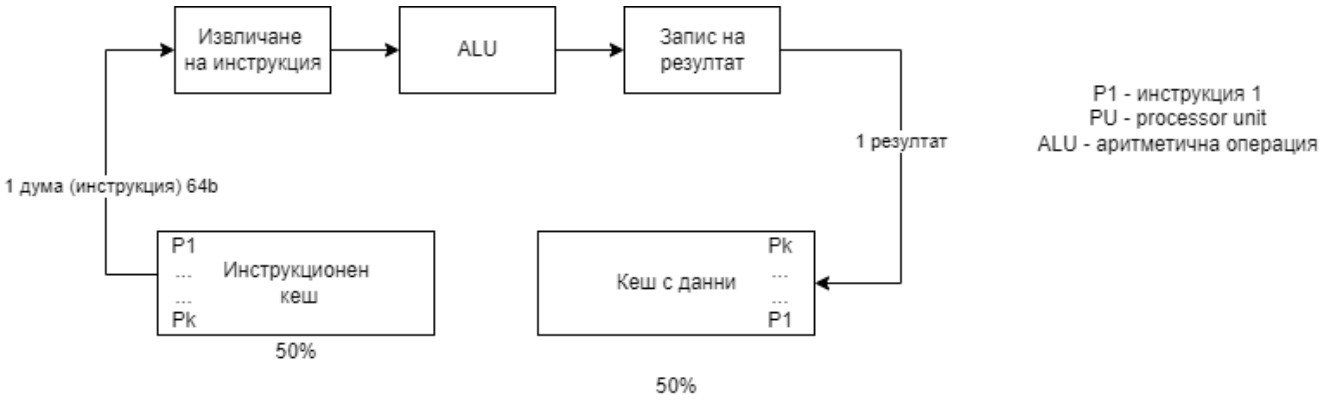
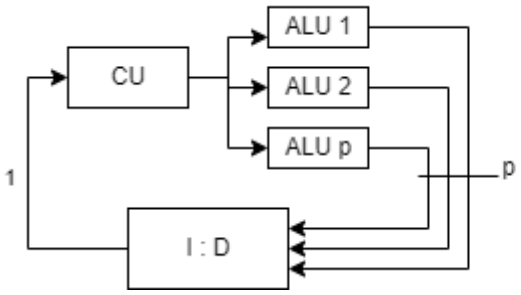
Тема 1. Класове машинни архитектури - мултикомпютри и мултипроцесори

Класове компютърни архитектури:
Архитектура - съдържа компоненти и организации на една система
Фон Нойманова архитектура - представя еднопроцесна програма, реализираща един алгоритъм
Програма - управляващ инструкционен поток на връзки, мрежи и некласическа организация - систолични, потокови, логически и редукционни модели
Класификация на Флин - класификация на паралелните архитектури на основата на управлението на потока инструкции и потока данни
Процесор - устройство, което изпълнява инструкция по инструкция, в реда в който ги е наредил компилатора
Инструкционен конвейер - състои се от 3 фази. Първата извлича инструкция от инструкционния кеш, където са подредени последователно. Втората обработва, ALU. Третата записва резултата. Темпът на работа на инструкционния конвейер се определя от най-бавната фаза, затова е препоръчително да работят с почти еднакво закъснение. Когато ядрото на ОС изпълнява инструкция, то я регистрира в собствената си област на кеша с данни. Ако се опита да я регистрира с друга област, се получава грешка по памет и процесът спира. Мегабайтите се делят на броя ядра, и оттам отново по равно
SISD - класическа фон нойманова обработка, в която няма паралелизъм. Броя фази при мобилните телефони е 4, докато в настолните и сървърните процесори стига до 14.

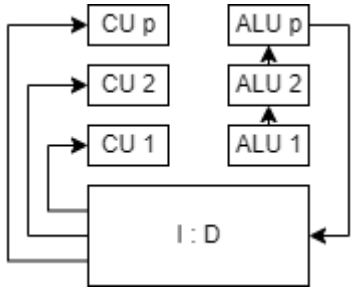


SIMD - достъпа до елементите е произволен. Имаме суперкомпютри - друго име на SIMD. Векторни машини - специализирани машини, които се занимават с единствено с векторна обработка - еднотипни елементи с еднаква дължина, фина грануларност. Паралелизмът е по данни (масивен) - матричен контекст и фон нойманова архитектура.

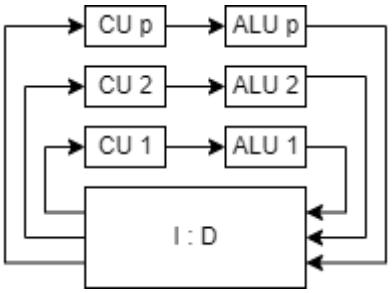


— всеки си чете в своя област на един и същи алгоритъм
— Моделът включва контролно устр. (CU) за дешифриране и зареждане на инстр,

MISD - кешът също е разделен на 2 части, но тук инструкционния конвейер е рефлектиран p на брой пъти. Има само 2 фази: извличане, която е свързана с p различни адреса на инструкционния кеш, всеки конвейер чете различна част, имаме и p на брой ALU. Първият ALU чете един пакет от данни от кеша с данни. Върху този пакет може да се изпълнят повече инструкции. След това минава на следващия пакет. Темпът се определя от най-бавната фаза. Машината е конвейер с паралелизъм по управление на p на брой равни фази на изчислимост. Предназначена е за поточни данни. Характерна е за обработката в реално време.



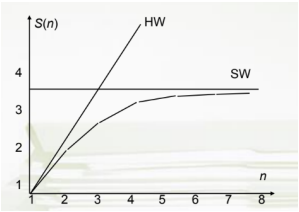
MIMD - мултипроцесори (с обща памет UMA) и мултикомпютри (без обща памет NUMA). Универсален клас. Всеки процесор чете самостоятелен поток инструкции и самостоятелно чете операнди и записва резултат. Известен е още като симетричен мултипроцесинг (Multicore). Мултикомпютър е набор от компютърни възли. Мрежово програмиране. Добавяме имена на нашите процеси, те се регистрират в локалния процес, а той ги изпраща до всички в мрежата.



Хардуерен и софтуерен паралелизъм - за паралелното изпълнение на програма са нужни едновременно апаратна и програмна поддръжка

- хардуерния паралелизъм се обуславя от завимостта по архитектура и ресурси, които са баланс между цената и производителността. Това задава зависимостта по ресурси. След като имаме хардуерен паралелизъм можем да напишем най-основните концепции на паралелизъм по данни и по управление - това е софтуерен паралелизъм.

Ускорение и ефективност - p - паралелизъм. За всеки обработващ процес се засича времето. Очаква се T2 да е 2 пъти по-бърз от T1 и тн. В идеалния случай, колкото е паралелизма толкова и ускорението, което се случва когато притежава свойството линейност. Имаме хардуерен лимит на ускорението. Колкото по-голям обхват до по-скалируема е нашата паралелна програма. Софт. лимит е независим за паралелизма (S(p) = p). Ускорение S(n) = T1/Tn, ефективност E(n) = S(n)/n



Делена на обработката: грануларност

- фина грануларност - делене на обработка на ниво инструкция - не работим с нея, на ниво компилатор е
- средна грануларност - на ниво на многозадачни системи
- едра грануларност - на ниво независима програма

Колкото по-фина е грануларността толкова по голям е паралелизма и комуникационния свръхтовар. Свръхтоварът идва от опита да се синхронизират процесите и да се балансира ускорението. Грануларността е основно средство при управлението. Не се измерва с число, а с качествена оценка - фина, средна и едра

Систолични матрици - специализирани машини, които имат комуникационни канали. Обработват 1 програма. Модификация на MISD

Разширена таксономия на Flyn-Johnson - в нея има споделена мултипроцесорна памет и разширена споделена мултикомпютърна памет. Те са допълнително разпределение на метода на обмена - общи променливи(при мултипроц.) и обмен на съобщения (при мултикомп.). Между тях има middleware, който осигурява общите променливи. Разпределена памет - набор от автономни компютри, които са самостоятелни възли и ресурсите им се делят на клиентски и сървърни. MIMD се разделя на GMSV и DMSV (при мултипроц.), GNMP и DNMP (при мултикомп.)

Потокови ядра - подменят последователния модел от класическите фон нойманови арх. по начин, който позволява паралелизъм на ниво инструкция. Операндите се използват веднага при наличие на операционен ресурс. Готови са тези инструкции, които имат начални ст-ти. Те могат да се изпълняват паралелно. От тях се генерират междинни или крайни резултати. Потоково управление - имаме винаги при функционалните езици, където има еднократно присвояване.

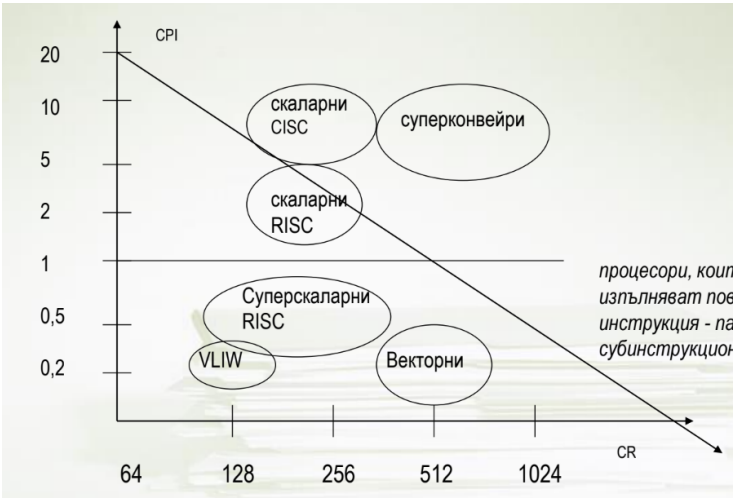
Тема 2. Процесорни архитектури - технологично пространство

Процесорни архитектури - mainframe - широк арх. клас от компютри. Суперскаларните CISC и RISC, процесорите VLIW, векторни проц арх. и суперконвейерни проц. арх.

