МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

О. В. Русанова,
О. В. Корочкін

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

ЧАСТИНА 2

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Комп'ютерні системи та мережі»
спеціальності 123 Комп'ютерна інженерія

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент Тарасенко-Клятченко О.В., доцент, к.т.н., доцент кафедри системного програмування і спеціалізованих комп'ютерних систем

Відповідальний редактор Кулаков Ю.О., професор, д.т.н., професор кафедри обчислювальної техніки

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 26.05.2022 р.) за поданням Вченої ради факультету інформатики та обчислювальної техніки (протокол № 8 від 18.04.2022 р.)

Розглянута проблема підвищення реальної продуктивності комп'ютерних систем за рахунок ефективного планування обчислень. Наведені класифікації методів і задач планування. Викладені вихідні дані, етапи та різні алгоритми планування. Особлива увага приділяється особливостям алгоритмів планування для різних типів однорідних та неоднорідних паралельних систем, а також розподілених GRID та CLOUD систем. Посібник призначений для здобувачів ступеня магістра за спеціальністю 123 Комп'ютерна інженерія, а також буде корисним при виконанні магістерських дисертацій присвячених розробці засобів підвищення продуктивності для сучасних високопродуктивних паралельних комп'ютерних систем.

Реєстр. № НП 21/22-488. Обсяг 5,2 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

I. Вступ

1. Організація обчислювальних процесів в паралельних КС

Принципова відмінність операційних систем (ОС) паралельних комп'ютерних систем (КС) від традиційних полягає в управлінні декількома одночасно виконуваними процесами, що обробляють інформацію, засобами передачі повідомлень між цими процесами і засобами синхронізації цих процесів. При розгляді сутності таких ОС необхідно відповісти на два основні питання:

1. На яких (якому) процесорах (-рі) будуть виконуватися функції ОС? Це проблема конфігурації ОС.

2. Які функціональні відмінності ОС паралельних КС і традиційних ОС?

2. Конфігурації ОС для паралельних КС

Існує пряма залежність між конфігурацією КС і її ОС. Розглянемо основні конфігурації КС і відповідні їм ОС.

КС з конфігурацією *«головний-підпорядкований»* або з несиметричною архітектурою. У цьому випадку всі функції ОС виконуються на головному процесорі. Підпорядковані процесори використовуються тільки для виконання прикладних програм. Така несиметрична конфігурація легко реалізується як розширення однопроцесорної (традиційної) КС.

Переваги даного підходу полягають у простоті (низька вартість) і у використанні тільки одного екземпляра керуючих програм (економія оперативної пам'яті). Головний процесор підтримує всі черги робіт для всіх процесорів і звільняє від планування підпорядковані процесори.

Недоліком є збільшення числа переривань, оскільки багато переривань підпорядкованих процесорів повинні передаватися головному процесору також через переривання. Оскільки це вимагатиме додаткових ресурсів, а частота переривань може виявитися високою, то головний процесор не зможе виконувати нічого, крім перемикання процесів. У

даному випадку головний процесор настільки перевантажений функціями планувальника, що не виконує жодних робіт користувачів.

Серйозним недоліком також є те, що збій головного процесора може призвести до зупинки всієї системи (низька надійність). Ще один **недолік** полягає в тому, що негнучкість даного підходу може призвести до дуже неефективного управління ресурсами.

Наведемо приклади КС, що використовують описаний підхід до створення ОС. Досить типовим прикладом є матричні КС. Так у системі ILLIAC-IV, крім матриці процесорів (8*8), існував керуючий процесор (ЕОМ В-6700), на якому виконувалися всі функції ОС. У КС ПС-2000 множиною виконавчих ПЕ (64), управляла моніторна підсистема (у вигляді 2х міні ЕОМ СМ-2), яка виконувала всі функції ОС. У КС DAP всіх поколінь також матриця процесорів (64*64) призначена для програм користувача, а комп'ютер ICL2900 використовується в якості ведучої машини для виконання системних програм ОС. І, нарешті, матрична КС MPP, складається з процесорної матриці 128*128 і ведучої ЕОМ VAX 11/780. Всі функції ОС виконуються на провідній ЕОМ.

Іншим прикладом є векторні КС. КС Cray 2,3,4 є несиметричними. Так, система Cray2 складається з 4 підпорядкованих ПЕ, кожен по структурі подібний до Cray1, і одного головного процесора, який призначений для координації системних функцій і управління системою, а також для обслуговування запитів на організацію взаємодії між її компонентами (тобто для виконання функцій ОС).

КС Facom складається також з векторного і інтерфейсного процесорів. Векторний процесор складається зі скалярних і векторних виконавчих пристроїв з ОП, призначених для виконання програм користувача, а інтерфейсний процесор - для функцій ОС.

Менш чутливою до збоїв є конфігурація, в якій ОС може виконуватися на кожному процесорі. Таку конфігурацію називають *гнучко зв'язаною*. Управляючі програми повинні бути або повторно вхідними, щоб їх можна було паралельно виконувати на різних процесорах, або вони повинні зберігатися в багатьох екземплярах. Хоча кожен процесор має свої черги процесів і свої таблиці, ряд глобальних таблиць повинен бути використаний спільно. Це породжує проблему цілісності даних і синхронізації процесів. При такій конфігурації важко виявити збій процесора та ініціювати його роботу заново. Процес зазвичай протягом усього часу існування залишається пов'язаним тільки з одним процесором. При використанні ефективних алгоритмів розподілу процесів між процесорами (повністю автономні процеси) даний підхід може суттєво підвищити пропускну здатність системи. Головною **перевагою** даного

підходу є висока надійність. **Недоліками** гнучко пов'язаних конфігурацій є великі витрати пам'яті, а також вузька спеціалізація застосування - тільки для слабко-пов'язаних завдань. Цей підхід був започаткованиий у багатомашинних обчислювальних комплексах.

Підвищення ефективності роботи ОС можна досягнути, розглядаючи процесори як множину ресурсів. Замість того, щоб виконувати на кожному процесорі одні й ті самі керуючі програми, функції ОС можна розподілити між ними. Існує два варіанти розподілених ОС:

- *Динамічно розподілені* ОС.
- *Статично розподілені* ОС.

У першому випадку будь-який процесор може виконувати різні керуючі програми ОС і в симетричній конфігурації це призводить до так званого рівноправного режиму. Усі черги і таблиці зберігаються в спільній пам'яті, так само як і єдина копія ОС, що повторно використовується. Оскільки процесори трактуються як множина ресурсів, процесу не обов'язково виконуватися завжди на одному і тому ж процесорі. В результаті досягається більш збалансоване завантаження, а в разі збою відбувається тільки зниження продуктивності. Оскільки керуюча програма і дані знаходяться в загальній пам'яті, конкуренція за їх використання дуже велика. Найбільш легко така конфігурація реалізується в тому випадку, коли всі процесори ідентичні. Прикладом такої динамічно розподіленої ОС є симетричні системи Cray - X/MP і Cray - Y/MP, які складаються з 4(8) ідентичних процесорів, причому кожен з них може виконувати або задачі користувача (користувацькі), або ж системні функції ОС (моніторний режим). Залежно від ситуації в моніторному режимі може перебувати будь-яке число процесорів. Будь-який процесор в моніторному режимі може перевести в цей режим будь-який процесор, який знаходиться в режимі користувача.

Статично розподілена конфігурація ОС досить характерна для систем з масовим паралелізмом, у складі якої величезна кількість процесорів (десятки, сотні тисяч, мільйони). Вони, як правило, об'єднані в кластерну архітектуру. У кожному кластері є свій так званий керуючий процесор (локальний хост-процесор). Між локальними хост-процесорами розподілені функції ОС, а кожен процесор управляє своїми ресурсами. Прикладом статично розподіленої ОС є Helios для трансп'ютерних КС. В ній немає централізованого управління. В результаті з'являється можливість підвищити надійність ОС, зменшити вплив окремих відмов, пов'язаних, наприклад, із зміною архітектури системи. У кожен процесор системи завантажується ядро ОС, яке управляє ресурсами цього процесора.

ОС Helios може працювати, коли кожен процесор має локальну пам'ять ємністю не менше 256 Кбайт. Helios оперує програмними модулями, що називаються завданнями (tasks). Кожна задача є або прикладною (клієнт) або системною (сервер) для хост-процесорів. Є менеджер об'єднань завдань (Task force manager). Розподілений сервер (безліч хост-процесорів) виконує завдання менеджера, тобто автоматично розподіляє об'єднання (групи) завдань по мережі і стежить за балансом навантаження на ОС.

Існує варіант конфігурації, що поєднує риси першої і третьої конфігурацій. Іншими словами, це системи типу «головний - підпорядкований» (несиметричні), але головних процесорів в системі певна кількість. Функції ОС розподілені між цими головними процесорами. Таким чином, структура ОС паралельних КС може бути близька до традиційної, яка є розподіленою, тобто виконуваною на декількох процесорах. Прикладами такого підходу є КС з декількома периферійними процесорами і множиною центральних процесорів з ОП типу ПС-3000, CYBER-171 та ін.

Підводячи підсумок вищесказаного, слід зазначити, що найбільш перспективними конфігураціями, в даний час, є розподілені ОС.

Крім паралельних КС, в останні роки швидкий розвиток отримали комп'ютерні мережі та розподілені системи. Виникає питання про відмінні характеристики ОС, що застосовуються в перерахованих системах. Дана інформація представлена в таблиці 1.

Таблиця 1.

Характеристика	**Мережеві ОС**	**ОС для розподілених систем**	**ОС для паралельних КС**
Комп'ютерна система є віртуально одноядерним комп'ютером	-	+	+
На всіх процесорах одна і та ж ОС	-	+	+

Кількість копій ОС в пам'яті	N	N	1
Засіб комутації	Поділювані файли	Повідомлення	Поділювана пам'ять або повідомлення
Чи потрібне узгодження мережевого протоколу	+	+	-
Єдина черга виконання процесів	-	-	+ -

3. Функції розподілених ОС

Насамперед слід зазначити, що при роботі багатопроцесорних КС виникає чотири різних види паралелізму обчислювальних процесів:

1. Між процесами всередині однієї задачі;
2. Між процесами, що належать різним завданням;
3. Між процесами задач користувача і процесами ОС;
4. Між процесами самої ОС.

У зв'язку з цим в розподілених ОС виникає ряд функцій, не властивих ОС однопроцесорним КС. До них відносяться:

- ініціація і завершення паралельно виконуваних процесів;
- обмін повідомленнями між ними та їх синхронізація;
- розподіл обчислювальних ресурсів для самої розподіленої ОС.

Крім цього в таких ОС ускладнюється вирішення або набуває великого значення ряд проблем, що існують в традиційних ОС. До таких проблем відносяться: розподіл ресурсів між задачами та процесами, небезпека виникнення глухих кутів та інше.

Перераховані функції необхідно реалізовувати для ОС багатопроцесорних КС. Для матричних і векторних КС ОС за функціями практично не відрізняється від традиційних. Для Dataflow систем додатковою функцією є розподіл готових команд між процесорами [1] .

II. Планування та організація паралельних процесів

Розглянемо основні терміни, що використовуються при плануванні обчислень. До основних термінів відносяться планування, диспетчеризація, призначення, розподіл, відображення, розклад. Термін *розподілу* ресурсів об'єднує дві проблеми: планування і диспетчеризацію. *Планування (scheduling)* - це виділення «віртуальних» ресурсів, необхідних для виконання процесів після їх ініціалізації. До таких ресурсів відносяться процесори, оперативна пам'ять, зовнішня пам'ять, файли і ін. *Диспетчеризація* - це виділення фактичних ресурсів для виконання процесів. Основна відмінність між плануванням і диспетчеризацією полягає в тому, що для будь-якої роботи планування виконується один раз, а диспетчеризація може виконуватися багаторазово. Синонімом поняття виділення ресурсів служить термін *призначення (allocation)*. Часто в літературі планування для КС із масовим паралелізмом *(massively parallel processing)(MPP)* (наприклад, трансп'ютерних) називають *відображенням(mapping)*. У даному випадку ототожнюють два різних поняття: планування і відображення. Термін відображення використовують також при синтезі систолічних структур, який означає розподіл обчислювальних робіт по процесорах і генерацію (створення) міжпроцесорних зв'язків для реалізації заданого обчислювального алгоритму. У цьому випадку відображення означає оптимальне налаштування (пристосування) структури до алгоритму конкретної обчислювальної задачі. При плануванні ж ми вирішуємо задачу оптимального пристосування обчислювального алгоритму до структури КС. У зв'язку з цим сформулюємо різницю між плануванням і відображенням. *Планування* - це розподіл обчислювального алгоритму по процесорах в КС з фіксованою структурою зв'язків, а *відображення* - це розподіл обчислювальних робіт по процесорах реконфігурованої КС (або

встановлення конфігурації КС для оптимального рішення заданого обчислювального алгоритму).

Теоретичною базою для диспетчеризації та планування є *теорія розкладу*. Задача складання розкладу зазвичай ототожнюється з плануванням обчислювальних робіт на ресурси.

4. Класифікація методів і задач планування

Існує статичне і динамічне планування.

Статичне планування проводиться до виконання обчислювальної задачі, отже, воно не є складовою частиною ОС.

Динамічне ж планування проводиться під час виконання завдання і є складовою частиною ОС. Його методи покладені в основу роботи планувальника і диспетчера ОС.

Відомо, що задачі планування відносяться до класу **NP-повних** і точного вирішення в загальному випадку не мають. Для отримання рішення, близького до оптимального результату, необхідний досить складний аналіз параметрів, як обчислювального алгоритму розв'язуваної задачі, так і структури паралельної або розподіленої КС, що вимагає великих часових витрат. При статичному плануванні витрати часу на його роботу не настільки критичні, як при динамічному плануванні. Тому алгоритми статичного планування, як правило, складніші і дають більш якісний результат. Алгоритми ж динамічного планування через жорсткі часові обмеження дають менш якісні результати. Для поліпшення якості динамічного планування використовуються алгоритми балансування навантаження (load balancing), за допомогою яких проводитися перерозподіл завдань по процесорах і тим самим вирівнюється навантаження процесорних елементів КС, що призводить до підвищення ефективності їх роботи та до покращення результатів динамічного планування.

Результати статичного планування доцільно використовувати при вирішенні таких практичних завдань:

- Формування бібліотек готових рішень найбільш часто використовуваних завдань. Такі бібліотеки є автоматизованими засобами паралельного програмування при створенні файлів конфігурації.
- Автоматизована побудова масштабованих комп'ютерних систем мінімальної вартості, що забезпечують виконання прикладних задач за заданий час (високорівневий синтез - High level synthesis).
- Налаштування конфігурації реконфігурованих КС.
- Побудова ефективної комутаційної підсистеми для MPP системи.

- Вибір мінімальної підмножини процесорів масштабованої системи, що забезпечує виконання прикладних задач за мінімальний час.
- Вибір елементної бази для системи із заданою топологією.
- Аналіз різних характеристик роботи МРР і розподілених систем при виконанні прикладних задач (топологічних характеристик, алгоритмів маршрутизації, комутаційних моделей).
- Оптимальне виконання запитів за рахунок ефективного планування для паралельних баз даних.

Останнім часом з'являється напрям, що поєднує особливості як статичного, так і динамічного підходів. Так, для формування пріоритетів процесів можна використовувати методи статичного планування, а при призначенні можна застосувати методи динамічного планування.

Розглянемо класифікацію методів статичного планування (рис. 1). Як зазначалося вище, планування відноситься до класу NP-повних задач. На сьогодні відомо лише чотири окремих випадки точного рішення задачі планування, таких як:

- планування завдання, представленої у вигляді дерева з ідентичними вагами дуг на n-процесорній КС без урахування топології [3];
- планування для двопроцесорної КС [3];
- планування для двошарового конвеєра [3];
- планування задачі, представленої «interval order» графом на n-процесорній КС без урахування топології [3].

Тому в даний час дослідники зосереджують свою увагу на розвитку евристичних субоптимальних алгоритмах планування. Всі евристичні підходи можуть бути розділені на три групи: генетичні, кластерні і спискові.

Генетичні алгоритми доцільно застосовувати у випадках невеликого діапазону рішень задачі планування. Якщо ж існує велика кількість варіантів вирішення задачі планування, генетичні алгоритми працюють не краще, ніж алгоритми випадкового пошуку рішень. При плануванні для МРР систем необхідно враховувати як структуру обчислювальної задачі, так і топологію системи. Крім того, ці системи, як правило, складаються з великого числа процесорів. Це призводить до величезного числа варіантів рішень задачі планування. Тому ми вважаємо, що генетичні підходи не підходять повною мірою для застосування в подібних КС.

Існує два підходи при реалізації кластерних алгоритмів. У першому з них спочатку проводитися розподіл обчислень на повнозв'язну систему без обмеження кількості процесорів, а потім обчислення об'єднуються в кластери, число яких визначається кількістю процесорів в системі. У другому підході спочатку проводиться розподіл обчислень на один процесор при їх послідовній обробці, а потім проводиться перерозподіл обчислень на інші процесори системи, зменшуючи загальний час виконання

обчислювальної задачі. Ці підходи можуть бути використані тільки як процедури для попереднього аналізу якості розпаралелювання обчислювальної задачі. У разі, коли є велике число процесорів КС, існує величезний простір варіантів формування кластерів. У цих умовах отримати якісний результат планування дуже проблематично. Більш того, кластерні алгоритми не враховують топологію КС і тому можуть бути використані лише для повнозв'язних МРР систем.

Рис. 1. Класифікація методів статичного планування

Сенс спискових алгоритмів полягає в наступному. Спочатку формується черга обчислювальних робіт (підзадач), що формують задачу, відповідно до їх пріоритетів. Потім виконується процедура призначення обчислювальних робіт по процесорах. Алгоритми спискового планування відрізняються один від одного способами визначення пріоритетів обчислювальних робіт, а також підходами, що застосовуються процедурою призначення. Остання може бути виконана з урахуванням або без урахування топології КС, а також комунікаційної вартості. Безумовно, кращі результати дають евристичні спискові алгоритми, з яких такі

параметри як топологія системи та час, що витрачається на пересилання між залежними процесами, враховуються повною мірою. Таким чином, ми вважаємо, що спискові методи є найбільш придатними для МРР систем.

Розглянемо класифікацію завдань планування, представлену на рис. 2. Насамперед, усі завдання планування можна розділити на два класи: задачі планування для КС з необмеженим і з обмеженим числом процесорних елементів (ПЕ). Рішення задач, які відносяться до першого класу, становлять інтерес лише для теоретичних досліджень. У цьому випадку може бути вивчена природа обчислювальної задачі: ступінь її розпаралелювання; мінімальний час її виконання; необхідне число процесорів для мінімального виконання завдання та ін.

Практичний інтерес представляють задачі планування, орієнтовані на обмежене число ПЕ. Ці завдання можна розділити на дві групи: завдання без урахування і з урахуванням часу, що витрачається на пересилання даних між послідовними процесами. Рішення перших завдань передбачає призначення на будь-який вільний процесор і тому може представляти інтерес лише для вивчення і аналізу різних підходів формування черг процесів. Завдання ж з урахуванням часу пересилань, в свою чергу, можна розділити на задачі з рівним і різним часом, що витрачається на пересилання даних. Перші з них можуть бути використані при плануванні обчислень в паралельних системах із загальною оперативною пам'яттю (SMP системах) або повнозв'язних системах з роздільною оперативною пам'яттю (МРР системах). Рішення других завдань може використовуватися для більш широкого спектра систем. Ці завдання без урахування топології орієнтовані для планування в SMP і в повнозв'язних МРР системах, а з урахуванням топології - для планування в системах будь-яких топологій. У свою чергу завдання з урахуванням топології можуть орієнтуватися на паралельні МРР системи (однорідні) або розподілені системи (неоднорідні). Крім того, рішення таких задач може мати рішення тільки для конкретної топології системи або мати універсальне рішення для системи з будь-якою топологією.

Як показують дослідження останніх років, найбільш придатними для МРР систем є універсальні, з точки зору топології, евристичні підходи планування, в яких враховуються різні за обсягом і відповідно за часом надсилання дані. Тому наші дослідження присвячені в першу чергу саме таким підходам.

Рис. 2. Класифікація завдань планування

5. Вихідні дані планування і відображення паралельних процесів у КС

У відповідності з теорією розкладу, задача планування вважається заданою, якщо визначені [3],[6],[7]:

1. обчислювальні роботи та їх операції, що підлягають виконанню;
2. кількість і типи машин, що виконують операції;
3. порядок проходження машин (конвеєрний, випадковий, довільний);
4. критерій оцінки планування.

5.1 Обчислювальні роботи та їх операції, що підлягають виконанню

При плануванні обчислювальних робіт у паралельних і розподілених системах, роботи, що підлягають виконанню, та їх операції задаються в залежності від типу планування (динамічне або статичне).

Більшість методів динамічного планування не орієнтовані на аналіз всієї сукупності обчислювальних робіт та їх взаємозв'язків, а враховують лише ті обчислювальні роботи, які готові до виконання в момент планування. Тому в даному випадку задаються множина готових обчислювальних робіт, їх пріоритети, а також такі параметри, як їх трудомісткість і час їх готовності до виконання.

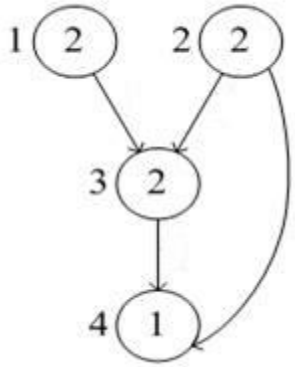
Методи статичного планування орієнтовані на аналіз всієї сукупності обчислювальних робіт та їх взаємозв'язків. Тому обчислювальні роботи в даному випадку задаються у вигляді графів. Залежно від цього, для яких систем проводиться планування, можуть відрізнятися поняття обчислювальних робіт, а також тип графа.

Так, для систем Dataflow конвеєрних КС обчислювальними роботами є команди і відповідні їм операції, а для багатопроцесорних,MPP та розподілених КС обчислювальними роботами є гілки алгоритму і відповідні їм процеси.

Крім цього, на графове представлення обчислювальних робіт, як правило, накладаються обмеження:

- графи розглядають ациклічні (всі циклічні ділянки до процедури планування підлягають розпаралелюванню або векторизації);
- як правило розглядаються інформаційні графи, а не інформаційно-логічні (хоча є способи планування і для цих графів) особливо, коли мова йде про планування для багатопроцесорних КС.

Отже при плануванні для Dataflow і ККС (конвеєрних КС) розглядають помічені, орієнтовані інформаційні графи ЯПФ (ярусно-паралельна форма) виду $G_T = \{T_i\}$, де T_i - множина вершин, наприклад:



Кожна вершина відповідає виконанню якої-небудь операції. Кожна операція, в залежності від типу, має свою вагу, тобто трудомісткість виконання. Дуги, як правило, в таких графах не зважені, тому що такі системи відносяться до КС із спільною оперативною пам'яттю (ОП) і вважається, що навіть якщо обсяг переданих даних відрізняється, йому відповідає одне звернення до ОП. Спрощений варіант цього графа - це граф з рівними вагами вершин (практично такий варіант можна розглядати у випадку виконання даної обчислювальної задачі з використанням RISC архітектури, де основні команди і відповідні їм операції виконуються за один і той же час).

При плануванні для МКМД КС із спільною пам'яттю обчислювальні роботи описуються за допомогою ЯПФ графа, що складається з множини зважених вершин, дуги яких означають інформаційні зв'язки між вершинами. Вершинам такого графа відповідають або обчислювальні завдання, або гілки однієї і тієї ж обчислювальної задачі. Якщо розглядати неоднорідні КС, то для кожної вершини може вказуватись вага для кожного типу ПЕ, і нескінченність в разі неможливості виконання даної вершини на якому-небудь ПЕ.

Для МКМД КС з роздільною (локальною) ОП множина обчислювальних робіт може бути описана 3 способами:

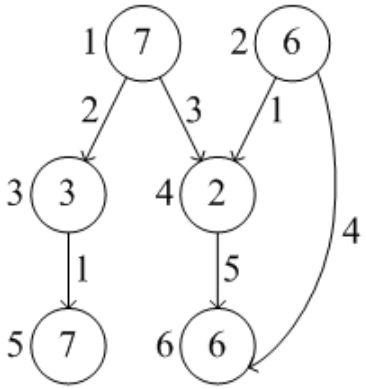
1. за допомогою ациклічно зваженого ЯПФ графа зі зваженими дугами, які визначають обсяг даних, що пересилаються (без урахування відстані між процесорами, між якими ця пересилка проводиться, тобто без урахування топології КС і відповідно маршруту пересилки);

2. за допомогою ієрархії графів. Цей підхід найчастіше використовується для кластерних КС, в яких поєднуються два типи паралелізму, а саме паралелізм незалежних обчислювальних завдань і паралелізм незалежних гілок кожної обчислювальної задачі. У цьому випадку застосовується *2 рівні графів*:

І рівень - граф сукупності обчислювальних задач, який є ациклічно зваженим ЯПФ графом із зваженими дугами. Вершинами цього графа є обчислювальні задачі.

ІІ рівень графів - це графи окремих обчислювальних задач. Кожна обчислювальна задача, в свою чергу, описується ациклічно зваженим орієнтованим ЯПФ графом зі зваженими дугами. Кожній вершині такого графу відповідає обчислення будь-якої гілки алгоритму. Число графів ІІ-го рівня дорівнює кількості типів обчислювальних завдань, що входять до складу графа І-го рівня.

