Виконання умови гарантує відсутність втрат пакетів на досліджуваному мережевому пристрої. Умова вводиться для уникнення перевантаження пропускної здатності каналу зв’язку, що виділяється для передавання пакетів тієї чи іншої черги мережевого пристрою в процесі управління ресурсами. У ході виконання умов за рахунок непердбачуваної зміни характеру мережевого трафіку на вузлі виникають черги та пов'язані з ними затримки пакетів. Для забезпечення допустимих затримок у вузлах вибирають оптимальну загальну буферну ємність. На цьому етапі починають з’являтися проблеми щодо вибору оптимальних відносно мінімальних затримок та втрат пакетів ємностей мережевих буферів. Таким чином, для кожної черги визначимо її поточну завантаженість і максимальну ємність, позначивши їх відповідно через nj і njmax (j = 1, ... , N). Запишемо умови запобігання перевантаження окремих черг по їх пропускній здатності умовами уникнення перевантаження черг по їх довжині і тепер завдання зводиться лише до вибору аналітичного виразу для розрахунку середньої довжини черги в процесі обслуговування пакетів у мережевих вузлах.

При цьому кожному типу трафіку, а значить і кожній черзі, може відповідати своя модель обслуговування, не обов'язково відповідаючи переліком існуючих варіантів СМО. З точки зору забезпечення гарантій QoS за параметром середньої затримки в ряді випадків зручніша сумарна умова (2), де середня затримка обслуговування в тій чи іншій черзі може бути розрахована за відомою середній довжині черги на основі формули Літтла для будь-якої СМО. Використання системи умов є актуальною у випадку, коли чисельні значення необхідної середньої затримки (як параметра QoS) нормовані по окремих ділянках мережі. Тоді в процесі управління чергами важливо не перевищити ці задані для окремо взятої пари вузлів каналу значення середньої затримки пакетів, що особливо характерно при вирішенні задач щодо забезпечення гарантованого QoS в рамках архітектурної моделі досліджуваної мережі.

## Модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг

Представлений оригінальний алгоритм керування чергами інформаційних потоків, що базується на одному з відомих алгоритмів CBWFQ та запропонованого методу. Розробка алгоритму полягає у забезпеченні максимальної доступності високопріоритетної послуги зв’язку при збереженні ефективного розподілу мережного ресурсу іншим потоком, який забезпечується зваженим механізмом кругового обслуговування черг. З метою підвищення якості обслуговування та мінімізації втрати пакетів інформації вводиться контроль за часом перебування в черзі пріоритетних пакетів. Узагальнений алгоритм обслуговування трафіку для моделі системи з m пріоритетними групами виглядатиме аналогічно. Утворюється m черг відповідно до пріоритетної групи. На початку кожної ітерації передачі інформації для кожного пакету пріоритетних черг оцінюється час перебування в черзі, якщо час перебування в черзі більше ніж ∆tm , тоді ці пакети класифікуються як прострочені та ставляться в першу позицію найкоротшої черги, що має пріоритет менший. Такі дії виконуються послідовно для всіх прострочених пакетів всіх пріоритетних груп. Отже, розміри черг повинні бути такими, щоб виконувалась умова, що час який буде чекати пакет, що останній став в чергу, повинен бути не більше ∆tmmax, (часу максимальної затримки рафіку p-ї пріоритетної групи).

Після того, як розмір черг був встановлений є потреба роз’яснити поведінку системи в цілому. При надходженні пакетів до вузла вони ставляться в чергу відповідно до пріоритетної групи. Як вже було з’ясовано для пріоритетних пакетів задаються інтервали часу ∆tm та ∆tmmax , на основі даних про них формуються черги. Відомо, що ∆tm < ∆tmmax. Пакети, що знаходяться в черзі довше ∆tm вважаються простроченими. Для того, щоб на кожній ітерації не виявляти прострочені пакети (виконання зайвих дій), є можливість ставити мітки тим пакетам, що потрапили в хвіст черги і явно стануть простроченими (отримають мітку lim), якщо будуть послідовно обслуговуватися в своїй черзі.