Процесорни архитектури - основни характеристики са CISC и RISC,които са скаларни процесори. Всички процесори се нанасят в технологичното пространство, което се представя с диаграма в която има:

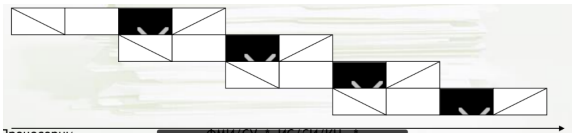
- тактова честота (CR), която зависи от технологията на изработка на процесора и логическата архитектура на конвейерите. Колкото по-висока е тактовата честота, толкова по-бързо конвейера ще обработва последователни инструкции.
- CPI - броя тактове за изпълнение на 1 инструкция. На всеки такт се изпълнява и завършва 1 инструкция. Времето за обработка не е 1 такт, а броя фази в инструкционния конвейер (обикновено 3), т.е 1 инструкция

седи толкова колкото е дълъг конвейера. RISC процесорите имат между 3 и 5 фази, независимо от честотата на работа. Скаларните CISC процесори са с от 6 до 8 фази. Няма конвейер с 1 фаза.



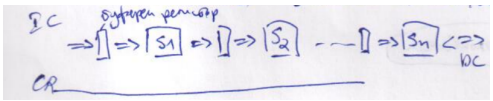
Фази на инструкционния конвейер - прието е да се разглеждат 4 фази: извличане, декодиране, изпълнение и запис. Декодирането установява фазата на изпълнение и необходимите ресурси - къса адреси в контрола за данни. Фазата изпълнение се състои от 10-15 ALU-та, извършващи различни операции. Накрая се генерира логическа операция.
Времедиаграма на инструкционния конвейер (карта на заетостта) - представлява карта на ресурсите на инструкционния конвейер - това са фазите. Закъснението между 2 последователни инструкции е една фаза при скаларните процесори. Ако няма зависимост между инструкциите се получава плътна карта на ресурсите. Иначе може да се получи конфликт - закъснение между фазите

Пълна заетост - при разделен кеш

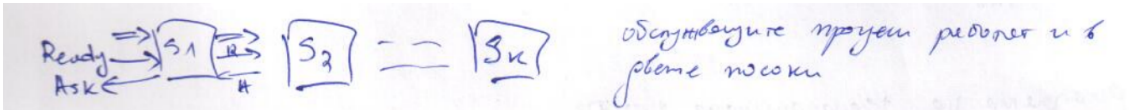

Закъснение между фазите

Синхронни линейни конвейерни процесори - случай, при който подаваме тактовата честота (clockrate), която управлява фазите и е настроен на темпа на работа на най-бавното устройство.

- синхронните ЛКП - служат за синхронизация на процесите. Фазата с най-голямо закъснение определя общия такт и производителност. Появява се фазово отместване на такта - команда за разпространение на сигнала.
- статични - изпълняват аритметични, обменни инструкции



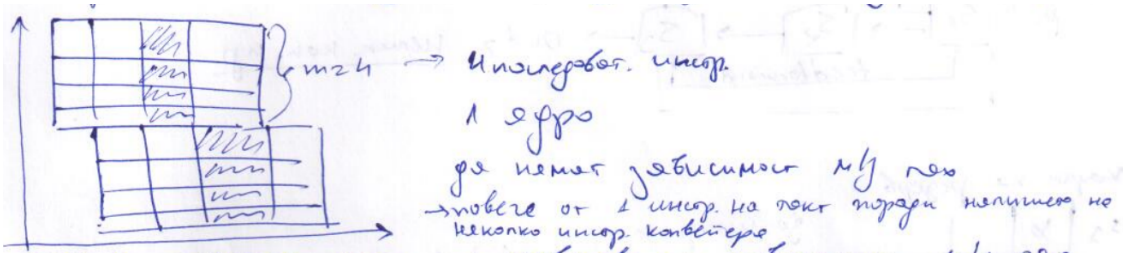
- асинхронният процесор работи с темпа на най-бавната фаза. Последователни процеси си предават частично обработени данни. Тази архитектура се използва във всички протоколни стекове - двупосочни конвейери.



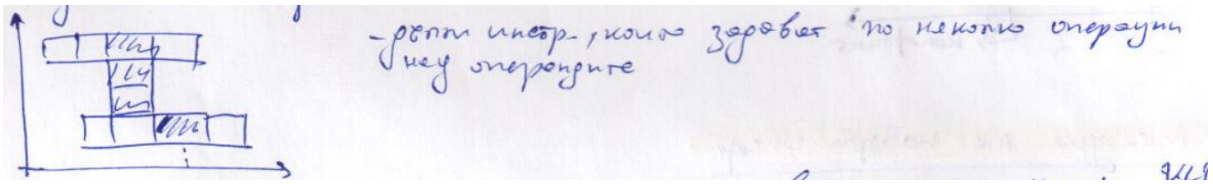
Нелинейни конвейерни процесори - всяка инструкция си следва фазите, но има такива, които приключват по-рано. Движението може да бъде различно: към предходни или прескачане на фаза. При CISC може да имаме фази с къси адреси, т.е. не трябва да минават през ALU. Има и операнди с плаваща запетая.
-> карти на резервацията - дава информация за инструкцията. За различните фази може да варира по управление и по време(тактове). Дава съвместимостта на последователни фази по управление

Архитектура на набора инструкции - класовете RISC и CISC се различават по параметрите: формат на инструкциите и данните, режими на адресация, регистърно адресиране, управление на изпълнението на програмата. RISC са къси инструкции, пригодени да изпълняват малък брой - 32 бита инструкция. Нямаме вариант за регистриране на сложни адреси и за отместване. Компилатора генерира 1 инструкция, но в нея се правят няколко операции, но се избират най-важните. 2-3 пъти повече инструкции се генерират за RISC в сравнение със CISC. От друга страна траят по-малко. CISC обикновено има 8 фази, но инструкциите са по-бавни.
Intel Pentium - въвеждат се суперскаларни процесори с ниво на инструкционен паралелизъм m = 2
Суперскаларни процесори(RISC и CISC) - за всяка нишка имаме data cache(50%) и instruction cache(50%).
Декомпозиция по данни:

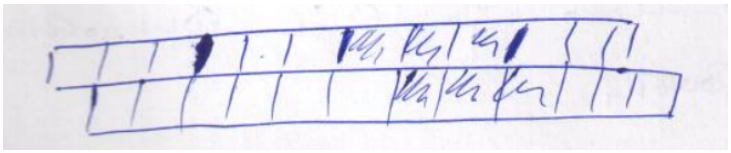
- модел: набор от параметри, които отразяват някакъв реален процес. Колкото повече са параметрите, толкова по-сложно е описанието на процеса.
- суперскаларност: вместо 1 конвейер да се използват 2 инструкционни конвейера, т.е прочитат се 2 поредни инструкции за изпълнение върху конвейера.



След като се подредят инструкциите се проверява за зависимости между тях и се дистанцират по-зависимите. Зависимите се изместват по-надолу.
VLIW процесори - сходни със суперскаларните по: инструкцията не е 1 процесорна дума, а са 2-3. Разлики: 1 VLIW инструкция замества няколко суперскаларни. Също така се прочита 1 дълга инструкция, но в нея се декодират няколко последователни инструкции.



Векторни процесори - могат да имат същата времедиаграма като на VLIW само че 1 стандартна инструкция се прочита, но се изпълнява върху няколко (двойки) аргументи: зареждане, запис на вектор от паметта, ескалиране, редукция.
Суперконвейерни архитектури - суперковейризация: имаме 1 изходен конвейер, който е имал 4 фази. С k означаваме броя фази (k = 4). Прилагайки коефициента на суперконвеирзация n = 3 -> новите фази стават 3*4=12



Тема 3: Паралелно програмиране. Паралелни и разпределени процеси. Паралелни алгоритми - принципи, проектиране, метрика. Среди и езици за паралелно програмиране. Синхронизация на паралелните процеси.

Паралелно = Конкурентно != Разпределено
Декомпозиция по данни -> за паралелно(мултипроц.)
Декомпозиция по управление -> разпределено(мултикомпютър)

Последователни и паралелни програми
Среда за последователно програмиране - когато програмата се състои от 1 процес, (фон Нойманова програма) и този процес се пусне сам да се изпълни на 1 машина. Няма едновременно стартирани. Изпълнението на всяка инструкция е последователно и независимо от изпълнението на други инструкции. Еднакви резултати. При среда с мултипрограмиране програмата отново се състои от 1 процес. Резултатите също за еднакви, но времето за изчислението им е различно. Между отделните процеси има зависимост.
Среда за паралелно програмиране - Програмата се състои от множество паралелни процеси. Тя включва освен управляващ код и данни, също и инструкции за синхронизиране и обмен между процесите, които представляват нейния планиращ процес. Резултата може да зависи от работата на планиращия процес.

Паралелни процеси
Процеси, изплъняващи програмата в средите за паралелна обработка. Могат да бъдат алтернативно:
Реплики - изпълняващи еднаква подпрограма върху различни данни. Модел SPMD - синхронизацията се извършва на ниво подпрограма, а не инструкция. Затова SPMD се изпълнява на MIMD компютри.
Различни подпрограми - модел MPMD: отделните подпрограми, процеси се пораждат като дъщери на един главен процес.

Граф на процесите - скаларни зависимости
Зависимостта по данни и управление се изследва на различни нива. Блок, израз, променлива, чрез графи.
Пример за серия изрази:
S1: A = B + C
S2: B = A + E
S3: A = A + B

Изразите се изобразяват като възли в графа на зависимостите, а другите са зависимостите, като началото на дъга е променлива(аргумент или стойност на израз), а край - същата променлива от следващ израз.