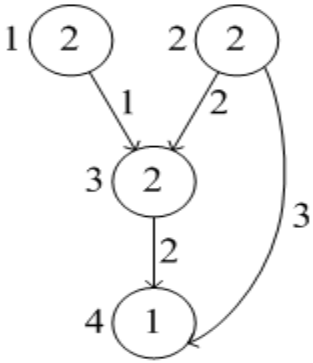
Приклад графа І-го рівня



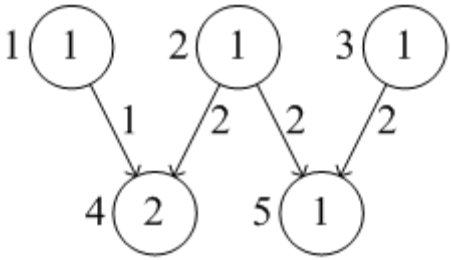
Нехай є 4 типи обчислювальних завдань:
1 і 5 вершини - І тип
2 і 6 вершини - ІІ тип
3 вершина - ІІІ тип
4 вершина - ІV тип

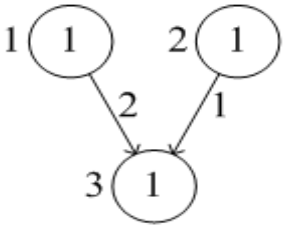
Граф ІІ-го рівня

I тип

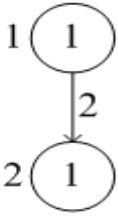


II тип





III тип

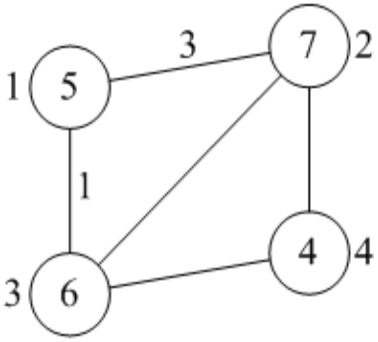


IV тип

Для неоднорідних КС, так як і в попередньому випадку, вказується не одна вага, а вектор ваг для різних типів ПЕ.

3. за допомогою неорієнтованого графа. Цей підхід застосовувався для опису множини процесів при їх виконанні в трансп'ютерних КС.

Кожна вершина відповідає виконанню будь-якого процесу, вага вершини - загальна трудомісткість процесу. Крім цього, для планування додатково необхідна наступна інформація: моменти часу обміну з кожним адресатом і обсяг інформації (після яких тактів обробки відбувається обмін, з якою вершиною, якого обсягу) процесів при їх виконанні в трансп'ютерних КС.



Наприклад, для 1-й вершини відомо, що ваги дуг дорівнюють сумарному обсягу даних, що пересилаються.

2 такта обробки
такт обміну з 2 вершиною з об'ємом 5
1 такт обробки
такт обміну з 3 вершиною з об'ємом 1
2 такта обробки
такт обміну з 2 вершиною з об'ємом 2

Цей граф неорієнтований з тієї причини, що кожен процес може обмінюватися (передавати і приймати інформацію) з іншим процесом кілька разів. З точки зору планування таке подання множини обчислювальних робіт є найбільш складним.

2. Кількість і типи машин, що виконують обчислювальні роботи можуть бути задані різним чином. Це залежить від 3-х властивостей КС:

- технічної характеристики ПЕ;
- однорідність ПЕ;
- тип ОП (загальна, роздільна і розподілена).

Більшість КС загального призначення складаються з однорідних процесорів. У цьому випадку параметр "тип машини" залежить тільки від технічних характеристик ПЕ. Технічними характеристиками ПЕ є:

- продуктивність процесора;
- кількість фізичних каналів пересилання даних (лінків);

- швидкість передачі даних по лінках (кількість одиниць інформації за одиницю часу) і можливість пересилання в різних напрямках (дуплексні і напівдуплексні);
- наявність або відсутність в кожному ПЕ процесора вводу-виводу, що визначає одночасність або розподіл у часі обчислень і пересилань даних.

Для більшості процесорів кількість фізичних каналів даних дорівнює одному. Для трансп'ютерів і деяких спеціалізованих процесорів (наприклад, деяких моделей ADSP) число лінків коливається в діапазоні від 4 до 6.

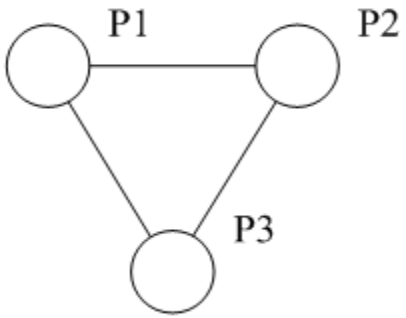
Для неоднорідних КС (в якості типу можуть бути включені обсяг ОП, продуктивність, число КВВ та ін) кожного ПЕ.

Якщо розглядати КС із загальною (спільною або розподіленою) пам'яттю, то вказується лише число ПЕ і їх характеристики, а для неоднорідних - число і тип ПЕ. Для конвеєрних КС вказується кількість шарів.

Як відомо, КС з роздільною ОП можуть бути різних типів:

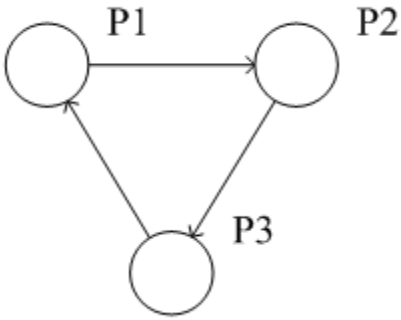
- з постійними (статичними) зв'язками;
- кластерні;
- з динамічними (реконфігурованими) зв'язками ;
- з комбінованими зв'язками.

Для КС з постійними зв'язками топологія КС задається за допомогою графа системи. Як правило, граф КС являє собою неорієнтований граф виду:



Вершинам такого графу відповідають ПЕ КС. Дугам цього графу відповідають двонаправлені зв'язки між ПЕ. Можуть бути зазначені ваги дуг, відповідні пропускній здатності каналів (якщо вони різні), а якщо пропускні здатності всіх каналів ідентичні, то ваги можуть не вказуватись.

Якщо ж у системі однонаправлені зв'язки (для конвеєрів), то граф КС являє собою орієнтований граф виду:



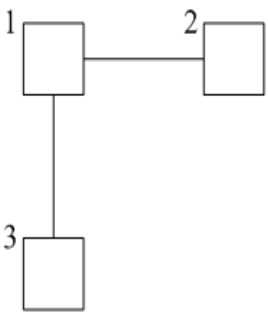
Крім цього, у разі неоднорідності КС, кожна вершина може характеризуватися вагою - характеристикою. КС кластерного типу, за аналогією з ієрархічними графами задач, можуть бути представлені ієрархічними графами систем. Так, наприклад, якщо система складається з двох рівнів кластерів, то вона може бути задана дворівневими графами.

Граф I-го рівня описує взаємозв'язок між кластерами процесорів. Цей граф неорієнтований, його вершинами є кластери.

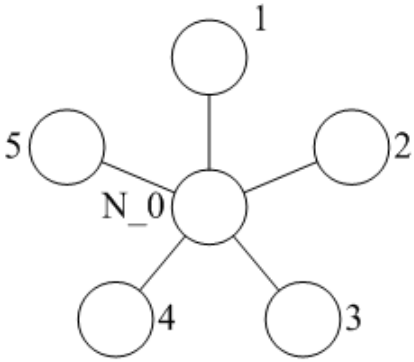
Графи II-го рівня описують взаємозв'язки між процесорами в кластерах КС.

N_0 - позначення спрощень ПЕ, через який відбувається зв'язок з іншими кластерами.

Граф I-го рівня



Граф II-го рівня



Реконфігуровані КС описуються за допомогою повнозв'язних неорієнтованих графів.

3. Порядок проходження машин може бути конвеєрний, випадковий і довільний. Найчастіше використовується 1 і 3 порядки проходження. 1-й порядок використовується для планування робіт у конвеєрних і векторних КС, а 3-й для всіх інших. З точки зору розкладу це дві різних за складністю задачі. Конвеєрний порядок проходження передбачає однаковий і певний порядок для всіх обчислювальних робіт. Конвеєр - це одновходова система. Планування в цьому випадку зводиться до формування черги обчислювальних робіт. Призначення проводиться з цієї черги і кожна обчислювальна робота виконується в конвеєрі ідентично. Визначається лише порядок призначення на конвеєр. Довільний порядок проходження означає мультипроцесорну обробку. У цьому випадку є багатовхідна система і будь-яка обчислювальна робота може бути (у разі однорідності КС) призначена на будь-який процесор. При цьому планування включає не тільки формування черги обчислювальних робіт, а й алгоритм призначення кожної обчислювальної роботи на процесор. Таким чином в цьому випадку планування більш складне, порівняно з конвеєрним порядком.

4. Критерії оцінки планування. Для паралельних і розподілених систем існує **3 основні критерії оптимізації** при рішенні завдань планування, диспетчеризації і відображення:

1. мінімальний час виконання обчислювальних робіт у КС при заданій вартості (тобто із заданими параметрами, головними з яких є число процесорів і структура взаємозв'язків). Цей критерій оптимізації найчастіше використовується для алгоритмів планування обчислень;
2. мінімальна вартість КС, що забезпечує виконання обчислювальних робіт за заданий час. У цьому випадку найбільш поширеним варіантом є ототожнення мінімальної вартості з мінімальним числом процесорів;
3. об'єднання: мінімальний час, якого можна добитися при мінімальній вартості. Цей критерій є найкращим, оскільки не завжди збільшення кількості процесорів у КС призводить до мінімізації часу виконання будь-якого завдання. В той же час досягнення цього критерію оптимізації потребує більш складних алгоритмів.

Крім цих критеріїв можуть бути ще й інші.

Перший критерій використовується при вирішенні задачі оптимального використання ресурсів КС, а другий - при оптимальному рішенні задачі синтезу КС, а також при оптимальному використанні ресурсів КС, що використовує паралелізм незалежних обчислювальних завдань (наприклад, в кластерних КС).

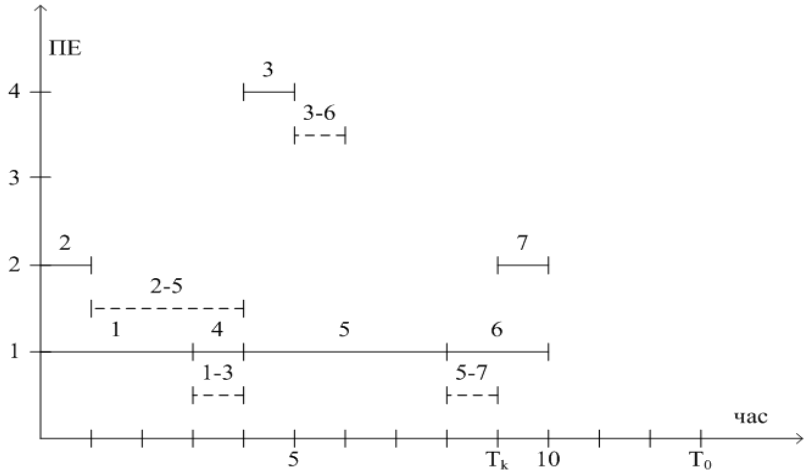
Крім розглянутих вхідних даних для вирішення завдання планування слід враховувати таку характеристику, як співвідношення

розмірності обчислювальної задачі і системи. З урахуванням цієї характеристики існують підходи, що враховують розподіл множини обчислювальних робіт на один процесор (мультипроцесорний режим), розподіл на процесор тільки однієї обчислювальної роботи, (при умові, коли розмірність задачі дорівнює розмірності системи) і комбінований варіант.

Результат планування може бути представлений за допомогою діаграми Ганта. Діаграма Ганта представляє собою розклад роботи всіх процесорів системи і для кожного з них потактовий порядок виконання всіх завдань, призначених на даний процесор. Однак традиційна діаграма Ганта містить не всю інформацію про дані, що пересилаються з одного процесора на інший. Нами запропонована модифікована діаграма Ганта [7], в якій розклад роботи кожного процесора включає не тільки чергу виконуваних задач, а й порядок пересланих даних всіх його каналів. Приклад таких діаграм представлений на рис. 3.

час	ПЕ1	ПЕ2	ПЕ3	ПЕ4
1	1	2		
2	1	2-5		
3	1	2-5		
4	4	2-5		
5	5			3
6	5			3-6
7	5			
8	5			
9	6			
10	6	7		

(а)



(б)

Рис. 3. Приклади результатів планування: традиційна діаграма Ганта (а) і модифікована діаграма Ганта (б)

5.2 Етапи планування

Етапи планування залежать від того, який задано порядок проходження машин, тобто конвеєрний або довільний. При конвеєрному порядку (для конвеєрних і векторних КС) планування зводиться до формування черги готових обчислювальних робіт (команд). При довільному порядку (для паралельних та розподілених КС) будемо розглядати процедуру планування, що складається з двох етапів:

- формування черги готових обчислювальних робіт;
- призначення обчислювальних робіт на процесори.

У процедурах планування реалізація цих етапів відрізняється один від одного. Крім цього існує два основних підходи виконання цих етапів: послідовний і комплексний. Перший підхід простіше в реалізації, але призводить до менш ефективного результату, а другий - більш складний, але ближче до оптимального результату.

Далі будемо розглядати особливості реалізації даних етапів для динамічного планування, а також для статичного планування КС різних типів.

При динамічному плануванні формування черги обчислювальних робіт проводиться на підставі наступних параметрів: заданих пріоритетів, за часом готовності до виконання, в порядку зменшення часу виконання, в порядку зростання часу, довільно. Реалізація такого алгоритму не вимагає великих часових витрат і не відрізняється складністю. Однак таке формування черги не сприяє отриманню близького до оптимального результату планування (мінімальний час виконання обчислювальних задач). Існують дані статичних досліджень різних способів формування черг, відповідно до яких перераховані способи значно поступаються більш складним способам формування черг, які використовуються у статичному плануванні. Перерахуємо результати в порядку зменшення ефективності (з перерахованих вище, крім пріоритетного):

- довільний вибір (випадковим чином);
- у порядку зменшення часу виконання;
- у порядку зростання часу виконання;
- у порядку зростання часу готовності до виконання.

Однак ці дослідження проводилися не для всіх типів КС. Тому таке дослідження є цікавим і актуальним для всіх типів паралельних та розподілених КС.

Призначення обчислювальних робіт на процесори при динамічному плануванні може здійснюється двома способами: централізовано і децентралізовано.

Відомо, що задачі планування відносяться до NP-повних. Точного рішення в загальному випадку немає. Для отримання більш точного (близького до оптимального) результату необхідний досить складний аналіз параметрів обчислювального алгоритму і структури КС, що вимагає великих витрат часу. При статичному плануванні витрати часу на роботу алгоритму не настільки критичні, як при динамічному. Тому алгоритми статичного планування як правило більш складні і дають більш якісний результат. Алгоритми ж динамічного планування через часові обмеження дають менш якісні результати. Для поліпшення якості використовують алгоритми балансування навантаження, за допомогою яких вирівнюється навантаження ПЕ КС, що підвищує ефективність роботи КС і призводить до поліпшення результатів планування. Останнім часом намітився напрям комбінованого підходу до планування, використовуючи для визначення пріоритетів процесів і формування черги процесів, а призначення довільними методами динамічного планування.

Існує *два способи динамічної диспетчеризації* та планування: централізований і децентралізований. У першому випадку спеціальний керуючий процесор вирішує ці завдання. При цьому існує більш повне охоплення станів усіх засобів КС, що дає більш точні результати. Але цей підхід ненадійний і неживучий. Децентралізована диспетчеризація

передбачає можливість самостійного звернення кожного процесора до загальної черги процесів. Звернення проводиться в моменти звільнення. Цей спосіб дає більш високу надійність, проте дає менш якісні результати планування.

Основна мета призначення - забезпечення мінімізації пересилань. При динамічному плануванні призначення здійснюється, як правило, довільним чином на звільнений ПЕ. Це не завжди призведе до забезпечення мінімізації пересилань. Однак обмежений час роботи алгоритму призначення не дає можливості ускладнювати його.

6. Методи статичного планування для паралельних і розподілених КС.

6.1 Характеристики графа завдань.

Основними характеристиками графів завдань є:

- критичний шлях (по числу вершин) графа;
- критичний шлях (за часом) графа;
- критичний шлях і-ї вершини (по числу вершин) до початку графа;
- критичний шлях і-ї вершини (за часом) до початку графа;
- критичний шлях і-ї вершини (по числу вершин) до кінця графа;
- критичний шлях і-ї вершини (за часом) до кінця графа;
- число процесорів (теоретично мінімальне і критичне число процесорів);
- ранній термін виконання вершин;
- пізній термін виконання вершин;
- трудомісткість (вага) вершин графа;
- число зв'язків (зв'язність);
- число приймачів (кількість вихідних дуг);
- максимальний і мінімальний обсяг пересилань (теоретично мінімальний обсяг пересилань дорівнює нулю, а максимальний обсяг сумі пересилань, коли всі вершини графу задач виконуються на різних ПЕ).

Перераховані характеристики в основному враховуються при реалізації першого етапу планування, а остання характеристика враховується для другого етапу планування.

Характеристика мінімального та критичного числа ПЕ враховується в тому випадку, коли критерій оптимізації – мінімальна вартість КС при заданому часі виконання задачі. У цьому випадку перш за

все визначають критичний шлях графа ($t_{кр}$), який порівнюють із заданим часом ($t_{zad.}$). Якщо заданий час менше за критичний , тоді задача не має рішення. В іншому випадку визначаємо теоретично мінімальне значення кількості ПЕ, що забезпечують виконання алгоритму за $t_{zad.}$, яке не враховує пересилання між ПЕ, за такою формулою $\left\lceil \sum_{i=1}^{n} t_i \Big/ t_{zad.} \right\rceil$. Часто критичну кількість ПЕ визначають як ширину графа, однак це має сенс тільки для синхронних КС. Крім цього, щоб правильно визначити максимальну ширину графа, слід «вирівняти» ширину кожного ярусу шляхом переміщення транзитних вершин і тільки тоді визначати ширину графа. Для асинхронних КС ширина графа ні про що не свідчить. Тому, щоб знайти мінімальне число ПЕ при $t_{zad.}$ для асинхронних КС єдиним способом є перебір числа ПЕ, починаючи з мінімальної кількості ПЕ, що визначається за вищенаведеною формулою.

Залежність часу виконання завдання (t) від числа ПЕ (n) можливо трьома способами, наведеними на рис.4

Якщо крива залежності відповідає I варіанту, це означає, що для виконання завдання не має сенсу використовувати паралельну КС. Якщо крива залежності відповідає II варіанту, це означає, що її мінімальна точка відповідає мінімальному числу ПЕ. Якщо ж крива залежності відповідає III варіанту, то можна знайти її мінімальну точку при мінімальній кількості ПЕ

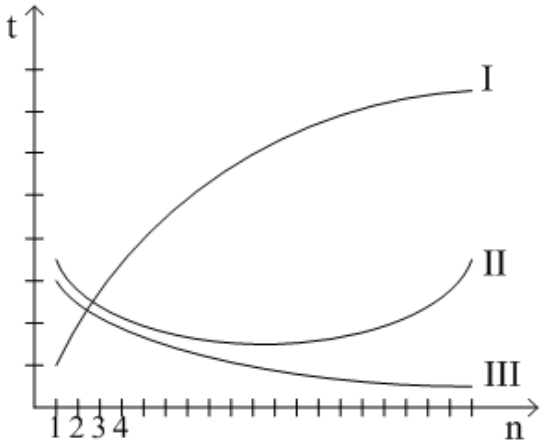
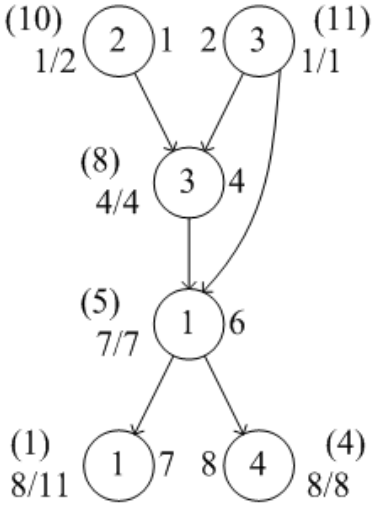


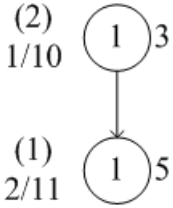
Рис.4 Варіанти залежності часу виконання завдання від числа ПЕ у КС

Для $n=n_{min}$ слід виконати планування за критерієм t_{min}. Потім порівняти отриманий час із $t_{zad.}$, якщо $t>t_{zad.}$, то n збільшується і виконується знову планування. Якщо отримане $t_2< t_1$, то пошук продовжується, а якщо ні, то пошук припиняється.

Таким самим чином можна вирішити двохкритеріальну задачу, а саме, знайти КС з min числом ПЕ, яка виконує завдання за t_{min}. Алгоритм той же, що й вище, тільки при переборі отриманий час t потрібно порівнювати з t_{min}, а не з $t_{зад.}$. Графік I може бути отриманий при сильній зв'язності елементів алгоритму і при низькоефективному алгоритмі планування.

Приклад визначення ранніх і пізніх термінів виконання.





Для визначення раннього терміну виконання і-ої вершини будемо використовувати наступну формулу:

$T_{p.cp_i} = T_{кр.n_i} + 1$, де $T_{кр.n_i}$ - критичний шлях по часу і-ої вершини до початку графа задачі. Наприклад $T_{p.cp_1} = 1$, $T_{p.cp_7} = 8$.

Для визначення пізнього терміну виконання і-ої вершини будемо використовувати наступну формулу:

$T_{n.cp_i} = (T_{кр.гр} - T_{кр.к_i}) + 1$, де $T_{кр.гр}$ - критичний шлях графу по часу; $T_{кр.к_i}$ - критичний шлях по часу і-ої вершини до кінця графа задачі.

Для даного графу $T_{кр.гр} = 11$. $T_{n.cp_1} = 2$, $T_{n.cp_7} = 11$.

Характеристиками графу систем є число ПЕ, діаметр, середній діаметр КС і відстань між ПЕ. Формально КС з одним і тим же числом ПЕ відрізняється діаметром і відстанню між ПЕ. Ці характеристики впливають на час пересилань.

Реальний час пересилань між двома процесами визначається як добуток обсягу пересилань (по графу задач) і відстані між ПЕ, на які призначені процеси, у графі системи. Таким чином характеристики графу системи повинні бути враховані при реалізації II етапу планування для забезпечення мінімального часу пересилань. Слід зазначити, що для КС із

загальною (що розділяється) пам'яттю відстань і діаметр між ПЕ одні й ті ж, як у повнозв'язних КС (*d=1*).

6.2 Методи формування черг готових процесів для MIMD систем.

Відома досить велика кількість евристичних алгоритмів формування черг готових процесів (17 з них запропоновані для виконання лабораторних робіт). У кожному з них аналізується одна або кілька перерахованих характеристик графа завдання. Частина з них могла бути реалізована для статичного (тому використовуються характеристики алгоритму, структура якого відома заздалегідь), а частина динамічного або статичного планування.

Крім цього всі ці алгоритми розрізняються по складності, а також по ефективності. Аналітичне порівняння ефективності алгоритмів досить складне. Тому були проведені статистичні дослідження ефективності цих алгоритмів при їх роботі на MIMD системах із спільною ОП. Для MIMD систем з роздільною ОП таких даних немає, оскільки отримати їх також досить важко (для різних топологій ефективність одних і тих самих алгоритмів формування черг може бути різною).

У наступній таблиці 2 наведено методи формування черг готових процесівув порядку спадання їх ефективності на MIMD систем із спільною ОП.

Таблиця 2

Ефективність	Складність	Метод формування черг	Використані характеристики графа	Тип планування
1	2	3	4	5

1	4	У порядку спадання пронормованої суми критичних по t (часу) і за кількістю вершин шляхів до кінця графа задачі	Критичний шлях по часу і кількості вершин	статичне
2	4	У порядку зростання різниці між пізнім та раннім строками виконання вершин графу задачі	Критичний шлях по часу і кількості вершин.	статичне
3	3	У порядку спадання критичного по часу шляхів до кінця графа завдання	Критичний шлях і-х вершин по часу	статичне
4	4	У порядку спадання критичного по числу вершин шляхів до кінця завдання, а при однакових значеннях в порядку спадання зв'язності вершин	Критичний шлях за кількістю вершин від заданих (зв'язність)	статичне

5	4	У першу чергу вершини, що знаходяться на критичному шляху графа (за кількістю вершин), а інші в порядку спадання критичного по числу вершин шляхів до кінця графа завдань	Критичний шлях графа Критичні шляхи і-х готових процесів.	статичне статичне
6	3	У порядку спадання критичних за кількістю вершин шляхів до кінця графа завдання	Критичний шлях за кількістю вершин	статичне
7	4	У порядку зростання критичного за кількістю вершин шляхів до початку графа, а при однакових	Критичний шлях по кількості вершин та їх зв'язність	статичне

		значеннях в порядку спадання зв'язності		
8	4	У порядку зростання критичного шляху за кількістю вершин шляхів до початку графа, а при однакових значеннях в порядку спадання часу виконання (ваги вершин)	Критичний шлях по кількості вершин та їх вага	статичне
9	3	У порядку зростання критичного за кількістю вершин шляхів до початку графа завдань	Критичний шлях по кількості вершин	статичне
10	4	У порядку спадання зв'язності	Критичний шлях по кількості вершин та їх зв'язність	статичне
11	4	У порядку спадання зв'язності вершин, а при рівних значеннях в порядку зростання критичного за кількістю вершин шляхів до початку графа завдання	Критичний шлях по кількості вершин та їх зв'язність	статичне
12	2	У порядку спадання числа вихідних дуг (приймачів)	Кількість вихідних дуг вершин	статичне
13	1	Випадковим чином		статичне, динамічне

14	2	У порядку спадання часу виконання вершин (ваги вершин)	Вага вершин	статичне, динамічне
15	2	У порядку зростання часу виконання вершин (ваги вершин)	Вага вершин	статичне, динамічне
16	3	У порядку зростання критичних по часу шляхів до початку графа завдання	Критичні по часу шляхи вершин від початку графа задачі.	статичне
17	2	У порядку зростання часу готовності		статичне, динамічне

У другій колонці вказані оцінки рівня складності (чим більша цифра, тим складніший алгоритм). Якщо перед виконанням користувацької задачі відома його структура (граф завдання), то в принципі будь-який з алгоритмів може бути використаний для динамічного планування. У цьому випадку все залежить від того, чи не призведе використання того чи іншого алгоритму до порушення часових обмежень на роботу диспетчера.

