ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
МОН УКРАЇНИ

ІНСТИТУТ ПРОБЛЕМ МОДЕЛЮВАННЯ В ЕНЕРГЕТИЦІ
ІМ. Г.Є. ПУХОВА НАН УКРАЇНИ

Кваліфікаційна наукова праця
на правах рукопису

УЗДЕНОВ ТАРАС АМУРОВИЧ

УДК 004.75 : 519.876

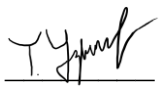
ДИСЕРТАЦІЯ

МЕТОДИ ДИСПЕТЧЕРИЗАЦІЇ ЗАВДАНЬ ДЛЯ GRID-СИСТЕМ З НЕВІДЧУЖУВАНИМИ РЕСУРСАМИ

Спеціальність 05.13.05 – комп'ютерні системи та компоненти
Галузь знань – інформаційні технології

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

 Т. А. Узденов

Науковий керівник – Данильченко Олександр Михайлович,
кандидат технічних наук, доцент

Київ – 2021

АНОТАЦІЯ

Узденов Т. А. Методи диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 «Комп'ютерні системи та компоненти». Державний університет «Житомирська політехніка» МОН України. Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України, Київ, 2021.

Дисертаційна робота присвячена підвищенню ефективності використання GRID-систем з невідчужуваними ресурсами шляхом дослідження та розроблення методів для вирішення задачі диспетчеризації завдань, як одній з основних проблем, що виникає при побудові таких систем.

Для досягнення мети було проведено порівняльний аналіз сучасних підходів до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами, який показав, що дослідниками з усього світу було запропоновано чимало нових методів, для вирішення даної задачі, але в реально діючих системах найчастіше використовується метод FCFS. І пов’язано це з тим, що даний метод є дуже простим в розробці та надійним в роботі. Використання ж інших методів сильно ускладнює систему, що робить її менш надійною. Враховуючи, що такі системи і так є досить нестабільними, в міру багатьох факторів, то зрозуміло, чому розробники відмовляються від складних методів і віддають перевагу FCFS. З цього випливає висновок, що нові методи потрібні, але ключовими характеристиками, якими вони повинні володіти, це простота реалізації, та краща продуктивність (в порівнянні з FCFS). Тому розроблення та дослідження методів диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами є актуальним науковим завданням, що потребує вирішення.

В роботі запропоновано новий підхід до вирішення задачі диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами, який пропонує здійснювати розподіл завдань між обчислювальними вузлами відповідно до закону балансу сил. Суть цього закону полягає в наступному: для кожної сили існує інша сила, яка врівноважує її. Обидві сили приблизно однакові і вони постійно змінюють свої потенціали. Коли потенціал першої сили збільшується, регулятор автоматично включається і відбувається процес вирівнювання потенціалів. Цей закон є універсальним і є більш високого рівня ніж третій закон Ньютона, так як він передбачає наявність ще третьої сили в даній системі сил, третя сила виконує роль регулювальника (диспетчера), задача якого є збереження системи протидіючих сил.

Ідея полягає в тому, щоб розглянути завдання, які потребують вирішення, як одну силу, а обчислювальні вузли, на яких вони повинні виконуватися як іншу силу, які протидіють одна одній. Роль диспетчера має виконувати програма, що працює за певним алгоритмом. І розподіляти завдання таким чином, щоб підтримувати баланс сил. Враховуючи, що такі поняття, як сила обчислювальної задачі та сила обчислювального вузла досить абстрактні поняття, пропонується використовувати поняття потужності завдання та потужності вузла. І здійснювати розподіл вже згідно балансу цих потужностей.

На основі запропонованого підходу розроблено метод диспетчеризації завдань FSA, що не розпаралелюються, для GRID-систем з невідчужуваними ресурсами. Розглядається черга завдань, що потребують вирішення, кожне з яких може бути виконане тільки на одному з вузлів. Розподіл відбувається з урахуванням потужностей кожного з вузлів та потужності завдання і здійснюється відповідно до їх співвідношення. Також знайдено дві можливі модифікації методу FSA (FSA Min, FSA Max) за рахунок комбінації з загальновідомими методами Min-Min та Max-Min.