Тобто всі пакети, що потрапили на місця черг в інтервалі [Np(m), Nplim(m)] , отримають мітку lim (прострочені). Якщо запропонувати перенесення цих пакетів одразу після їх надходження до черг наслідування, тоді збережеться цілісність задачі, суттєво спростяться розрахунки часу затримки повідомлень, та зменшиться кількість втрачених пакетів з пріоритетних груп, тому що останні позиції черг пріоритетних пакетів будуть порожніми. Робота запропонованого алгоритму для системи з трьома чергами: відео потоків (Video), голосових потоків (Voice) та потоків Інтернет трафіку (Data). Група черг наслідування складається з черги Інтернет трафіку. Група наслідування складається з черг непріоритетних пакетів або низько пріоритетних пакетів. Якщо Інтернет трафік класифікований деяким чином, і пакети різних класів Інтернет трафіку мають різні пріоритети, тоді всі або деякі з них можуть потрапити до групи наслідування. В рамках даної роботи робиться припущення, що події пов’язані з потраплянням пакетів до однієї з черг групи наслідування вважаються рівноймовірними.

Модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг у вузлах корпоративної мережі алгоритм дає змогу ефективно передавати пакети пріоритетних типів трафіку, при цьому не завдаючи суттєвої шкоди для передачі непріоритетних потоків.

## Імітаційна модель системи оброблення інформаційних потоків за удосконаленим алгоритмом управління черг у маршрутизаторах

Модифікований метод управління чергами в мультисервісних вузлах телекомунікаційної мережі. Одна з ключових відмінностей пропонованого рішення полягає в тому, що ведеться контроль за часом перебування в чергах пакетів і, в умовах перевищення допустимого часу очікування, пакет передається на чергу із нижчим пріоритетом та першочерговим обслуговування, що дає змогу підвищити ефективність розподілу мережевих ресурсів за критерієм якості обслуговування. Розроблено імітаційну модель маршрутизатора, який володіє функцією адаптивного вибору алгоритму обслуговування черг в умовах виникаючого явища випадкового сплеску трафіку характерних для мультисервісних мереж нового покоління. Адекватність розроблених моделей підтверджено на основі дослідження імовірнісних властивостей трафіку мультисервісної корпоративної мережі. В результаті проведення імітаційного моделювання доведено, що застосування розробленого методу управління трафіком у вузлах мережі надання інфокомунікаційних послуг в мережах нового покоління, призведе до покращення якості обслуговування потоків реального часу. А саме зменшено затримку обслуговування інформаційних послуг реального часу з кінця в кінець до 2 разів та зменшено ймовірність втрат пакетів на 3% із застосуванням удосконаленого алгоритму зваженого кругового обслуговування черг в мережевих вузлах мультисервісної мережі.

Це в свою чергу призводить до вимушеного відкидання пакетів згідного алгоритму WRED та виникнення значних втрат пакетів голосу, що вплине на якість розмови користувачів в NGN мережах. Внаслідок застосування удосконаленого алгоритму та із вище запропонованою моделю обслуговування, зменшується кількість пакетів у буфері голосового потоку до допустимого значення. А прострочені пакети надсилаються у вільно доступні черги нищої пріоритетності із вищою пропускною здатністю, що в свою чергу призводить до

зменшення втрат та середньої затримки обслуговування у вузлі NGN мережі у 2 рази. На практиці застосування запропонованого модифікованого алгоритму обслуговування черг у мережевих вузлах NGN мережі, дасть змогу зменшити тривалість оброблення пакетів та ймовірність втрат даних потоку реального часу із високим пріоритетом не погіршуючи якості низько пріоритетним.

