю заміняючи звичайний CPU за рахунок реалізації деяких системних викликів (в термінології GPU системними викликами є, наприклад, malloc і free, реалізованих в CUDA 3.2) і додавання полегшеного енергоефективного CPU-ядра в GPU (архітектура Maxwell). Важливою перевагою CUDA є використання для програмування GPU мов високого рівня. В даний час існують компілятори C++ і Fortran. Ці мови розширюються невеликою множиною нових конструкцій: атрибути функцій і змінних, вбудовані змінні і типи даних, оператор запуску ядра.

42. OpenCL - відкритий стандарт і API для апаратного прискорення обчислення на GPU та інших однорідних обчислювальних системах

OpenCL (від англ. Open Computing Language) — фреймворк для створення комп'ютерних програм, пов'язаних з паралельними обчисленнями на різних графічних (англ. GPU) і центральних процесорах (англ. CPU). У фреймворк OpenCL входять мова програмування, яка базується на стандарті C99, та інтерфейс програмування комп'ютерних програм (англ. API). OpenCL забезпечує паралельність на рівні інструкцій та на рівні даних і є реалізацією техніки GPGPU. OpenCL — повністю відкритий стандарт, його використання доступне на базі вільних ліцензій.

Мета OpenCL полягає в тому, щоб доповнити OpenGL і OpenAL, які є відкритими галузевими стандартами для тривимірної комп'ютерної графіки і звуку, користуючись можливостями GPU.

OpenCL замислювався як технологія для створення додатків, які могли б виконуватися в гетерогенному середовищі. Більше того, він розроблений так, щоб забезпечувати комфортну роботу з такими пристроями, які зараз знаходяться тільки в планах і навіть з тими, які ще ніхто не придумав. Для координації роботи всіх цих пристроїв гетерогенній системі завжди є головний пристрій, який

взаємодіє з рештою посередництвом OpenCL API . Такий пристрій називається «хост», він визначається поза OpenCL.

Тому OpenCL виходить з найбільш загальних передумов, що дають уявлення про пристрій з підтримкою OpenCL: так як цей пристрій передбачається використовувати для обчислень - в ньому є якийсь «процесор» в загальному розумінні цього слова. Щось, що може виконувати команди. Крім обчислювальних ресурсів пристрій має якийсь обсяг пам'яті. OpenCL надає програмісту низькорівневий API, через який він взаємодіє з ресурсами пристрою. OpenCL API може або безпосередньо підтримуватися пристроєм, або працювати через проміжний API (як у випадку NVidia: OpenCL працює поверх CUDA Driver API, підтримуваний пристроями), це залежить від конкретної реалізації не описується стандартом.

Для опису основних ідей OpenCL скористаємося ієрархією з 4х моделей:
Модель платформи (Platform Model);
Модель пам'яті (Memory Model);
Модель виконання (Execution Model);
Програмна модель (Programming Model);

Модель платформи (Platform Model).

Платформа OpenCL складається з хоста з'єднаного з пристроями, що підтримують OpenCL. Кожний OpenCL-пристрій складається з обчислювальних блоків (Compute Unit), які далі поділяються на один або більше елементи-обробники (Processing Elements, далі PE).

Модель виконання (Execution Model).

Виконання OpenCL-програми складається з двох частин: хостової частини програми і kernels (ядра), що виконуються на OpenCL-пристрої. Хостова частина програми визначає контекст, в якому виконуються kernel'и, і управляє їх виконанням.

Модель пам'яті (Memory Model).

Work-Item, що виконує kernel, може використовувати чотири різних типи пам'яті:

- Глобальна пам'ять. Ця пам'ять надає доступ на читання і запис елементам всіх груп.
- Константна пам'ять. Область глобальної пам'яті, яка залишається постійною під час виконання kernel'а.
- Локальна пам'ять. Область пам'яті, локальна для групи. Приватна (private) пам'ять. Область пам'яті, що належить Work-Item.

Програмна модель. (Programming Model)

Модель виконання OpenCL підтримує дві програмні моделі: паралелізм даних (Data Parallel) і паралелізм завдань (Task Parallel), так само підтримуються гібридні моделі.