На основі запропонованого підходу розроблено метод FSA_P диспетчеризації завдань, що легко можуть бути розпаралелені, для GRID-

систем з невідчужуваними ресурсами. Розглядається черга завдань, що потребують вирішення, кожне з яких може бути розпаралелене між вузлами системи. Розпаралелення відбувається з урахуванням потужностей кожного з вузлів і здійснюється пропорційно до їх величини.

Після розробки нових методів виникла необхідність дослідити їх ефективність та порівняти з FCFS. Для цього було здійснено аналіз програмних засобів, що б дозволили симулювати роботу GRID-системи з невідчужуваними ресурсами та здійснювати розподіл завдань відповідно до розроблених методів та порівняти результати з результатами роботи при розподілі завдань методом FCFS. Зокрема, з існуючих програмних засобів було виділено: OMNET ++ (програмний пакет для моделювання роботи великих комп'ютерних мереж), Bricks (система оцінки продуктивності мережі), MicroGRID (моделювання GRID експериментів в віртуальному середовищі), SimGRID (інструмент для вивчення розподілених алгоритмів), GRIDSim (програмний пакет для дослідження алгоритмів планування навантаження в GRID-системах та кластерах). Розглядались дані засоби на предмет можливості враховувати потужності вузлів та завдань при розподілі завдань між вузлами, в результаті встановлено, що такої можливості (без додаткової розробки функціоналу) надати вони не можуть.

Тому було розроблено програмний комплекс «Симулятор GRID-системи з невідчужуваними ресурсами» для дослідження ефективності методів диспетчеризації завдань, який побудований на клієнт-серверній архітектурі, що дозволяє проводити дослідження методів диспетчеризації, як на багатьох ПК (за умови їх підключення до локальної або глобальної комп'ютерної мережі), так і на одному. Для прикладу, на ПК з характеристиками: Intel Core i7-7500U (2.7-3.5 ГГц) / RAM 24 ГБ / SSD 512 МБ / NVidia GeForce 940MX, 2 ГБ адекватно можна симулювати роботу GRID-системи з кількістю вузлів до 100.

Пропонується клієнт-серверна архітектурна модель для створення програмного забезпечення для розподілених обчислень в комп'ютерних

мережах та Internet, а також для створення GRID-систем з невідчужуваними ресурсами. Дана модель побудована на базі WCF сервісу.

Наукова новизна полягає в наступному: вперше запропоновано підхід до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами, в якому, на відміну від існуючих, пропонується здійснювати розподіл завдань відповідно до закону балансу сил, що забезпечує розроблення простих та ефективних методів диспетчеризації; розроблено метод диспетчеризації в GRID-системах з невідчужуваними ресурсами завдань, що легко розпаралелюються (FSA_P), який, на відміну від існуючих, враховує різну обчислювальну потужність вузлів, що забезпечує близьку до максимально можливої продуктивності системи; розроблено метод диспетчеризації в GRID-системах з невідчужуваними ресурсами завдань, які не розпаралелюються (FSA), який, на відміну від існуючих, враховує різну обчислювальну продуктивність вузлів та різні властивості завдань, що забезпечує зменшення часу виконання черги завдань; запропоновано дві модифікації методу FSA – FSA Min та FSA Max, в яких для початкового розподілу завдань використовуються ідеї методів Min-Min та Max-Min відповідно, що забезпечує зменшення часу виконання черги завдань в порівнянні з методом FSA при початковому стані системи, коли кількість завдань дорівнює або менше кількості вузлів.

Як показали експерименти, розподіл завдань в GRID-системі з невідчужуваними ресурсами методом FSA дозволяє значно підвищити ефективність використання ресурсів системи та забезпечити скорочення загального часу виконання всіх завдань з черги у 2,3 рази (у порівнянні з методом FCFS), для черги з послідовних завдань при умові, що кількість завдань перевищує кількість вузлів.

Практичне значення отриманих результатів полягає в тому, що. запропонований в роботі підхід та розроблені на його основі методи можуть бути використані для вирішення задачі диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами.

Створений програмний комплекс SGRIDAR-1 можна використовувати: для дослідження та аналізу поведінки методів диспетчеризації при зміні різних умов (кількості та об’єму завдань, кількості та потужності вузлів); проводити попередні дослідження щодо ефективності різних методів диспетчеризації для заданих архітектур комп'ютерних систем та завдань, що мають на них виконуватись.