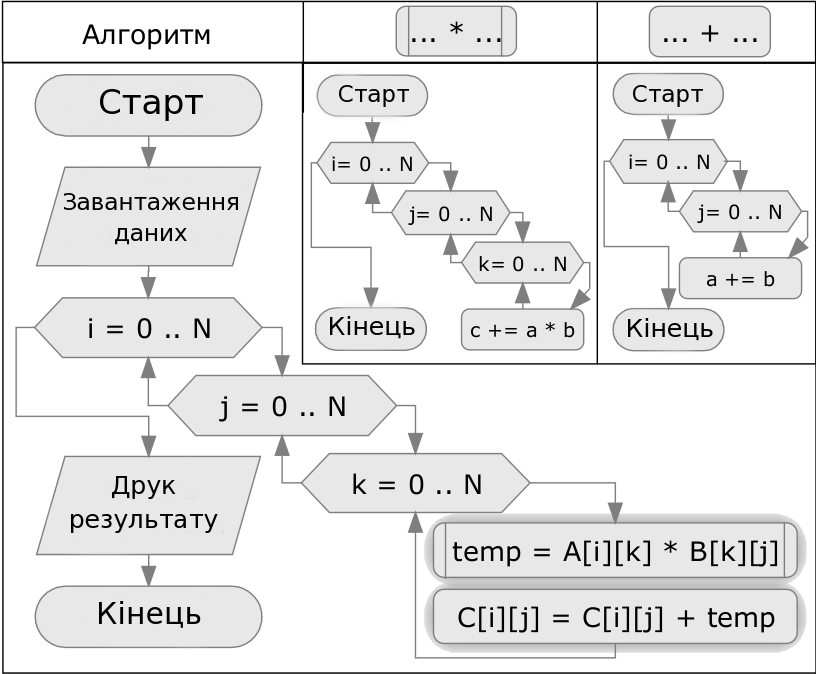
Результат оцінки тривалості оброблення пакетів маршрутизатора рівня ядра, щляхом пінгування від сервера на якому встановлена система моніторингу. Внаслідок чого спостерігається затримка, що перевищує 10,2 мс та 2% втрат пакетів. Що при передаванні потоків реального часу є не допустим, якщо на шляху до адресата стоять аналогічні вузли з однотипним навантаження та кількість їх перевищує 10. Отже, забезпечити гарантовану якість потоків реального часу в даній мережі при кількості хопів більше 10 із застосуванням існуючих алгоритмах CBWFQ є неможливим. Що підтверджує доцільність впровадження у NGN мережах запропонованих рішень розглянутих у дисертаційній роботі.

## Вимоги до оформлення інтерфейсу для розпаралелювання послідовних програм

Система DDCI, що реалізує підхід прозорого автоматичного динамічного розпаралелювання обчислень для блокових послідовних алгоритмів, вже зарекомендовала себе як ефективне і зручне рішення для автоматичного перетворення послідовних алгоритмів в паралельні аналоги. Обмеження системи, пов'язані з розпаралелюванням тільки блокових послідовних алгоритмів, є обґрунтованими і не являють незручностей для програмування, що було доведено в теоретичних і експериментальних дослідженнях. У системі, процес динамічного побудови графа распараллелівать програми в головній мірі залежить від обраного методу інтегрування DDCI-системи в код послідовних програм. Запропонований метод інтеграції DDCI-системи має багато спільного з підходом, використовуваним в Т-Системі, але ті відмінності, які присутні, вносять серйозні зміни в процеси динамічного побудови і розпаралелювання графа послідовної програми.

Розгляд проблеми динамічного побудови графа послідовної програми з вибору завдання, на якій наочно можна буде продемонструвати всі висунуті методи. Блоковий алгоритм твори двох квадратних матриць:

* має поліноміальних складність завдання O(n3) в залежності від розмірності матриць
* для завдання часто доводиться використовувати паралельну реалізацію
* блок-схему алгоритму легко аналізувати (рис. 1)

  
Illustration 1: Блок-схема блочного алгоритму перемноження двох квадратних матриць

На рис. 1 демонструються три блок-схеми, сумарно реалізують алгоритм твори двох матриць. Основна блок-схема програми позначена як «Алгоритм». Дві малих блок-схеми з лівої частини малюнка реалізують алгоритми функцій, що відповідають за твір і складання двох матриць. Виклики даних функцій можна побачити в основний блок-схемі в правому нижньому кутку. Таке розбиття алгоритму на три блок-схеми не випадково, кожна частина алгоритму має свої унікальні особливості.

«Алгоритм», в подальшому «керуючий код». Дана частина алгоритму в післяпослідовно реалізації завдання відповідає в першу чергу за зв'язок виконуваної програми з різними локальними сховищами даних, наприклад: параметри запуску програми, доступ до консолі, доступ до файлової системи, мережеві протоколи, специализированне дослідницьке обладнання. Наслідком є ​​неможливість виконання деяких команд даної частини алгоритму на віддалених обчислювальних станциях, а, значить, і неможливість неконтрольованого автоматичного розпаралелювання програми.