Якщо ж до вирішення завдань їх взаємозв'язок наперед невідомий, тобто граф задач відсутній, то для динамічного планування можуть бути використані тільки ті методи, які вказані в останній колонці таблиці. Як говорилося вище, на практиці використовуються 13, 14, 15 і 17 методи.

Вхідними даними для процесу формування черги є ациклічний орієнтований граф задачі з заданими вагами вершин (трудомісткістю підзадач).

Алгоритми формування черги не враховують вагу дуг. Ці характеристики використовуються для алгоритмів призначення.

Результатом формування черги є послідовність вершин з характеристиками, за якими відбулось сортування.

Графи мають наступні характеристики:

- критичний шлях графу - максимальна сума ваг вершин, що належать шляху від початку до кінця графу;
- критичний шлях по часу від заданої вершини до кінця графу задачі (максимальна сума ваг вершин, що належать шляху від заданої вершини до кінця графу);
- критичний шлях по часу від заданої вершини до початку графу задачі (максимальна сума ваг вершин, що належать шляху від початку графу задачі до даної вершини);
- кількість вершин критичного шляху графу (максимальна кількість вершин, що належать шляху від початку до кінця граф);
- кількість вершин критичного шляху по часу від заданої вершини до кінця графу задачі (максимальна кількість вершин, що належать шляху від заданої вершини до кінця графу, враховуючи задану вершину);
- кількість вершин критичного шляху по часу від даної вершини до початку графу задачі (максимальна кількість вершин, що належать шляху від початку графу задачі до даної вершини, не враховуючи задану вершину);
- ранній строк -мінімальний момент часу, коли вершина може починати виконання, тобто для неї сформовані вхідні дані;
- пізній строк - момент часу, пізніше якого виконання задачі за мінімальний час стає неможливим;
- зв'язність вершин - сума кількості вхідних і вихідних дуг вершини;
- вага вершин (час виконання);
- кількість вхідних і вихідних дуг.

Дано граф задач, який зображений на рис. 5.

Розглянемо наступні приклади формування черги для заданого графу задач (рис.5).

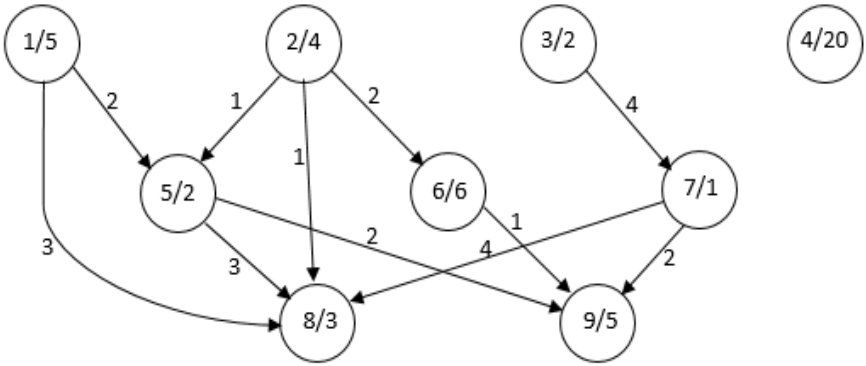


Рис.5 Приклад графу завдання

1) У порядку спадання пронормованої суми критичних по часу і по кількості вершин шляхів до кінця графа задачі.

У даному випадку використаними характеристиками будуть критичний шлях графа та вершин по часу.

Для розрахунку пронормованої суми для кожної вершини використаємо формулу: $Pr_i = \frac{T_{\text{кр}_i\text{к}}}{T_{\text{кр гр}}} + \frac{N_{\text{кр}_i\text{к}}}{N_{\text{кр гр}}}$

, де

$T_{\text{кр}_i\text{к}}$ - критичний шлях по часу від заданої вершини до кінця графу задачі,

$T_{\text{кр гр}}$- критичний шлях графу по часу,

$N_{\text{кр}_i\text{к}}$ - критичний шлях по кількості вершин від заданої вершини до кінця графу задачі,

$N_{\text{кр гр}}$-- критичний шлях графу по кількості вершин

Проведемо розрахунки за формулою та запишемо їх до таблиці.

Вершина	$T_{\text{кр}_i\text{к}}$	$N_{\text{кр}_i\text{к}}$	Pr_i
1	12	3	**1,6**
2	15	3	**1,75**
3	8	3	**1,4**
4	20	1	**1,33**
5	7	2	**1,02**
6	11	2	**1,22**
7	6	2	**0,97**
8	3	1	**0,48**
9	5	1	**0,58**

$$T_{\text{кр гр}} = max(T_{\text{кр}_i\text{к}}) = 20$$
$$N_{\text{кр гр}} = \left(N_{\text{кр}_i\text{к}}\right) = 3$$

Згідно обрахунків, отримуємо результат, відсортувавши вершини за спаданням пронормованої суми критичних по часу і по кількості вершин шляхів до кінця графа задачі.

Результат у форматі «номер вершини (використані характеристики графу)»: **2(1,75), 1(1,6), 3(1,4), 4(1,33), 6(1,22), 5(1,02), 7(0,98), 9(0,58), 8(0,48)**

2) У порядку зростання різниці між пізнім та раннім строками виконання вершин графу задачі

Для обчислення раннього і пізнього строку кожної вершини скористаємось формулами:

$$\text{Р}_{\text{ср}i} = T_{\text{кр}_i\text{поч}} + 1$$

де $T_{\text{кр}_i\text{поч}}$ - критичний шлях по часу від початку графу задачі до заданої вершини

$$\text{П}_{\text{ср}i} = (T_{\text{кр гр}} - T_{\text{кр}_i\text{к}}) + 1$$

Вершина	$T_{\text{кр}_i\text{к}}$	$T_{\text{кр}_i\text{поч}}$	$\text{Р}_{\text{ср}i}$	$\text{П}_{\text{ср}i}$	$\text{П}_{\text{ср}i} - \text{Р}_{\text{ср}i}$
1	12	0	1	9	**8**
2	15	0	1	6	**5**
3	8	0	1	13	**12**
4	20	0	1	1	**0**
5	7	5	6	14	**8**
6	11	4	5	10	**5**
7	6	2	3	15	**12**
8	3	7	8	18	**10**
9	5	10	11	16	**5**

$$T_{\text{кр гр}} = 20$$

Згідно підрахунків, отримуємо результат, відсортувавши вершини за зростанням різниці між пізнім і раннім строками виконання вершин графу задачі.

Результат у форматі «номер вершини (використані характеристики графу)»: **4(0), 2(5), 6(5), 9(5), 1(8), 5(8), 8(10), 3(12), 7(12)**

3) У порядку спадання критичного по часу шляхів до кінця графа задачі.

У даному випадку використовуємо критичний шлях вершин по часу до кінця графа задачі ($T_{\text{кр}_i\text{к}}$):

Вершина	$T_{\text{кр}_i\text{к}}$
1	**12**
2	**15**
3	**8**
4	**20**
5	**7**
6	**11**
7	**6**
8	**3**
9	**5**

Згідно підрахунків, отримуємо результат, відсортувавши вершини за спаданням критичних по часу шляхів до кінця графу задачі.

Результат у форматі «номер вершини (використані характеристики графу)»: **4(20), 2(15), 1(12), 6(11), 3(8), 5(7), 7(6), 9(5), 8(3)**

4) У порядку спадання критичного по кількості вершин шляхів до кінця графа задачі, а при рівних значеннях – в порядку спадання зв'язності вершин.

У даному випадку використаними характеристиками будуть:

$T_{\text{кр}_i\text{к}}$ - критичний шлях по кількості вершин від заданої вершини до кінця графу задачі та S_{v_i} - зв'язність вершини (сума вхідних і вихідних дуг вершини). Проведемо розрахунки та запишемо їх до таблиці.

Вершина	$T_{\text{кр}_{i\text{к}}}$	S_{v_i}
1	**3**	**2**
2	**3**	**3**
3	**3**	**1**
4	**1**	**0**
5	**2**	**4**
6	**2**	**2**
7	**2**	**3**
8	**1**	**4**
9	**1**	**3**

Спочатку сортуємо вершини графу по спаданню критичного по кількості вершин шляхів до кінця графа задачі ($N_{\text{кр}_{i\text{к}}}$), а потім в порядку спадання зв'язності вершин.

Результат у форматі «номер вершини (використані характеристики графу)»: **2(3,3), 1(3,2), 3(3,1), 5(2,4), 7(2,3), 6(2,2), 8(1,4), 9(1,3), 4(1,0)**

5) У першу чергу вершини, що знаходяться на критичному шляху графа (по кількості вершин), а інші – в порядку спадання критичного по числу вершин шляхів до кінця графа задач.

Спочатку знаходимо критичний шлях графу. У нашому випадку їх є 4: **1,5,8**; **2,5,8**; **2,6,9**; **3,7,9**. Якщо критичних шляхів декілька, то обираємо будь-який із існуючих, наприклад, **1,5,8.** Знаходимо кількість вершин критичного шляху по часу від заданої вершини до кінця графу задачі ($N_{\text{кр}_{i\text{к}}}$):

Вершина	$N_{\text{кр}_i\text{к}}$
1	**3**
2	**3**
3	**3**
4	**1**
5	**2**
6	**2**
7	**2**
8	**1**
9	**1**

На першому місці будуть вершини, що знаходяться на критичному шляху. Далі вершини сортуємо в порядку спадання критичного по числу вершин шляхів до кінця графа задач.

Результат у форматі «номер вершини (використані характеристики графа)»: **1(3), 5(2), 8(1), 2(3), 3(3), 6(2), 7(2), 4(1), 9(1)**

6) У порядку спадання критичних по кількості вершин шляхів до кінця графа задачі.

Сортуємо вершини по значенню $N_{\text{кр}_i\text{к}}$

Вершина	$N_{\text{кр}_i\text{к}}$
1	**3**
2	**3**
3	**3**
4	**1**
5	**2**
6	**2**
7	**2**
8	**1**
9	**1**

Результат у форматі «номер вершини (використані характеристики графу)»: **1(3), 2(3), 3(3), 5(2), 6(2), 7(2), 4(1), 8(1), 9(1)**

7) В порядку зростання критичного по кількості вершин шляхів від початку графа, а при рівних значеннях у порядку спадання зв'язності

У даному випадку будемо використовувати наступні характеристики: $N_{\text{кр}_i\text{п}}$- критичний шлях по кількості вершин від початку графу задачі до заданої вершини (максимальна кількість вершин, що передують заданій);

S_{v_i} - зв'язність вершини

Запишемо ці характеристики до таблиці:

Вершина	$N_{\text{кр}_i\text{п}}$	S_{v_i}
1	**0**	**2**
2	**0**	**3**
3	**0**	**1**
4	**0**	**0**
5	**1**	**4**
6	**1**	**2**
7	**1**	**3**
8	**2**	**4**
9	**2**	**3**

Спочатку сортуємо вершини графу по зростанню критичного по кількості вершин шляхів до початку графа задачі ($N_{\text{кр}_i\text{п}}$), а потім в порядку спадання зв'язності вершин.

Результат у форматі «номер вершини (використані характеристики графу)»: **2(0,3), 1(0,2), 3(0,1), 4(0,0), 5(1,4), 7(1,3), 6(1,2), 8(2,4), 9(2,3)**

8) У порядку зростання критичного шляху по кількості вершин від початку графа, а при рівних значеннях – в порядку спадання ваги вершин

Вершина	$N_{\text{кр}_i\text{п}}$	Вага веришини
1	**0**	**5**
2	**0**	**4**
3	**0**	**2**
4	**0**	**20**
5	**1**	**2**
6	**1**	**6**
7	**1**	**1**
8	**2**	**3**
9	**2**	**5**

Спочатку сортуємо вершини графу по зростанню критичного по кількості вершин шляхів до початку графа задачі ($N_{\text{кр}_i\text{п}}$), а потім в порядку спадання ваги вершин.

Результат у форматі «номер вершини (використані характеристики графа)»: **4(0,20), 1(0,5), 2(0,4), 3(0,2),6(1,6), 5(1,2), 7(1,1), 9(2,5), 8(2,3)**

9) У порядку зростання по кількості вершин критичних шляхів від початку графа задачі. Сортуємо по $N_{\text{кр}_i\text{п}}$

Вершина	$N_{\text{кр}_i\text{п}}$
1	**0**
2	**0**
3	**0**
4	**0**
5	**1**
6	**1**
7	**1**
8	**2**

9	**2**

Результат у форматі «номер вершини (використані характеристики графу)»: **1(0), 2(0), 3(0), 4(0), 5(1), 6(1), 7(1), 8(2), 9(2)**

10) У порядку спадання зв'язності, а при рівних значеннях – в порядку спадання критичного по кількості вершин шляхів до кінця графа.

Сортуємо в порядку спадання зв'язності S_{v_i}, якщо перший показник однаковий, то беремо до уваги $N_{\text{кр}_i\text{к}}$ у порядку спадання:

Вершина	S_{v_i}	$N_{\text{кр}_i\text{к}}$
5	**4**	**2**
8	**4**	**1**
2	**3**	**3**
7	**3**	**2**
9	**3**	**1**
1	**2**	**3**
6	**2**	**2**
3	**1**	**3**
4	**0**	**1**

Результат у форматі «номер вершини (використані характеристики графу)»: **5(4;2), 8(4;1), 2(3;3), 7(3;2), 9(3;1), 1(2;3),6(2;2), 3(1;3), 4(0;1)**

11) У порядку спадання зв'язності вершин, а при однакових значеннях – в порядку зростання критичного по кількості вершин шляхів від початку графа задачі.

Сортуємо в порядку спадання зв'язності S_{v_i}, якщо перший показник однаковий, то беремо до уваги $N_{\text{кр}_i\text{п}}$ у порядку зростання:

Вершина	S_{v_i}	$N_{\text{кр}_i\text{п}}$
5	**4**	**1**
8	**4**	**2**
2	**3**	**0**
7	**3**	**1**
9	**3**	**2**

1	**2**	**0**
6	**2**	**1**
3	**1**	**0**
4	**0**	**0**

Результат у форматі «номер вершини (використані характеристики графу)»: **5(4;1), 8(4;2), 2(3;0), 7(3;1), 9(3;2), 1(2;0),6(2;1), 3(1;0), 4(0;0)**

12) У порядку спадання кількості вихідних дуг вершин:

Результат у форматі «номер вершини (використані характеристики графа)»: **2(3), 1(2), 5(2), 7(2), 3(1), 6(1),4(0), 8(0), 9(0)**

13) Випадковим чином.

Для формування черги застосовується метод random, використовуючи як параметр кількість вершин графу.

14) У порядку спадання ваги вершин

Результат у форматі «номер вершини (використані характеристики графу)»: **4(20), 6(6), 1(5), 9(5), 2(4), 8(3),3(2), 5(2), 7(1)**

15) У порядку зростання ваги вершин

Результат у форматі «номер вершини (використані характеристики графу)»: **7(1), 5(2), 3(2), 8(3), 2(4), 9(5),1(5), 6(6), 4(20)**

16) У порядку зростання критичного по часу шляхів вершин від початку графу задачі. Сортуємо по $\boldsymbol{T}_{\text{кр}\,i^{\text{п}}}$ по зростанню

Вершина	$\boldsymbol{T}_{\text{кр}\,i^{\text{п}}}$
1	**0**
2	**0**
3	**0**
4	**0**
7	**2**
6	**4**
5	**5**
8	**7**

9	**10**

Результат у форматі «номер вершини (використані характеристики графу)»: **1(0), 2(0), 3(0), 4(0), 7(2), 6(4), 5(5), 8(7), 9(10)**

17) У порядку зростання часу готовності.

Цей метод використовується під час моделювання. Даний алгоритм є псевдо динамічним. На початку формування черги обираються всі вершини першого ярусу. Після їх виконання 'активізуються' вершини з другого ярусу, і так далі.

6.3 Призначення обчислювальних робіт на процесори

Насамперед, визначимо, яким чином впливає спосіб призначення на процесори на час виконання задачі. Відомо, що час виконання алгоритму є сумою двох складових: часу, що витрачається власне на обчислення і часу, що витрачається на пересилання даних. Спосіб призначення впливає на значення другої складової. З точки зору вирішення задачі призначення будемо розрізняти алгоритми, орієнтовані на системи з розподіленою (загальною) пам'яттю і алгоритми, орієнтовані на системи з роздільною пам'яттю.

Розглянемо проблему призначення на системи з розподіленою пам'яттю. У даному випадку мінімізація пересилань може бути забезпечена за рахунок мінімізації числа звернень до пам'яті, що розділяється (за рахунок використання кеш-пам'яті в кожному процесорі), а також мінімізація числа конфліктів до розподіленої пам'яті.

Розглянемо проблему мінімізації числа конфліктів.

Існує теорема відповідно до якої, при виникненні конфлікту слід віддавати перевагу тій обчислювальної роботі (процесу), яка рідше вступає в конфлікт з іншими процесами. Реалізація цієї теореми заснована на припущенні про наявність достовірної статистики звернень процесів до ресурсів. Однак таку статистику, як правило, важко отримати. Тому існує підхід, який дає субоптимальний результат. Для такого алгоритму використовують таблиці, в яких розташовуються імена конфліктуючих (блокованих) процесів по кожному з ресурсів. Алгоритм мінімізації алгоритмів розглядається окремо для кожного з ресурсів.

Зміна таблиці блокованих процесів відбувається в ті моменти часу, коли ресурс виділяється якомусь процесору. Можливі дві ситуації: в момент часу ресурс потрібен одному процесу або множині процесів.

У першій ситуації і-й процес, якому виділений ресурс, стає в таблиці на перше місце, а інші процеси з першого по (і-1)-й зсуваються на одну позицію вниз.

У другій ситуації, коли в один і той самий момент часу ресурс потрібен множині процесів, ресурс виділяється тому з них, чиє ім'я в таблиці стоїть останнім. Після цього ім'я даного процесора стає в таблиці на перше місце.

Приклади:

Нехай початковий стан таблиці $T = \{x_1, x_2, x_3, x_7, x_9, x_{10}\}$ де x_i – ім'я i-го процеса.

1. В t-й момент часу ресурс необхідний процесу x_7. Тоді, після надання даному процесу ресурсу, стан таблиці має вигляд:

$$T = \{x_7, x_1, x_2, x_3, x_9, x_{10}\}$$

2. В t-й момент ресурс необхідний множині процесів $\{x_2, x_7, x_9\}$. Ресурс буде виділений x_9, а стан таблиці буде сформовано наступним чином:

$$T = \{x_9, x_1, x_2, x_3, x_7, x_{10}\}$$

Цей алгоритм був статистично випробуваний і порівнювався з призначенням випадковим чином. Результати позитивні, тобто застосування описаного алгоритму дозволяє зменшити число конфліктів і тим самим зменшити час виконання обчислювальних алгоритмів у КС із загальною або розподіленою пам'яттю. Для розподіленої пам'яті формується n таблиць блокованих процесів, де n - кількість банків роздподіленої пам'яті.

6.3.1 Алгоритми призначення для КС зі спільною (розподіленою) пам'яттю

1. Алгоритм випадкового призначення (на будь-який вільний процесор).

2. Призначення на перший звільнений процесор. У цьому випадку, при призначенні, з вільних процесорів обирається той, який звільнився і простоює довше інших. Такий підхід дозволяє збалансувати завантаження всіх процесорів системи. Проілюструємо роботу цього алгоритму на прикладі. Нехай задано граф задач, представлений на рис.6, припустимо, що ми маємо наступну чергу 1(1), 2(2), 3(2), 4(1), 5(1) та 3 ПЕ. Результат роботи цього алгоритму призначення продемонстровано на рис.7.

3. Алгоритм «сусіднього» призначення. У відповідності з даним алгоритмом обчислювальна робота призначається на той процесор, в пам'яті якого міститься максимальний обсяг вихідних даних. Для реалізації такого алгоритму необхідна наявність швидкої локальної пам'яті великого обсягу в процесорах системи. Алгоритм «сусіднього» призначення дозволяє мінімізувати час пересилань за рахунок мінімізації звернення до

розподіленої оперативної пам'яті і зменшення кількості конфліктів. Крім цього, час пересилань залежить від наявності автономних контролерів або процесорів вводу-виводу, а також застосовуваної комунікаційної моделі. Наявність автономних контролерів дозволяє процесору паралельно виконувати обробку інформації і обмін даними з іншими процесорами через оперативну пам'ять. З точки зору комунікаційної моделі будемо розглядати два варіанти:

- пересилання даних для обчислювальної роботи починає здійснюватися тільки після формування всіх її вихідних даних;
- пересилання даних для обчислювальної роботи здійснюється по міру готовності її вихідних даних асинхронно «з випередженням».

Виходячи з вищевикладеного, перерахуємо наступні варіанти алгоритмів «сусіднього» призначення:

a) «сусіднє» призначення для систем без автономних контролерів вводу-виводу із застосуванням першого варіанту комунікаційної моделі (пересилання без «випередження»);

b) «сусіднє» призначення для систем з автономними контролерами введення-виведення із застосуванням першого варіанту комунікаційної моделі;

c) «сусіднє» призначення для систем з автономними контролерами введення-виведення із застосуванням другого варіанту комунікаційної моделі (пересилання з «випередженням»).

Проілюструємо роботу цих алгоритмів на прикладі графу задачі, представленого на рис. 6. Маємо наступну чергу - 1(1), 2(2), 3(2), 4(1), 5(1) та три ПЕ. Результати роботи перерахованих варіантів «сусіднього» призначення представлені відповідно на рис. 8-10.

4. Алгоритм «сусіднього» призначення з урахуванням зайнятих процесорів. Даний алгоритм відрізняється від попереднього тим, що при призначенні обирається той процесор, який забезпечує мінімальний стартовий час виконання обчислювальної роботи незалежно від того, зайнятий він чи вільний у момент призначення. Результати роботи цього алгоритму для графу задачі (рис.11) представлені на рис. 12.

Крім того, розглянутий вище підхід щодо мінімізації кількості конфліктів може служити доповненням до будь-якого з алгоритмів призначення і може застосовуватись у тих випадках, коли проводитися призначення рівно пріоритетних процесів, з точки зору алгоритму формування черги. Іншими словами, такий підхід може бути покладений в основу створення алгоритму формування черги або є доповненням до інших

алгоритмів призначення і може застосовуватись для рівно пріорітетних процесів.

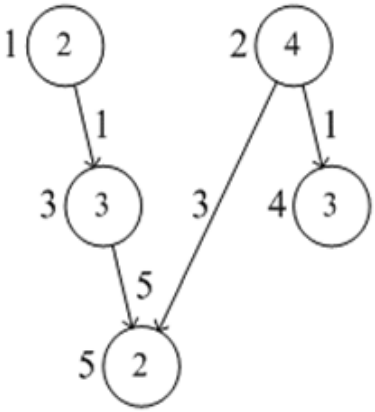


Рис. 6 Граф задачі

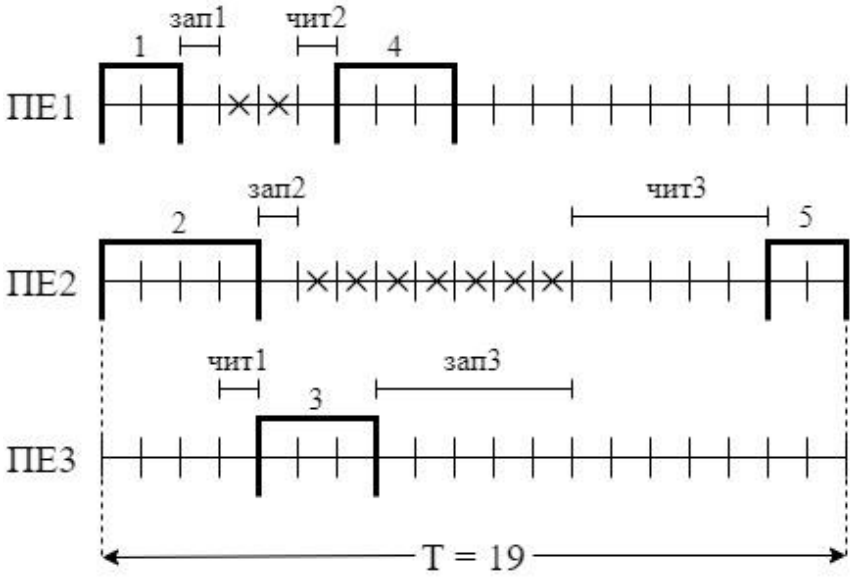


Рис. 7 Призначення на перший вільний процесор

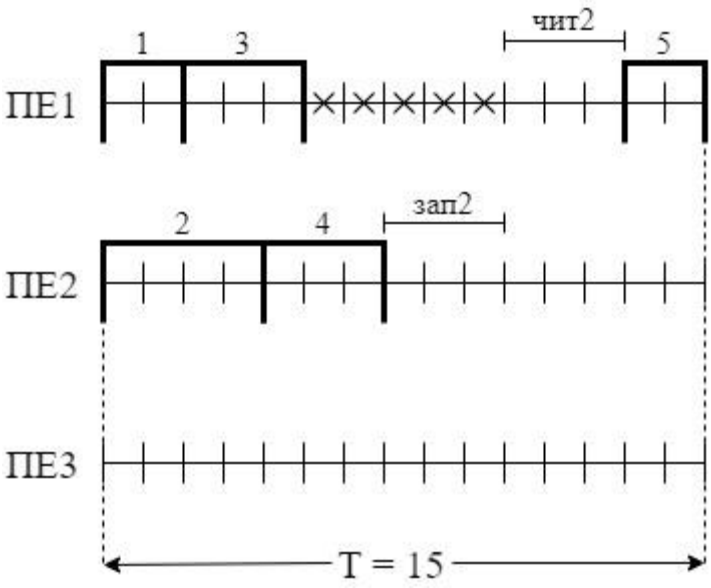


Рис. 8 «Сусіднє» призначення без автономного контролера введення-виведення, пересилання без «випередження»

Відповідно до алгоритмів «сусіднього» призначення використовувати ПЕ3 немає сенсу, оскільки обчислювальна робота призначається на той процесор, в пам'яті якого міститься максимальний обсяг вихідних даних. Тобто, для виконання третьої підзадачі буде більш пріоритетним ПЕ1, тому що дані для її виконання там уже є і не потрібно витрачати час на їх пересилку. Аналогічно для четвертої підзадачі, тому ПЕ3 можна взагалі не використовувати, там можуть виконуватись якісь інші фонові процеси.

