ПАРАМЕТРИ НА ПАРАЛЕЛНАТА И РАЗПРЕДЕЛАНА ОБРАБОТКА, МЕТРИКА, МЕТОДИ ЗА АНАЛИЗ.

Развитието на паралелната обработка е свързано със следните принципи и закони.

- **Принцип на неограничения паралелизъм -** Ако съществува паралелен алгоритъм за изпълнение на дадена задача, то той се реализира в хипотетична паралелна КС, за която няма ограничения относно функционалните възможности.
- **Закон на Amdahl -** Всеки процес, подлежащ на разпаралелване, се състои от една част, която може да бъде изпълнена паралелно и от друга част, която трябва да се изпълни последователно. Системната производителност не може да надвиши определено ниво, определено от наличието на последователно-изпълнима част от програмата

Паралелната обработка е свързана с изпълнение на паралелна програма в подходяща парлелна КС (ПКС).Една задача може да бъде изчислена паралелно, ако може да се състави **паралелен алгоритъм**, допускащ реализация върху хипотетична паралелна КС.

Проектирането на ПА минава през следните фази:

- **Разделяне** (**partitioning**) – декомпозоция на проблема по данни или по функции.
- **Комуникации** (**communication**) – формулира информационните или контролните зависимости между отделните подзадания. Комуникациите са представят като канали и съобщения (т.е. данни и команди), които се предават по тези канали.
- **Форматиране** (**agglomeration**) – формулираните подзадания и прилежащите им комуникации, се групират в задания, при което се отчитат характеристиките на архитектурата на обработка – основно брой процесори/възли и комуникационен модел – и 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) и технологично оптимизиране.
- **Разпределение** (**mapping**) – незадължителна фаза, която се състои в разпределение на форматираните задания по обработващите възли на системата със 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. За разлика от тях обаче при паралелните алгоритми сложността е функция на архитектурата и на средата за паралелна обработка.

- **Степен на паралелизъм** – максималния брой операции, които могат да се изпълнят паралелно при обработката на алгоритъма – това е архитектурна величина.
- **Ускорение (acceleration)**
  - $S_p = \frac{T_1}{T_p}$, където T_1 е последователното време за изпълнение, а T_p е паралелното време за изпълнение
- **Ефективност (eficiency)**
  - $E_p = \frac{S_p}{p}$, където S_p е ускорението, а p е броят използвани компютри
- **Цена при обработка (cost)**
  - $C_p = p.T_p$ **,** където p е брой процесори, а T_p е време за изпълнение на една операция
- **Коефициент на използване (utilization)**
  - $U_p = \frac{O_p}{C_p}$, където O_p е брой на операциите с p процесора, а C_p е цената на обработка
- **Темп на обработка (execution rate)**
  - Представя се с няколко скали и изборът на скала зависи от типа ПА
- **Излишък (redundancy)**
  - $R_p = \frac{O_p}{O_1}$ **,** където O_p е брой на операциитес p процесора за паралелна обработка, а O_1 е брой на операциите с един процесор за последователна обработка

Коректността на даден паралелен алгоритъм е архитектурно независима, но неговата ефективност зависи от изпълнителната платформа, поради което е целесъобразно сложността му да се оценява и като функция на разпределението (**mapping**).

- **алгоритмичната сложност** - оценява времевите и пространствените характеристики на обработката
- **времева сложност** се задава като брой елементарни операции и комуникации (от който се получава времето за обработка в дадена архитектура),
- **пространствената сложност** - в брой алоцирани регистри и клетки памет.
- **процесорна сложност –** прилага се за по-обща оценка на ПА

МОДЕЛИ НА РАЗПРЕДЕЛЕНИТЕ СОФТУЕРНИ АРХИТЕКТУРИ

Софтуерната архитектура – представя/моделира програмния проект, като рапределен процес от софтуерни компоненти.