Другою відмітною особливістю даної частини алгоритму є споживана обчислювальна потужність. Якщо в блок-схемі виклики розрахункових функцій

«Множення» і «Додавання» замінити заглушками, то стане видно, що в незалежності від розмірності матриць ця частина алгоритму завжди виконується однаково швидко і не вносить відчутної надбавки до часу роботи алгоритму в цілому. Такі особливості по даному конкретному прикладу означають тільки одне: розпаралелювання керуючої частини коду не є першочерговим завданням, так як це не принесе відчутного зростання в швидкості роботи паралельного аналога. Тепер можна зробити деякі проміжні висновки:

1. Автоматичне розпаралелювання розглянутої частини алгоритму ускладнене через проблеми, пов'язані з локальними дескрипторами ресурсів, що використовуються для зв'язку розпаралелювань послідовної програми із зовнішніми сховищами даних.
2. Існує велика кількість різновидів локальних дескрипторів, і навіть якщо більшість з них є стандартними, то завжди залишається можливість установки на комп'ютері спеціалізованого програмного забезпечення або дослідницького обладнання, що використовує свої «унікальні» дескриптори. В результаті детектування локальних дескрипторів, не вводячи в інтерфейс распараллелівать системи спеціальних директив, неможливо.
3. Якщо всі основні обчислення распараллелівать блочного алгоритму знаходяться в чистих функціях, то виклики функцій можна замінити «заглушками», а автоматичне розпаралелювання цій частині алгоритму є необов'язковим.
4. Вимога винесення основних обчислень з основного графа програми в окремі чисті функції є не більше ніж частиною інтерфейсу рішення, а, значить, можна досягти для переважної числа блокових послідовних алгоритмів, що підлягають автоматичному распараллеливанию.

Результатом зроблених висновків є формулювання першого вимоги для послідовних алгоритмів: все автоматично динамічно распаралелені програми повинні використовувати основну частину графа алгоритму для зв'язку з зовнішніми джерелами даних і використовувати мінімум обчислень, що не інкапсульованих в окремі чисті функції.

«Множення», в подальшому «розрахункові функції». У запропонованому завданні багаторазове обчислення функції, що відповідає за множення двох блоків від початкових матриць, займає близько 99% часу роботи всього алгоритму. Надалі, при розгляді розрахункових функцій, буде матися на увазі безліч помічених чистих функцій з послідовного блочного алгоритму будь-якого завдання, які виконують всі основні обчислення, і при цьому їх виконання займає близько 99% часу роботи всій завдання. Для всіх розрахункових функцій в межах реалізованого підходу також висуваються певні вимоги, і їх повний виклад вже наводилося в попередніх роботах. Коротко можна сказати, що розглянута операція «перемноження блоків матриць» повністю відповідає всім висунутим раніше вимогам:

1. Враховано всі обмеження для «чистої функції», згідно з якими функція може працювати тільки з тими даними, які були передані в якості параметрів.
2. Функція відповідає всім нормам по «ефективності чистих функцій», так як кількість переданих даних в функцію зростає квадратично, а кількість обчислень - кубічно щодо боку перемножуєте квадратних матриць.

На закінчення можна сказати, що даний клас функцій найбільше потребує ефективного розпаралелювання, так як сумарний час роботи розрахункових функцій - це основна складова будь-якої распараллелівать завдання.

«Додавання», в подальшому сполучні функції. Такі функції в попередніх авторських роботах розглядалися як окремі випадки рідко використовуваних низькоефективних чистих функцій. На рішення виділити їх в окремий клас вплинули наступні моменти:

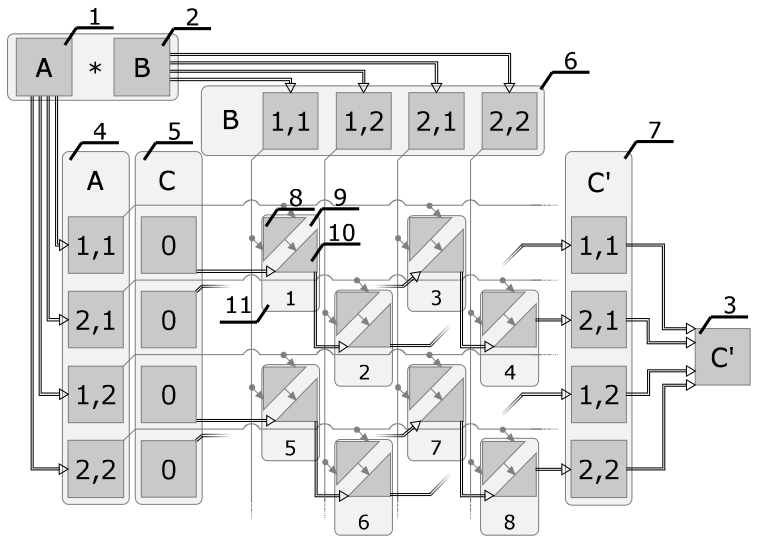
1. Такі функції завжди мають низьку ефективність, яка призводить до уповільнення роботи распараллелівать програми, причому твердження вірне, тільки якщо вважати їх розрахунковими функціями і використовувати для них такі ж методи планування.
2. Так як складність використовуваного в них алгоритму близька до лінійної щодо кількості переданих даних, то немає явної необхідності обчислювати їх всього один раз і надалі використовувати результати їх роботи. Більш ефективним підходом для них можна вважати багаторазовий перерахунок на різних станціях, так як це не додасть суттєвих змін до спільного часу роботи всього алгоритму.
3. Відмова від обліку таких функцій як повноправних вершин графа розпаралеленої частини програми економить кількість використовуваної розподіленої пам'яті для зберігання проміжних розрахунків. Зменшення кількості розподіленої пам'яті, в свою чергу, призводить до зниження кількості межпроцессорного трафіку.