Типове скаларни зависимости
Зависимост по данни - резултата от израза е аргумент на следващ израз. Тази зависимост е непреодолима.
Антизависимост - аргументи на израз е резултат от следващ израз. Може да се преодолее чрез репликиране на променливите.
Зависимост по изход - резултатите от два израза се записват в една и съща променлива, преодолимо.
Зависимост по вход - два израза имат общ аргумент. Няма значение при съвременни програмни системи(поради средства за конкурентен достъп)
Зависимост по управление - условно изпълнение на израз, където условието е резултат от предходен израз.
*За по-висок паралел без 2 и 3.
Типове едномерни векторни зависимости
-Независими SPMD: for i= 0, 499 a[i] = a[i] + C
-Зависими SPMD: for i= 0, 499 a[i] = a[mod(i-1|500)] + C
-Антизависими SPMD: for i= 0, 499 a[i] = a[mod(i+1|500)] + C

Модели обща памет
В паралелните системи достъпът до общата памет и ресурси е конкурентен и се базира на схемите за PRAM.
В модела PRAM се прилагат и схеми за отстраняване на конфликтен конкурентен достъп до общото адресно пространство.
-EREW - резервиране на конкурентния достъп до даден адрес за двата типа операции
-CREW - няколко процесора могат да четат едновременно даден адрес
-ERCW - допускат се няколко едновременни операции на запис, на монополно четене
-CRCW - конкурентните операции са без ограничение

Модел с обмен на съобщение
Всяка двойка процеси е свързана с комуникационен канал, представен в точно 1 променлива, последователните съобщения са стойностите на тези променливи. Дефинирано е състояние на канала. Асинхронният и синхронният канал са с еднакъв режим на достъп.

Паралелни алгоритми - те са междинно звено във веригата на паралелната обработка(между изчислителния проблем и паралелната система)

Паралелният алгоритъ е абстрактно представяне на изчислителен проблем като набор от процеси за едновременно изпълнение.
Основни характеристики

-Брой процеси и логическата им организация
-Разпределение на данните
-Точки на синхронизация(оптимизиране)
-Модел на междупроцесорния обмен

Фази на проектирането на паралелния алгоритъм
Разделяне - декомпозиция на проблема
По данни(главно SPMD)
По функции(главно MPMD)
Целта е да се дефинират множество подзадание. Резултатът от фазата е дефиниране на отделните задания.
Комуникации - формулира информационните или контролните зависимости между отделните подзадания.
Комуникациите се представят като канали и съобщения. Оценка на паралелни алгоритми по комуникационна сложност.
Формиране - след оценка на изчислителната и комуникационната сложност на формулираните подзадания и прилежащите им комуникации, те се групират в задания, при което се отчитат характеристиките на архитектурата на обработка - брой процесори/възли и комуникационен модел
В резултат се постига оптимизиране по следните характеристики:
Грануларност и балансираност
Евентуално репликиране на данни и подзадания
Оптимизиране на комуникациите
Евентуално запазване на линейност(скалируемост)
Технологично оптимизиране

Разпределение(незадължителна фаза) - състои се в разпределение на формираните задания по обработващите възли на системата с кодиране на съответното решение.

Метрика и анализ на производителността
Сложността на последователните алгоритми се оценява като функция само на размера на проблемната област и следователно може да се оцени абстрактно от архитектурата. При паралелните алгоритми тя е функция на архитектурата и на средата за паралелната обработка. Основен фактор при паралелните алгоритми е степента на паралелизъм P -максималният брой операции, които могат да се изпълняват при обработката на алгоритъм. При размер на проблема W не повече от P(W) процесора могат да се използват ефективно.

Закон на AMDAHL(1967) - при наличие на 2 интензивности(темпове) на обработка на даден проблем - високо паралелна Rh и ниско паралелна Rl, които са в съотношение f:(1-f) по брой на генерирани резултати(междинни и крайни) - общата интензивност на обработката е:
R(f) = [f(Rh + (1-f)/Rl]-1
за f->1: R(f) -> Rh и при f->0 : R(f) -> Rl

Цена по коефициент на използване
Цена(cost) при обработка на паралелни алг. с p процесора за Tp единици време(единица време е времето за изпълнение на 1 елементарна операция) е Cp = pTp. Cp е критерий за максимален брой операции, които биха могли да се извършат за времето на обработка на съответния пар. алг.
Коефициент на използване - състоящ се от Op на брой операции и p процесора е Up = Op/Cp = Op/(pTp)
Up е отношението на действителните към потенциалните операции при обработка

Еталонни паралелни алгоритми - SPMD
Асинхронни SPMD - тест на Манделброт и Фрактални(себеподобни) изчисления. В двумерното пространство за всяка точка се извършва r0 = C - const.
Върху него => Zn+1=Zn2+C, за Z0=C …
….
Въртим така докато разликата между две z влезе в един интервал(дистанция) или стигнем края на теста, който обикновено стига до 256 итерации. Между 1 и 256 итерации за всяка точка. Разликата между 2 итерации се кодира с един цвят от 256-цветна палитра. Обикновено с черно оцветяваме, когато за 256 итерации не сме стигнали края, с бял цвят за 1 итерация.
Предимство:
Един израз в цикъл(while)
Нямаме начален контекст(входна матрица)

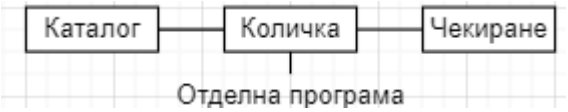
С по-фина грануларност и разпределена статично правим статично балансиране на товара.
Локално-синхронни SPMD
Симулацията Wator(игри раждане-смърт, генетични алгоритми).
Симулацията на обектите, при което има релация на съседство между тях. Те могат да се местят в 4те посоки, освен ако няма пречка в някоя посока(фигура). Обектите имат средно време на живот, може и средно време на

размножаване.(Маркиране на някои от съседите). Допълнително имаме и средно време на плавна смърт. В една дума от 64b можем да кодираме 32 позиции. Ако имаме 4 процеса, първият ще има съседи 2рия и 4ия. Те могат да се обменят на всяка стъпка. Всички процеси вървят заедно.
Сортировка - сортиращи мрежи - особен вариант на насочен граф. Базира се на компаратор. И възли с 2 входа, които могат да се заемат от 2 елемента на несортирани последователн и на изход вече са сортирани.
С декомозиция по данни за P процеса- сортировките са независими, могат да се извършват наведнъж
Odd-even - позволява паралелна обработка
мехурче - паралелно : 2n -3 стъпки
последователност = (n-1)* (n/2) стъпки
Степенен ред e^x = SUM((x^k)/k), x принадлежи -+безкр
Трябва да представим числото е като ред.
Трябва ни факториела от 0 до 10 000, който да поделим на p процеса. Всеки процес умножава своите частични факториели. Всеки следващ процес чака предишните.
Хистограмата ни трябва, когато правим стохастичен процес. Ускоренията не се представят с нея.
Глобално-синхронни SPMD - имаме и тела в двумерно пространство. Гравитация в равнина. Всяко тяло има начален импулс(произведение от вектора - скорост и маса и начална точка). Ако върху едно тяло не действа сила, то запазва своето равномерно разположение, запазва траекторията си, Иначе телата си влияят. Ако искаме да сметнем някой момент трябва да знаем координатите им в предходен момент и каква е масата им.

Multithreading - едновременно обработване на 2 преоцеса от 1 суперскаларно ядро, посредством 2 логически ядра.

Тема 4. Модели на разпределена софтуерна архитектура с UML

Модели на софтуерна архитектура - Софтуерната архитектура представя програмния проект като съставен от софтуерни компоненти. Моделирането на PCA е първата и най-важна фаза на проектиране, настройка, тестване, разгръщане и документация на разпределени среди за обслужване. Моделът описва декомпозицията на процеса на компоненти, функционалната им композиция, прилагане на арх. стил, качествени атрибути на услугите (QoS).
Представяне на софтуерните модели - използват се графи и техни разширения. Описанието е чрез диаграми. За по-пълно функционално и нефункционално описание на проекта се прилагат многомерни модели, като 4+1 модели: логически изглед, изглед процеси, изглед проектиране, физически изглед, потребителски интерфейсни изгледи.
Блокова диаграма - показва как декомпозираме приложението си. Представлява граф на зависимостите.



UML модели на софтуерните архитектури - За ОО спецификация са 2 групи диаграми:
- структурни - статично описание на елементите в системата с помощта на йерархични класове и статични връзки между тях - наследяване, асоциация, агрегация, обмен (class, object, composite, component, package, deployment)
- функционални диаграми - динамично описание - функциите на инстанциите на обект (use case, activity, state machine, sequence, communication)

Клас диаграми - най-разпространено описание при всеки модел. Класовете се описват с техните атрибути, тип, интерфейс, методи, свойства. Описват се е отношенията между класовете
Обектни диаграми - извличат се от клас диаграмите. Описват обектите като инстанции на класове
Composite диаграми - описват връзката между обектите. Обектите и връзката се означават с етикети.
Компонентни диаграми - компонентите са изпълними модули за преизползване при проектиране. В UML те са със скрита структура. Представя съответствието между изискваните и имплементираните интерфейси. Компонентите може да са готови и специфични.
Пакетни диаграми - физическо описание на проекта, описваме папките в които са структурирани диаграмите ни
Диаграма на разгръщането - представя възлите, които ще изпълняват нашите декомпозирани модули
Use case - описва потребителските сценарии на приложение на системата като граф от актьори, потребителски случаи и връзки между тях. Колкото са актьорите, толкова са ролите.
Диаграма на дейностите - дейностите са представени в тяхната последователност и са дадени зависимостите между две дейности. Високо ниво на представяне на алгоритъм
Диаграми на машина на състоянието - представят състоянието на обслужващите устройства или софтуерни модули в проект. Набор от състоянията и преходите между тях.
Диаграма за преглед на взаимодействието - взаимодействието като последователност от нашите диаграми маркирани с указател - sd, cd, ad
Диаграма на последователността - описват относителната последователност от процеси във времето. Показва недефинирано закъснение. Може да се върне или да не се върне резултат
Времеви диаграми - описва графика на състоянията от машината на състоянията
Развоен, процесен и физически изглед -
- развоен - описва потребителските фази на приложението, както и основните нефункционални изисквания с use case диаграма

- процесен - описва декомпозицията на разпределеното приложение с оглед на реализираните фази с клас диаграма + последователност или дейности
- физически - описва цялата PCA + приложението - инсталация, конфигурация, разгръщане

Тема 5. Модели на разпределена софтуерна архитектура - обектни, потокови и контекстни

Обектно ориентирани архитектури

ОО - принципи: прилагат се 3 основни принципа:

- капсулиране: видимост на функциите и прозрачност да имплементацията

- последователност: йерархии от класове

- полиморфизъм

ОО - софтуерна архитектура - абстрактни типове данни, класовете са техни имплементации с публичен интерфейс от атрибути и операции. Обектите са имплементации на класове

- статични отношение между класовете: композиция, наследствие, операции, асоциации

- динамични отношения: обмен на съобщения

ОО - проектиране - целта му е да декомпозира системата на технологични модули-класове. Високо и ниско ниво на проектиране

- високо - идентифицира класовете, напр. с приложение за CRC - карти и клас - диаграми за статичните отношения между класовете

- ниско - детайлизира проектираните класове и тяхното динамично взаимодействие

  - стъпка 1: CRC (Class - Responsibility - Collaboration) карти с описание на името, функцията, задължение и колаборащи класове за тях.