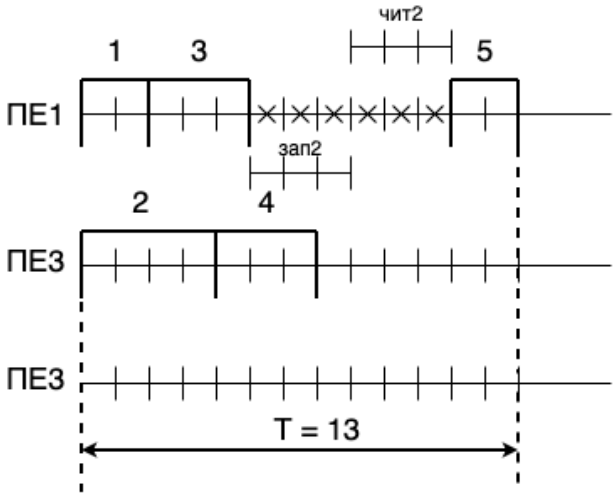


Рис. 9 «Сусіднє» призначення з контролером введення-виведення, пересилання без «випередження»

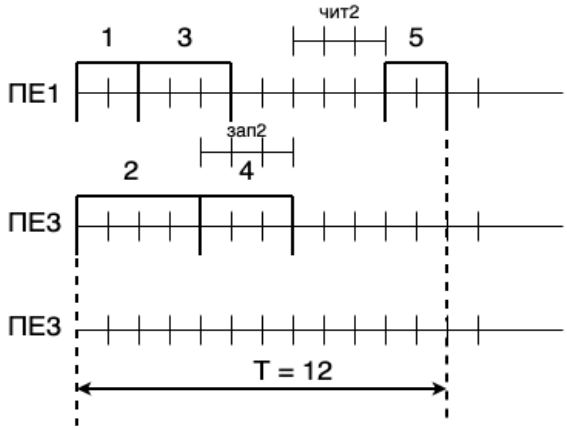


Рис.10 «Сусіднє» призначення з контролером введення-виведення, пересилання з «випередженням»

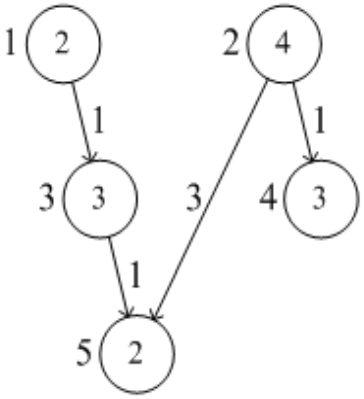


Рис. 11 Граф задачі

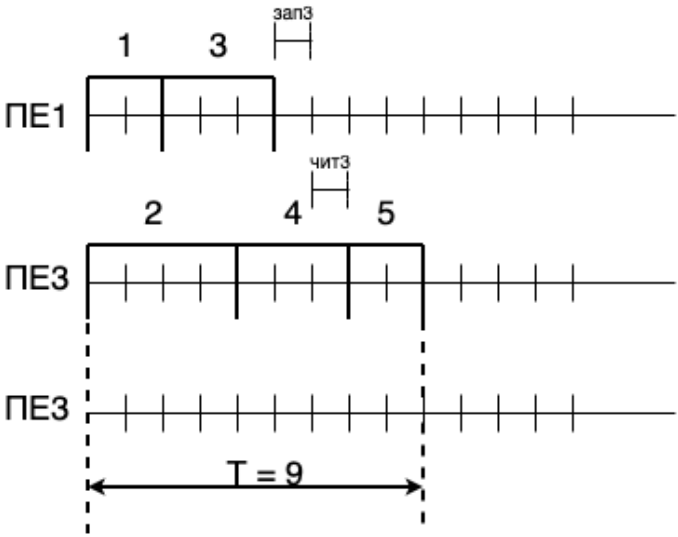


Рис. 12 «Сусіднє» призначення з урахуванням зайнятих процесорів, з автономним контролером введення-виведення

6.3.2 Призначення обчислювальних робіт на процесори для комп'ютерних систем із роздільною (локальною) пам'яттю.

При вирішенні проблеми призначення обчислювальних робіт на процесори для КС із локальною пам'яттю існує дві проблеми:

1. Призначення незалежних за даними обчислювальних робіт, що знаходяться на першому ярусі графа завдання;
2. Призначення всіх інших обчислювальних робіт (залежних за даними від інших), що знаходяться на інших ярусах графа завдання.

Перша проблема вирішується на підставі топологічних характеристик КС, тобто для процесорів визначаються пріоритети, відповідно до яких виконуються призначення обчислювальних робіт у порядку спадання їх терміновості. Для визначення пріоритетності процесорів використовуються, як правило, такі характеристики, як середня відстань від даного процесора до інших або зв'язність процесора в графі системи.

Другу проблему розглянемо більш детально. У цьому випадку мінімізація загального часу пересилань може бути забезпечена за рахунок:

- мінімізації кількості пересилань;
- мінімізація витрат через конфлікти до каналів пересилки;
- оптимальної маршрутизації (мінімізація часу пересилки);

Перше завдання вирішується так само як і в КС з роздільною пам'яттю.

Друге завдання вирішується, як правило, так. Канали пересилань займаються відповідно до черги процесів, тобто для більш пріоритетних процесів дані повинні бути передані раніше і для цих пересилань канали насамперед займаються.

Перша і друга задачі вирішуються з різною складністю.

Розглянемо наступні алгоритми призначення.

1. Алгоритм випадкового призначення

На практиці, як згадувалося раніше, часто використовується алгоритм випадково розподілу по процесорах з подальшим вирівнюванням навантаження за допомогою динамічного балансування навантаження. Цей підхід часто використовується при динамічному плануванні, а також у якості найпростішого при статичному плануванні.

2. Алгоритм призначення на перший звільнений процесор

У цьому випадку при призначенні обирається вільний процесор, який раніше за інших звільнився від попередніх обчислень. Такий алгоритм призначений для вирівнювання завантаження процесорів.

3. Алгоритм призначення з урахуванням пріоритетів процесорів (по зв'язності).

Даний алгоритм згадувався вище для призначення обчислювальних робіт першого ярусу графа завдання. Однак цей алгоритм може бути використаний для всіх обчислювальних робіт графа завдання.

4. Алгоритми «сусіднього» призначення.

Одним з ефективних підходів мінімізації пересилань є алгоритм «сусіднього» призначення. Суть цього підходу в наступному. Сусідні в графі задачі вершини (обчислювальні роботи) розміщуються на мінімальній відстані між собою в графі системи. У цьому випадку при призначенні враховуються, як правило, тільки вільні процесори.

Розглянемо кілька варіантів сусіднього призначення.

4.1. Алгоритми «сусіднього» призначення без пересилань «з випередженням».

У даному випадку використовується комунікаційна модель, коли тільки після формування всіх даних починається пересилання для будь-якої обчислювальної роботи.

Нехай для призначуваної і-ої обчислювальної роботи існує один попередник, тобто дані сформовані в одному процесорі. Тоді призначення здійснюється на один з m вільних процесорів, для якого затримка початкового моменту виконання обчислювальної роботи $T_{s,i}$ має мінімальне значення відповідно до формули:

$$T_{s,i} = \min_{j=1,m} \{e_i * r_{k,j}\} \quad (1)$$

де r_{kj} – мінімальна відстань між k-м процесором, у якому сформовані вихідні дані та j-м вільним процесором; e_i - час, що витрачається на пересилання вихідних даних для i-ї обчислювальної роботи між сусідніми процесорами.

Нехай для призначуваної i-ї обчислювальної роботи існує z попередників, тобто дані можуть бути сформовані в z процесорах. Тоді

призначення здійснюється на один з m звільнених процесорів, для якого $T_{s,i}$ має мінімальне значення відповідно до формули:

$$T_{s,i} = \min_{j=1,m} \left\{ \sum_{c=1}^{z} \left\{ e_{i,c} * r_{k,c,j} \right\} \right\} \quad (2)$$

при одному каналі у процесора.

$$T_{s,i} = \min_{j=1,n} \left\{ \max_{c=1,z} \left\{ e_{i,c} * r_{k,c,j} \right\} \right\} \quad (3)$$

при множині каналів у процесора.

де $r_{k,c,j}$ – мінімальна відстань між k-м процесором, в якому сформовані вихідні дані від c-го попередника і j-м вільним процесором; e_i - час, що витрачається на пересилання вихідних даних для і-ї обчислювальної роботи від *c*-го попередника між сусідніми процесорами.

Приклади

Нехай є КС, що складається з дев'яти процесорів, об'єднаних в топологію «решітка» (рис.13).

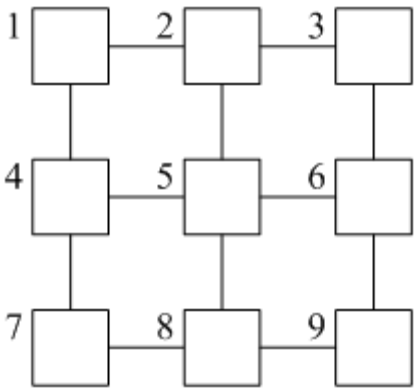


Рис. 13. Приклад графу системи

Нехай для призначуваної і-й обчислювальної роботи існує один попередник, який виконувався в шостому процесорі. У момент призначення і-ї обчислювальної роботи вільні перший і восьмий процесори. Згідно з формулою (1).

$$T_{s,i} = \min_{j=1,8} \{e_i * 3, e_i * 2\}$$

тобто для призначення обираємо восьмий процесор, оскільки $r_{6,8}=2$, $r_{6,3}=3$.

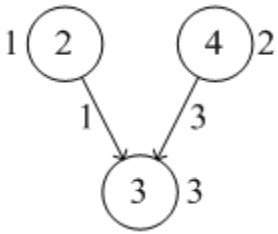


Рис.14. Приклад графу задачі

Нехай для призначуваної 3-ї обчислювальної роботи існує два попередника відповідно до графу задачі (рис.14), причому 1-а обчислювальна робота виконувалася на другому процесорі, а 2-а на шостому процесорі. У момент призначення 3-ї обчислювальної роботи вільні 5-й і 9-й процесори.

Тоді відповідно до формули (2)

$$T_{s,3} = \min_{j=5,9} \{1*1+3*1, 1*3+3*1\}$$

, коли процесори мають один фізичний канал. У цьому випадку для призначення обираємо п'ятий процесор.

Якщо процесори мають множину фізичних каналів, то згідно з формулою (3)

$$T_{S,3} = \min_{j=5,9} \{\max\{2,3\}, \max\{3,3\}\}$$

У цьому випадку для призначення 5-й і 9-й процесори рівноправні. При призначенні на 5-й процесор діаграма Ганта має вигляд, зображений на рис.15.

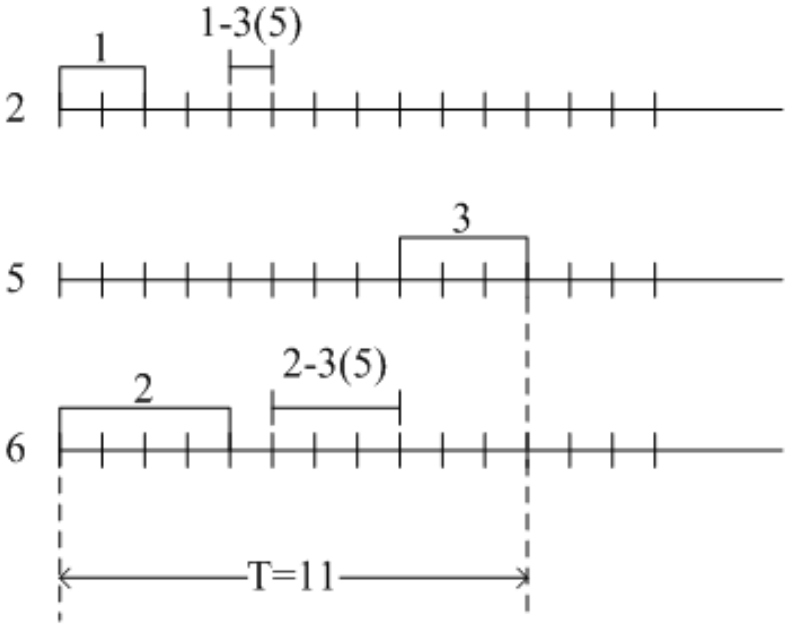


Рис.15. Діаграма Ганта при виконанні графу задачі (рис.14) на КС (рис.13) з використанням пересилань «без випередження» (процесори мають один фізичний канал)

4.2. Алгоритм сусіднього призначення з пересилками «з випередженням».

У даному випадку використовується комунікаційна модель, аналогічна тій, що була описана в алгоритмі для КС із спільною пам'яттю, де дані передаються асинхронно відразу після їх формування. Для розглянутого вище прикладу діаграма Ганта має вигляд, представлений на рис.16. У цьому випадку дані від 1-ї вершини передаються відразу після їх формування. Це призводить до зменшення загального часу виконання завдання. Тут T=10, а для пересилань «без випередження» T=11 (рис.15).

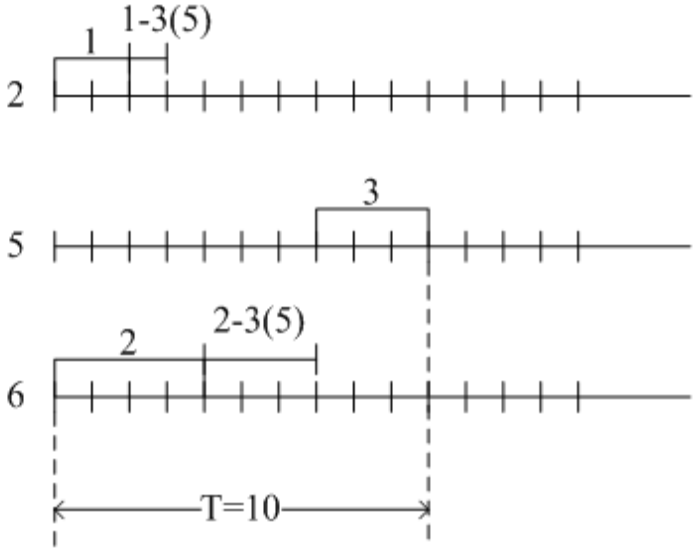


Рис.16. Діаграма Ганта при виконанні графу задачі (рис.14) на КС (рис.13) із пересилками «з випередженням»

4.3. Алгоритм сусіднього призначення з використанням моделювання для визначення початкового часу виконання обчислювальних робіт.

У попередньому алгоритмі для визначення затримки початкового часу виконання обчислювальних робіт використовувалися формули (1)-(3). Однак у цих випадках не враховувалися втрати часу через можливі конфлікти до каналів з пересилання повідомлень (передбачається, що в один і той же момент часу канал використовується для пересилання тільки одного повідомлення), кількості каналів у процесора, паралельних пересилок даних по каналах різних процесорів (несусідніх), а також можливості асинхронності пересилань «з випередженням». Усі ці фактори можна врахувати тільки при моделюванні пересилань. Таким чином даний алгоритм заснований на тому, що для кожного з вільних процесорів моделюються пересилання для розглянутої обчислювальної роботи і обирається той з них, який забезпечує мінімальний початковий час її (обчислювальної роботи) виконання. При моделюванні можуть враховуватися два варіанти комунікаційних моделей: пересилання «без випередження» і пересилки «з випередженням».

У розглянутих вище алгоритмах «сусіднього» призначення використовувалися тільні вільні процесори.

В цьому їх основний недолік. Оскільки можлива ситуація, коли в момент призначення вільним може виявитися процесор, що знаходиться на значній відстані від процесора джерела даних і пересилання даних означає значні втрати часу і завантаженість каналів даних. У цьому випадку часто більш вигідним виявляється очікування звільнення процесорів, що знаходяться на більш близькій відстані. Виходячи з вищевикладеного, для забезпечення мінімального часу, що витрачається на пересилання варто враховувати як вільні, так і зайняті процесори.

4.4. Алгоритм оптимізованого «сусіднього» призначення, в якому враховуються всі процесори незалежно від їх стану в момент призначення (зайнятий або вільний).

У цьому випадку найпростіший алгоритм призначення полягає в наступному. Для кожного процесора визначається початковий момент звільнення процесора, а також сумарний час пересилань необхідних вихідних даних з процесорів, де вони сформовані. Ці показники сумуються і визначається початковий час виконання розглянутої обчислювальної роботи. Призначення проводиться на той процесор, де початковий час має мінімальне значення.

У разі, коли і-а призначувана обчислювальна робота має одного попередника, мінімальне значення початкового моменту часу виконання і-ї роботи (Ts, i) визначається за формулами:

$$T_{s,i} = \min_{j=1,n} \left\{ \max \left\{ T_j, T_f + e_i * r_{k,j} \right\} \right\}$$

для пересилань з випередженням.

де n - число процесорів в КС; T_j – час звільнення j-го процесора; T_f – час формування даних для і-го процесора.

У разі, коли і-а призначувана обчислювальна робота має z попередників, то

$$T_{s,i} = \min_{j=1,n} \left\{ \max_{c=1,z} \left\{ T_j, \sum_{c=1}^{z} T_f + e_{i,c} * r_{k,c,j} \right\} \right\}$$

для пересилань без випередження.

$$T_{s,i} = \min_{j=1,n} \left\{ \max_{c=1,z} \left\{ T_j, \sum_{c=1}^{z} T_{f,c} + e_{i,c} * r_{k,c,j} \right\} \right\}$$

для пересилань з випередженням

Приклади

Нехай є граф системи на рис.17 і граф завдання на рис. 18.

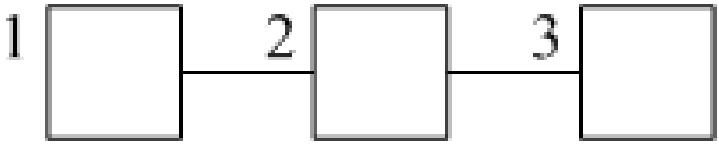


Рис. 17 Приклад графу системи

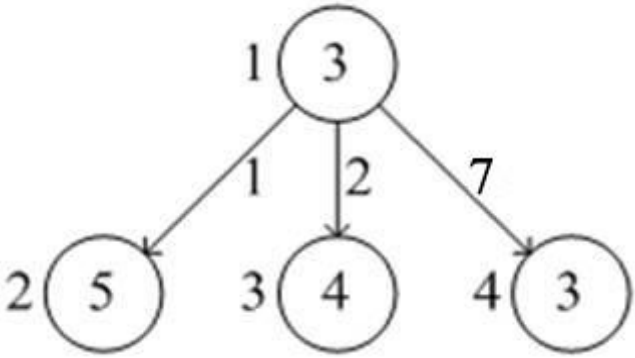


Рис. 18 Приклад графу задачі

Нехай є черга вершин графу задачі (відповідно до 3-го алгоритму): 1(8), 2(5), 3(4), 4(3).

Нехай 1-а вершина виконується на другому процесорі (з максимальною зв'язністю). Тоді відповідно до черги інших вершин і з урахуванням алгоритму «сусіднього» призначення (4.2) діаграма Ганта має вигляд, представлений на рис.19.

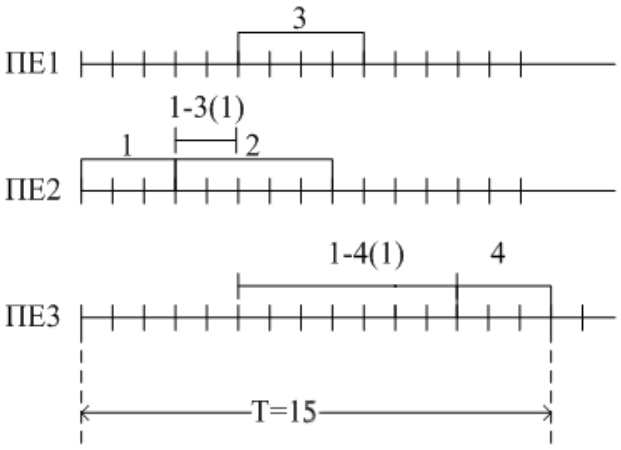


Рис.19 Діаграма Ганта з використанням алгоритму 4.2

Відповідно до алгоритму оптимізації сусіднього призначення з урахуванням зайнятих процесорів (4.4), діаграма Ганта буде мати вигляд, представлений на рис.20.

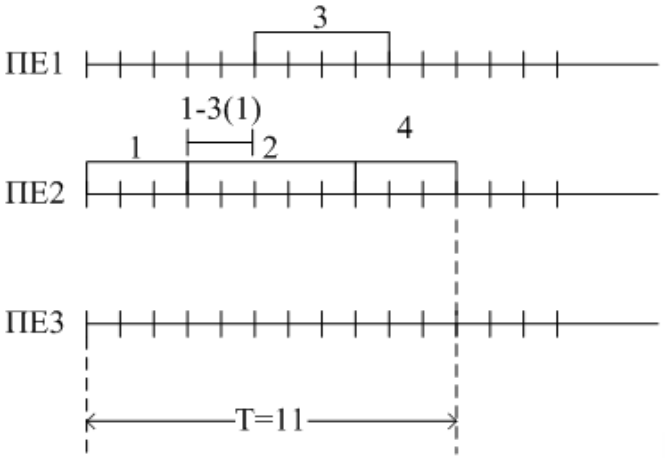


Рис. 20 Діаграма Ганта з використанням алгоритму 4.4

Недоліком наведеного підходу є повний перебір процесорів при призначенні.

Це призводить до великих часових витрат, тим більше для КС з великою кількістю процесорів. Тому слід шукати підходи, які забезпечують не повний, а спрямований перебір процесорів для призначення. Розглянемо один з таких алгоритмів. Спочатку усі процесори розбиваються на групи. У кожну групу повинні входити процесори, що мають однакову відстань до процесора – джерел(-а) даних. Пошук процесорів починається з групи, що має мінімальну відстань (або з процесора - джерела). Визначається момент часу його звільнення. Оскільки в даному випадку час на пересилання дорівнює нулю, то час звільнення і буде часом початкового виконання першочергової обчислювальної роботи. Після цього отриманий час слід порівняти з пізнім терміном виконання і якщо запізнення немає, то виконуємо призначення на цей процесор. Якщо ж запізнення є, то пошук триває. Обирається група процесорів з відстанню, що дорівнює одиниці, визначається процесор, що має мінімальне значення часу звільнення і для нього визначається початковий час виконання процесу , як сума часу звільнення і часу пересилання даних. Далі, як у попередньому випадку, проводимо порівняння з пізнім терміном виконання і якщо запізнень немає, то проводиться призначення, а якщо є, то пошук продовжується в іншій групі. Таким чином проводиться перебір груп процесорів до тих пір, поки не буде знайдений процесор, на якому розглянутий процес виконується без запізнення, або у разі перебору всіх груп буде знайдений процесор, на якому розглянутий процес виконується з мінімальним запізненням.

На рис.21. наведений приклад КС, коли 3-й процесор містить вихідні дані для призначуваної обчислювальної роботи.

Тоді всі процесори розподіляються за такими групами:

I – 3 процесор
II – 2 і 6 процесори
III – 1, 5 і 9 процесори
IV – 4 і 8 процесори
V – 7 процесор

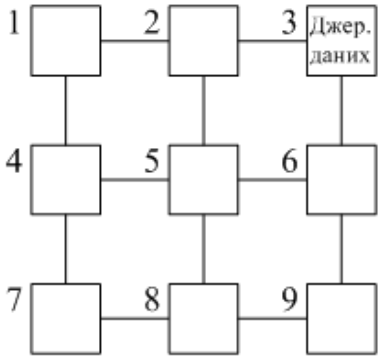


Рис. 21 Приклад КС

Аналогічним чином можна описати алгоритм призначення, коли є множина процесорів-джерел даних. Групи визначаються трохи інакше.

Приклад

Нехай 1 і 3 процесори-джерела даних, тоді:

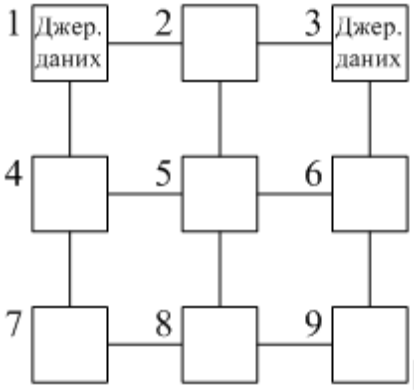


Рис. 22 Приклад КС

I група – 1, 2, 3 (d=2)
II група -5, 4, 6 (d=4)
III група – 7, 8, 9 (d=6),

а далі алгоритм аналогічний.