Крім того, SGRIDAR-1 має зручний та зрозумілий інтерфейс, а клієнти (вузли) реалізовані окремими програмами, всі обчислення проходять в інтерактивному режимі. Сукупність цих факторів дозволяє використовувати даний програмний продукт в освітніх цілях.

Враховуючи, що SGRIDAR-1 реалізований на основі клієнт-серверної архітектури, його клієнти можна запускати на багатьох персональних комп’ютерах, що об’єднані локальної або глобальною мережею, а програмний компонент Host реалізований як служба WCF і її можна розміщувати на веб-сервері, то після незначного доопрацювання під конкретну задачу, його можна використовувати для проведення тих чи інших обчислень, вже не як симулятор, а як повноцінна GRID-система з невідчужуваними ресурсами.

Запропоновану в роботі клієнт-серверну архітектурну модель можливо використовувати для побудови нових програмних засобів, що потребують використання розподілених обчислень для свого повноцінного функціонування.

Ключові слова: диспетчеризація завдань, методи планування, FCFS, GRID-система з невідчужуваними ресурсами, програма-симулятор, розподілені обчислення.

ABSTRACT

Uzdenov T. A. Task scheduling methods for GRID-systems with non-alienable resources. – Qualifying scientific work as manuscript.

The dissertation on competition of a scientific degree of the candidate of technical sciences on a specialty 05.13.05 – Computer Systems and Components. – State University "Zhytomyr Polytechnic" Ministry of Education and Science of Ukraine, Pukhov Institute for Modelling in Energy Engineering, National Academy of Sciences of Ukraine, Kyiv, 2021.

The dissertation is devoted to improving the efficiency of GRID-systems with non-alienable resources by researching and developing methods for solving the problem of task scheduling, as one of the main problems that arises when building such systems.

To achieve this goal, a comparative analysis of modern approaches to solving the problem of task scheduling for GRID-systems with non-alienable resources was shown, which showed that researchers from around the world have proposed many new methods to solve this problem, but in real systems the FCFS method is most often used. And this is due to the fact that this method is very easy to develop and reliable. The use of other methods greatly complicates the system, making it less reliable. Given that such systems are already quite unstable, due to many factors, it is clear why developers abandon complex methods and prefer FCFS. This leads to the conclusion that new methods are needed, but the key characteristics they must have are ease of simplicity and better performance (compared to FCFS). Therefore, the development and research of task scheduling methods for GRID-systems with inalienable resources is an urgent scientific task that needs to be solved.

The paper proposes a new approach to solving the problem of task scheduling for GRID-systems with non-alienable resources, which proposes the distribution of tasks between computing nodes in accordance with the law of balance of forces. The essence of this law is as follows: for each force there is a different force that balances it. Both forces are approximately the same and they are constantly changing their potential. When the potential of the first force increases, the regulator automatically turns on and the process of equalization of potentials takes place. This law is universal and is of a higher level than Newton's

third law, as it provides for the presence of a third force in this system of forces, the third force acts as a regulator (controller), whose task is to preserve the system of opposing forces.

The idea is to consider tasks that need to be solved as one force, and the computational nodes on which they must run as another force that oppose each other. The dispatcher should be performed by a program that works according to a certain algorithm. And distribute the tasks in such a way as to maintain a balance of forces. Given that concepts such as the force of the computational task and the force of the computing node are quite abstract concepts, it is proposed to use the concepts of task power and node power. And to carry out distribution already according to balance of these capacities.

Based on the proposed approach, a method for scheduling non-parallel FSA tasks for GRID-systems with non-alienable resources has been developed. The sequence of tasks that need to be solved is considered, each of which can be performed only on one of the nodes. The distribution takes into account the power of each of the nodes and the power of the task and is carried out according to their ratio. Two possible modifications of the FSA method (FSA Min, FSA Max) were also found due to the combination with the well-known Min-Min and Max-Min methods.

Based on the proposed approach, a scheduling method has been developed for easily parallel tasks (FSA_P) for GRID-systems with non-alienable resources. The sequence of tasks that need to be solved is considered, each of which can be parallelized between the nodes of the system. Parallelization takes into account the power of each of the nodes and is carried out in proportion to their size.