  - стъпка 2: Взаимодействието между обектите от стъпка 1. Диаграми и послед. и комуникация.

  - стъпка 3: Описва динамичното поведение на по-сложните класове за целия им цикъл на живот с диаграми на състоянието.

  - стъпка 4: Подробно описание на интерфейсите на всеки клас.

Потокови архитектури (Data Flow)

Представляват обработката като последователностс от трансформации в/у последователност от набори структурирани еднотипни данни.

Системата се декомпозира на функционални модули или подсистеми.

Основният паралелизъм е по данни, защото ритъмът на обработка се задава от наличието на данни за обработка.

Декомпозиция по управление

Декомпозиция на последователни пакети

- Категории потокови архитектури - топологията на пренос на данните между модулите се задава с блок-диаграми. Обработката е асинхронна. Можем да разглеждаме блок-диаграмите като последователност от фазите при конвейери. Имаме средно време на закъснение.

! По механизма за свързване между модулите се разграничават:

- пакетна обработка }

- филтрирани канали } Всички представляват 1 конвейер

- контролни процеси }

- Пакетна обработка - най-старият модел на СА за обслужване със стандартен файлов IO (input output) и редуктори. Данните се записват на изходящ и входящ файл. Приложението е скрипт с команди за изпълнение на съответните модули в UNIX, DOS, TCI/Тк.

Данните се представят като програми, които се активират със скрипт или като резидентни модули, които сканират входните си файлове. Неприложима СА за интерактивен интерфейс. Широко приложение за асинхронни паралелни процеси. Системи Condor, Boine, SETI@home. Има отлагане във времето (отложен режим offline), когато сървърът се освободи, довършва обработката.

- филтрирани канали (online интерактивен режим) - Приложението се декомпозиция на източник на данните, филтри, канали и консуматор на данните. Данните са последователни FIFO потоци от байтове, символи или запаси. Филтрите трансформират потока данни без необходимост да изчакват готовност на целия пакет. Записват изходните данни в пател, който ги предава на друг асинхронно - работещ филтър

(Активен филтър - pull и push върху пасивни канали)

(Пасивен - предоставя push/pull интерфейси на каналите)

Каналите преместват потока данни, които се обменят м/у два филтъра

- Контролни СА в реално време - прилагат се вградените системи (BAC). Дефиниран е срок, в който трябва да се мине през съответната фаза.

Процесът се разделя на няколко модула от 2 типа:

- контролни: за следене и манипулиране на променливите на средата и състоянието

- изпълнителски: за управление на архитектурата

- връзките между модулите - чрез поточни данни

! По какво се различават: по скоростта (offline, online, real time)

Типове контролни потоци при КСА

- контролни променливи: измерват се текущо от сензорите и се съпоставят с целеви стойности

- входни променливи според проблемите област (скорост, налягане, температура, влажност, GPS координати)

Контекстни архитектури (Data Centric)

СУБД-та с техните начини за съхранение на данни, каквито са релационните бази данни. Характеризират се с централизирано хранилище на данните, които са достъпни за всички компоненти на системата, така че декомпозиция е на модул за управление на достъпа до данните и агенти, които извършват операции върху тях.

В чист вид не се предвиждат преки комуникации между информ. агенти.

! Модулът данни изпълнява операции по извличане или регистриране и промяна на записа по два възможни модела:

- хранилище с активни агенти (обикновено СУДБ, CORBA). Агентите са равнопоставени (еднаква логика на транзакциите, peer-to-peer)

- черна дъска - агентите са абонати за събития (event listener), които настъпват при промяна в данните и на които абонатите отговарят реактивно

Заявките на транзакции (message - passing), защото е разпределена IT архитектура. Заявките са нужни за достъп до данните.

Транзакциите (пакет или група операции) в себе си пакетират до 2-3-4 операции върху данните, до които СУБД-то им дава достъп - четене и запис.

Заявките постъпват конкурентно и могат да бъдат много. Могат да дойдат 10 транзакции наведнъж, но ние трябва да ги обслужим сякаш са последователни. Чрез времева мярка можем да ги подредим по време на пристигане. По-лесно е да я сложим на сървъра. Използваме времевата мярка като аргумент на нашата обработка.

- контекстни архитектури с хранилище - за разлика от потоковите архитектури за пакетно обслужване на транзакции тези архитектури поддържат интерактивните интерфейси

Релационните СУБД са обичайната платформа за имплементация на тези архитектури, тъй като поддържат свързаност (консистентност) на разпределение на достъпа до данните, както и множество системни средства за операции, базирани на метаданни

- разпределено хранилище - за по-висока отказоустойчивост и защита на данните. Основен недостатък е статичната структура на данните

- контекстни архитектури с черна дъска - Ориентирани са към проблеми, режими, свързани с методите по AI и мултидисциплинарни обработки. Декомпозират решаването на проблеми на 2+ дяла:

- черна дъска, съхраняваща данни - факти, хипотези

- източници на знание - паралелно работещи агенти, които съхраняват различни данни (знание) от проблемната област

- контролер - система за начално зареждане и управление на разпределеното приложение

Запазва се блок-диаграмата на класовете, но контролираният поток е самостоятелността от ЧД към ИЗ (?). Обръщенията възникват при промени в данните и се предават към абонираните за тези промени ИЗ, които изпълняват логическите правила за избор в тях. Този асиметричен механизъм за обмен е известен като модел publish

- диаграми на контекстни архитектури с черна дъска

- класове-източници - агенти. Съхраняват специфичните правила за логическите изводи. Регистрират се в съответната ЧД. Абонират се за оповестяването на промени в данните на ЧД и евентуално генериране на реакции с изменение в локалния си или общ контекст.

- контролер - инициира ЧД, множеството на ИЗ. Инспектира състоянието им и публикува крайното решение.

Активният агент е сървърът, т.е. черната дъска, т.е. СУБД-то. Имаме начални данни в процеса. Върху този контекст се дефинират теми.

Критични зони с взаимно изключване

В унипроцесорите те се управляват от механизмите на ключалки семафори и монитори. В РС тези подходи се имплементират от централизирани алгоритми за управление на достъпа.

Взаимно изключване

- централизирано взаимно изключване - базира се на излъчен координатор, към който се отправят заявките за достъп до критичната зона. Заявките се потвърждават по реда на постъпване. Процесите с непотвърдени заявки изчакват. След освобождаване на критичната зона, чакащият (блокиран) заявител получава потвърждение (и достъп).

- разпределено взаимно изключване - Базира се на тотално подреждане на събитията с надеждни (потвърдени) групови комуникации. Заявителят изпраща съобщение с името на критичните зони, свое id и лок. вр. Всеки получател връща ОК съобщение, ако не е, или не чака достъп в тази критична зона. Ако е в критична зона, не отговаря, ако е изпратил собствена заявка за същата критична зона, сравнява

двете времеви мярки и ако има по-късна мярка, изпраща ОК на заявителя, или не отговаря. Заявителят изчаква ОК от всички останали процеси и заема критичната зона.

- Резервирано взаимно изключване (Token Ring) - Базира се на логаритмично подреждане на процесите в пръстен. Стартиращият процес освобождава съобщението token. Служебното съобщение се предава последователно между процесите, давайки право на достъп на текущия процес до критичната зона, след излизане от което съобщението-token се предава към следващия процес в пръстена.

Разпределени транзакции

- транзакциите са механизъм за синхронизация на съвместната работа на устройствата в системата. Или се изпълняват докрай, или процесите се връщат в състоянието преди началото на транзакциите.

- свойства на транзакциите (ACID) блокови:

  - атомарност (прозрачност)

  - логичност (consistent): съхраняване на системен контекст

  - изолираност: конкретна транзакция се изпълнява последователно

  - устойчивост (durable): не се променя след изпълнение

ACID (блокови) - не допускат съхраняване и достъп до междинни резултати

- обхват на КАро е черна дъска - подходяща за комплексни неизследвани особено мултидисциплинарни проблеми, които са без детерминистично решение. Може да се генерират оптимално или няколко субоптимални решения. За разпределена обработка с умерена скалируемост поради централизирания контекст.

! Гопок - диаграма на КАЧО на система за туристически консултации

Контролният процес има потребителски интерфейс. Черна дъска е представена от СУБД. Агентите са разделени тематично. В черната дъска се съхраняват различни данни/факти от заявките за обслужване. Клиентът има интерфейс към контролния процес. Изпраща заявка, в която се съдържа различни процеси към агентите. Тази заявка се разделя на тематични заявки към съответните агенти.

Обменът означава message-topic. Тук всеки процес е равнопоставен - няма дефиниран интуитивен реактивен процес. Няма клиент, няма сървър. Всеки процес се явява абонат на поне една тема. Всеки процес, който е абонат на една тема има право да изпраща съобщение и да получава всички съобщения, публикувани по тази тема.

- Клас диаграма на компютърната архитектура с черна дъска

За основни типа (и на брой) процеси:

- черна дъска (СУБД)

- агенти (Knowledge …)

- Control (контролер и процес)

Тази архитектура не е непременно ориентирана към ИТ обслужване (за разлика от хранилище). Тук обикновено се обработват мултидисциплинарни обработки. Най-напред се зареждат начални данни в този процес (може и да не е СУБД - при транзактивно обслужване). Върху този контекст се дефинират определен брой теми (topics). KSi процесите се явяват абонати при определени теми. Те са специализирани в своята логика (в своите процеси) в своя алгоритъм. След началното зареждане черната дъска изпраща съобщение за началното състояние на данните, но тематично. След като извършват (KS) обработката на база някакво съобщение, те връщат резултати, изразяващи се в заявки за промяна на съществуващото съдържание в черната дъска. Тези заявки за съдържанието водят до промяна в началното състояние, което води до нова вълна от съобщения пък към KSi за това какви промени в съдържанието са настъпили, които касаят други операции. И това продължава, докато се достигне предварителното условие за край, чиято цел е да се достигне оптималност или суботпималност. За да е оптимално трябва всички критерии да са спазени. Условието за край обикновено се дава като функция на контролния процес, която проверява дали е достигнато многомерното условие за край. Локално, в отделните KSi, няма решение на това дали сме стигнали условието за край, защото те не

са тематично разделени. Контролът обикновено се изпълнява на host компютър, който следи обслужването и получените съобщения за междинните цели и крайни резултати.