OBJECT-ORIENTED ARCHITECTURE

Обектно-ориентираният архитектурен стил е основан на разделянето на отговорностите на системата в индивидуални повтроно използваеми и независими обекти, всяка съдържаща данните и поведението, свързани с обекта. Тази архитектура вижда системата като група от сътрудничещи си обекти, вместо множество от инструкции. Ключовите принципи на този стил са:

- **Абстракция** - опростява сложната реалност чрез моделиране на класове подходящи за разрешаването на определен проблем.
- **Композиция** – обектите могат да се сглобят от други обекти и тези вътрешни обекти може да се скрият от други класове или да са достъпни чрез интерфейси
- **Наследственост** – обектите могат да наследяват други обекти и да използват техните функционалности или да им припишат ново

- **Капсулация** – обектите скриват вътрешните си детайли и могат да са достъпни само чрез
- **Полиморфизъм** – Това свойство дава възможност различни операции да имат едно и също име. Той позволява един метод на класа да реагира различно, в зависимост от използвания обект.

DATA FLOW SOFTWARE ARCHITECTURE STYLE

Този архитектурен стил вижда цялата система като последователност от трансформации на последователен набор от данни, където данните и операциите над тях са независими един от друг. Софтуерната система се декомпозира на обработваеми елементи, където данните насочват и контролират реда на изчислителния процес. Всеки компонент трансформира входните данни в съответните изходни данни.

- **Пакетна обработка**
  - Независимите програми биха обработили данните, предоставящи файл като изход
  - Този файл би станал вход за друга независима програма, която би го прочела, обработила и формирала друг изходен файл
- **Pipes and Filters** - Използва се за разделяне на големи задачи за обработква в последователност от малки, независими стъпки (филтри), които са свързани чрез канали (Pipes).

DATA-CENTERED STYLE

DATA STORE се намира в центъра на архитектурата и е достъпна често от други компоненти, които се обновяват, добавят, изтриват или модифицират по друг начин на данни в рамките на магазина. Контекстните архитектури насърчават integrability, което означава, че съществуващи компоненти могат да бъдат променени и да могат се добавят нови клиентни компоненти към архитектурата, без да се интересуваме от другите клиенти.

- **Хранилище (Repository)** - Подсистемите си комуникират единствено чрез хранилището. Сравнително независими са. Контролът върху поведението на системата може да се осъществява както от хранилището (при настъпването на някакво събитие в него), така и от подсистемите, които да освобождават ключове в хранилището и по този начин да дават зелена светлина на друга подсистема да ползва съответните ресурси.
- **Blackboard** - Тази архитектура има компонент „черна дъска", който се държи като централно хранилище, и множество други компоненти, които са зависими от общата структура данни, съхранявани в „черната дъска".

HIERARCHICAL ARCHITECTURE

Тази архитектура се характеризира с това, че вижда цялата система като йерархична структура. Системата е декомпозирана в логически модули на разлчини нива в йерархията. Модулите от различните нива са свързани чрез явни и неявни извиквания от методи. Т.е. модулите от по-ниските нива предоставят услуги на непосредствените модули от по-високо ниво, които извикват методи или процедури от ниското ниво.

- **Master-Slave** - В този архитектурен стил, „робите" предоставят услуги на своите „господари", и господарите избират определен резултат сред робите чрез различни стратегии. Робите могат да изпълнят същата задача с различни алгоритми и методи или чрез напълно различни функционалности.
- **Layered** - Тази архитектура се декомпозира на ниски и високи нива в йерархията;
- **Virtual Machine** - Виртуалната машина е изградена на базата на съществуващи системи и предоставя виртуална абстракция, множество от атрибути и операции. Обикновено разделя приложната среда от програмната.

ASYNCHRONOUS SOFTWARE ARCHITECTURE

Архитектури базирани на неявни обръщения между обслужващите процеси. Обемнът може да бъде online или offline – втория има буфер, докато първия няма. Активния процес генерира съобщения ,а пасивните ги получават и евентуално реагират.

- **Nonbuffered Event-Based** - Разделят системата на два дяла – източници на събития и слушатели на събития. Процеса на регистриран ена събития, свързва тези два дяла. Няма буфер между тях
- **Buffered Message-Based** - Разделя системата на три дяла: генератори на съобщения, консуматори на съобщения и услуга за асинхронен буфериран обмен на съобщения.