7. Особливості планування для неоднорідних КС

Будемо розрізняти наступні типи неоднорідних КС:

- Неоднорідність при використанні чіпів з різними технічними характеристиками (кількість ядер, продуктивність ядер, комунікаційна вартість, енергоефективність)
- Функціональна неоднорідність

Для неоднорідності першого типу специфіка вхідних даних при виконанні задачі планування полягає у наступному:

1. Використовується **зважений** помічений граф системи;
2. На основі графів задачі і системи для кожної підзадачі формується **вектор часів їх виконання** на різних ядрах системи, а для пересилок **вектор комунікаційних вартостей**.

Для неоднорідності другого типу специфіка вхідних даних полягає у наступному:

1. У графі задачі можуть бути задані різні типи вершин відповідно до їх функціонального призначення;
2. У графі системи існують різні типи вершин відповідно до їх функціонального призначення

Специфіка алгоритмів планування для неоднорідності першого типу – при виборі ресурсу необхідно орієнтуватись на той процесор (ядро), що забезпечує **мінімальний час завершення виконання підзадачі**

Специфіка алгоритмів планування для неоднорідності другого типу - при виборі ресурсу необхідно орієнтуватись на відповідний тип або одночасне використання ресурсів різних типів. Алгоритми реалізації планування для неоднорідних КС розглянуті аторами у роботах [9],[12].

8. Приклад виконання контрольної роботи.

Завдання. Вирішити завдання оптимального планування обчислень в комп'ютерній системі з роздільною пам'яттю. Критерій оптимізації - мінімальний час рішення задачі.

Для вирішення даного завдання необхідно виконати наступні підзадачі:

- Визначити мінімальний час вирішення завдання ($T_{кр\ гр}$).

- Визначити мінімальне число процесорів (N_{min}), необхідне для вирішення завдання за мінімальний час і порівняти з заданим в системі.
- Визначити пріоритети всіх вершин графа завдання з урахуванням їх критичних шляхів. Сформувати чергу вершин графа задачі.
- Визначити пріоритети процесорів системи на підставі їх зв'язності. Сформувати чергу процесорів.
- Виконати призначення обчислювальних робіт задачі на процесори системи з урахуванням сформованої черги і мінімізації часу пересилань даних. Представити результати планування у вигляді діаграми Ганта.
- Виконати аналіз отриманих результатів, визначити коефіцієнти прискорення та ефективності роботи системи, а також порівняти отриманий час вирішення завдання з мінімальним значенням.

Вхідні дані:

- граф системи (рис. 23)

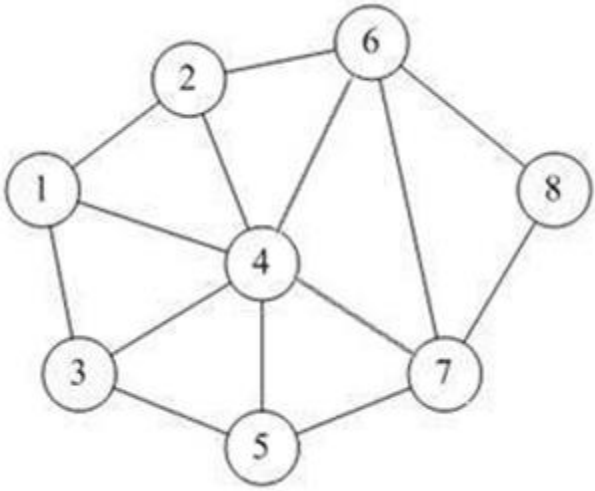


Рис. 23 Граф системи

- граф задачі (рис. 24)

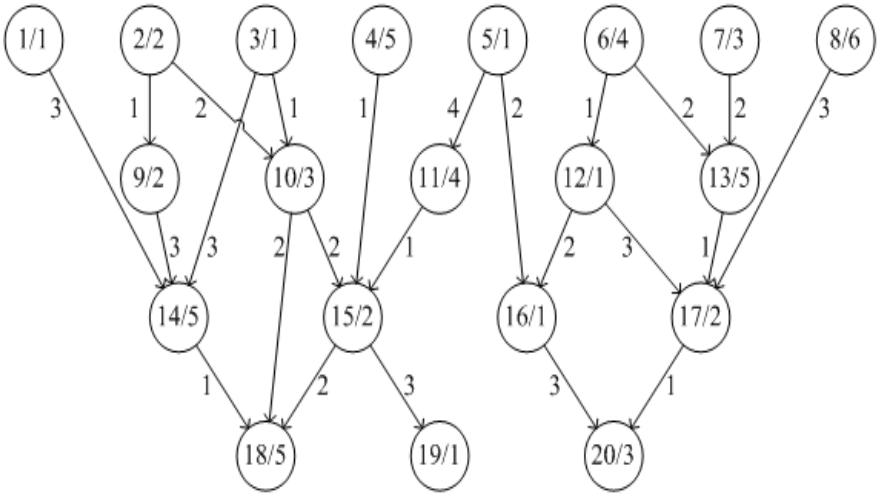


Рис. 24 Граф задачі

Розглянемо вказівки до виконання контрольного завдання:

- При плануванні застосувати списковий метод, що складається з двох етапів: формування черги готових обчислювальних робіт (вершин графа задачі) і призначення обчислювальних робіт на процесори.
- При формуванні черги застосувати алгоритм №3, розглянутий у п.п.6.2.
- При виконанні другого етапу застосувати алгоритм «сусіднього» призначення.
- Вважати, що кожен процесор системи має число фізичних каналів, що дорівнює кількості його зв'язків з іншими процесорами відповідно до заданої топології.Кожен канал підтримується автономним контролером введення-виведення, що забезпечує кожному процесору можливість одночасної обробки та пересилання повідомлень з сусідніми процесорами.
- Для вершин першого ярусу графа завдання рекомендується застосувати призначення на процесори з урахуванням їх пріоритетів по зв'язності.
- При пересиланні повідомлень рекомендується застосовувати другу комунікаційну модель «з випередженням».
- Можливе застосування дуплексного варіанту пересилання повідомлень.

Послідовність виконання завдання:

1. Для графу задачі на рис.24 є два критичних шляхи, що проходять через вершини 2 – 9 – 14 – 18 і 6 – 13 – 17 – 20.

$T_{кр\ гр} = 14$

2. Мінімальне число процесорів, необхідних для вирішення задачі за мінімальний час визначається за формулою:

$$\mathrm{N\ min} = \left\lceil \frac{\sum_{i=1}^{m} Wi}{T_{кр\ гр}} \right\rceil,$$

де m – число вершин у графі задачі, W_i – вага і-ї вершини графа задачі.

Для графу задачі на рис. 24

$$\mathrm{N\ min} = \left\lceil \frac{57}{14} \right\rceil = 5$$

У заданому графі системи є вісім процесорів, значить рішення задачі за $T_{кр\ гр}$ можливе.

3. Визначимо пріоритети всіх вершин графу задачі на підставі критичних шляхів:

Pr_1=11; Pr_2=14; Pr_3=11; Pr_4=12; Pr_5=12; Pr_6=14; Pr_7=13; Pr_8=11; Pr_9=12; Pr_{10}=10; Pr_{11}=11; Pr_{12}=6; Pr_{13}=10; Pr_{14}=10; Pr_{15}=7; Pr_{16}=4; Pr_{17}=5; Pr_{18}=5; Pr_{19}=1; Pr_{20}=3.

Побудуємо чергу обчислювальних робіт за спаданням пріоритетів:

2 (14), 6 (14), 7 (13), 4 (12), 5 (12), 9 (12), 1 (11), 3 (11), 8 (11), 11 (11), 10 (10), 13 (10), 14 (10), 15 (7), 12 (6), 17 (5), 18 (5), 16 (4), 20 (3), 19 (1).

4. Пріоритети процесорів по зв'язності для графуусистеми на рис.23 мають наступний вигляд:

Pr_1=3; Pr_2=3; Pr_3=3; Pr_4=6; Pr_5=3; Pr_6=4; Pr_7=4; Pr_8=2.

Побудуємо чергу процесорів системи за спаданням пріоритетів:

4 (6), 6 (4), 7 (4), 1 (3), 2 (3), 3 (3), 5 (3), 8 (2).

5. Виконаємо призначення обчислювальних робіт (вершин графа завдання) на процесори і результати представимо у вигляді діаграми Ганта (рис. 25).

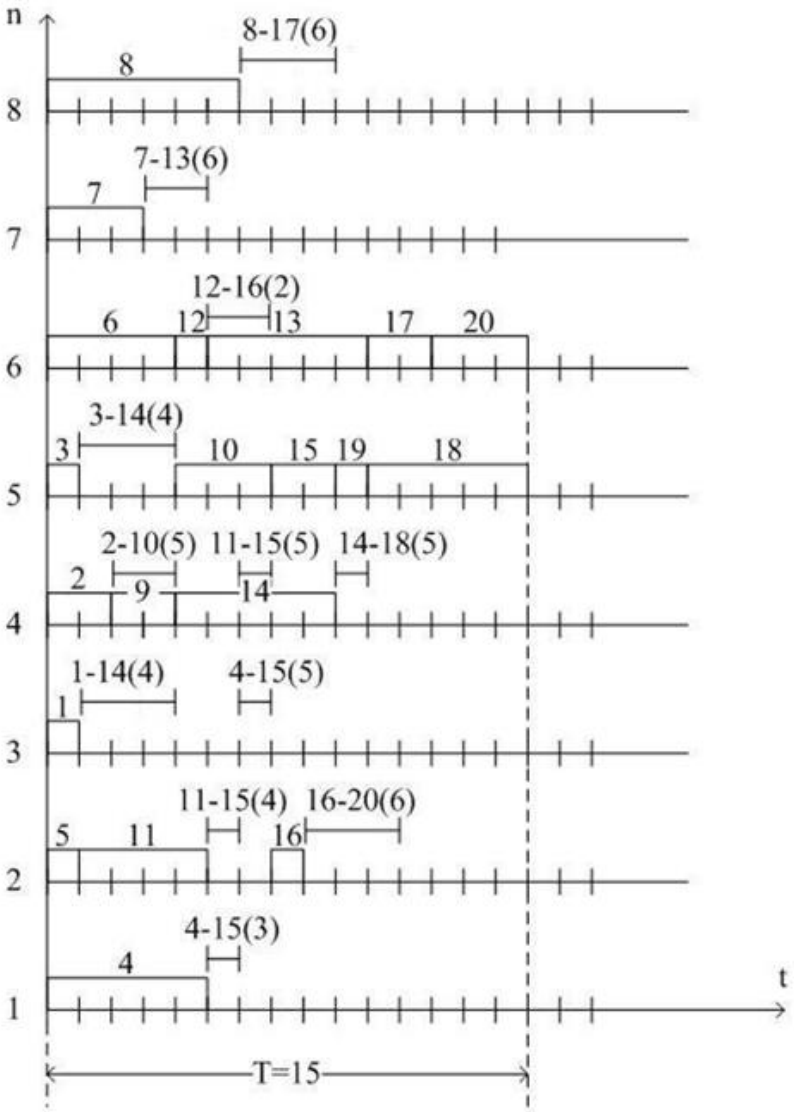


Рис. 25 Діаграма Ганта

6. Відповідно до отриманої діаграми Ганта, реальний час виконання завдання ($T_р$) дорівнює 15, що на 1 більше, ніж значення

критичного шляху. Перевищення реального часу пов'язане з сильною зв'язністю задачі (> 50%).

Коефіцієнт прискорення (Кп) виконання завдання визначається, як

$$K_п = \frac{\sum_{i=1}^{m} W_i}{T_p} = \frac{57}{15} = 3,6$$

Коефіцієнт ефективності роботи системи визначається за формулою:

$$K_e = \frac{K_п}{n} = \frac{3,6}{8} = 0,45$$

9. Динамічне балансування завантаження в MIMD-системах з роздільною пам'яттю.

Зазвичай проблема балансування завантаження розглядається для систем з роздільною пам'яттю.

Балансування завантаження є одним із наближених методів відображення або планування паралельних процесів на паралельну архітектуру КС. При цьому процеси призначають на процесори таким чином, щоб їх обчислювальне завантаження було оптимальним. Критеріями оптимізації завантаження може бути рівне число процесів, рівне обчислювальне завантаження процесів, а також двохкритеріальна оптимізація, коли крім обчислювального завантаження процесів, враховується комунікаційне завантаження (мінімальний час на пересилання).

Розрізняють статичне і динамічне балансування завантаження. Алгоритми статичного планування і відображення фактично вирішують проблему статичного балансування завантаження. Алгоритми ж динамічного балансування є частиною ОС і використовуються як додаткові засоби динамічного планування з метою отримання більш якісного результату розподілу процесів по процесорах. Найчастіше у цьому випадку в якості алгоритму динамічного планування використовується випадковий вибір.

Найбільш вагому практичну цінність представляють алгоритми балансування завантаження для тих науково-технічних завдань (фізики, моделювання, хімії) для вирішення яких заздалегідь невідома кількість процесів (частіше, коли мова йде про ітераційні методи вирішення - моделювання при проектуванні літальних апаратів та ін.).

Існує три типи алгоритмів динамічного балансування завантаження:

1. Централізований. У цьому випадку балансування виконується на основі інформації про поточне завантаження всіх процесів КС.

2. Децентралізований. У цьому випадку кожен ПЕ виконує балансування на основі інформації про завантаження тільки своїх сусідів.

3. Комбінований або напівдецентралізований. У цьому випадку КС ділиться на групи (кластери). Усередині кожної групи використовується централізований алгоритм.

Таким поняттям при реалізації будь-якого підходу є завантажувальний контролер (load controller) (програма управління завантаженням). Залежно від підходу в системі існує один або множина завантажень контролерів. У I підході - один, в II - число контролерів дорівнює числу ПЕ, а в III - числу груп ПЕ.

Основні функції завантажувального контролера:

1. початковий прийом або зміна інформації про процеси в мережі КС;

2. аналіз черг процесів залежно від критерію;

3. прийняття рішення про переміщення процесів залежно від типу алгоритму балансування.

Початкове завантаження процесів відбувається відповідно до застосовуваної стратегії динамічного планування в розглянутій КС. Як згадувалося вище, найчастіше застосовується розподіл по процесорах випадковим (довільним) чином. Можуть бути й інші дисципліни, які ми розглядали вище. Крім цього, завантаження процесів може бути проведене статично і динамічно. У першому випадку відбувається завантаження (розподіл) по процесорах всіх процесів, що включають задачі незалежно від того, готові вони до виконання чи ні. У другому випадку розподіл по процесорах проводиться у міру готовності процесів до виконання. В обох випадках кожен процесор повинен мати дві черги: активних (готових) процесів і пасивних. У першому випадку пасивними будуть вважатися процеси, які не готові до виконання спочатку, а також процеси, які очікують пересилань даних. У другоиу випадку пасивні - це процеси, які очікують пересилань даних.

Зміна інформації про процеси в КС може проводиться різними способами:

1. постійно (кожен такт);

2. періодично, тобто через задані проміжки часу;

3. коли контролер завантаження будь-якого ПЕ запитує інформацію завантаження інших ПЕ;

4. при переміщенні процесів від одного ПЕ до іншого.

Аналіз черг процесів проводиться або в різних ПЕ (при децентралізованому і комбінованому алгоритмах) або в одному ПЕ (при централізованому підході) з урахуванням прийнятого критерію оптимізації.

Якщо в якості критерію оптимізації використовується рівна кількість процесів, що обробляються в процесорах КС, то при централізованому підході на початку визначається середня кількість готових процесів або існує гранична довжина черги готових процесів, що знаходяться в системі. Потім проводиться перерозподіл процесів з тих ПЕ, де число процесів перевищує середнє значення (граничне значення) в ті ПЕ, де число таких процесів нижче середнього значення (граничного значення).

Для децентралізованого підходу в кожному ПЕ визначається середня кількість готових процесів, що знаходяться в сусідніх ПЕ. Потім проводиться перерозподіл так само як і в попередньому випадку, але тільки серед сусідніх ПЕ. Як видно цей алгоритм працює швидше, але менш точний за результатами. Якщо ж у якості критерія оптимізації використовується сумарний час обробки процесів, то при централізованому алгоритмі визначається середній сумарний час виконання готових процесів, що знаходяться в системі. Потім проводиться перерозподіл процесів. У децентралізованому алгоритмі для кожного ПЕ визначається середній сумарний час виконання готових процесів, що знаходяться по сусідству і потім проводиться їх перерозподіл.

Якщо в якості критерію використовується рівний сумарний час виконання і мінімальний час пересилань, то зазвичай використовується централізований алгоритм балансування. У цьому випадку на початку визначається середній час виконання процесів в системі. Потім визначаються множини ПЕ - джерел зайвих процесів і ПЕ - приймачі додаткових процесів. Далі визначаються пари ПЕ - джерел - приймачів таким чином, щоб мінімізувати час пересилань.

10. Функція призначення модулів ОС по процесорах.

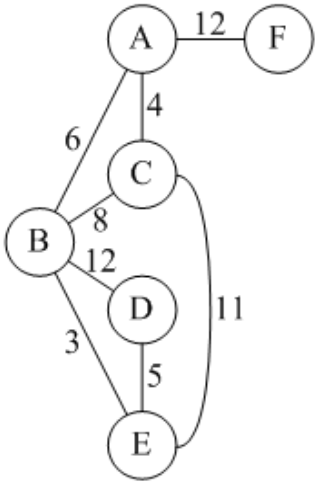
Існує три підходи до призначення модулів ОС по процесорах: «жорстке» закріплення - визначення модулів, які постійно виконуються на певних процесорах(Hellios), рівноправне виконання модулів ОС на процесорах (тобто відсутність «жорсткого» призначення модулів за процесорами, що характерно для Cray X / MP та інших симетричних КС) і комбінований варіант, тобто частина модулів закріплені за окремими процесорами, інші в процесі роботи системи можуть бути призначені на будь-які процесори.

З точки зору максимальної продуктивності і відповідно ефективності КС перший варіант є найбільш жорстким, але функція призначення як така відсутня, вона визначається раз і назавжди при розробці ОС. Через нерівноправності завантаження процесорів модулями ОС, а також через тимчасові витрати, пов'язані з пересиланням даних, ефективність у деяких випадках може бути низькою.

Другий варіант є достатньо гнучким, проте формально вирішити завдання призначення оптимальним чином досить складно. Це комбінаторна NP-повна задача, як і всі задачі призначення. Тому практично для симетричних КС це завдання зводиться до вирішення завдання балансового завантаження процесорів.

Третій варіант є також досить гнучким, але з теоретичної точки зору є трохи простішим в порівнянні з другим, хоча і в цьому випадку поки немає універсального методу призначення на n процесорів.

Розглянемо сутність цього методу для двопроцесорних КС. Нехай є розподілена система, що складається з різних по продуктивності ПЕ. Деякі модулі ОС жорстко закріплені за певними процесорами. В нижче наведеному методі ми не будемо враховувати окремо втрати часу на синхронізацію. Отже, наше завдання - розподілити модулі ОС по процесорах таким чином, щоб мінімізувати загальний час їх виконання. Це завдання вирішується методом знаходження мінімального розрізу потоку в мережі. Уявімо програму ОС у вигляді графа, наприклад



Де A, B, C, D, E, F - модулі ОС, ваги дуг - часова трудомісткість міжмодульних зв'язків.

У таблиці 3 визначається час на виконання модулів на різних процесорах.

Таблиця 3

Тип модуля	ПЕ1	ПЕ2

A	5	10
B	2	-
C	4	4
D	6	3
E	5	2
F	-	4

З таблиці 3 видно, що B закріплений за ПЕ1, а F - за ПЕ2.

Відзначимо, що загальний час виконання всієї програми ОС складається з сумарного часу виконання всіх модулів плюс часові витрати на міжмодульні пересилки. Якщо модулі виконуються на одному процесорі, то час на пересилання між ними дорівнює нулю.

З точки зору мінімізації сумарного часу виконання доцільно модулі розподілити таким чином:

на першому ПЕ - B, A, C
(по таблиці)
на другому ПЕ - D, E, F.

З точки зору мінімізації часу пересилань доцільно модулі розподіляти так:

на першому ПЕ - B, C, D, E
(по графу)
на другому ПЕ - F, A.

Ми отримали різні рішення, причому жодне з них не вирішує проблему мінімізації загального часу. Запропонуємо наступний алгоритм. Будується модифікований граф, в якому окрім вершин відповідних модулям введені вершини S_1 і S_2, що представляють процесори P_1 і P_2. Усі вершини, відповідні модулям пов'язані з вершинами S_1 і S_2. Вага дуг дорівнює часу виконання відповідних вершин на процесорах. Дуги між вершинами - модулями означають час міжмодульних пересилань. У даному випадку будь-який розріз такого графа однозначно відповідає розподілу модулів по процесорах, а вага розрізу є загальною ціною (часом) виконання програми ОС при даному розподілі по процесорах. Модифікований граф для вищенаведеного прикладу зображено на рис.26

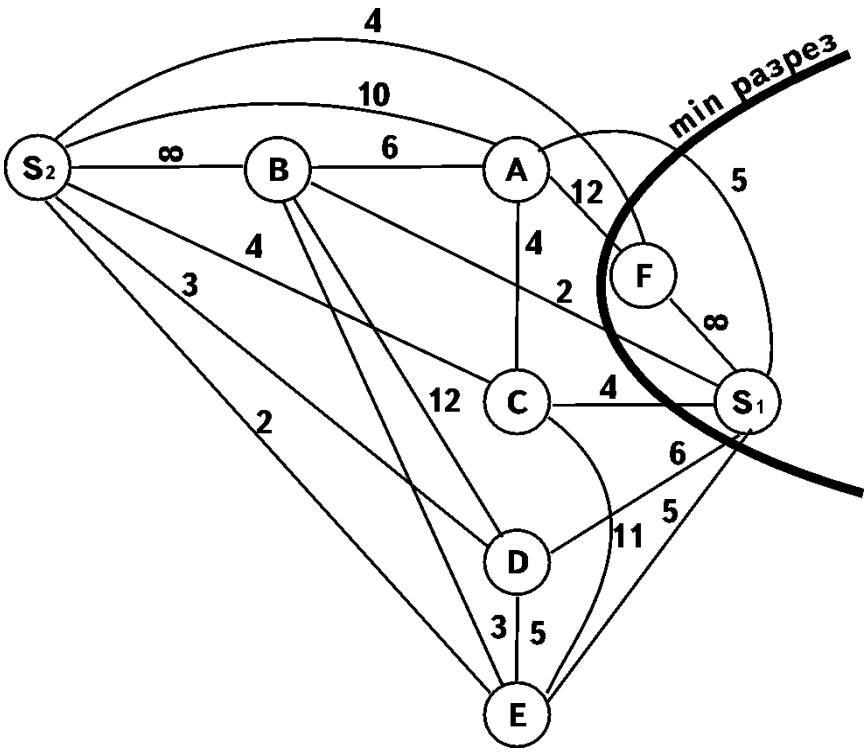


Рис.26 Модифікований граф

Мінімальна сумарна ціна(мінімальний розріз даного графу) складає 5 + 4 + 12 + 2 + 4 + 6 + 5 = 38 У цьому випадку на ПЕ1 виконуються модулі - A, C, D, E, B, а на ПЕ2 – F. Будь-який інший розріз має сумарну ціну більше вищезазначеної мінімальної сумарної ціни.

Визначення мінімального розрізу для двопроцесорних КС має точне рішення. Для n - процесорних КС такого точного рішення немає. Рішення може бути прийняте лише евристичним шляхом.

11. Специфіка планування у Grid системах

Одним з основних методів підвищення реальної продуктивності традиційних паралельних, кластерних і розподілених систем є ефективне планування обчислювальних ресурсів. Дане завдання відноситься до класу NP-повних і в загальному випадку точного рішення не має. Тому дослідження в цій області зосереджені на пошуку евристичних підходів, за допомогою яких можуть бути отримані квазіоптимальні результати. Рішення даного завдання планування ускладнюється бурхливим зростанням кількості комп'ютерів в кластерних системах і появою Grid систем. Більш того, алгоритми планування безпосередньо пов'язані з основними характеристиками паралельних (симетричних мультипроцесорних систем (SMP) і систем з масовим паралелізмом (MPP)), кластерних і Grid систем. У таблиці 4 представлені основні характеристики перерахованих систем і відповідні їм алгоритми планування [4].

Таким чином, на підставі попередніх досліджень можна зробити висновок, що традиційні моделі планування для паралельних систем в загальному випадку недостатні для Grid систем. Причини полягають в наступних передбачуваних властивостях традиційних паралельних систем [5]:

• Усі ресурси постійно знаходяться в межах одного адміністративного домену.

• Для забезпечення образу окремої системи, планувальник управляє всіма ресурсами.

• Сукупність ресурсів постійна.

• Конфліктні ситуації, викликані багатьма додатками, як правило планувальником не розглядаються. Передбачається, що в системі обробляється єдине завдання.

• Завдання і його дані знаходяться на одному і тому ж вузлі.

Таблиця 4 Розвиток алгоритмів планування в паралельних і розподілених системах

Тип архітектури	SMP и MPP	Кластери	Grid
1	2	3	4
Хронологія	Кінець 1970-х	Кінець 1980-х	Середина 1990-х
Типові засоби комутації	Шина, комутатор	Комерційні LAN, ATM	WAN/ Інтернет
Вартість комутації	Дуже низька	Низька	Висока
Гетерогенність взаємозв'язку	Ні	Низька	Висока

Гетерогенність вузлів	Ні	Низька	Висока
Єдина система бачення	Так	Так	Ні
Ресурсний пул Статичний / Динамічний	Зумовлений і статичний	Зумовлений і статичний	Не визначений і динамічний
1	2	3	4
Політика управління ресурсами	Однакова	Однакова	Різноманітна
Типовий алгоритм планування	Однорідний алгоритм планування	Неоднорідний алгоритм планування	Grid алгоритм планування

На жаль, всі ці властивості не характерні для Grid систем. У даних системах багато унікальних характеристик, які набагато ускладнюють виконання завдання планування більш складним завданням [3]. Розглянемо основні з них.

