

От консултацията с Васил:

Имаме разпределено изчисление, когато процесите се изпълняват на различни сървъри, а паралелна обработка, когато имаме повече от едно ядро.

При паралелната обработка имаме n на брой процеса; имаме две неща, които ни забавят - време за синхронизация и .. :д

параметри на паралелна обработка са:

ускорение (speed) - $S_n = T_1/T_n$

- тук може да начертаем координатна система, на която по x имаме n - броя процеси, и S_n по y , т.е представяме ускорението като функция на n ; ъглополовящата на квадранта представя перфектното ускорение; чертаем ускорението като близко до тази ъглополовяща, ако имаме линейност

ефективност - $E_n = S_n / n$

Примерна задача: Да се намери сумата на N числа (или някакъв друг прост алгоритъм)

$$T_1 = N$$

$$T_n = N/p + p$$

разделяме n -те събираеми на броя процеси + p операции за да получим крайния резултат. Това щеше да бъде формулата ако имаме следната топология - мрежа без broadcasting. Ако имаме топология звезда или пълен граф (всеки със всеки), тогава:

$T_n = N/p + p + 1$ (закъснение за комуникация, всички изпращат парциалните суми на един процес, който ги събира). Могат да се направят оптимизации - парциалните суми да не се събират последователно, а също да се разпаралелят и парциалните суми да се събират по двойки и т.н да се раздробят йерархично т.е вместо n ще имаме $\ln(n)$.

Избираме си стойности за N , например $N=1000, 10\ 000, 100\ 000$, и за $p = 2, 4, 8, 16$. Трябва разликата да е голяма. Изчисляваме S_n и можем да направим една координатна система за да нанесем резултатите.

Достатъчно е да напишем за най-простия вариант.

Сложността ни е T_n , паралелизмът $P = p$

методи: експериментални, еталонни и аналитични (статистически, теория на вероятностите)

1. Параметри на паралелна и разпределена обработка

1.1 Паралелни алгоритми

Представяват абстрактно описание на изчислителен проблем като набор от процеси за едновременно изпълнение.

1.2 Метрики и анализ на производителността на ПА

Когато разглеждаме сложността на един паралелен алгоритъм **не** можем да пренебрегнем архитектурата и средата на действие, както при последователните алгоритми. От особено значение са също степента на паралелизъм (**P=p**), т.е. максималният брой операции, които могат да се изпълнят паралелно при обработка на алгоритъма. Също така трябва да се има в предвид, че всеки ПА се състои от една част, която може да се изпълни паралелно и такава част, която трябва да се изпълни последователно, т.е. системната производителност не може да надвиши определена стойност, която зависи от частта за последователно изпълнение.

Нека:

- **p** е брой процесори, като се приема, че всички те са с еднаква производителност
- **T₁** е времето за последователно изпълнение на една операция
- **T_p** е времето за последователно изпълнение на една операция
- **O₁** е брой операции с 1 процесор (за последователна обработка)
- **O_p** е брой операции с **p** процесора

Тогава имаме:

- **ускорение (speed up):**

$$S_p = \frac{T_1}{T_p}$$

- **ефективност (efficiency) < 100%:** характеризира частта от времето, през която процесорните елементи се използват ефективно

$$E_p = \frac{S_p}{p}$$

- **цена (cost) на използване:** критерий за броя операции, които биха могли да се извършат за времето на обработка на ПА

$$C_p = p \cdot T_p$$

- **коэффициент на използване (utilization):** отношението на действителните към потенциалните операции при изпълнението на съответния ПА

$$U_p = \frac{O_p}{C_p}$$

- **излишък (redundancy):**

$$R_p = \frac{O_p}{O_1}$$

1.3 Методи за анализ - биват аналитични, еталонни и експериментални

2. Модели на разпределене софтуерни архитектури

2.1 Обектни - представяне на компонентите и връзките между тях; основни принципи на ООП, анализ и проектиране, декомпозиция на модули, видове диаграми, които се използват за представяне на обектно-ориентираната архитектура, изобразяване и използване на наследяване/композиция, агрегация и асоциация, предимства и недостатъци

2.2 Потокови - основният паралелизъм е по данни (data flow); обработката се представя като последователни действия, които се извършват върху еднотипни данни

- според механизма на свързване между модулите разграничаваме следните типове:

- пакетна обработка: този тип е неприложим за интерактивни проложения, но има широко приложение за асинхронни паралелни процесори; данните се декомпонират като множество входни файлове, а обработващите модули се репликират в множество възели

- канали и филтри: тук имаме последователност от FIFO потоци от входни данни (буфери, опашки), филтри, които трансформират потока данни и записват изходните данни във филтри, които пак извършват преноса към следващия филтър. Тази архитектура може да се представи с диаграма на последователността (или блокова). Пример за обработката на `who | wc -l` в UNIX: данните са последователност от потребители, а филтри могат да са произволни процеси в основен и фонов режим

- контролни процеси: прилагат се във вградени системи; имаме поток от входни променливи, които могат да са променливи на средата, и обработващи модули; пример - ABS система в автомобилите

2.3 Контекстни - характеризират се с централизирано хранилище на данните, до което всички компоненти на системата има достъп; казваме, че имаме данни и агенти, които изпълняват операции върху тях. В зависимост от това кой иницира комуникацията имаме два основни модела:

- хранилище (repository), при който инициативата е на агентите (например CORBA, UDDI хранилища); могат да се кажат предимства и недостатъци на този модел (отказоустойчивост, репликации, но и трудна и скъпа промяна на модела на данните)