INTERACTION-ORIENTED SOFTWARE ARCHITECTURE

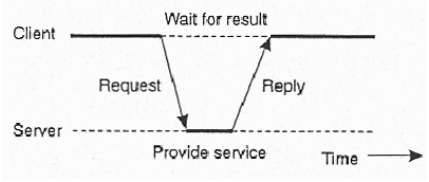
Разделя системата на три главни дяла: модул за данни, контролен модул и презентационен модул (изглед). Всеки модул си има свои отговорности. Модулът за данни предоставя абстракцията на данните и основаната бизнес логика за обработка на данните. Презентационния модул е отговорен за визуалното или звуково представяне на данни и може да използва потребителски интерфейси. Контролиращият модул установява потока на контрол, включващ избор, комуникация между модулите, изпращането на работа и инициализация на определени данни и действия за конфигуриране на действията.

ОРГАНИЗАЦИЯ НА РАЗПРЕДЕЛЕНИТЕ ПРИЛОЖЕНИЯ

КЛИЕНТ-СЪРВЪР, ДВУСЛОЙНИ АРХИТЕКТУРИ

При основния клиент-сървър модел процесите са обособени в две групи, които може да се припокриват.

- **Сървърът** е процес, който имплементира конкретна услуга (например сървър за база от данни).
- **Клиентът** е процес, който изисква някаква услуга от сървъра чрез изпращане на заявка и след това чака отговор от сървъра.



Взаимодействие между клиент и сървър

Комуникацията между клиента и сървъра може да се осъществи с прости протоколи без изграждане на връзка (connectionless protocoles – **UDP**) или чрез използване на протоколи, които установяват връзка (connection-oriented procotols – **TCP**).

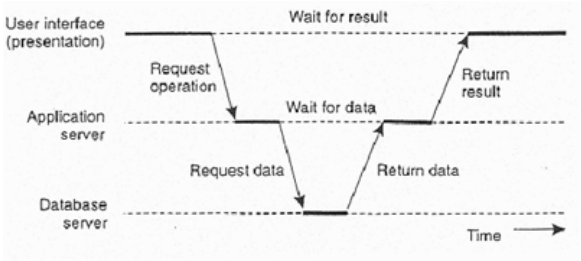
ТРИСЛОЙНА АРХИТЕКТУРА.

Голяма част от клиенти-сървър приложенията са ориентирани към предоставянето на достъп до някаква информация (база от данни). Имайки предвид това, може да разграничим три основни слоя:

- **потребителски интерфейс** – съдържа програми, които позволяват на потребителя да взаимодейства с приложението. Представя на потребителя информацията, получена от изпълнението на приложението.
- **слой с бизнес логиката** – съдържа логиката за обработка на информацията.

- **слой с данните** – съхранява информацията, с която приложението работи. Негова задача е да пази данните консистентни. Най-често съдържа база от данни.

В този случай слоят с бизнес логиката се намира на отделен сървър, който играе ролята и на клиент, когато взаимодейства със сървъра с данните.



Комуникация в трислойна архитектура

Причината за появяването на многослойните архитектури е разделянето на приложения на три слоя.

- **вертикално разпределение** - разпределената обработка означава организирането на клиент-сървър приложението в многослойна архитектура
- **хоризонтално разпределение** - клиентът или сървърът е физически разпределен, като всяка отделна логическа част оперира върху част от данни

КОМУНИКАЦИОННИ МОДЕЛИ В КЛИЕНТ-СЪРВЪР АРХИТЕКТУРИТЕ

RPC (**Remote Procedure Call**) – Отдалечено извикване на процедури. Идеята е извикването на отдалечена процедура да изглежда като извикване на локална процедура (да се постигне прозрачност).

RMI (**Remote Method Invocation**) – Този модел е развитие на RPC чрез използване на принципите на обектно-ориентираното програмиране. В този случай се извикват методи на разпределени обекти. Свързването с тях става като в адресното пространство на клиента се зарежда интерфейса на обекта (proxy – изпълнява ролята на клиентския стъб).

MOM (**Message-Oriented Middleware**) – Комуникация, базирана на обмяната на съобщения. Компютрите са свързани посредством комуникационна инфраструктура (комуникационни сървъри). Всеки клиент предоставя интерфейс към комуникационната система за изпращане на съобщения.