Неоднорідність (гетерогенність) і автономність. Ще до появи Grid систем неоднорідність враховувалася в деяких алгоритмах планування для кластерів. Однак повністю вимоги неоднорідності для Grid систем ці алгоритми не враховували. У Grid системах неоднорідність характерна як для обчислювальних ресурсів, так і для комунікаційних мереж. Оскільки обчислювальні ресурси розподілені в різних доменах Інтернету, комунікаційні мережі можуть значно відрізнятися по пропускній здатності і використовуваним протоколах комунікації. Обчислювальні ресурси різнорідні в тому, що вони можуть мати різні апаратні засоби, такі як набір команд, комп'ютерну архітектуру, кількість процесорів, обсяг пам'яті, продуктивність процесорів, а також різне програмне забезпечення, таке як операційні системи, файлові системи, програмне забезпечення управління кластерами і т. д. У традиційних паралельних і розподілених системах, обчислювальні ресурси зазвичай управляються централізовано. Планувальник не тільки має повну інформацію про всі виконання / затримки завдань і використані ресурси, але також і керує чергою завдання і пулом ресурсу. Це дозволяє легко передбачити поведінку ресурсів, і призначити завдання на ресурси відповідно до певних вимог їх виконання. У Grid системі ресурси зазвичай автономні і планувальник не має повного управління ресурсами. Він не може порушити локальну політику використання ресурсів. Це призводить до складнощів оцінки точної вартості виконання завдання на різних вузлах для планувальника Grid системи. Автономність також призводить до різноманітності підходів, що

застосовуються в локальному управлінні ресурсами і політиці управління доступом (пріоритетні параметри налаштування для різних додатків і методів резервування ресурсу). Таким чином, планувальник Grid мережі зобов'язаний бути адаптивним до різних видів локальної політики ресурсів. Неоднорідність і автономність з точки зору користувача Grid системи представлені різними параметрами, такими як, параметри ресурсів, моделі виконання програми та критерії оптимізації.

Динамічність середовища. У традиційних паралельних обчислювальних середовищах, таких як кластери, безліч ресурсів постійні (статичні). Для Grid системи характерна динамічність середовища, тобто кількість і відповідно продуктивність доступних ресурсів постійно змінюється. З одного боку нові ресурси можуть приєднатися до Grid системи, а з іншого боку, деякі ресурси можуть стати недоступними через такі проблеми, як відмова мережі. Крім того, можливості ресурсів можуть змінюватися в часі, через автономності вузлів і конкуренції програм за ресурси. Ефективний планувальник повинен бути адаптивним до такої динамічної поведінки. Після того, як новий ресурс приєднається до Grid системи, планувальник повинен виявити це автоматично і використовувати новий ресурс в подальшому при плануванні. Коли обчислювальний ресурс стає недоступним в результаті несподіваної відмови, повинні бути використані механізми, такі як контрольна точка або перепланування, щоб гарантувати надійність Grid систем. Та ж динамічність характерна і для комунікаційних мереж, які об'єднують ресурси Grid системи: доступна пропускна здатність може бути сильно змінена потоками Інтернет-трафіку. Це призводить до коливання продуктивності Grid системи, ускладнює оцінку продуктивності мережі і знижує ефективність планування обчислень з використанням класичних підходів. Алгоритм планування повинен бути адаптивним до такої динамічної поведінки (наприклад, резервування ресурсу і перепланування).

Глобальна розподіленість. У традиційних паралельних системах програмні коди додатків і їх вхідні / вихідні дані знаходяться зазвичай на одному і тому ж вузлі. В цьому випадку вартістю пересилання вхідних / вихідних даних або нехтують, або приймають за константу, певну перед виконанням. Алгоритми планування цю вартість не розглядають. Оскільки Grid система складається з великої кількості різнорідних обчислювальних вузлів (від суперкомп'ютерів до настільних комп'ютерів), пов'язаних через глобальні мережі, в якій великі комунікаційні витрати, вартістю пересилання вхідних / вихідних даних додатків знехтувати не можна. Тому планувальник повинен враховувати вартість таких пересилань при призначенні ресурсу, інакше перевага від вибору обчислювального ресурсу,

який може забезпечити високу продуктивність, може бути нейтралізовано високою вартістю доступу до вузла зберігання даних.

11.1 Основні способи планування в Grid системах

В даний час існує три основних способи планування в Grid системі: централізований, децентралізований і ієрархічний [4]. Проаналізуємо кожен з них.

При централізованому способі всі призначені для користувача додатки надсилаються централізованим планувальником (рис.27). Існує єдина черга в централізованому планувальнику для вхідних додатків. Як правило, локальний вузол не має своєї черги, і не виконує функцію планування. Вузол тільки отримує завдання від центрального планувальника і виконує їх. Перевагою такого способу є досить висока ефективність планування, пов'язана з тим, що планувальник в даному випадку має інформацію про всіх доступних ресурсів і вхідних додатків. З іншого боку, централізований планувальник може виявитися вузьким місцем з точки зору як надійності, так і продуктивності. Наприклад, якщо планувальник буде відключений через відмову мережі, то система стає непрацездатною. Збільшення черги додатків може призвести до значного зростання часу обслуговування завдань і в результаті до зниження реальної продуктивності системи. Таким чином, централізований спосіб планування погано масштабується зі збільшенням кількості ресурсів, тобто цей спосіб придатний лише для Grid систем з обмеженим числом вузлів.

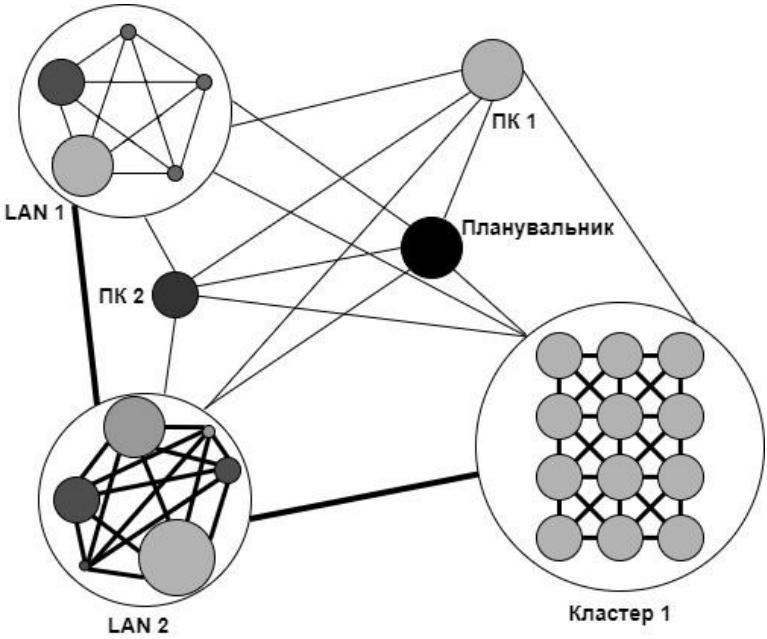


Рис. 27 Модель централізованого планування Grid системи

При децентралізованому способі, функція планування виконується на кожному вузлі системи (рис.28). У цьому випадку будь-який вузол Grid системи працює і як планувальник, і як обчислювальний ресурс. Призначені для користувача програми передаються локальному планувальнику Grid мережі, де виконуються програми. Локальний планувальник відповідає за планування локальних додатків і обслуговує локальну чергу власних додатків. Крім того, він повинен бути в змозі відповісти на запити зовнішніх додатків, приймаючи або відхиляючи їх. Оскільки відповідальність планування розподілена, відмова окремого планувальника торкається роботи інших. Таким чином, децентралізований спосіб забезпечує кращу відмовостійкість і надійність, в порівнянні з централізованим. Але брак глобального планувальника, який знає інформацію про всі додатки і ресурси, зазвичай призводить до низької ефективності. У той же час можливе застосування різної політики планування в різних локальних вузлах. Тому автономність вузла може бути досягнута легко, оскільки локальні планувальники можуть бути спеціалізовані для потреб власника вузла. Даний спосіб може бути застосований для Grid систем з

необмеженою масштабованістю, однак ефективність планування може бути досить низькою.

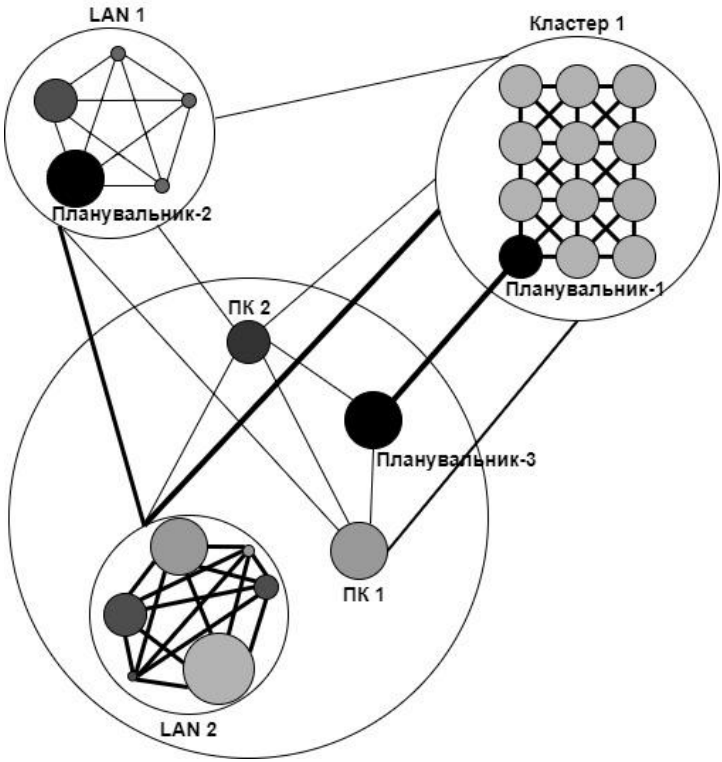
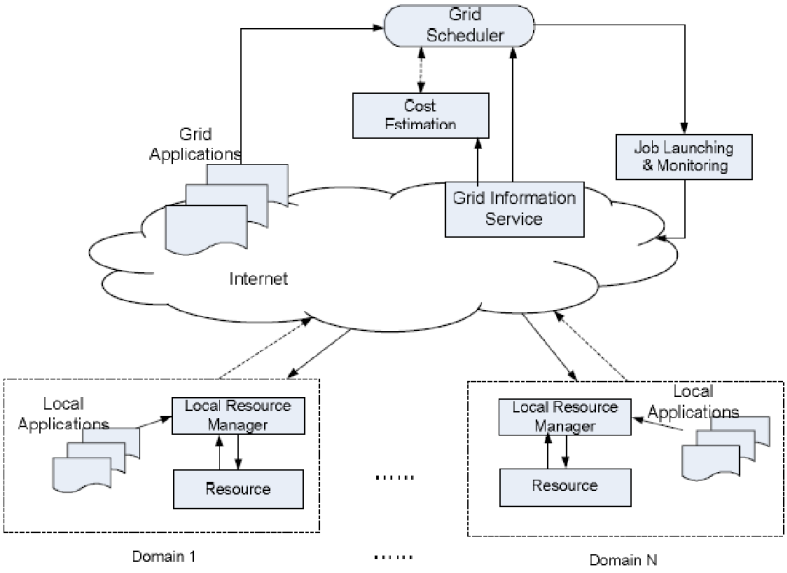


Рис. 28 Модель децентралізованого планування Grid системи

При ієрархічному методі процес планування завдань розподілений на двох рівнях: глобальному і локальному. Розглянемо модель планування Grid системи, яка підтримує ієрархічний підхід, зображену на рис. 29 (а). У даній моделі функціональні компоненти пов'язані двома типами потоку даних: інформаційний потік ресурсу або додатки (пунктирні лінії) і потік управління завданнями або плануванням завдань (суцільні лінії).



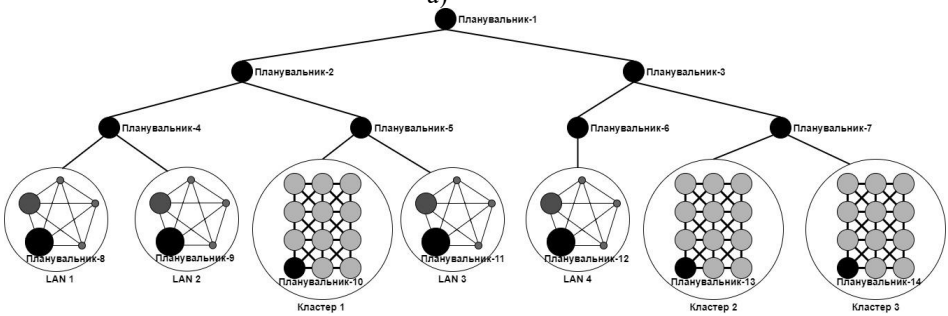


Рис. 29 Модель ієрархічного планування Grid системи

Планувальник Grid системи отримує додатки від користувачів, обирає ресурси для цих додатків відповідно до отриманої інформації від модуля інформаційної служби Grid мережі, і, нарешті, генерує призначення додатку на ресурс, засноване на певних цільових функціях і передбаченій продуктивності ресурсу. Як зазначалося вище, процес обробки завдань відбувається на двох рівнях: глобальному і локальному. На глобальному рівні управління завданнями здійснює метапланувальник Grid (Grid Scheduler), а на локальному - менеджер ресурсів (Local Resource Manager)

(рис.29 б). На відміну від централізованих метапланувальників, Grid системи зазвичай не можуть безпосередньо управляти ресурсами системи, але працюють як брокери або агенти. Розглянемо основні компоненти моделі на глобальному рівні. Інформація про стан доступних ресурсів дуже важлива для метапланувальника Grid системи для того, щоб виконати ефективне планування, особливо з урахуванням неоднорідної і динамічної природи Grid мережі. Роль інформаційної служби Grid системи (Grid information service (GIS)) - забезпечити такою інформацією планувальників Grid. GIS відповідальна за збір і пророкування інформації про стан ресурсу, таку як продуктивність процесора (процесорів) вузлів, розмір пам'яті, пропускна здатність мережі, готовність програмного забезпечення та завантаження вузла в певний період. GIS відповідає на запити для отримання інформації про ресурс або передає інформацію користувачам. Система моніторингу і контролю Globus (The Globus Monitoring and Discovery System (MDS)) - приклад GIS.

Крім інформації про ресурс від GIS, властивості додатків (наприклад, приблизна кількість команд, вимоги до пам'яті та зберігання, залежність підзадач в завданні і обсяги комунікації) і продуктивність ресурсу для різних видів додатків також необхідні для виконання ефективного планування. Отримання зазначених властивостей може бути забезпечено компонентою профілювання додатка (Application profiling (AP)), а вимір продуктивності ресурсу для даного типу завдання - компонентою тестування (analogical benchmarking (AB)). На основі інформації, отриманої від AP і AB, а також використовуваної моделі продуктивності [5] проводиться оцінка вартості планування вузлів-кандидатів на виконання програми, з яких планувальник обирає оптимальний відповідно до цільової функції.

Модуль запуску і контролю (Launching and Monitoring LM) (також відомий як "компонувальник") здійснює передачу додатків на обрані ресурси, пересилаючи вхідні дані і файли для виконання, а в разі потреби і контроль виконання додатків. LM - Globus GRAM (Grid Resource Allocation and Management) - приклад такого модуля .

Локальний менеджер Ресурсів (Local Resource Manager (LRM)) головним чином відповідає за виконання зовнішніх (отриманих від метапланувальника) і локальних завдань, а також передає повідомлення для GIS інформації про стан ресурсів. У межах домену один або множина місцевих планувальників працюють з конкретною локальною політикою управління ресурсами. Приклади таких локальних планувальників включають OpenPBS і Condor. Для збору інформації про локальний ресурс LRM використовують такі інструментальні засоби, як Network Weather Service, Hawkeye і Ganglia .

Таким чином ієрархічний спосіб планування об'єднує переваги як централізованого, так і децентралізованого способів. Він забезпечує високу ефективність планування, підтримує високу надійність і може бути застосований для Grid систем з необмеженою масштабованістю, що складаються з множини суперкомп'ютерних центрів (кластерів і паралельних комп'ютерних систем різної складності). Його основним недоліком є складність. Приклад реалізації ієрархічного способу планування розглянутий авторами у роботі [11]

11.2 Класифікація алгоритмів планування для GRID систем

На підставі відомих класифікацій алгоритмів планування для Grid систем, наведених у роботах [3], [4], пропонуємо розрізняти такі алгоритми за такими ознаками:

• за часом виконання: динамічні, статичні та комбіновані;

• за критеріями оптимізації планування: максимізація користувальницької продуктивності, максимізація системної продуктивності, оптимізація економічних витрат;

• за типом розкладів: однопроцесорний або багатопроцесорний;

• за структурою завдань: послідовний потік завдань, паралельний потік незалежних завдань, паралельно-послідовний потік залежних завдань;

• за методами обслуговування завдань: розподіл ресурсів, розподіл часу;

• за точністю планування: точні та субоптимальні.

Розглянемо докладніше класифікацію по кожному з наведених ознак.

Головна відмінність динамічного та статичного алгоритмів – це час, коли приймаються рішення щодо планування обчислень. Статичне планування виконується до виконання обчислювального завдання, тоді як динамічне планування здійснюється під час виконання завдань.

У разі статичного планування, інформація щодо всіх ресурсів у Grid системи, а також усіх підзадач завдання, імовірно є доступною до процедури планування. На відміну від цього, у разі динамічного планування така інформація може бути відсутня. Динамічне планування застосовується для низки додатків, для яких неможливо визначити заздалегідь їх час виконання, а також число ітерацій обчислення підзавдань. З іншого боку для отримання рішення, близького до оптимального результату, необхідний досить складний аналіз таких параметрів, як структура задачі, що

вирішується, так і структури Grid системи, що вимагає великих тимчасових витрат. При статичному плануванні витрати часу на його роботу менш критичні, ніж при динамічному плануванні. Тому алгоритми статичного планування, як правило, складніші і дають якісніший результат. Алгоритми ж динамічного планування через жорсткі часові обмеження дають менш якісні результати. Для поліпшення якості динамічного планування може використовуватися балансове планування (load balancing), за допомогою якого проводиться перерозподіл завдань по вузлах і тим самим вирівнюється навантаження Grid системи, що призводить до підвищення ефективності роботи мережі та покращення результатів динамічного планування. Як статичне, і динамічне планування широко застосовуються в Grid системах. Наприклад, статичні алгоритми планування наведені у роботах [11]-[13], а динамічні алгоритми планування представлені у роботі [14]. Крім того, динамічні алгоритми реалізовані в таких системах, як Legion та Condor.

Останнім часом намітився розвиток гібридних алгоритмів, що поєднують особливості та переваги як статичного, так і динамічного підходу. У цьому випадку, визначаються підзадачі завдання із заздалегідь відомими параметрами та підзадачі з невизначеними параметрами. Для перших із них застосовується статичне планування, а для других – динамічне. Приклад такого підходу наведено у роботі [15].

Як зазначалося вище, при плануванні в Grid системах найчастіше застосовується один з наступних критеріїв оптимізації - максимізація продуктивності користувача, максимізація системної продуктивності або оптимізація економічних витрат.

У разі застосування першого критерію оптимізації, алгоритм планування повинен забезпечити максимальну реальну продуктивність Grid системи при виконанні конкретної задачі. При цьому основна мета – отримати мінімальний час виконання цієї задачі (makespan) та забезпечити максимальне прискорення у системі. Тому найкращий результат дають алгоритми, які при виборі ресурсу для конкретної програми не розглядають інші задачі, що надходять.

У разі застосування другого критерію алгоритм планування повинен забезпечити максимальну реальну продуктивність Grid системи при виконанні множини задач, що надходять (системну продуктивність) у Grid систему від низки користувачів. Для досягнення зазначеної мети необхідно максимізувати пропускну здатність системи та мінімізувати середній час виконання програм.

У разі застосування третього критерію оптимізації алгоритм планування повинен забезпечити оптимальну економічну оцінку (Quality of Services (QoS)) , наприклад, таку як, мінімальна вартість, яку користувач заплатить за використання ресурсу, що забезпечує заданий термін

виконання програми. При цьому для кожного ресурсу визначено його вартість та продуктивність.

За типом розкладів алгоритми планування поділяються на однопроцесорні чи багатопроцесорні. У першому випадку всі задачі виконуються послідовно на однопроцесорному вузлі, у другому випадку незалежні задачі або підзадачі однієї й тієї самої задачі одночасно обробляються на різних процесорах (комп'ютерах) паралельної системи, кластера або розподіленої системи вузла Grid системи.

Для реалізації однопроцесорних розкладів використовується або єдина задача, або послідовний потік завдань. Для реалізації багатопроцесорних розкладів застосовуються додатки у вигляді паралельного потоку задач (множина незалежних задач) або додатки у вигляді паралельно-послідовного потоку підзадач (множина залежних підзадач). Зрозуміло, перший варіант є частковим випадком другого. Тому представляє особливий інтерес розробка багатопроцесорних алгоритмів планування додатків у вигляді паралельно-послідовних потоків підзадач.

Розглянемо класифікацію алгоритмів планування для GRID систем з точки зору типів завдань, зображену на рис.30

Рис.30 Класифікація алгоритмів планування з точки зору типів завдань

За методами обслуговування завдань існує дві групи алгоритмів планування.. Алгоритми першої групи засновані на ідеї поділу простору ресурсів (space-sharing) між завданнями, згідно з якими кожне завдання отримує необхідний обсяг ресурсів на потрібний час в ексклюзивному режимі. Алгоритми ж другої групи використовують розподіл часу процесорів (time-sharing) між декількома завданнями. Це означає, що кілька

завдань у довільний час можуть розділяти одні й самі обчислювальні ресурси.

Загальна класифікація алгоритмів планування завдань в Grid системах зображена на рис.31

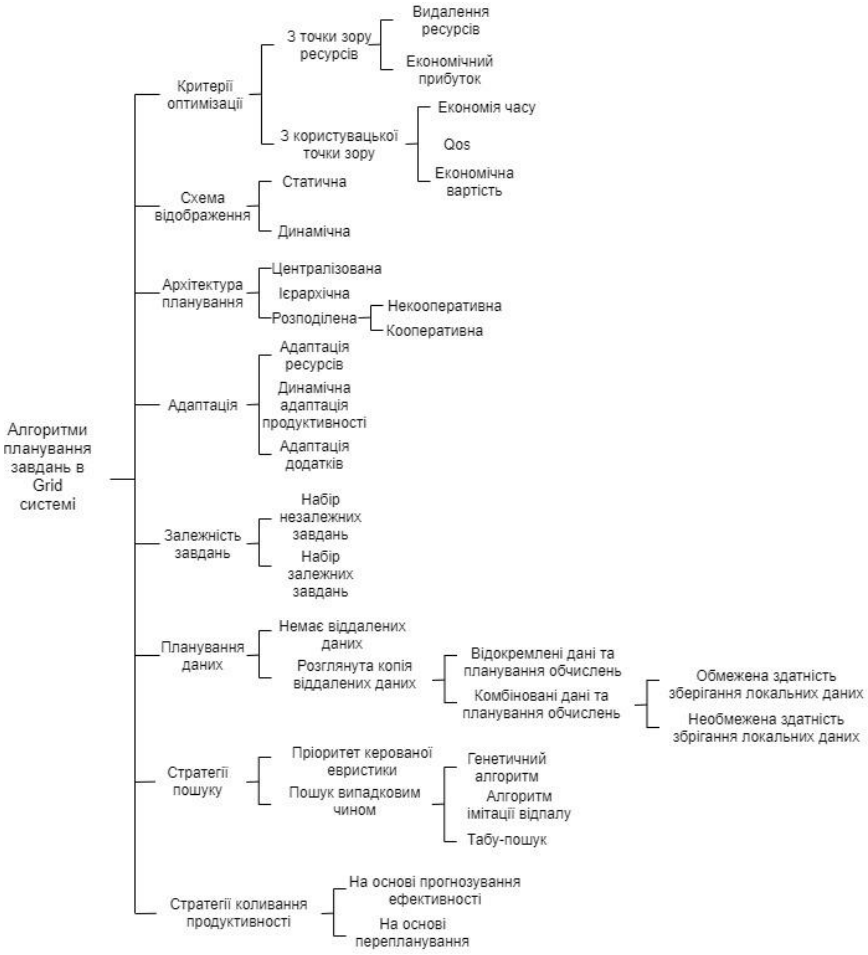


Рис.31 Систематика алгоритмів планування завдань в Grid системах

Вирішення проблеми планування для хмарних обчислень достатньо схоже на планування Grid системах . Класифікація методів планування для хмарних обчислень зображена нп рис. 32.

Рис. 32 Класифікація методів планування в середовищі хмарних обчислень

12. Особливості планування в паралельних базах даних.

12.1 Причини (мотиви) виникнення РБД і ПБД

1. Багато важливих проблем вимагають використання великих баз даних (БД) в реальному часі. Для цього одним з варіантів скорочення часу виконання запитів застосування засобів індексування, хешування, попереднього сортування.

Однак, не завжди заздалегідь відомий набір запитів, більш того, існують випадки, коли запити породжуються на основі результатів уже виконаних запитів, що має місце в сучасних системах підтримки прийняття рішень. У цьому випадку скорочення часу виконання запитів можливо тільки за рахунок їх паралельного виконання.