- черна дърка с инициатива на модула за данни, при който агентите се абонират за промени, настъпили върху данните (publish/subscribe); свързването е слабо и не е необходимо заключване на данните за конкурентен достъп, както при хранилищата; източникът на данни може да бъде само един модул и другите да са абонирани за промени или да са съчетани двата принципа - хранилище и черна дърка

2.4 Йерархични - популярен архитектурен модел, реализиран в много ОС и протоколи (OSI model, TCP/IP). Основната идея е функционалността да се групира и

раздели на слоеве, които енкапсулират детайлите по реализацията си и предоставят интерфейси за комуникация на слоя на по-горно ниво. Приложими са при повишаване на нивото на абстракция - принципът на проектиране е отгоре-надолу. Твърде многото слоеве могат значително да намалят производителността на системата за сметка на по-абстрактния поглед над архитектурата; Пример - JVM; йерархичен модел Master/Slaves - вариант на йерархична архитектура с подпрограми, при която в контролния модул (master) се реализират техники за осигуряване на определени нефункционални изисквания (отказоустойчивост, надеждност) и се разпределят заявките към съответните подпрограми за обработка (slaves)

2.5 Асинхронни - асинхронна комуникация, която може да бъде в реално време (online), т.е процесите, които си взаимодействат трябва да са активни едновременно, или offline - поддържане на някакъв буфер за временно съхранение на информацията; може да имаме publish-subscribe обмен (1-n) или Message Queue за обмен 1-1; при publish-subscribe небуферираната комуникация трябва да имаме и модул за регистрация, които трябва да е синхронен, което е минус; при MOM имаме надежден обмен и скалируемост

2.6 Интерактивни - при приложения, които предоставят интерфейс, с който потребителите взаимодействат интензивно; за поддържане на множество изгледи на даден набор от данни се използват шаблони за проектиране като MVC и PAC (Presentation-Abstraction-Controller). Характерното е, че системата се разделя на функционални модули за данни, изглед и управление. Ще разгледаме по-подробно какви са основните принципи на MVC..

И да не забравиш да се усмихваш по-често :)))))) Няма :)))

3. Организация на разпределените приложения

3.1 Клиент сървър

Моделът клиент/сървър е в основата на днешните разпределени системи. Приложенията се моделират като съвкупност от услуги, предоставяни от сървъри и клиенти, които използват тези услуги. Клиентите трябва да знаят за сървърите, но обратното не е вярно.

Клиентът и сървърът могат да са на един и същ компютър или на различни компютри, свързани в мрежа. Мрежата прави възможна отдалечената клиент/сървър комуникация. Комуникацията може да бъде разпределена в три основни слоя: презентационен, логически и слой за управление на данни. В зависимост от това на колко слоя е разделена клиент-сървър комуникацията имаме 2-слойна, 3-слойна (най-често) и n-слойна архитектура. При 2-слойната архитектура имаме един сървър и множество клиенти, които изпращат заявки към него. В зависимост от това, дали обработката на данните се извършва при клиента или на сървъра имаме съответно thin-client модел и fat-client модел. При n-слойните архитектури се добавят междинни нива (middleware) и

всеки от слоевете на приложението се изпълнява на отделен процесор като по този начин се постига по-добра производителност, по-голяма сигурност, както и по-лесни за менажиране клиенти. В частност, при 3-слойната имаме 3 нива на достъп.

3.2 Peer-to-peer

Разпределена (децентрализирана) мрежова архитектура, съставена от участници, които отдават част от своите ресурси (като процесорна сила, дисково пространство, ширина на мрежовата връзка) директно на разположение на другите мрежови участници. Съществуват и полуцентрализирани p2p системи, при които има сървър за откриване на участници в мрежата.

3.3 Сървъри за приложения и web-servers

Основната им задача е да предоставят ефективно изпълнение на процедури (програми, скриптове), от които съответните приложения се възползват. Сървърите за приложения служат като съвкупност от компоненти/услуги, които са достъпни чрез специално дефинирано API. **Уеб сървърът** е приложна програма, която позволява на даден компютър да предоставя информация на други компютри, под формата на страници с хипертекст (HTML). За пренасянето на информацията се използва протокола HTTP. Често Уеб сървърът се използва за предоставяне на информация и в други формати — изображения, XML документи или други видове файлове, дефинирани съгласно MIME.

3.4 Метасистеми и грид

Грид-технологията е съвкупност от компютърни ресурси, разположени на различни места, за да се постигне общата цел. Броят им може да варира от няколко до хиляди. За разлика от супер-компютрите, при които се разчита на много процесори, свързани с високоскоростна шина, при грид изчисленията имаме отделни компютри, свързани в мрежа, всеки от които извършва част от задачата.

3.5 SOA

Основната идея е да се предоставят уеб-услуги, които представляват стандартен начин за преизползване и споделяне на някакъв компонент/функционалност в мрежата. SOA дефинира начина, по който две изчислителни единици, например програми, взаимодействат по такъв начин, че едната извършва единица работа от името на другата. Начинът на взаимодействие се дефинира посредством език за описание на услуги. Сервизно-ориентирани системи могат да съчетават в себе си услуги, предоставяни от различни доставчици.

3.6 АОА

Аспектно-ориентирана архитектура позволява моделиране на нефункционалните изисквания и качествени характеристики на дадена система, известни като аспекти. Тези модели не се занимават пряко с конкретна функционалност. Това са архитектурни съображения, които не могат да бъдат обособени в един модул, а трябва да се приложат върху няколко такива.

3.7 МОА

Основната идея е създаването на системи на базата на абстрактни модели и диаграми (например клас диаграми), при това чрез средства за автоматизирано генериране на код.

#nimoa