P2P

Peer to peer, представлява разпределена 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 без необходимост от централна мрежова инстанция. Участниците са едновременно "снабдители" и "потребители" на ресурси в контраст на традиционния клиентско-сървърен модел, където само сървърите дават (снабдяват), а клиентите (clients) консумират.

СЪРВЪРИ ЗА ПРИЛОЖЕНИЯ И WEB-СЪРВЪРИ

Сървърът за приложения е сървър, който е специализиран в изпълнението на определени приложения. Сървърът за приложения представлява съвкупност от компоненти, които могат да бъдат достъпени чрез платформено независими интерфейси за програмиране (**API** – Application programming interface).

Web-сървърите са специализиран вариант на сървърите за приложения.

- Съдържанието, което се доставя, е основно HTML текст, но може да съдържа изображения, мултимедия, документи или други видове файлове, дефинирани съгласно MIME.
- Освен статичното съдържание, предоставят и интерфейс към сървърни web-приложения чрез някой от стандартните езици/интерфейси.
- Повечето web-сървъри предоставят и допълнителни системни функции като контрол на трафика, компресия/декомпресия на съдържанието, виртуален хостинг.

МЕТАСИСТЕМИ И ГРИД СИСТЕМИ

Разпределените системи, които се използват за извършване на сложни изчисления могат да бъдат разделени в две категории – **клъстерни системи** и **грид системи**.

- **Клъстерните системи** - състоят се от множество подобни компютри, които са свързани помежду си чрез високоскоростна локална мрежа.
- **Грид системи** - изградени са от различни компютри, с различни операционни системи, намират се в различни административни области, имат различни политики за сигурност.

СЕРВИЗНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Сервизно-ориентираната архитектура (SOA) представлява съвкупност от принципи за изграждането и поддръжката на бизнес процеси в големи разпределени системи.

Базира се на три ключови концепции:

- **Услуги** - представляват софтуерни компоненти, които имат определен функционален смисъл (често енкапсулират бизнес концепция от високо ниво).
- **Сервизна шина** - инфраструктура, която позволява съвместната работа между разпределени системи. Улеснява разпределянето на бизнес процесите върху много системи, които използват различни платформи и технологии.
- **Слабото свързване** - концепция за намаляване на зависимостите в системата. Тъй като процесите са разпределени върху множество компютри, е важно да се намали ефектът от промени и грешки.

МОДЕЛНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Моделно-ориентираната архитектура е софтуерна архитектура за автоматизирано проектиране на приложения.

- базира на принципите на моделното проектиране (**Model-driven engineering**) – изграждането на абстрактни модели, които да описват системата от гледна точка на конкретната област.
- цели на тази архитектура - разделянето на дизайна от конкретната имплементация.

АСПЕКТНО-ОРИЕНТИРАНИ АРХИТЕКТУРИ

Аспектно-ориентираната архитектура е софтуерна архитектура, която се базира на принципите на аспектно-ориентираното програмиране. Основаната цел е да се подобри структурата на приложението и да се улесни добавянето на нови функционалности. В общия случай разделяме отделните изисквания в отделни модули. В процеса на това разделяне се появяват много съображения, които не могат да бъдат адекватно обособени в отделни компоненти – нар. се **аспекти**. Аспектите представляват общи функционалности, които се прилагат в няколко модула.

СОФТУЕРНИ АГЕНТИ

Софтуерните агенти са автономни процеси, които могат да изпълняват някакви задания заедно с други агенти (които може да са отдалечени). Основните свойства на софтуерните агенти са:

- **автономност** – могат да извършват някакви задачи и да взимат решения без човешка намеса;
- **комуникативност** – могат да взаимодействат и да обменят информация с други агенти или с потребителя;
- **реактивност** – възприемат средата, която се намират и могат да отговарят на промените в нея;
- **проактивност** – могат да променят параметрите на средата, в която се намират;
- **мобилност** – миграция между различни възли. Това свойство не е характерно за всички софтуери агенти;
- **адаптивност** – възможност за приспособяване към средата (еволюция на реакциите при еднакви параметри на средата). Това свойство не е характерно за всички софтуерни агенти;
- **непреходност** – многократно изпълнение на едни и същи функции. Това свойство не е характерно за всички софтуерни агенти.