В результаті можна зробити наступний висновок: виділення класу сполучних функцій може підвищити ефективність роботи автоматично распараллелівать програм за умови, що надасть права мати окремі методи планування для виконання функцій даного типу. Також це призводить до додаткових інтерфейсним вимогам, які полягають в самостійної позначці програмістом всіх сполучних функцій в його послідовній програмі, так як система автоматичного розпаралелювання самостійно не зможе відрізнити сполучні і розрахункові функції.

## Управління динамічним графом розпаралелювання частини програми

Розглянемо процес формування графа распараллелівать частини послідовного алгоритму з боку планувальника. Планувальник влаштований так, що він є незалежною системою і пов'язаний з керуючою частиною коду через механізми формування і модифікації динамічного графа. Як говорилося раніше, керуючий код відповідає за процес динамічного додавання нових елементів в граф програми і при необхідності може очікувати закінчення виконання заданої вершини графа. Плануюча система відповідає за запуск чистих функцій (вершин графа) на доступних обчислювальних станціях, а також за подальше видалення вже порахованих елементів з графа. Плануюча система відповідає за запуск чистих функцій (вершин графа) на доступних обчислювальних станціях, а також за подальше видалення вже порахованих елементів з графа. При такому підході динамічно будується граф має багато схожого з буфером зворотного магазинного типу (FIFO). Різниця полягає лише в тому, що плануюча система не зобов'язана вибирати елементи динамічного графа в тому ж порядку, в якому вони були додані в граф. При цьому практичне використання системи показало, то для більшості блокових алгоритмів мінімізація межпроцессорного трафіку і динамічне балансування навантаження призводять до того, що планувальник подає на виконання чисті функції приблизно в такому ж порядку, в якому вони додавалися в динамічний граф.

Тепер розглянемо граф паралельної частини алгоритму, який реалізує блочне твір двох квадратних матриць. Для випадку розбиття кожної матриці на 4 однакових квадратних блоку граф буде виглядати наступним чином (рис. 3).

  
рис 3: Граф розпаралелювання частини блочного алгоритму перемноження двох матриць

Центральна частина графа, що складається з однотипних елементів під номером «9», формується в процесі виконання трьох циклів з «Міткою-3» в блокхеме алгоритму. Позначки «8» і «10» відповідають функціям перемноження і складання двох квадратних блоків матриць відповідно. Порядок, в якому елементи будуть додаватися в динамічний граф програми, також показаний на схемі (нумерація з позначкою «11»). З порядку додавання елементів в динамічний граф відразу видно серйозна проблема, з якою стикається плануюча система. Проблема полягає в тому, що проведення будь-якого ефективного розподілу обчислень для всіх розрахункових станцій під час інтенсивного формування графа програми безглуздо, тому що згодом додавання нових елементів в динамічний граф програми призведе до несумісності кінцевого графа із запланованими раніше операціями.

Більшість сучасних систем динамічного розпаралелювання використовують для планування тільки пасивні алгоритми аналізу графа. Такі алгоритми проводять спрощену оптимізацію розподілу обчислень для розрахункових станцій, так як проводити якісний аналіз по распараллеливанию безглуздо через мінливої форми графа.

Набір елементів під номером «7» відповідає результатам перемноження двох матриць. Тут результат представлений у вигляді набору блоків, з яких складається результуюча матриця. Всі чотири блоки до закінчення обчислень будуть знаходитися в області розподіленої пам'яті на різних обчислювальних станціях. Об'єднання їх в цілісну матрицю під номером «3» проводиться на етапі блок-схеми з «Міткою-6». Процес перенесення даних з розподіленого сховища в область керуючого коду, як говорилося вище, включає в себе призупинення керуючого коду на тривалий проміжок часу. Саме цю «припинення» і пропонується використовувати для запуску активних алгоритмів розпаралелювання, так як саме в цей проміжок часу за зміни динамічного графа відповідає тільки планувальник.