After developing new methods, it became necessary to investigate their effectiveness and compare with FCFS. To do this, the analysis of software that would simulate the operation of the GRID-system with non-alienable resources and allocate tasks in accordance with the developed methods and compare the results with the results of work on the allocation of tasks by FCFS. In particular, from the existing software were selected: OMNET ++ (software package for

modeling the operation of large computer networks), Bricks (network performance evaluation system), MicroGRID (simulation of GRID experiments in a virtual environment), SimGRID (tool for studying distributed algorithms), GRIDSim (software package for the study of load scheduling algorithms in GRID-systems and clusters). These tools were considered for the possibility of taking into account the powers of nodes and tasks in the distribution of tasks between nodes, as a result, it was found that such an opportunity (without additional development of functionality) they cannot provide.

Therefore, the software package "GRID-system simulator with non-alienable resources" was developed to study the effectiveness of task scheduling methods. It is built on a client-server architecture, which allows you to conduct research on scheduling methods, both on many PCs (provided they are connected to a local or global computer network), and on one. For example, on a PC with the following characteristics: Intel Core i7-7500U (2.7-3.5 GHz) / RAM 24 GB / SSD 512 MB / NVidia GeForce 940MX, 2 GB, you can adequately simulate the GRID-system with up to 100 nodes.

A client-server architectural model is proposed to create software for distributed computing in computer networks and the Internet. And also for creation of GRID-systems with non-alienable resources. This model is based on the WCF service.

The scientific novelty of the study is as follows: for the first time an approach to solving the problem of task scheduling in GRID-systems with non-alienable resources is proposed, in which, unlike the existing ones, it is proposed to distribute tasks according to the law of balance of forces, which provides development of simple and effective scheduling methods; is developed a method of scheduling in GRID-systems with non-alienable resources for easily parallel tasks (FSA_P), which, in contrast to existing ones, takes into account the different computing power of nodes, which provides close to the maximum possible system performance; developed a method of scheduling in GRID-systems with non-alienable resources of tasks that are not parallel (FSA), which, in contrast to

existing ones, takes into account different computational performance of nodes and different properties of tasks, which reduces the execution of the queue of tasks; two modifications of the FSA method are proposed - FSA Min and FSA Max, in which the ideas of Min-Min and Max-Min methods are used for the initial distribution of tasks, respectively, which reduces the execution time of the queue compared to the FSA method at the initial state of the system, when the number of tasks is equal to or less than the number of nodes.

Experiments have shown that the correct distribution of tasks in the GRID-system with non-alienable resources can significantly increase the efficiency of system resources, what to reduce the total execution time of all tasks from the queue by 2,3 times (compared to the FCFS method) for the queue of consecutive tasks at provided that the number of tasks exceeds the number of nodes.

The practical significance of the results is that the approach proposed in the work and the methods developed on its basis can be used to solve the problem of task scheduling for GRID-systems with non-alienable resources.

The created software complex SGRIDAR-1 can be used: for research and analysis of behavior of methods of dispatching at change of various conditions (quantity and volume of tasks, quantity and powers of nodes); to conduct preliminary research on the effectiveness of various scheduling methods for given architectures of computer systems and the tasks to be performed on them.

In addition, SGRIDAR-1 has a user-friendly interface, and clients (nodes) are implemented by separate programs, all calculations are performed online. The combination of these factors allows you to use this software product for educational purposes.

Given that SGRIDAR-1 is implemented on the basis of client-server architecture, its clients can be accessed on many personal computers connected by a local or wide area network, and the Host software component is implemented as a WCF service and can be hosted on a web server, then after their minor refinement, it can be used to perform certain calculations, no longer as a simulator, but as a full-fledged GRID-system with non-alienable resources.

The client-server architectural model proposed in the work can be used to build new software solutions based on it, which require distributed computing for its full functioning.

Keywords: task scheduling, scheduling methods, FCFS, GRID-system, simulator program, distributed computing.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Uzdenov T. (2021) A New Task Scheduling Algorithm for GRID Systems with Non-alienable Resources. Studies in Systems, Decision and Control, vol 346, pp. 207-220. https://doi.org/10.1007/978-3-030-69189-9_12. (Scopus, ISSN 2198-4182)

2. Uzdenov T.A. (2021) Task scheduling in Desktop GRID by FSA method: a practical example. CEUR Workshop Proceedings, vol. 2850, pp. 97-109. http://ceur-ws.org/Vol-2850. (Scopus, ISSN 1613-0073)

3. Данильченко О.М. Моделі системи масового обслуговування для розрахунку навантаженості багатоагентних систем / О.М. Данильченко, Д.Г. Літвинчук, Т.А. Узденов // Вісник ЖДТУ №4(55) – ЖДТУ, 2010. – Т.II. – С. 86-93.