Іншою обставиною, що сприяв розвитку паралелізму і розподілу БД, є масове застосування комп'ютерних мереж і паралельних КС. Стимулюючими ідеями розвитку розподілених БД (РБД) і паралельних БД (ПБД) є підвищення продуктивності, надійності і масштабованості перерахованих апаратних платформ. Розглянемо визначення РБД і ПБД.

РБД - це сукупність багатьох взаємопов'язаних баз даних, розподілених у комп'ютерній системі, апаратно являє собою комп'ютерну мережу.

ПБД - це сукупність багатьох взаємопов'язаних баз даних, розподілених в багатопроцесорній системі (паралельні КС).

Розглянемо спільні та відмінні характеристики РБД і ПБД.

Загальні характеристики:

1. РБД і ПБД - це саме БД, а не колекція файлів, індивідуально зберігаємих в різних вузлах системи. Розподілені дані пов'язані між собою за допомогою деякого структурного формалізму (такий як реляційна модель), а для доступу до них є єдиний високорівневий інтерфейс.

2. Розподіл даних по вузлах невидимий для користувачів. Це властивість називається прозорістю. Прозорість означає, що користувачі мають справу з єдиним логічним образом БД і здійснюють доступ до розподільних даних так само, як вони б зберігалися централізовано.

3. Комп'ютерна система складається з множини вузлів прийому запитів (можливо порожнього) і непорожньої множини вузлів даних.

Вузли даних мають засоби зберігання власне даних, а вузли прийому запитів – не мають, на них лише прийоми, що реалізують інтерфейс користувача для доступу до даних.

Відмінні характеристики РБД і ПБД:

1. Вузли комп'ютерної системи для РБД логічно є незалежними комп'ютерами, пов'язані мережею, на яких встановлені власні ОС і можуть виконувати незалежні додатки.

Таким чином, комп'ютерна система для РБД має слабозв'язаний характер.

Вузли ж ПБД - це процесори багатопроцесорних КС, керовані єдиною ОС.

2. У ПЗ БД можуть бути передбачені три види паралелізму:

- міжзапитний (одночасне виконання запитів, що належать різним транзакціям (неподільна одиниця виконання операції над БД, в результаті якої вона залишається в коректному стані);

- внутрішньозапитний (одночасне виконання операцій належать одному запиту);

- внутрішньоопераційний (одночасне виконання безлічі субопераций належать одній операції).

РБД, як правило, можуть підтримувати перші два види паралелізму, а ПБД - усі три.

12.2 Архітектури КС ПБД

ПБД можуть використовувати такі структури КС (рис.28):

- з роздільними оперативною і зовнішньою (дисковою) пам'яттю (1);

- з роздільною зовнішньою пам'яттю і розподіленою оперативною пам'яттю (2);

- з розподіленими оперативною і зовнішньою пам'яттю (3);

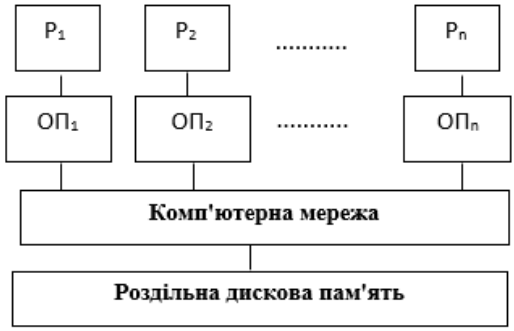
- комбіновані структури, в яких частина пам'яті розподілено, а частина розділяється групами (кластерами) процесорів; Омега з 2-х кластерів, кожен з яких відноситься до другого типу структури.

Диски використовуються для зберігання даних, а тип оперативної пам'яті відображає спосіб обміну даними.

IBM/370, Diqital VAX,

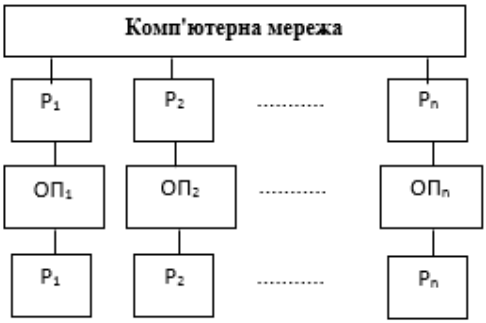
Segment Symmevry

(1)



IBM Syspbx

(2)



Teradata

Tandem,

nCule, VAXcluster

(3)

ПБД **Teradata**, Gamma,

Tandem, Bubba, Arbre ...

Рис.28 Варіанти структур КС для ПБД

12.3 Паралелізм в ПСУБД

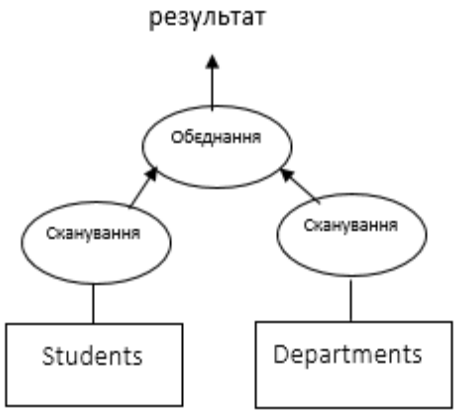
Модель даних SQL була спочатку запропонована з метою збільшити продуктивність програмістів за допомогою непроцедурної мови баз даних. Додатковою перевагою стала незалежність даних, так як програмісти не визначають, яким чином обробляти запит, то програми на SQL не втрачають актуальності в міру розвитку логічних і фізичних схем баз даних.

Несподіваною перевагою реляційної моделі є паралелізм. Реляційні запити просто створені для паралелізму, оскільки в дійсності вони є реляційними операторами, які застосовуються до дуже великих наборів даних.

Реляційні запити можуть виконуватися у вигляді графа потоку даних, а оскільки на вході СУБД запит подається в декларативному вигляді (скажімо - SQL), то є велика свобода по вибору способу його виконання на внутрішньому рівні СУБД .

Представивши запит у вигляді графа реляційних операцій, можна виділити незалежні ланцюжки, найчастіше пов'язані з обробкою різних таблиць. Природно, що маючи необхідні ресурси, виникає можливість виконати такі ланцюжки паралельно, прискоривши таким чином виконання запиту в цілому. Звернемося до прикладу:

```
SELECT  * FROM  students, departments
    WHERE  students.id = departments.std_id
```



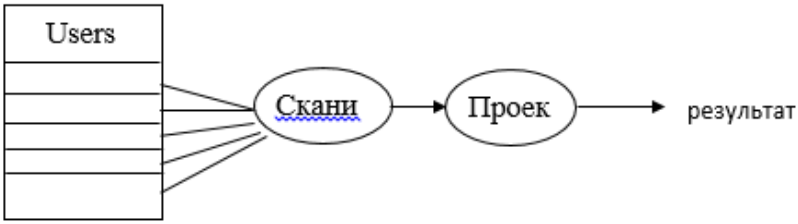
Можна помітити, що операції сканування таблиць Students і Departments не залежать по даними один від одного, отже, нам ніщо не завадить виконати їх паралельно. Даний вид паралелізму в ПСУБД називається незалежним.

Реляційні операції мають властивість замкнутості, тобто вхідними даними будь-якої операції є відношення (таблиці) і в результаті виконання операції ми отримуємо відношення (таблицю). Таким чином, два оператора можуть працювати паралельно, якщо ми вихід одного оператора направимо на вхід іншого, що в ідеалі забезпечує прискорення в два рази.

Візьмемо приклад:

SELECT name FROM users WHERE name = "Bobby"

Цей запит можна представити у вигляді такого графа:



Такий вид паралелізму називається конвеєрним, або вертикальним (якщо розгорнути малюнок на 90 градусів). Його недоліком є те, що графи реляційних операторів рідко мають велику довжину, до того ж, деякі оператори не йдуть на вихід, поки не здійсниться все введення (join, sort - в найпростішому варіанті). Дуже часто час виконання одного оператора більше інших, тому якщо такий оператор знаходиться в конвеєрному ланцюжку, то виграш, отриманий від конвеєризації, буде невеликий.

Якщо ж розділити вхідні дані оператора за наявними дискам (забезпечити можливість паралельної обробки шматочків таблиць), то можна розбити реляційний оператор на кілька незалежних, кожен з яких, працює з частиною даних. Візьмемо запит з попереднього прикладу, але уявімо, що таблиця Users розбита на три рівні частини, при чому обробка кожної частини може виконуватися незалежно від інших.

В цьому випадку граф виконання запиту буде виглядати таким чином:

Цей вид паралелізму називається рознесеним, або горизонтальним (якщо розгорнути малюнок на 90 градусів). Рознесений паралелізм пропонує кращі можливості для прискорення і розширюваності. Беручи великі реляційні оператори і розділяючи їх входи і виходи за принципом "розділяй і володарюй", можна перетворити одну велику роботу в безліч незалежних невеликих робіт. Однак, як видно з прикладу, поділ призводить до необхідності введення додаткового оператора злиття, який об'єднує кілька паралельних потоків даних в один.

До того ж слід зазначити, що для реалізації таких операцій як з'єднання і сортування, наприклад, буде потрібна розробка нових паралельних алгоритмів, ефективно використовують такий вид виконання, оскільки в класичній нотації вони не допускають розбиття вхідних потоків даних і для виконання вимагають повного завершення введення.
Таким чином, конвеєрний і незалежний паралелізм можна віднести до міжопераційного паралелізму, а розділений паралелізм - внутрішньоопераційного, оскільки в цьому випадку реляційний оператор розбивається на безліч незалежних.

Ключем до використання внутрішньоопераційного паралелізму є поділ даних, що зберігаються в БД.

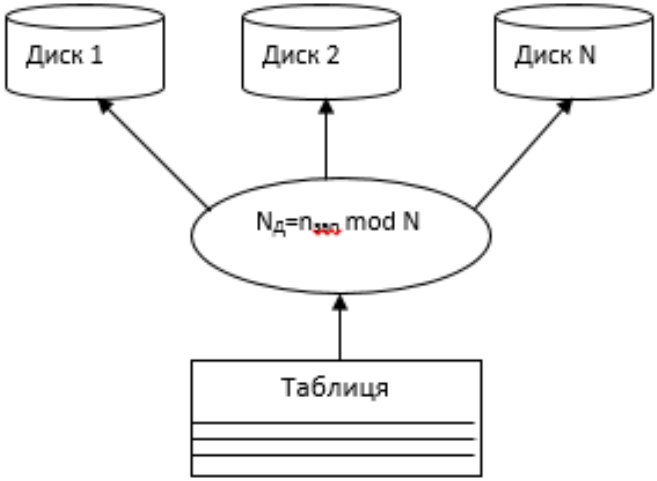
Способи поділу даних.
тиражування (реплікація);
фрагментація:
- горизонтальна за рахунок селекції;
- вертикальна за рахунок проекції.

Способи фрагментації:
- по кільцю;
- поділ з хешем (за значенням хеш-функції);
- поділ на основі діапазону значень (дана кластеризація по вживаності).

Поділ даних. Поділ відношень передбачає розподіл його кортежів між декількома дисками. Поділ даних йде від централізованих систем, які змушені розділяти файли, тому що файл занадто великий для одного диска або тому що неможливо забезпечити прийнятну швидкість доступу до файлу на одному диску. Поділ даних використовується в розподілених базах даних, коли частини відношень вміщаються в різні вузли мережі. Поділ даних дозволяє паралельним системам баз даних використовувати пропускну здатність введення / виведення декількох дискових пристроїв шляхом паралельного читання і запису. Такий підхід забезпечує більш широку пропускну здатність введення / виведення, ніж у систем, що використовують RAID (дискові масиви), без застосування будь-якої спеціальної апаратури.

Найпростіша стратегія поділу полягає в тому, що кортежі (записи) таблиці розподіляються по дискам за принципом кільця. Це кращі результати в тому випадку, якщо призначені для користувача запити, вимагають повного перегляду всіх записів відношень, оскільки рівномірно

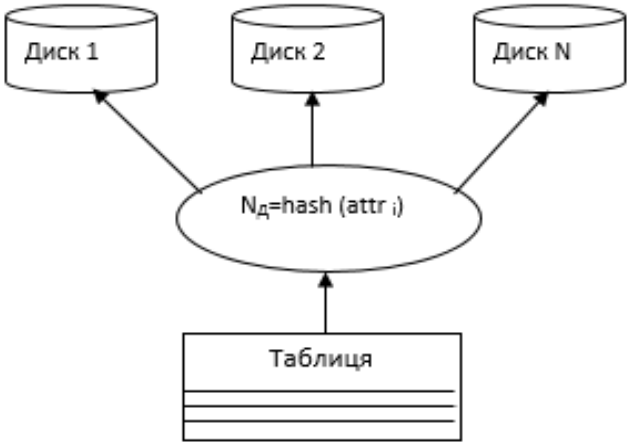
розподіляє кортежі по дискам, ніж балансує їх завантаження і призводить до максимального прискорення.



У разі ж асоціативного перегляду таблиць (пошуку всіх записів, які відповідають умові) зростає можливість перекосу при обробці, тобто якщо всі потрібні кортежі знаходяться на 1-м диску обробка все одно буде вестися на всіх дисках, що дуже неефективно.

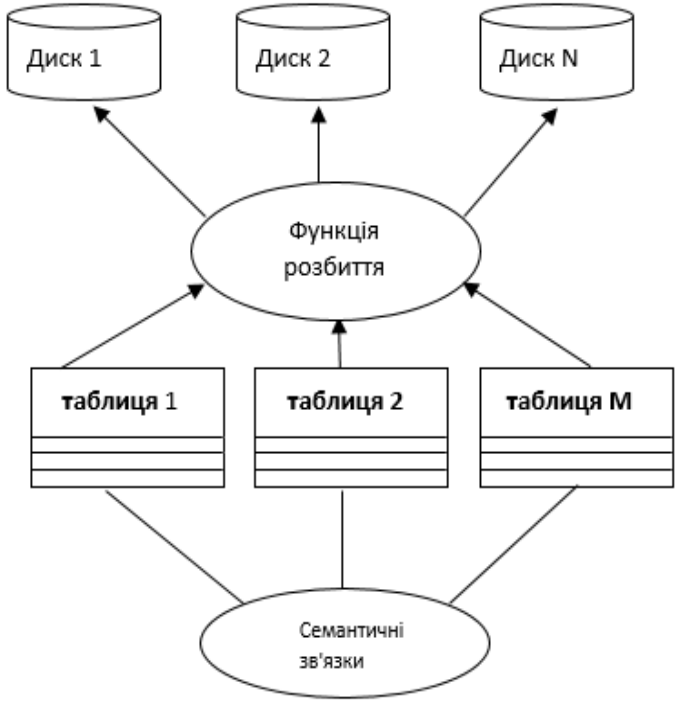
У цих випадках підходить поділ хешем. При цьому кортежі (записи) розподіляються залежно від хеш-функції, застосованої до значення атрибута кожного кортежу. Результатом виконання функції буде конкретний диск, на якому розміститься кортеж.

При такому поділі, асоціативний пошук буде направлений до конкретного диску, що виключить витрати на запуск і пошук інформації на інших дисках, при чому не на шкоду послідовному перегляду всіх кортежів відношень. Основною проблемою в цьому випадку є вибір такої хеш-функції, яка б, по-перше, розбивала кортежі якомога більш рівномірно по дисках, а по-друге забезпечувала ефективний асоціативний пошук, що в цілому не так просто. Так само не можна забувати, що отримані переваги працюють тільки у разі запитів, пов'язаних з атрибутом, за яким здійснювалось хешування, в інших варіантах запитів виграшу не буде, тому важливо, щоб в такій ситуації хешування не приводило до великих втрат в продуктивності (тобто забезпечити якомога більш рівномірний розподіл записів).



У системах БД дуже часто при поділі увага приділяється семантичним зв'язкам між таблицями. Так, БД містить таблиці студентів і факультетів (студент вчиться на факультеті, а на факультеті навчається багато студентів), таблицю студентів можна розділити на підставі лексикографічного порядку (за алфавітом), а можна розділити на підставі факультетів, в яких вони навчаються. Подібний метод використовується в системах БД для індексації і розміщення записів семантично пов'язаних таблиць на фізичних сторінках для прискорення операцій читання / запису. Називається він дуже просто – кластеризацією.

Як видно, такий поділ багато в чому пов'язаний зі специфікою прикладної програми, що використовує БД і може дати дуже гарні результати при паралельній обробці.



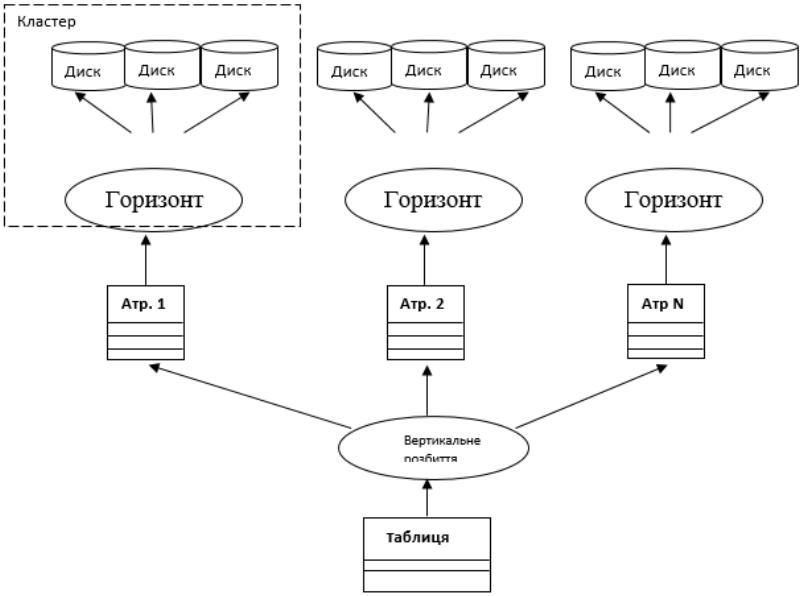
Однак такий розподіл може призвести до перекосу даних, тобто коли вся інформація потрапляє в один розділ (всі студенти знаходяться в БД вчаться на електрогазозварювальному факультеті, або ж всі студенти перебувають у родинних стосунках і мають однакові прізвища) і до перекосу виконання, коли вся обробка проводиться на одному розділі. У такій ситуації можна накопичувати статистику звернень до кожного кортежу відношень і на її підставі робити динамічний перерозподіл даних,

щоб збалансувати завантаження кожного дискового пристрою (розділу). Такий метод використаний в ПСУБД Bumba .

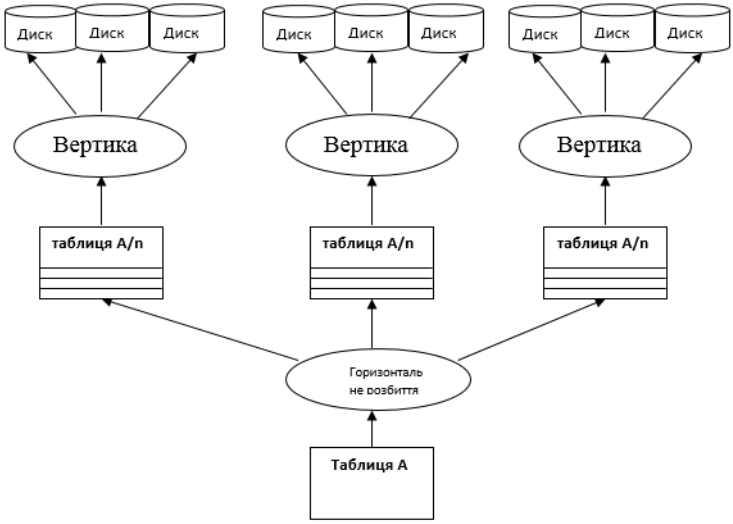
Негативним наслідком кластерного поділу є необхідність введення нових призначених для користувача функцій, призначених для розподілу даних, в результаті чого відбувається відхід від універсальності в сторону проблемно орієнтованої обробки таблиць, або ведення і накопичення складної статистичної інформації з подальшим перерозподілом даних.

У найпростішому варіанті розбиття можна використовувати хеш-функцію, яка б рівномірно розподіляла дані у випадковому порядку, такий варіант, як показує практика, є досить ефективним і простим в реалізації.

У разі використання розбиття методом проектування, таблиці БД розподіляються по дисковим пристроям таким чином, щоб збільшити ефективність виконання запитів з проекцією, які зустрічаються досить часто. Розбиття відбувається наступним чином: з вихідної таблиці вибираються всі значення певного атрибута і поміщаються на диск, потім - значення іншого атрибута - на інший диск, і т.д. по всіх атрибутам. Логічно, що такий метод дуже хороший при обробці запитів з проекцією (SELECT name, FIO FROM ...), оскільки з обробки відразу ж виключаються всі атрибути, які не включені в проекцію. Найгіршим варіантом в цьому випадку буде виконання запитів без умови проектування (SELECT * FROM...), оскільки буде потрібна обробка всіх дисків, на яких розподілена БД. Як недолік можна також відзначити той факт, що при такій розбивці задіюється кількість дисків, яка дорівнює кількості атрибутів у відношенні, тому обробка таблиці (введення / виведення), що складається з двох атрибутів, і більйона записів буде прискорена всього в 2 рази (т. я . задіяно 2 диски), а хотілося б більше. Але рішення існує! Необхідно використовувати комбінований метод, тобто вертикально-горизонтальний або горизонтально-вертикальний або перпендикулярний (в крайньому випадку). Дуже важливо відзначити, що при цьому переваги обох способів розбиття додаються, і ми отримуємо одну велику перевагу. Якщо визначити безліч дисків, на яких розміщується фрагмент таблиці, отриманий таким способом, як кластер, то визначення місцезнаходження даних, необхідних для обробки, буде відбуватися у дві фази: визначення кластера (по первинному методу) і визначення конкретного диска всередині кластера (по вторинному методу), при чому використання такого синтезу методів відбувається без шкоди ефективності кожного окремо. Основною метою, що досягається цим методом, є максимальне звуження множини дисків, задіяних при виконанні конкретного запиту. Також важливим для впровадження паралелізму є те, що розбиття може проводиться на будь-яку кількість дискових пристроїв, доступних в системі, а не обмежується, як у випадку простої проекції.



Вертикально-горизонтальне розбиття



Горизонтально-вертикальне розбиття

Таким чином, хоча концепція поділу проста і легко здійсненна, вона породжує ряд нових питань, що стосуються проектування БД. Для кожного відношення тепер має бути присутня стратегія поділу і набір дискових фрагментів. Збільшення ж фрагментації може призвести до зменшення часу відповіді для конкретного запиту, але збільшити пропускну здатність системи. При послідовних переглядах даних час відповіді зменшується, оскільки для виконання використовуються велике число дисків і процесорів. А при асоціативних переглядах час відповіді може істотно зменшиться, тому що в кожному вузлі зберігається менше число кортежів і, отже, зменшується розмір індексу, який повинен бути використаний для пошуку.

Однак, починаючи з деякого моменту, при подальшому поділі час відповіді на запит починає зростати. Це відбувається, коли час запуску запиту в вузлі стає істотною часткою реального часу виконання запиту, тому при розбитті важливо знайти золоту середину, при якій досягається максимальна ефективність.

12.4 Загальна схема виконання запитів

На рис.29 зображена загальна схема виконання запитів. Для реляційних систем оптимізація є як проблемою, так і способом підвищення продуктивності. Реляційні мови запитів забезпечують високорівневий "декларативний" інтерфейс для доступу до даних, що зберігаються в реляційних базах даних. Згодом з'явилася мова SQL як стандарт реляційних мов запитів. Двома ключовими підкомпонентами компонента обчислення запитів в SQL-орієнтованій системі баз даних є оптимізатор запитів і підсистема виконання запитів [2].

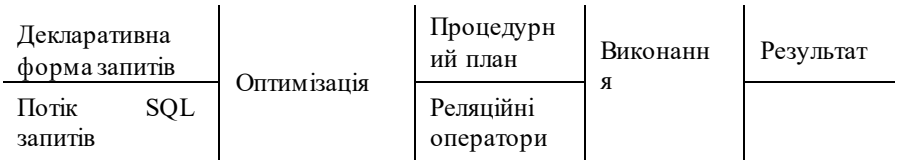


Рис.29 Загальна схема виконання запитів

Оптимізатор запитів відповідає за генерацію введення для підсистеми виконання. Він приймає на вході уявлення SQL-запиту після граматичного розбору і відповідає за генерацію ефективного плану виконання даного SQL-запиту, виходячи з тих планів, які складають

простір можливих планів виконання. Завдання оптимізатора нетривіальне, тому що для заданого SQL-запиту може існувати велика кількість можливих дерев операцій:

• Алгебраїчне подання цього запиту може бути перетворено в багато інших логічно еквівалентних алгебраїчних уявлень; наприклад, Join (Join (A, B), C) = Join (Join (B, C), A)

• Для даного алгебраїчного уявлення може існувати багато дерев операцій, що реалізують вираження алгебри; наприклад, в системі баз даних зазвичай підтримується кілька алгоритмів з'єднання.

Сформулювати задачу оптимізації можна таким чином: для даного запиту q, простору можливих планів виконання E і оцінної функції cost (p), що визначає вартість виконання кожного плану p належить E, необхідно знайти такий план Pnice, що виконує q, у якого оцінна функція cost (Pnice) має мінімальне значення.

Таким чином, до оптимізації запитів можна ставитися як до складної пошукової проблеми. Для того щоб вирішити цю проблему, нам потрібно забезпечити:

• Простір планів (простір пошуку).