## Планувальник

На основі системи динамічного розпаралелювання обчислень можна використовувати пасивні та активні алгоритми розпаралелювання в межах одного завдання. Проблему одночасного використання обох підходів з метою поліпшення ефективності роботи комплексного планувальника для динамічного розпаралелювання обчислень.

Виділимо два основних види ресурсів, за ефективне використання яких відповідає планувальник:

* обчислювальні ресурси включають в себе всю обчислювальну потужність використовуваної багатопроцесорної системи;
* транспортні ресурси включають в себе можливості транспортної системи, використовуваної для передачі даних між обчислювальними станціями.

З метою ефективного управління розглянутими ресурсами пропонується

створити кілька черг з різним пріоритетом для можливості ефективного розподілу всіх наявних ресурсів і мінімізації простоїв.

Обчислювальні ресурси:

* високий пріоритет - використовується для всіх завдань, запланованих активним планувальником. Це необхідно для того, щоб активний планувальник в разі нестачі вільних обчислювальних станцій міг витісняти завдання пасивного планувальника;
* середній пріоритет - використовується для задач, запланованих пасивним планувальником. Черги високого і середнього пріоритетів використовують один і той же обчислювальний ресурс, і якщо чергу високого пріоритету непорожній, то все завдання з черги середнього пріоритету припиняються або можуть бути скасовані і перенесені на інші обчислювальні станції.

Транспортні ресурси:

* високий пріоритет - використовується для активного планувальника, дана чергу витісняє всі наступні черги;
* середній пріоритет - використовується для пасивного планувальника, а також для витіснення черзі низького пріоритету;
* низький пріоритет - черга використовується для фонового алгоритму дублювання розподілених даних. Черга активна під час простою каналів, коли черги високого і середнього пріоритетів порожні.

Для ефективного використання всіх ресурсів також пропонується розбити планувальник на три незалежні системи: активний планувальник, пасивний планувальник і система фонового дублювання даних.

Активний планувальник. Використовується тільки під час призупинення керуючого коду, так як тільки в цей момент часу можна проводити складний аналіз графа распараллелівать програми, не побоюючись за серйозні і непередбачені наступні зміни динамічного графа з боку керуючого коду. При цьому під час своєї роботи алгоритм може використовувати вільні обчислювальні ресурси, виділені раніше для виконання керуючого коду. Пропонується використовувати даний планувальник для ефективного розпаралелювання тільки тієї частини динамічного графа, результати розрахунку якої очікує керуючий потік, а інша частина графа буде як і раніше распараллелівать пасивним планувальником. Такий підхід спрямований на мінімізацію часу простою керуючого коду і, отже, на мінімізацію часу роботи паралельної реалізації програми в цілому.

Пасивний планувальник. Використовується постійно і не використовує складні алгоритми аналізу графа. Пропонується для реалізації планувальника використовувати основні ідеї Т-Системи: планувальник повинен стежити, щоб якомога менше обчислювальних ресурсів простоювало. При цьому, на відміну від Т-Системи, пропонується ввести критерій, який буде відображати, скільки подальших розрахунків залежить від результатів виконання кожної окремої вершини графа. Критерій пропонується використовувати для того, щоб поліпшити початкову ідею Т-Системи за рахунок того, що на виконання спочатку будуть ставитися ті вершини графа, результатів виконання яких

очікує більше число наступних чистих функцій. Таке нововведення дозволить змусити пасивний планувальник в першу чергу слідувати по шляху мінімізації діагоналі графа, що, в свою чергу, призведе до підвищення ефективності розпаралелювання.

Система фонового дублювання даних для розподіленого сховища. Метою алгоритму є кероване дублювання проміжних даних між обчислювальними станціями. Такий підхід, по-перше, підвищує ймовірність того, що велика частина даних, необхідна для запуску чистої функції, заздалегідь буде перебувати на потрібній обчислювальної станції. По-друге, при запланованій передачі даних між обчислювальними станціями з'являється можливість вибрати для передачі відсутніх даних обчислювальну станцію з найбільш низьким завантаженням транспортного каналу на даний момент, а не передавати дані з тією єдиною станції, на якій дані були пораховані, без можливості адекватної балансування навантаження на транспортні канали. При цьому такий алгоритм не заважає роботі пасивного і активного планувальників, так як використовує тільки простоюють транспортні ресурси.