Модели PCA - йерархични, синхронни интерактивни

Йерархични архитектури

Декомпозират системата по управление на йерархични модели, тоест функциите се групират по йерархичен принцип на няколко нива

Координацията е между модули от различни нива(вертикална свързаност) и се базира на явни обръщения(заявка-отговор). Ниските нива функционират като услуги към непосредствените по-високи нива. Услугите са имплементирани като функции и процедури или пакети от класове.

Архитектурен модел на много ОС:

Базови услуги - модули за IO, транзакции

Междинен слой - ядро, поддръжка на логиката

Потребителски интерфейсен слой - екран

Йерархия с подпрограми - базира се на процедури със споделен достъп до данните. Декомпозицията е по управление като комплексната функционалност на приложението се разделя на по-малки функционални групи - процедури и подпрограми - с цел тяхното споделяне между различни изискващи ги модули.

Подпрограмите формират ациклична слоеста йерархия. Слабостта и е, че услугите на някое ниво трябва да са разгърнати, защото основната услуга е резидентна. Всичките слоеве не изпълняват логиката, а само транслират.

Диаграми на MSub архитектура - потокова диаграма се използва за начално моделиране на изискванията към системата. От нея се извличат:

Блокова диаграма на арх., съставена от контролни и диспечерски модули(подпрограми), съответстващи на трансформиращите и транзактивните възли на потоковата диаграма

Master/Slaves - това е вариант на архитектура с подпрограми, който е специализиран към поддържане на:

Отказоустойчивост и надеждност

Динамично балансиране на ускорено изпълнение на заявките

Декомпозицията е по данни на толкова части, колкото са slaves. Но това означава, че работим на най-едрата възможна грануларност, което предизвиква дисбаланс, когато някои процеси завършват преди други, което води до падане на нашия паралелизъм.

Диаграма на MC арх.

Приложими са при ОО имплементация

Проблем: достъпа до глобаните данни(модел на обща памет)

Слоести архитектури - групиране по различните нива в йерархията във функционално свързани слоеве от пакети класове, библиотеки от подпрограми.

Интерфейсът на слоя се състои от интерфейсите на включените в него компоненти, и изпълняваната от тях функционалност е протокола на слоя.

Обработката се декомпозира на заявки от по-висок слой към непосредствения по-нисък. Типично разслояване: потребителски интерфейс

Клас-диаграма на слоеста арх.

Компонентно-базирано разслояване - основен подход за капсулирането на услугите в слой е формирането на компонент, който се описва със своя интерфейс - jar

Компонентите на отделните слоеве формират пакети на платформата.

Всеки клас от компонента е достъпен за приложението чрез своя интерфейс.

Виртуални машини - слоест модел, който предоставя високо ниво на абстракция - скрива изчислителната платформа.

Нарича се машина, защото извършва услуги, които ги прави едно ядро на ОС, което за да изпълнява своите услуги трябва да се стартира.

Ако 1 ядро го натоварим да изпълнява друго ядро на друга ОС, се получава виртуална машина.

Предусловия за използване на виртуална машина - да са от един тип ВМ на Windows ще работи, ако отдолу има ВМ на Windows.

Емулация - изпълнение на функциите на дадена система от друга система(с принципно различни функции или организация). Емулация на UNIX върху MSDOS/Windows.

!Когато има пълно покритие казваме ВМ, иначе е емулация.

Виртуалните машини са се появили с клауд услугите в последното десетилетие

За всяко ВМ заделяме област от паметта на устройството: пясъчник - резервно работно пространство.

Когато използваме виртуални машини, данните са защитени и само ние имаме достъп до тях. Не може да се адресират чужди данни.

Асинхронни архитектури - Базират се на неявни асинхр. обръщения между обслужващите процеси. Асинхронният обмен може да бъде:

Свързан(online) - без буфериране - и двата процеса трябва да са активни, но не се блокират изчакващо в точката на обмен.

Независим(offline) - с опосредяващ обмена процес - буфер на съобщението

Приемащият процес може да не е активен в момента на изпращане на съобщението и обрат на процесът - буфер може да служи като

Централизатор(MessageTopic) на всички съобщения и да ги препраща тематично до абонатите 1 към много обмен

Резервирана опашка(MessageQueue) 1 към 1 обмен

*Асинхр. арх. още наричани Message-oriented middleware(MOM).

Имаме 3 нива на обмен:

Message-passing(MP) - работи по следния начин: Тръгваме от процес и програмираме го да работи с този MOM, казвайки му кой процес е и, че тръгва, при което в регистъра на процеси се казва, че стартира процеса с това име и той е на еди кой си IP адрес и порт.

Тази информация се записва локално и на машината, на която се разпространява това съобщение. След като не регистрира, той може изпраща съобщ. на различни процеси, Така работи MP(online)

Message-querying- пощенска кутия, която пази съобщ. при offline обмена за определено време

Message-topic - регистриране на процеси и абонати по тези теми, които да получат съобщението(тематичен обмен)

=> от тук не е задължително

Небуферирани асинхронни СА : системата се декомпозира на 2 части:

Генератори на събитие, служители на събитие и регистратори на събитие. Сравнително ниска производителност и голям системен свръхтовар.

Буферирани асинхронни СА: системата е контекстна, слабо свързана.

Декомпозира се на 3 части: генератори на съобщ., консуматори на съобщ., услуги за асинхронен буферен обмен на съобщ. - MOM

Висока скалируемост, надеждност, p2p приложение, поддържа опашка и тематичен обмен.

Обхват на асинхрон. СА: подходящо за слабо свързани системи с устойчив неявен обмен на съобщения., при които обменящите процеси са анонимни. Висока скалируемост.

Подходящи за пакетна обработка и за интегриране на процесни приложения в съвременни проекти.

Интерактивни софтуерни архитектури

Поддържат интензивен потребителските интерфейс. При тях винаги доминира MVC модел. Архитектурно разделяме потребителски интерфейс спрямо данните на 2 дяла:

Моделът е страната данни, VIEW е потребителски интерфейс, контролерът е модул за управление time-drive, замества(подсеща клиентът. Вместо клиентът сам да refresh-ва контролерът го прави вместо него. И ако е настъпила промяна се променя и view-то. Не е event-driven. Не е свързан с данните. Но се генерира свръхтовар при постъпване на много заявки. MVC е основен модел за сървърни приложения с Web клиенти за достъп. Динамично се представят промените в данните, тоест в реално време при отдалечени клиенти.

Контролерът и изгледът са 1 процес, регистриран в един модул C/V. Приложим е за по-прости приложения с компоненти GUI.

MVC II - контролерът и изгледът са самостоятелни. Допълнителни функции на контролерът е да се инициализира между изгледа и модела и управлява обмена между тях.

PAC(Presentation abstraction control) развитие на MVC, което поддържа агентен обмен на съобщения. Системата се състои от множество специализирани агенти, декомпозирани на трите модула P, A, C

Тука няма централизиран достъп до данните, а всеки агент може да съдържа данни. Презентационният модул на агенти е опция.

Контролният модул е задължителен, той управлява достъпа до функциите на агента.

Абстрактният модул капсулира данните и операциите на агента.

Обхват на PAC - прилагат се за интерактивни системи от кооперациращи специализирани информационни агенти.

Слабосвързана разпределена система - комуникациите са неблокиращи асинхронни.

Тема 6. Модели PCA - йерархични, асинхронни и интерактивни

Йерархични архитектури

- декомпозират системата по управление на йерархични модули, т.е. функциите (ф-ите) се групират по йерархичен принцип на няколко нива

  Координацията е между модули от различни нива (вертикална свързаност) и се базира на явни обръщения (заявко-отговор). Ниските нива функционират като услуги към непосредствените по-високи нива. Услугите са имплементирани като фигури и процедури или пакети от класове.

  Архитектурен модел на много ОС (?):

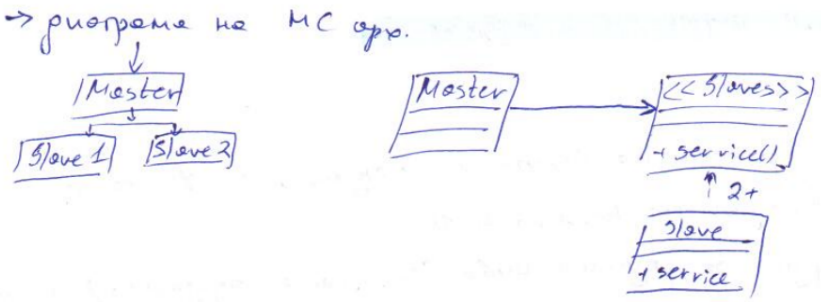
  - базови услуги - модули за IO, транзакции
  - междинен слой - ядро, поддръжка на логиката
  - потребителски интерфейси слой-екран
- йерархия с подпрограми (main/subroutine) - базира се на процедури със споделен достъп до данните. Декомпозицията е по управление, като комплексните функционалности на приложението се разделят на по-малки функционални групи - процедури и подпрограми - с цел тяхното споделяне между различни извикващи ги модули.

  Подпрограмите формулират ациклични слоеста йерархия. Слабостта ѝ е, че услугите на някое ниво трябва да са разгърнати, защото основната услуга е резидентна. Всичките слоеве не изпълняват логиката (?), а само транслират.

- диаграми на MSub-архитектура
  - Потокова диаграма се използва за начално моделиране на изискването към системата. От нея се извличат
  - блокова диаграма на архитектурата - съставена от контролни и диспечерски модули (подпрограми), съответстващи на трансформациите и транзактивните възли за потоковата диаграма
- Master/Slaves (усложнена SPMD (още 1 процес)) - това е вариант на архитектурата с подпрограми, който е специализиран към поддържане на:
  - отказоустойчивост и надеждност
  - динамично балансиране за ускорено изпълнение на заявките

  Декомпозицията е по данни на толкова части, колкото по slaves. Но това означава, че работим на най-едрата възможна грануларност, което предизвиква дисбаланс, когато някой процеси завършват преди други, което води до падане на нашия паралелизъм.