4. Данильченко О.М. Моделі розрахунку трудомісткості операцій комунікації / О.М. Данильченко, Д.Г. Літвинчук, Т.А. Узденов // Моделювання та інформаційні технології, 2011. – Вип. 59. – С. 72-76.

5. Данильченко О.М. Процес диспетчеризації для GRID-системи з невідчужуваними ресурсами / О.М. Данильченко, Т.А. Узденов // Вісник ЖДТУ №2(61) – ЖДТУ, 2012. – Т.II. – С. 147-154.

6. Узденов Т.А. Симулятор процесу диспетчеризації задач в GRID-системах з невідчужуваними ресурсами / Т.А. Узденов // Електронне моделювання. 2021. Т. 43. № 1, С. 87-97.

Наукові праці, які засвідчують апробацію матеріалів дисертації

7. Данильченко О.М. Моделювання програмного комплексу для дослідження паралельних алгоритмів на кластерній системі / О.М. Данильченко, Т.А. Узденов // Тези науково-практичної міжвузівської конференції, 18-19 січня 2010 року. Житомир: ЖДТУ, 2010. – С. 36-37.

8. Данильченко О.М. Проблеми розробки методів керування

паралельними завданнями та їх алгоритмічної підтримки для актуальної форми GRID / О.М. Данильченко, Т.А. Узденов // Тези XXXVI науково-практичної міжвузівської конференції, присвяченої Дню науки, 12-13 травня 2011 року. Житомир: ЖДТУ, 2011. – Т.І. – С. 51-52.

9. Узденов Т.А. Модель процесу диспетчеризації для організації розподілених обчислень на GRID-системах з невідчужуваними ресурсами / Т.А. Узденов, В.О. Артемчук // IV Международная научная конференция «МОДЕЛИРОВАНИЕ-2012». Сборник трудов конференции, К., 2012. – С. 71-74.

10. Узденов Т.А. Алгоритми диспетчеризації для GRID систем з невідчужуваними ресурсами / Т.А. Узденов // Збірник матеріалів VIII Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2020» (Київ, 21 жовтня 2020 р.). – К.: ФОП Ямчинський О.В. – С. 186-189.

11. Узденов Т.А. Огляд програм-симуляторів для дослідження алгоритмів диспетчеризації для GRID-систем / Т.А. Узденов // Тези доповідей III Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення», м. Житомир, 26-27 листопада 2020 р. – Житомир: Житомирська політехніка, 2020. – С. 21-22.

12. Узденов Т.А. Планування обчислень в Desktop GRID / Т.А. Узденов // Матеріали І міжнародної спеціалізованої наукової конференції «Актуальні питання механічної та електричної інженерії, транспортних технологій, електроніки, автоматизації та ІТ». 5 березня 2021 р., Хмельницький – МЦНД – С. 35-36.

ЗМІСТ

ВСТУП

Актуальність теми. GRID – географічно розподілені обчислювальні ресурси, які об’єднані в одну обчислювальну систему. Такі системи створюються для вирішення завдань, що не можуть бути вирішеними за розумний проміжок часу на одному чи навіть декількох ПК, об’єднаних в кластер. Системи GRID можуть включати в себе будь-які обчислювальні ресурси. Такі системи потребують значних фінансових та людських витрат, оскільки вартість високопродуктивного обладнання досить висока, крім того воно потребує окремих приміщень, а також залучення кваліфікованого персоналу. При цьому будь-яка сучасна організація має багато персональних комп’ютерів, на яких працює її персонал. Використання таких ПК не є максимально ефективним, так як більшість задач, що виконуються на них, не займають і 10 – 20% від максимальної продуктивності ПК. Тому є доцільність створювати на базі таких ПК обчислювальних систем, які б дозволяли виконувати інші завдання паралельно з поточними для кожного з ПК. Такі системи називаються GRID-системи з невідчужуваними ресурсами. Ще такі системи називають настільними (Desktop GRID). Перший великий волонтерський обчислювальний проєкт SETI@home був запущений в 1999 році, створивши основу для розвитку BOINC (Відкрита інфраструктура Берклі для мережевих обчислень).