• Метод оцінки вартості, щоб можна було оцінити кожен план в кожному просторі пошуку. Інтуїтивно, це оцінка ресурсів, необхідних для виконання плану.

• Алгоритм перебору, який може здійснювати пошук в просторі планів виконання.

Бажаним оптимізатором є такий, в якому:

1) простір пошуку включає плани з низькою вартістю;

2) метод оцінок є точним;

3) алгоритм перебору ефективний.

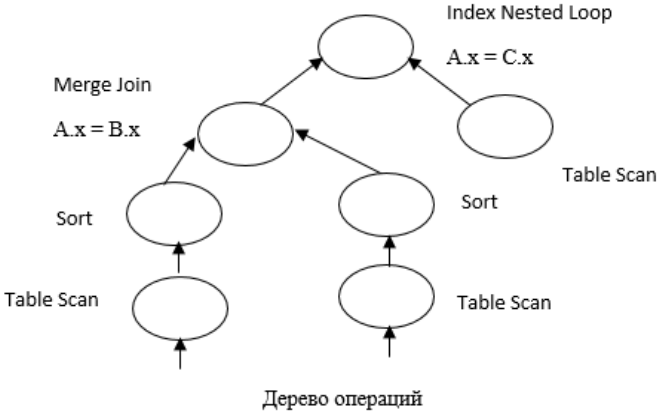
Кожне з цих трьох завдань нетривіальне, і через це побудова хорошого оптимізатора є величезною роботою.

Підсистема виконання запитів реалізує набір фізичних операцій. На вхід кожної операції надходять один або кілька потоків даних, і на виході формується потік даних. Прикладами фізичних операцій є сортування, послідовне сканування, індексне сканування, з'єднання методом вкладених циклів і з'єднання сортуванням і злиттям. Ці операції називаються фізичними, оскільки вони не обов'язково пов'язані один до

одного з реляційними операціями. Найпростіше представляти фізичні операції як шматки коду, які використовуються в якості будівельних блоків для забезпечення можливості виконання SQL-запитів. Абстрактним поданням такого виконання є дерево фізичних операцій. Дуги в дереві операцій представляють потоки даних між операціями. Підсистема виконання відповідає за виконання плану, що призводить до формування відповідей на запити. Отже, можливості підсистеми виконання запитів визначають структуру допустимих дерев операцій.

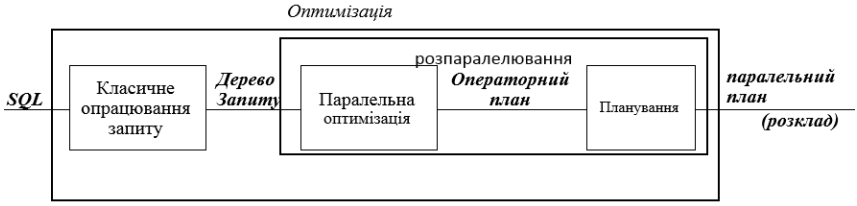
Приклад:

SELECT * FROM A,B,C WHERE A.x=B.x AND A.x=C.x



Дерево операций

Що стосується нашої теми, подібна схема обробки запитів зазнає певних змін, пов'язаних в першу чергу з розпаралелюванням операцій, а оскільки така функція покладається на оптимізатор, модернізована схема буде виглядати таким чином:



Класична обробка запитів

На етапі класичної обробки запиту виконується:

- Перетворення запиту з декларативної форми в дерево операцій, в термінах реляційної алгебри, тобто вершинами дерева є класичні реляційні оператори (Select - Project - Join і т.д.).

- Виконання деяких перетворень і оптимізації, спрямованих на представлення запиту в найбільш зручному вигляді для подальшої обробки.

Декларативна форма подання запитів (SQL) зручна в першу чергу для користувача, але не годиться для внутрішнього уявлення, оскільки володіє надмірністю (один і той же запит можна представити багатьма способами) і несприятлива для будь-яких перетворень (відсутній формальний мат. апарат), звідси - необхідність перетворення у внутрішню, більш зручну форму. Перша частина класичної обробки виконується на підставі формалізованих алгоритмів, і включає в себе лексичний і синтаксичний аналіз запиту в формі SQL з подальшим формуванням операційного дерева реляційних операцій. Вибір саме такої форми внутрішнього представлення заснований на існуванні реляційної алгебри, яка надає зручний математичний апарат для виконання всіляких перетворень і маніпуляцій із запитом. Варто також відзначити, що таке уявлення є проміжно-абстрактним (ми абстрагуємося від фізичної реалізації реляційних операцій), а вже для виконання необхідне подання запиту у вигляді фізичних операцій, що реалізуються віртуальною машиною СУБД (про це далі).

Друга частина класичної обробки включає в себе виконання деякої оптимізації, яка "гарантовано є хорошою", незалежно від реальних даних, що зберігаються в БД, і шляхів доступу до них. Суть у тому, що всі запити реляційної мови дозволяють висловити кількома різними способами. Продуктивність виконання запиту не повинна залежати від форми запису запиту, яку обрав користувач. Тому перше, що необхідно зробити - перевести запит в еквівалентну канонічну форму. Метою такого перетворення є виключення зовнішніх відмінностей в еквівалентних уявленнях запиту і, що більш важливо, пошук подання запиту, в деякому сенсі більш ефективного в порівнянні з вихідним поданням.

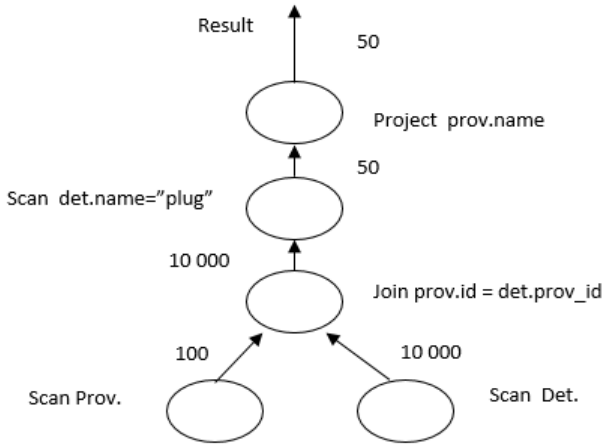
Канонічна форма: Нехай Q - множина запитів, і нехай існує поняття про еквівалентність запитів (запити q1 і q2 еквівалентні тоді і тільки тоді, коли дають ідентичні результати). Кажуть, що підмножина C множини Q є підмножиною канонічних форм для запитів з Q тоді і тільки тоді, коли кожному запиту q з Q відповідає тільки один запит c з C. Тоді кажуть, що запит c є канонічною формою запиту q.

Для виконання такого перетворення оптимізатор використовує добре відомі правила перетворення, або закони (у нашому випадку - правила

реляційної алгебри). Наприклад, операції об'єднання, перетину і з'єднання є комутативними і асоціативними. Подібні перетворення і правила використовуються у відносно арифметичних і логічних виразах, що зустрічаються в запитах. Розглянемо приклад подібного перетворення:

Нехай існує БД обліку деталей, що складається з таблиць постачальників і деталей. Деталь обов'язково має свого постачальника, а постачальник поставляє певну кількість деталей (зв'язок один - до - багатьох). Необхідно отримати імена всіх постачальників, що поставляють втулки для глибинних насосів:

```
SELECT providers.name FROM providers,details
        WHERE providers.id = details.prov_id
        AND details.name = "plug"
```



Отриманий граф виконання запиту ніяк не є оптимальним, оскільки вибірка після операції з'єднання знаходиться явно не на своєму місці. Припустимо у нас 10 000 деталей (з них - 50 втулок) та 100 постачальників, підрахуємо приблизну вартість виконання такого графа:

1. Вибірка всіх деталей і постачальників з таблиць: 100 + 10000 = 10100

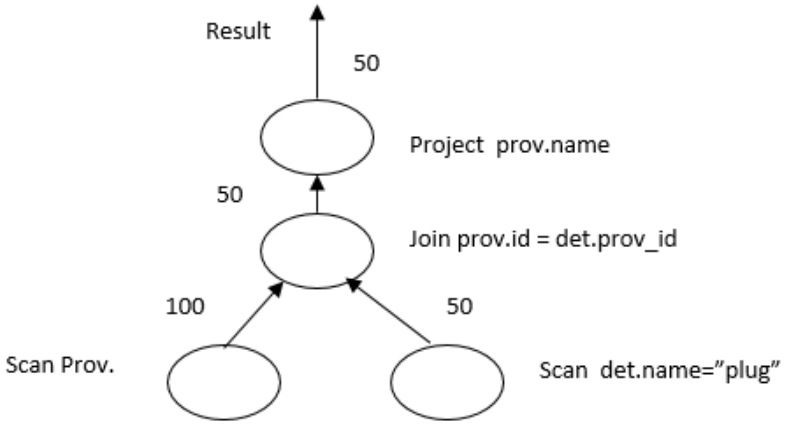
2. З'єднання відношень det і prov по атрибуту prov_id: 100 * 10000 = 1000000 (з'єднання вимагає перегляду кожної деталі по всім постачальникам)

3. Вибірка всіх деталей б відповідала умовам: 10 000

4. Проекція: 50

5. Загальна вартість - 1020150 у.о.

Сумісна вибірка деталей відповідала б умовам з вибіркою з таблиці:



Підрахуємо вартість у цьому випадку:

1. Вибірка всіх деталей і постачальників з таблиць: 100 + 10000 = 10100

2. З'єднання відношень det і prov по атрибуту prov_id: 100 * 50 = 5000 (з'єднання вимагає перегляду кожної деталі по всім постачальникам)

4. Проекція: 50

5. Загальна вартість - 15150 у.о.

Отже, отримане прискорення - 70 разів. З цього прикладу можна зробити висновок, що операція вибірки - звужує вихідний потік даних, тому, якщо є можливість, її треба якомога частіше застосовувати, і прагне розташувати потік в нижній частині дерева.

При обробці логічних виразів виконується їх приведення до КНФ (кон'юнктивна нормальна форма). Перевага КНФ полягає в тому, що КНФ-вираз істинний, тільки якщо істинні всі його окремі часткові кон'юнкції. Тому, при виконанні послідовно (або паралельно) обчислюються кон'юнкції в будь-якому порядку (від простих до складних), і якщо значення хоча б однієї з кон'юнкцій звернеться в 'брехня', то виконання всього висловлювання можна припиняти.

Існує ще один прийом, пов'язаний з перетворенням логічних виразів - транзитивне замикання предикатів. Припустимо, що A і B - атрибути двох різних відношень, тоді умова:

A> B AND B> 3 еквівалентно висловом:

A> B AND B> 3 AND A> 3, і тому може бути перетворена в цей вислів. Дана еквівалентність базується на тому, що операція> транзитивна. Подібне перетворення вельми корисне, тому що дозволяє створити додаткову вибірку (за допомогою умови A> 3), про доцільність якої було сказано раніше.

У разі арифметичних виразів, стратегія обробки аналогічна стратегії, використовуваної в компіляторах. По-перше, необхідно постаратися виконати максимум обчислень на етапі оптимізації (висловлювання на кшталт: (40 * 600- 2000) /8.432)), а по-друге, на підставі законів трансформації - звести до більш оптимальної форми (A * B + A * C = A * (B + C)).

Крім еквівалентних перетворень, існують ще семантичні перетворення, які розглянемо далі.

Семантичні перетворення: Перетворення, коректне в силу певної умови цілісності, називають семантичним перетворенням, а оптимізацію, отриману в цьому випадку - семантичною оптимізацією. Отримується в результаті подання запиту, який не є еквівалентним початковому, але гарантується, що результат виконання перетвореного запиту збігається з результатом запиту в початковій формі при дотриманні обмежень цілісності, існуючих в базі даних .

Візьмемо попередній приклад і припустимо, що БД обліку деталей задовольняє умові: "Всі гайки поставляє Петренко". У цьому випадку запит: "Вибрати всіх постачальників, що поставляють гайки",

```
SELECT  * FROM  providers,details
    WHERE  providers.id = details.prov_id
    AND  details.name = “гайка”
```

Зводиться до такого виду:

```
SELECT  * FROM  providers
    WHERE  providers.name = “Петренко”
```

Можна однозначно сказати, що другий запит явно має меншу вартість, ніж вихідний. Однак існує проблема накопичення інформації про подібні умови цілісності ("Всі гайки поставляє Петренко"), бо вона пов'язана з семантикою БД. Рішення може бути отримано у вигляді

надання функцій по занесенню таких умов в СУБД проектувальнику БД, або у вигляді постійного накопичення знань про БД в процесі роботи з нею (типу БЗ).

Таким чином, етап класичної обробки має велику важливість, оскільки стоїть першим в ланцюжку перетворення запиту від входу в систему до виконання, внаслідок чого, від нього багато в чому залежить ефективність подальших перетворень.

12.5 Паралельна оптимізація

Етап паралельної оптимізації є одним з найбільш трудомістких. Основними завданнями, які розв'язуються на цьому етапі є:

• формування множини паралельних планів виконання у термінах функціональності віртуальної машини СУБД (система команд);

• оцінка ефективності і вибір оптимального плану.

Як було зазначено раніше, на етапі класичної оптимізації ми абстрагувалися від фізичної реалізації реляційних операцій на конкретній СУБД. На цьому ж етапі якраз і відбувається вибір таких низькорівневих процедур, а в контексті ПБД, проводиться визначення операцій допускають розпаралелювання (види одного розглянуті раніше) і формування паралельного плану виконання, який здатна реалізувати підсистема виконання запитів. Відбувається дана дія таким чином: починаючи знизу дерева, в напрямку до кореня, послідовно з вихідного графа обираються вершини, після чого, для кожної, на підставі:

- властивостей вхідних потоків даних (впорядкованість, потужність (вага пересилання))

- особливостей вершин джерел і вершин приймачів даних для операції (взаємозалежність операцій)

- метаданих про відношення (наявність індексації атрибутів, кардинальні числа відношень, статистична інформація)

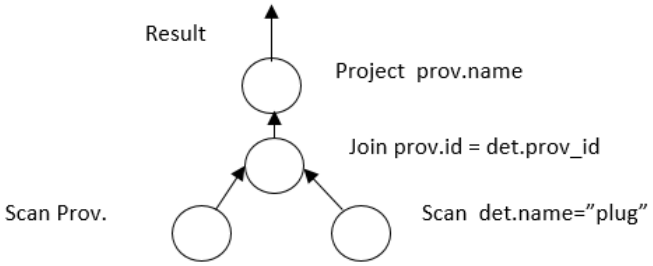
- інформації про розподіл даних відношень по дискам

виробляється генерація можливих (найкращих з можливих) варіантів фізичної реалізації, при чому одній реляційній операції може відповідати група взаємопов'язаних фізичних, так наприклад, операція сканування відношень на паралельній системі перетворюється в множину сканувань на кожному з дисків, що містить дані відношення, з подальшим злиттям результату. Процес має ітеративний характер, тобто на кожному наступному кроці ми враховуємо варіанти, отримані на попередніх, що сприяє постійному зростанню кількості варіантів з наближенням до кореня дерева. Розглянемо приклад.

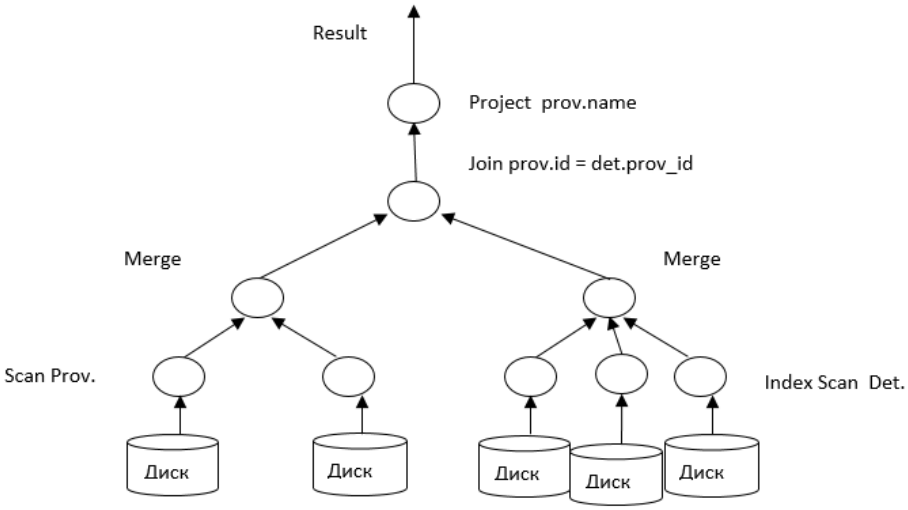
Розглянемо знайому БД про деталі і постачальників, і той же запит:

```
SELECT providers.name FROM providers,details
        WHERE providers.id = details.prov_id
        AND details.name = "plug"
```

Припустимо, що таблиця постачальників розподілена по двох дисках, а таблиця деталей - по трьох, крім того вона проіндексована по полю name. Вихідний граф реляційних операцій



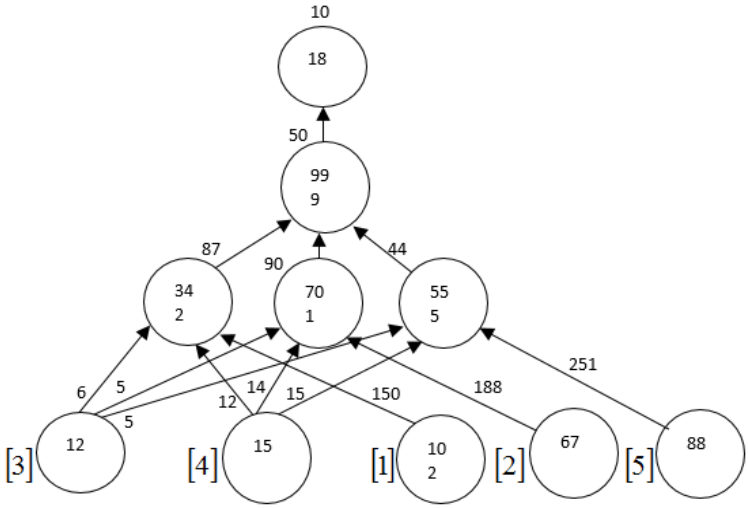
Розглянемо перетворення сканувань, при цьому, скористаємося інформацією про розподіл таблиць по дисках і наявності індексів, отримаємо:



Зауважимо, що варіант сканування таблиці деталей без використання індексу виключається з безлічі можливих, оскільки він свідомо неефективний.

Наступним кроком буде перетворення операції об'єднання (Join). Ця операція є однією з найбільш часто використовуваних в системах БД, та до того ж однією з найбільш трудомістких. Тому, нерідко час її виконання становить левову частку в загальному часу виконання запиту. Звідси велика кількість варіантів фізичної реалізації об'єднання, найпростішим з яких є метод вкладених циклів (nested loop join). Він складається в порівнянні кожного з'єднувального атрибута першого відношення з усіма атрибутами другого (два вкладених цикла), тобто якщо кардинальне число першого відношення - n, а другого - m, то вага операції об'єднання буде n * m * час виконання одного порівняння. Неважко помітити, що якщо об'єднувальні відношення будуть упорядковані, то ефективність виконання операції можна помітно підвищити. Звідси правило: операція об'єднання залежить від властивостей вхідних потоків даних. Так само існують оптимізовані методи виконання об'єднання: об'єднання з хешем (hash join), об'єднання з сортуванням (sort join) і багато інших ще більш витончених. З назв зрозуміло, що вони засновані на штучному приведенні вхідних потоків до вигляду, зручного для виконання об'єднання. Поява ПСУБД, сприяла виникненню нових, паралельних алгоритмів об'єднання, що дозволяють максимально ефективно використовувати ресурси паралельних КС.

Отже, на виході оптимізатора, після проведення оцінки і вибору того єдиного, найоптимальнішого плану, запит представляється в такий спосіб:



Анотована інформація по конкретних операціях в вершинах графа, опускається до моменту реального виконання, оскільки вона не потрібна на етапі планування, а цифри в дужках (біля вершин) позначають залежність оператора від місцезнаходження даних, що в свою чергу означає залежність місцезнаходження даних від конкретного процесора, в результаті чого ми отримуємо зв'язок оператора з процесором, тобто цифра в дужках - номер процесора. Однак подібний зв'язок може бути множиною, при використанні систем з розділеними ресурсами, тобто, наприклад, 5 процесорів під'єднані до 20-ти розділених дисків, тоді при необхідності обробки якихось оператором (з плану) інформації, що зберігається на одному з цих дисків , ми зможемо виконати обробку на будь-якому з 5-ти процесорів (в цьому випадку в дужках буде якесь безліч допустимих процесорів:). У наведеному прикладі - передбачається архітектура без спільного використання ресурсів, тобто один процесор - один диск. Таке позначення використовується в операторах, операндами яких є відношення, що зберігаються в БД, а до операторів, пов'язаних з проміжними результатами воно не відноситься.

12.6 Планування

На етапі планування проводиться призначення ресурсів на кожен з операторів паралельного плану, отриманого від оптимізатора. У процесі

планування використовується інформація про топології системи, пропускної спроможності каналів зв'язку, кількість процесорів, наявність каналів вводу / виводу у процесорів, типи зв'язків і т.д.

Алгоритми планування можуть варіюватися як від простих, що призначають заявку на перший вільний процесор, так і досить складних, які мінімізують пересилання даних і кількість використовуваних ресурсів. В цілому ж завдання планування є класичною, з тією лише особливістю, що в деяких випадках паралельний план може містити вже призначені оператори (див. п.п.12.5).

Можливою проблемою є неточність оцінок ваги вершин і пересилань в плані, що природно впливає на ефективність роботи планувальника. Рішенням може слугувати використання інформації, доступної в динаміці, але при цьому буде потрібно введення в схему обробки досить складного диспетчера, який виконував би призначення в залежності від реальної ситуації в системі, ніж балансував завантаження процесорів. При цьому планувальник можна полегшити, визначивши його функцію не конкретним призначенням процесора на оператор, а формуванням множини допустимих варіантів призначення, виходячи з яких і буде працювати диспетчер.

Література

1. Корочкін О.В., Русанова О.В.Паралельні та розподілені обчислення.Вибрані розділи: Навч.посібник. [Електронний ресурс] / О.В.Корочкін, Русанова О.В. –Електронні текстові дані (2 файла: 43,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 123 с.
2. Русанова О.В., Корочкін О.В.Програмне забезпечення комп'ютерних систем. Програмування та компіляція: Навч. посібник . [Електронний ресурс] / О.В.Корочкін, Русанова О.В. –Електронні текстові дані (1 файла: 1,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 95 с
3. Hesham El-Rewini, Ted G. Lewis Distributed and Parallel Computing, Manning Publications Co., 447 p., 1998.
4. Technical Report No. 2006-504 Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Fangpeng Dong and Selim G. Akl. School of Computing,Queen's University Kingston, Ontario January 2006.
5. F. Berman, High-Performance Schedulers, chapter in The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, 1998.
6. O Rusanova, A Korochkin Scheduling problems for parallel and distributed systems. Proceedings of the 1999 annual ACM SIGAda international conference on Ada.Vol.XIX,Issue 3,1999, pp.195-201.
7. G. Loutsky, O. Rusanova, «Scheduling Problems on the Parallel and Distributed Systems - an Overwiew» – Poland, Rzeszow, tom 1, pp.101-105 (Engl), 2000.
8. Valerii Demchyk, Vitalii Tyzun, Alexander Korochkin, Olga Rusanova. THE APPLICATION OF WCF TECHNOLOGY TO INCREASE THE EFFICIENCY OF PARALLEL COMPUTING IN CLOUD DISTRIBUTED COMPUTER SYSTEMS // Security, Fault Tolerance, Intelligence: proceedings of the International Conference ICSFTI2020, Kyiv, Ukraine, May 13, June 15, 2020. – Kyiv : Igor Sikorsky Kyiv Polytechnic Institute, Publishing House "Polytechnica", 2020. – P.209-215
9. Olga Rusanova, Igor Boyarshin, Anna Doroshenko. ENERGY-AWARE TASK SCHEDULING ALGORITHM FOR MOBILE COMPUTING // Security, Fault Tolerance, Intelligence: proceedings of the International Conference ICSFTI2020, Kyiv, Ukraine, May 13, June 15, 2020. – Kyiv :

Igor Sikorsky Kyiv Polytechnic Institute, Publishing House "Polytechnica", 2020. – P.107-113

10. O.V.Rusanova, A.P.Shevelo List scheduling algorithm modification for MPP Systems.Вісник НТУУ"КПИ".Сер. Інформатика, управління та обчислювальна техніка. - 2006. Випуск 45. -С.101 – 111.
11. Кулаков Ю.А.,Русанова О.В., Шевело А.П. Иерархический способ планирования для GRID. Вісник НТУУ"КПИ".Сер. Інформатика, управління та обчислювальна техніка. - 2009. Випуск 51. - С.57 – 66
12. Русанова О.В., Ярох Ю.Н. Планирование вычислений в гетерогенных кластерных системах. Вісник НТУУ"КПИ".Сер. Інформатика, управління та обчислювальна техніка. - 2011. Випуск 54. - С.155 –163.
13. Русанова О. В. Спосіб планування обчислень для мультиядерних кластерів/ О.В. Русанова// Вісник НТУУ «КПІ». Інформатика, управління таXIX техніка: Зб. наук. пр. –К.: Век+, -2016. –No 64. –С. 31-37.
14. N. Muthuvelu, J. Liu, N. L. Soe, S.r Venugopal, A. Sulistio and R. Buyya, A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids, Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research (AusGrid 2005), Newcastle, Australia, January 30 – February 4, 2005.
15. N. Spring and R. Wolski, Application Level Scheduling of Gene Sequence Comparison on Metacomputers, in the Proc. Of 1998 International Conference on Supercomputing (ICS'98), pp. 141-148, Melbourne, Australia, July 1998