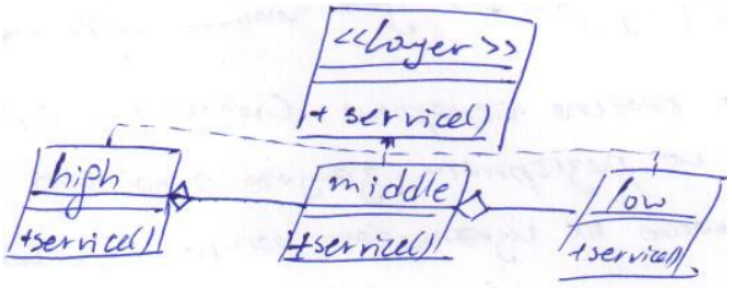
  - диаграма на MC архитектура



Приложими са ОО имплементации

Проблем: достъпът до глобалните данни (модел на обща памет)

- слоести архитектури - Групиране на различните нива в йерархията във функционално свързани слоеве от пакети класове, библиотеки от подпрограми. Интерфейсът на слоя се състои от интерфейсите на включените в него компоненти, и изпълняваната от тях функционалност е протоколът на слоя. Обработката се декомпозира на заявки от по-висок слой към непосредствения по-нисък. Типично разслояване: потребителски интерфейс <-> бизнес слой <-> базови условия <-> условия на ядрото
    - клас-диаграма на слоеста архитектура



    - компонентно-базирано разслояване - основен подход за капсулирането на услугите компонент, който се описва със своя интерфейс - .jar
      Компонентите на отделни слоеве формират пакета на платформата. Всеки клас от компонентите е достъпен за приложението чрез своя интерфейс.
- виртуални машини - слоест модел, който предоставя високо ниво на абстракция - скрива изпълнителната платформа

  Нарича се машина, защото извършва услуги, които ги прави едно ядро на ОС, което за да изпълнява своите услуги трябва да се стартира.

  Ако 1 ядро го натоварим да изпълнява друго ядро на друга ОС, се получава виртуална машина.

    - предусловия за използване на виртуална машина: да са от един тип виртуална машина на Windows ще работи, ако отдолу има виртуална машина на Windows
    - емулация - изпълнение на функциите на дадена система от друга система (с принципно различни функции или организация). Емулация на Unix върху MS-DOS/Windows

! Когато има пълно покритие е **виртуална машина**, иначе е **емулатор**

Виртуалните машини са се появили с cloud услугите в последното десетилетие.

За всяка виртуална машина заделяме област от паметта на устройството: пясъчник - резервно работно пространство.

Когато използваме виртуални машини, данните са защитени и само ние имаме достъп до тях. Не може да се зареждат чужди данни.

Асинхронни архитектури

Базират се на неявни асинхронно обръщение между обслужващите процеси. Асинхронният обмен може да бъде:

- свързан (online) - без буфериране - и двата процеса трябва да са активни, не се блокират изчакващото в точката на обмен
- независим (offline) - с опосредяващ обмена процес - буфер на съобщението. Приемащият съобщението и обрат на процеса - буфер може да служи като
    - ! централизатор (MessageTopic) на всички съобщения и да ги препраща тематично до абонатите. 1 към много обмен
    - ! резервирана опашка (MessageQueue) - 1 към обмен

Асинхронните архитектури, още наречени Message-oriented middleware (MOM)

! Имаме 3 нива на обмен:

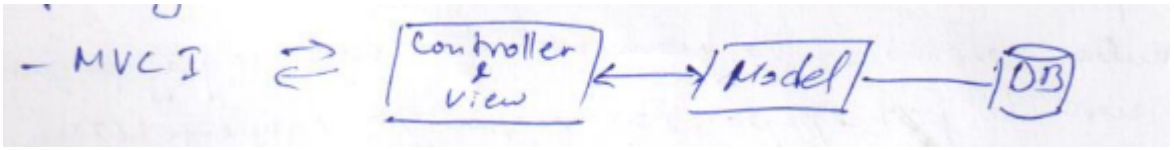
- **!** message-passing (MP) - работи по следния начин: Тръгваме от процес и В (?). Програмираме го да работи с този MOM, казвайки му кой процес е и, че тръгва, при което в регистъра на процеси се казва, че стартира процесът с това име и той е на еди кой си IP адрес и порт.

  Тази информация се записва локално на машината, на която се разпространява това съобщение. След което се регистрира, той може да изпраща съобщение на различни процеси. Така работи MP (online) <- само (?)

- message-queueing - пощенска кутия, която пази съобщенията при offline обмени за определено време
- message topic - регистриране на процеси и абонати по тези теми, които да получат съобщението (тематичен обмен)
  - от тук не е задължително
- небуферирани асинхронни СА: системата се декомпозира на 2+части:
  генератори на събитие, служители на събитие и регистратори на събитие
  Сравнително ниска производителност и голям системен свръхтовар
- буферни асинхронни СА: системата е контекстна, слабо свързана
  Декомпозира се на 3 части: оператори на съобщ., консуматори на съобщ., услуги за асинхронен буфериран обмен на съобщ.- MOM
  Висока скалируемост, надеждност, p2p приложения, поддържа опашка и тематичен обмен
- обхват на асинхронен СА - подходящи за слабо свързани системи с устойчив неявен обмен на съобщ., при които обменящите процеси са анонимни.
  Висока скалируемост
  Подходящи за пакетна обработка и за интегриране на наследени приложения в съвременни проекти.

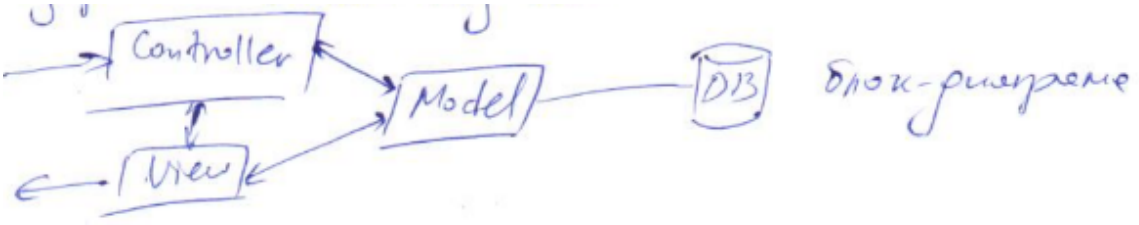
Интерактивни софтуерни архитектури

Поддържат интензивен потребителски интерфейс. При тях винаги доминира MVC модел на архитектурата. Разделяме потребителския интерфейс спрямо данните на 2 дяла: моделът е страната данни, VIEW е потребителския интерфейс, контролерът е модул за управление на процесите. Инициативен е клиентът. Контролерът е на принципа time-drive, замества (подсеща) клиента. Вместо клиентът сам да refresh-ва, контролерът го прави вместо него. И ако е настъпила промяна, се променя и view-то. Не е event-driven. Не е свързан с данните, но се генерира свръхтовар, при постъпване на много заявки, MVC е основен модел за сървърни приложения с Web-клиенти за достъп. Динамично се представят промените в данните, т.е в реално време при отдалечени клиенти

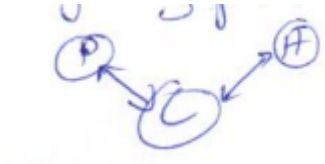


Контролерът и изгледът са 1 процес, регистрирани в 1 модул.(?) Приложим е за по-прости приложение с компактен GUI

- MVC II - контролерът и изгледът са самостоятелни. Допълнителна функция на контролера е да инициализира връзката между изгледа и модела и управлява обмена между тях.



- PAC (Presentation-Abstraction-Control) - развитие на MVC, което поддържа агенти обмен на съобщения. Системата се състои от множество специализирани агенти, декомпозирани на трите модула - P, A и C



Тук няма централизиран достъп до данните, а всеки агент може да съдържа данни. Презентационният модул на агенти е онзи контролният модул е задължителен: той управлява достъпа до функциите на агента.

Абстрактният модул капсулира данните и операциите на агентите -> обхват на РНС - прилагат се за интерактивни системи от кооперираща специализирани информационни агенти

Слабосвързана разпределена система - комуникациите са неблокиращи агенти.

Тема 7. Системни средства за синхронизация. Синхронизация и системно време. Протоколи за подреждане. Глобален статус. Взаимно изключване. Разпределени транзакции

Системно време и таймери - В разпределените системи (в сървърите) има едно определено време, което се нарича UTC и е предназначено за глобална координация. Всяка национална институция трябва да покрие територията си с предаватели на UTC и всеки сървър сравнява времето си с UTC. Системното време се отчита от таймер - кристален осцилатор + брояч + регистър за брой импулси за 1 секунда, с генерация на системно прекъсване. Системния часовник е процес, който отброява прекъсванията С по таймер. Таймерът мери системното време. Време диаграмата с импулси - броят на импулсите генерира локалния системен часовник (clock time). Различията на UTC и clock time се дължат на примеси в силициевия кристал. Реалните осцилатори в масовите компютри работят с относителна грешка

$\rho \approx 10 - 5$ т.е $1 - \rho <= dc/dt <= 1 + \rho$, ρ е максималното отклонение с възможно избързване или изоставане UTC се разпространява чрез късовълнови радиостанции.