Однією з основних задач, що виникають при створенні GRID-системи з невідчужуваними ресурсами, власне як і для GRID-систем, є задача диспетчеризації завдань. Отже, в GRID-системах повинен бути реалізований механізм планування. Він необхідний для розподілу завдань на виконання між вузлами системи, з метою мінімізації часу виконання роботи та балансування навантаження системи.

Диспетчеризація завдань в GRID-системах є достатньо складною задачею і на сьогоднішній час не існує чіткого і однозначного її рішення. Підходи, викладені в роботах Ankita Sahana S., Carastan-Santos D., De

Camargo R.Y., Trystram D., Zrigui S., Dheenadayalan K., Muralidhara V.N., Srinivasaraghavan G., Haruna A.A., Jung L.T., Zakaria N., Hlaing Y.T., Yee T.T., Kaur M., Khan Z.F., Pujiyanta A., Nugroho L.E., Kumar P.P., Cordeiro D., Каляева А.І., Чернова И.А., Минухина С.В., Коровина А.В. та інших вчених є або занадто складними в реалізації (наприклад, потрібно заздалегідь знати, скільки вузлів буде в системі на протязі всього часу обчислень) або не враховують деяких важливих факторів (наприклад, того, що такі системи є досить гнучкими і постійно змінюються характеристики вузлів, може змінюватися потужність вузла, або швидкість каналу зв’язку з ним), що практично унеможливлює їх застосування для GRID-систем з невідчужуваним ресурсами.

Тому розроблення та дослідження методів для вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами є актуальним науковим завданням, що потребує вирішення.

Зв'язок роботи з науковими програмами, планами, темами. У 2012 році колективом кафедри програмного забезпечення та обчислювальної техніки Житомирського державного технологічного університету успішно завершено виконання прикладного дослідження за державним замовленням “Розвиток методів комбінаторної оптимізації в задачах побудови замкнутих маршрутів на графах та мережах” (номер державної реєстрації: №111U001777). В якій здобувач приймав участь як один з виконавців.

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення ефективності використання GRID-систем з невідчужуваними ресурсами шляхом розроблення та дослідження методів диспетчеризації завдань, що враховують різнорідність ресурсів.

Для досягнення мети були поставлені та вирішені наступні основні задачі:

1) проведення порівняльного аналізу сучасних підходів вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами та існуючих програмних засобів для дослідження таких систем;

2) розроблення нового підходу до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами на основі закону балансу сил;

3) розроблення методу диспетчеризації завдань, що легко можуть бути розпаралелені, в GRID-системах з невідчужуваними ресурсами;

4) розроблення методу диспетчеризації завдань, що не розпаралелюються в GRID-системах з невідчужуваними ресурсами. Дослідження можливостей його модифікації за рахунок комбінації з іншими методами;

5) розроблення програмного комплексу для дослідження ефективності методів диспетчеризації завдань. Проведення обчислювальних експериментів.

Об'єктом дослідження є процес диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами.

Предметом дослідження є методи диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами.

Методи дослідження. Методи теорії масового обслуговування, пріоритетного планування, балансування навантаження в розподілених комп'ютерних системах та мережах, об'єктно-орієнтованої методології розробки і моделювання складних систем.

Наукова новизна одержаних результатів полягає в наступному:

1) вперше запропоновано підхід до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами, в якому, на відміну від існуючих, пропонується здійснювати розподіл завдань відповідно до закону балансу сил, що забезпечує розроблення простих та ефективних методів диспетчеризації;

2) розроблено метод диспетчеризації в GRID-системах з невідчужуваними ресурсами завдань, що легко розпаралелюються (FSA_P), який, на відміну від існуючих, враховує різну обчислювальну потужність вузлів, що забезпечує близьку до максимально можливої продуктивності

системи;

3) розроблено метод диспетчеризації в GRID-системах з невідчужуваними ресурсами завдань, які не розпаралелюються (FSA), який, на відміну від існуючих, враховує різну обчислювальну продуктивність вузлів та різні властивості завдань, що забезпечує зменшення часу виконання черги завдань;

4) запропоновано дві модифікації методу FSA – FSA Min та FSA Max, в яких для початкового розподілу завдань використовуються ідеї методів Min-Min та Max-Min відповідно, що забезпечує зменшення часу виконання черги завдань в порівнянні з методом FSA при початковому стані системи, коли кількість завдань дорівнює або менше кількості вузлів.