Синхронизиращи алгоритми за системно време -

- синхронизация - необходимо е при комуникации между процесите, подреждане на разпределени събития, използване на системното време като аргумент. Алгоритмите се базират алтернативно на времеви сървър, който се синхронизира с UTC.
- централизиране (сървърна синхронизация) - периодични заявки от системни възли към времевия сървър (пасивен сървър)
    - проблеми: закъснението в цикъла заявка - обслужване отговор. Затова корекцията се прави като полученото време от сървъра се добавя обикновено половината от закъснението на отговора. Периодичната проверка на локалните системни времена във възлите и изравняване към средна стойност (активен сървър)
- p2p синхронизация - базира се на периодично общодостъпно предаване на локалното време от всеки възел. След определено изчакване в началото на всеки период, възлите изчисляват локално време. Параметри: период на гласуване R, период на изчакване S << R и брой игнорирани екстремни стойности T.
- синхронизация с времеви марки (timestamps) - синхронизиращи съобщения между възлите с времеви марки на локалните логически времена. Ако получаващия процес има по-малка стойност на локалното логическо време от маркера на изпратеното съобщение, той коригира своя логически часовник (+) към стойност (марка +1). Изискване: няма 2 събития с еднакво С. Ако синхронизиращия процес изпраща, той ги дистанцира логически на 1 такт.
- Алгоритъм на Lamport - имаме 3 машини (възела) с техните clock time. В първия момент О виждаме, че и трите машини са сверени, clock time им е О. След това обаче нещата се развалят. Всеки показва различни единици в следващия момент. Алгоритъмът гласи, че съобщенията в графики, които се разменят между машините се използват и за корекция на локалното системно време. Потребителското съдържание в 1 съобщение се нарича payload, а пък допълнителните данни, които се залепват до него, за да мине, се нарича: signalling - системен/комуникационен свръхтовар.

Протоколи за подреждане - протокол за тотално подреждане - прилага логическата синхронизация с времеви марки за еднакво подреждане на събитията при групово предаване, при което изпращащият процес като член на групата получава своите съобщения в реда на изпращането им и без загуби. Всеки приемащ процес записва получените съобщения в локалния буфер по реда на времевите марки и потвърждава приемането до процесите в групата. Всички съобщения са групови.

- локални буфери - в различните мрежи съобщенията се получават по различно време затова се подреждат различно. Но след секунди съобщенията се преподреждат и в един момент на всички устройства имат еднаква подредба, маркировката е със скаларни времеви марки. Имаме алгоритъм на Lamport.
- протокол за съхранено подреждане - позволява освен тотално подреждане на събития при запазване на реда им в реално време, така и запазване на причинно-следствената връзка, т.е съхранено подреждане. Тук маркировката е векторна. Всеки процес поддържа свой вектор от броят, чиито елементи броя събития, настъпили в процесите със съответен индекс. Когато Pi изпраща съобщение m към него добавя и текущата стойност на своя вектор Vi, като векторна мярка Vt. По този начин получаващият съобщението m процес Pj е информиран за събития, възникнали във всички процеси преди да изпрати m - т.е общият брой събития от които изпращането m може да е следствие.
- вложени транзакции (nested) - дървовиден набор от суб транзакции, първата от които инициира няколко следващи и т.н. в съответствие с логическо и причинно следствено разпределение на цялата “супер транзакция”. Всяка суб транзакция се счита за изпълнен само ако главната е изпълнена, а ако не е, се заличават резултатите на успешно изпълнените дъщерни суб транзакции.
- разпределени транзакции(distributed) - при тях декомпозицията на супер транзакции в суб транзакции не следва логическо разделение, а се определя от структурата на разпределения контекст - СУБД-то.

Серийно планиране на конкурентни транзакции - Запазване на резултата от конкурентни транзакции такъв какъвто би бил при последователното им изпълнение.

- конфликтни операции - когато има 2 или повече конкурентни транзакции извършват операции върху общи данни и поне 1 е запис ;четене-запис; запис - запис конфликт

Два планиращи подхода:

- песимистичен - операциите се проверяват за конфликти преди изпълнението и се подреждат преди да се изпълнят
- оптимистичен - синхронизацията се случва след изпълнението им. Ако накрая се установи, че е имало конфликтни операции, поне 1 от транзакциите се отменя(абортира)

Песимистично планиране с времеви марки - прилага се алгоритъмът на Лампорт. Транзакцията T има своя времева мярка S в момента на изпращане. А обектите данни x също се маркират с времеви марки за четене Sr(x) и запис Sw(x). И когато дойде транзакцията T за четене x.read(T, x). Ако времевата мярка на x за запис е по-късна от тази на T, T се отменя, защото x вече е било променено след старта на T: S(T) < Sw(x). Ако T е по-късно от времевата мярка на x за запис: S(T) > Sw(x) -> T се потвърждава, като Sr(x) = max { S(T), Sr(x)}

- write(T,x) : S(T) < Sr(x) -> T се отменя, x е прочетен
- write(T,x) : S(T) > Sr(x) -> T се потвърждава, Sw(x) = max {S(T), Sr(x) }

Оптимистично планиране с времеви марки - конкурентните транзакции се изпълняват докрай без заключване и сравняване на времеви марки. В края на транзакцията се проверява дали нейните операции са консистентни на операциите на останалите конкурентни транзакции и при откриване на промяна в даден обект след стартирането на тази транзакция, тя се отменя. Висок паралелизъм, но при отмяна на T, тя се рестартира отначало.

Тема 8. Модели на разпределеното обслужване. Сървърни разпределени услуги: Клиент-сървър. Трислоен модел. Брокерен модел. Сервизно-базиран модел. Компонентно базиран модел

Сървърите на устройства, проектирани с параметри - те представляват услугите за достъп до данните. При по-сложни данни нашият еднослоен сървър се разслоява. Имаме два вида сървъри: на приложението и backend сървъри - на данните.

Брокери на услуги - представляват сървъри на приложения, които допълнително дават възможност на потребителя за избор на реда параметри до еднотипни услуги. Брокерът на услуги представлява системата за достъп до тези услуги, а потребителят със своята заявка класира и избира една от изброените услуги. При базирания модел имаме маршрутизация на съобщенията до достигане на данни, защото тук нямаме връзка между активния и реактивния процес.

Обхват - вече приложението става MPMD - Multiple program, Multiple data - система за масово обслужване. Характерното за сървърните такива е, че те са централизирани приложения. Централизирания модел означава, че е възможно да се получи свръхтовар. По-лесно проектиране, но проблема със скалируемостта, производителността.

- топология - звезда/дърво/нелинеен конвейер/граф. Може да се наложи създаване на реплики, поради големия брой заявки като всяка отива в различен поддържащ сървър с реплики.
- грануларност - от едната страна имаме броя нишки, а от другата - броя задания. Броят заявки не трябва да пада под броя нишки. Тук обаче гланулярността има по-различен смисъл, който отразява времето за изпълнение на услугите. Колкото по-бързо се изпълняват те, толкова по-фина е грануларността и обратно
- управление
    - по време -> по събитие , разпределено -> централизирано -> йерархия. Централизирано управление, но ако заявките станат много и се прояви свръхтовар, се прави йерархия от клъстери - част от заявките се копират в 1 клъстер, нови и нови, потребителски заявки в друг клъстер. Балансирано между клъстерите става на едно по-високо ниво. Така получаваме йерархия от клъстери, или клъстерифициран между няколко централизирани системи, за да разчупим свръхтовара.

Модели - клиент сървър: многослоен сървър; брокерно-базиране архитектура (CORBA); сервизно-базирана архитектура (SOA); web-услуги, грид-услуги

- характеристики на услугите - адресна прозрачност, откриваемост и режим на достъп и надеждност

Клиент-сървър - класически MPMD - модел за изпълнение на сложни и съставни услуги - клиент, граничещ с безсървърно обслужване (p2p) -> активен интерфейсен процес и реактивен процес за приложната логика и данни е сървър. В зависимост от това каква тежест прехвърляме на клиентската или сървърната среда, моделът може да се раздели на базата на различни начини

- тънък клиент - не е много разпространен в нашите инфраструктури. Той представлява един интерфейсен терминал, с което въвеждаме буквено цифрова информация и получаваме резултат - потребителски вход и изход. Тънкият клиент обслужва само входа и изхода. А това какво да се показва на екрана се изтегля в сървърната част. Всичко, което прави администратора се отразява на сървъра и то се показва на клиента, уеб браузър, уеб браузър с JS, файл сървър, разпределен файлов сървър.
- предимства - ночно специализиране, инфраструктурна гъвкавост, преизползване на сървърните компоненти, еволюция на услугите без участие на потребителя
- недостатъци - унификация на клиентската инфраструктура. Защита на достъпа и данните, скалируемост на сървъра, скъпа поддръжка.

Сървърната страна се организира от доставчика на услуги.

Многослоен сървър -

- Декомпозиция на сървъра на минимум 2 слоя:
    - междинен за приложната логика

- вътрешен слой - за достъпа до данни и комуникация

- итеративен сървър - сървър на приложението, при който постъпва 1 заявка за обслужване по инициатива на клиента. Сървърът я изпълнява и връща реакцията към потребителския процес. Ако междувременно е постъпила друга заявка от друг потребител. Заявките се нареждат в опашка и се изпълняват 1 по 1. Има 1 активен процес. Всеки процес е стабилен докато броя на постъпилите заявки за обслужване на единица време е по-малък от броя на обслужените заявки. Ако е обратното сървърът/процесът е в неустойчиво състояние и не може да се изчисли средното време за изпълнение на 1 заявка.

Системите за масово обслужване не работят с итеративен сървър, а повече с:

- рекурсивен сървър с нова нишка за всяка нова заявка и/или потребител. В тази сървърна архитектура всяка заявка генерира нов процес. Обслужването на всяка е независимо, както при итеративния.
- многонишкова междинна услуга за ускорение или скалируемост

Garbage collector-a се грижи за премахване на процеси, които са излишни.

Недостатъци: унификация на клиентската инфраструктурна защита на достъпа и данните. Скалируемост на сървъра.

Брокер на услуги - представляват директория на услугите, които се класират. В тях има търсене на услуги.

- стартирането на 1 услуга означава регистрирането и в процеса “брокер” от сървъра. Брокерът на услуги се записва, класира я по параметри, с които тя се е регистрирала и връща потвърждение. От тук нататък услугата е достъпна в брокера на услуги. През него тя се активизира при получаване на заявка и връща съобщение към клиентския процес. Освен клиентския процес и брокерът, тък имаме проксита(service proxy, client proxy). Обменът се базира на тях - допълнителен протокол изпълняващ функциите на де/сериализиране. Прокситата представляват: “Зареди и изпълни, от страна на клиента, процеси и интерфейси”.
  Сървърното прокси “Брокерът зарежда услугата”
  - междуброкерни мостове: преформатират съобщенията между брокери с различни технологии
  - недостатъци: свръхтовар и поддръжка

Системи с брокери на съобщения - Базират се на XML съобщения за обмен на форматиране данни между приложенията. Прилагат се при слабо свързани при системни процеси, т.е. с неявна адресация, с повече от 1 възможна инстанция - реплика, в 2+ административни области. Поддържат се от всички доставчици.

Тема 9. Безсървърно обслужване (p2p). Приложение и модули: p2p, мрежи върху IP маршрутизация, откриване, отказоустойчивост, надеждност, репутация, защита, p2p данни: разпределени хеш-таблици.

Безсърверно масово обслужване (БМО)

Трябва да се извърши в клиентската инфраструктура. БМО включва p2p процеси - равнопоставени процеси - с универсална специализация. Всеки процес на управление, маршрутизация, регистрация на ресурси и данни е разпределен т.е. всичко равнопоставени комуникиращи процеси трябва да ги управляват

Възел ~== процес с малка разлика.

Мрежата е инцидентна - устройствата се присъединяват ad hoc, т.е динамична свързаност на възлите. Всяка система е по-скалируема от централизирана.

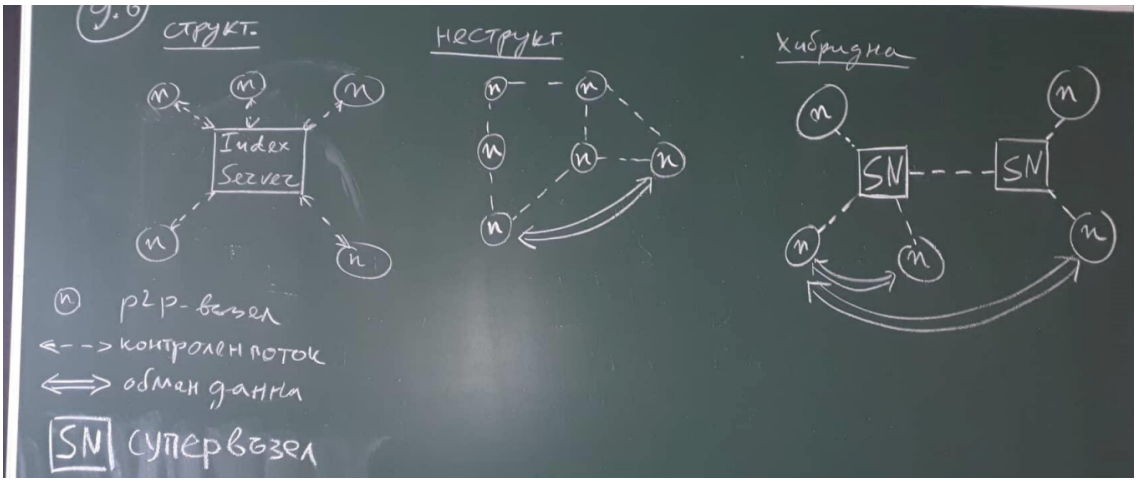
Анонимност на споделянето: влиза в пакета за защита на данните на отделните потребителски процеси. Кодиране на данни, оторизиране на потребители.

Комуникация в p2p процесът минава през няколко други, за да достигне до възела, до който са адресирани данните на нашия процес, т.е. други процеси могат да четат данните, които обменяме с други възли от цялата група

- юридическа регулация - докато доставчиците на сървърно-базирани ИТ услуги са регистрирани търговци, те носят отговорност за поддържането на данните техния сървър. Всичко това е регистрирано и подлежи на проверка и контрол иначе санкции - пиратското съдържание.
  За да може един процес да изпълнява сървърните функции е важно той да бъде активен постоянно.

Типове БсМО

- **наслоени мрежи за БсМО**
  - неструкторирани
    произволен граф на свързаност и производителност на комуникационните канали, спонтанно добавяне и отпадане на възли (типично за нововъзникнали БсМО)
  - структорирани - използва се топология за маршрутизация, обикновено кръг, хорда, дърво, верига. Мрежите с комуникация на съобщ. - отсъства понятието за съседство. В структорираната мрежа нашите възли-процеси имат съседи, които не са непременно в една мрежа (локална). Един възел за да комуникира със своите съседи, трябва да знае техните IP адреси и портове. Колкото повече съседи има един възел, токкова по-бързо ще открие резултата от търсенето си. Но експоненциално толкова свръхтовар ще си генерира в цялата мрежа.
    За да работи една структорирана мрежа, значи има определен брой процеси, участващи в топология (хорда) и чрез хорди се определя кои са съседите. Има лимит на бр. комуникиращи процеси.
  - хибридни: структурирани – неструктурирана супермрежа от супервъзли, участващи в по 1+ структурирана наслоена мрежа (супервъзел)



- **Разпределени хеш-таблици (DHT)** - данните са организирани на много по-ниско ниво. Тези таблици представляват списък на ключовете на споделеното съдържание (данни, в които от едната страна имаме ключ, а от другата - IP адреси в мрежата). Когато възникне въпросът за търсене на данни, трябва да знаем ключа (име на файл, комбинация от атрибути)
  на базата на атрибутите се генерира ключ и срещу него е адреса на възела и процеса, където е съответното съдържание.
  DHT е не-p2p услуга на междинния слой.
  DHT е стръктурен елемент на p2p мрежите.

Случаи на приложение

1. споделяне на файлове 1:1 - заявките се разпространяват в инцидентна верига от процеси, а при установено наличност на файла, обменът се извършва пряко между двата възела – Napster, Gnutella, eDonkey.
2. Разпространяване на съдържание (1+):*: процесите пре-предават полученото от тях съдържание докато то се разпространи до всички заявители – BitTorrent.
3. p2p чат (“instant messaging“): текстови съобщения и поточни данни между клиентските процеси без сървърна поддръжка – Skype, MSN.
4. Разпределена обработка: декомпозиция на данните на сложна задача и независимата им обработка от група еднакви процеси (SPMD) – SETI@home.
5. Съвместна работна среда: форматни “събития” и други съобщения се обменят групово между процесите – Microsoft SharePoint Workspace (преди Groove), (защитèн чат и обмен в различни административни области на инцидентно формирана група от клиентски процеси с пре-предаване).
6. Развойни платформи за ИТ-услуги – JXTA, MS .Net: с поддръжка и на p2p-проекти за стандартни приложноориентиани услуги: защита на достъпа и обменя на регистрираните (еднотипни, интерфейсни) процеси. Директория (<мр.адрес, рег. име >) на процесите. Директория на споделените файлове. Оторизацията е в централизирана мрежа от резидентни услуги. Обменът е пряк междъ регистр. прегодни процеси.

Bit Torrent протокол

• Протокол, технология и среда за споделяне на файлове в инцидентни (ad hoc) p2p мрежи
• 1/3 от Интернет трафика, но само 1/5 от висикоскоростния трафик (понеже се поддържа основно от потребителски мрежи и устройства)
• базира се на споделяне на дискови, компютърни, комуникационни ресурси с минимална сървърна поддръжка
• BT-клиента:
  - поддържа списък на локални работни копия на споделените файлове и прозрачно обслужва заявки (seeder)
  - генерира последователност от заявки за зареждане на файл
    - заявките са TCP-формат – към множество възли поддържащи отделни части от файла (при webбраузърите единична HTTP GET заявка към един сървер – дори и при реплики на файла) – посложно управление (signaling), но потенциално уплътнен трафик
    - негарантирана скорост и ред на зареждане – неподходящ за изпреварващо зареждане на поточни ММ данни (progressive streaming) – текущи разработки
    - планът на заявките е rarest-first
  - зарежда метаданни за заявен файл – torrent – списък с текущите клиенти, поддържащи копия на части от файла и адрес на сървера, координиращ процеса (tracker)
• BT-сървер – tracker:
директория с мета-данни (торенти); контролира коефициента на споделяне на клиентите (за избягване на leeching „пиявици“ – егоистично използване)

Skype протокол

- прилага хибриден модел топология, като различава 3 категории процеси („възли“) – възел, супервъзел и сървер на профилите
- Супервъзлите се подбират да имат физическо IP, подходяща локална и мрежова производителност и по режим на работа – прозрачно за потребителя/администратора
- за мултимедиен обмен (звук и/ли видео) възелът се свързва със супервъзел
- всеки възел (и супер-) има около 200 съседи с няколко супер („host cache”: IP-тата и скайп-портовете), като доставчикът на услугата е разгърнал и подходящ брой супервъзли в националната мрежа
- при начално инсталиране/свързване започва попълването на хост-кеша; наличността на профилите и заявките за откриване се съхраняват в супер-, а запазените профили – локално
- без защитна стена около един от комуникиращите профили (2+) съобщенията и сигналинга са TCP, а потоците – през UDP (ако маршрутизират през защитна среда – и потоците са върху TCP

Хетерогенност и скалируемост на p2p мрежите

- payload - Клиентско съдържание
- signaling - (комуникационен свръхтовар)

• хетерогенността се изразява в разликите в производителност на равнопоставените възли, платформа (ОС) и мрежова скорост – решават се със средствата на йерархичното разслояване – напр. като при скайп-протокола
•изисква се независимост от броя свързани процеси: повече възли увеличават сигналинга логаритмично, но също добавят нови ресурси
•ако времето за разпространяване не е логаритмична а линейна функция на p – обслужването се счита за нескалируемо
•маршрутизацията, основана на наводняване, не може да работи с логаритмичен сигналинг, ако не се йерархизира
•йерархизираното наводняване работи само ако • възлите са директно свързани със супервъзли и • наводняването се разпространява само между супервъзлите

Ефективност на търсенето и откриването

•обектите данни и техните списъци са разпределени между възлите и търсенето/откриването им предхожда достъпа до тях
•търсенето е сляпо или информирано

- при сляпо търсене възелът или потвърждава наличието на локален обект, или препраща заявката наводняващо до всички свои съседи, или я препраща до произволна част от съседите си без критерий за селекцията им (голям сигналинг)
- при информираното търсене възлите поддържат локално и списъците на съседите си през r стъпки (hops или „пръсти“) в топологичния граф, така че препращането не е наводняващо а фокусирано (голям свръхтовар от синхронизацията на външните списъци)

Анонимност

• анонимността в p2p се разглежда като скриване на:

- инициативния процес/потребител; реактивния процес/потребител; -> еднопосочна
- • двата; всички (групов) -> двупосочна

• защитена („лукова“, onion) маршрутизация за еднопосочна анонимност :