Практичне значення отриманих результатів. Запропонований в роботі підхід та розроблені на його основі методи можуть бути використані для вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами.

Створений програмний комплекс SGRIDAR-1 можливо використовувати для дослідження та аналізу поведінки методів диспетчеризації при зміні різних умов: кількості та об’єму завдань, кількості та потужності вузлів. За допомогою SGRIDAR-1 можна проводити попередні дослідження щодо ефективності різних методів диспетчеризації для заданих архітектур комп'ютерних систем та завдань, що мають на них виконуватись.

Крім того, даний програмний комплекс має зручний та зрозумілий інтерфейс, а клієнти (вузли) реалізовані окремими програмами, всі обчислення проходять в інтерактивному режимі. Сукупність цих факторів дозволяє використовувати даний програмний продукт в освітніх цілях.

Основні результати дисертаційної роботи впроваджені в Державному університеті «Житомирська політехніка».

Особистий внесок у роботи, виконані у співавторстві. Всі наукові результати, що подані у дисертації, одержані автором особисто. В роботах, опублікованих в співавторстві, особистий науковий внесок здобувача

складає: [3] – огляд підходів для вирішення задачі масового обслуговування, при умові задіяння в системі декількох агентів; [4] – аналіз та порівняння моделей оцінки трудомісткості операцій; [5] – розробка архітектурної моделі програмного комплексу для досліджень алгоритмів диспетчеризації, побудова схеми взаємодії компонентів системи; [7] – розробка архітектури програмного комплексу для дослідження паралельних алгоритмів на кластерній системі; [8] – аналіз проблем розробки методів керування паралельними завданнями та їх алгоритмічної підтримки для актуальної форми GRID; [9] – розробка моделі процесу диспетчеризації завдань для GRID-систем.

Апробація роботи. Основні положення дисертаційної роботи доповідалися та обговорювались на наступних наукових конференціях і семінарах: XXXIX науково-технічна конференція молодих вчених та спеціалістів Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України (м. Київ, 12 травня 2021 р.); «Workshop on the Quantum Information Technologies + Edge Computing Workshop» (м. Житомир, 1 квітня 2021 р.); міжнародна спеціалізована наукова конференція «Актуальні питання механічної та електричної інженерії, транспортних технологій, електроніки, автоматизації та ІТ» (м. Хмельницький, 5 березня 2021 р.); VIII Всеукраїнська науково-практична конференція молодих вчених «Наукова молодь-2020» (м. Київ, 2020 р.); III Всеукраїнська науково-практична інтернет-конференція здобувачів вищої освіти і молодих учених «Інформаційно-комп'ютерні технології: стан, досягнення та перспективи» (м. Житомир, 2020 р.); IV Міжнародна наукова конференція «МОДЕЛИРОВАНИЕ-2012» (м. Київ, 2012 р.); XXXVI науково-практична міжвузівська конференція, присвячена Дню науки (м. Житомир, 2011 р.); науково-практична міжвузівська конференція (м. Житомир, 2010 р.).

Публікації. Основний зміст роботи викладено в 12 наукових працях, серед яких – 2 статті в наукових періодичних виданнях, що індексуються

Scopus, 4 статті у наукових фахових виданнях, що рекомендовані МОН України, та 6 тез доповідей у збірниках матеріалів та тез конференцій.

Структура й обсяг роботи. Дисертація містить анотацію, вступ, чотири розділи, висновки, список використаних джерел із 148 найменувань та 5 додатків. Загальний обсяг дисертації становить 158 сторінок, з яких 115 сторінок основного тексту. Робота містить 34 малюнки, 6 таблиць.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД МЕТОДІВ ДИСПЕТЧЕРИЗАЦІЇ ЗАВДАНЬ В GRID-СИСТЕМАХ З НЕВІДЧУЖУВАНИМИ РЕСУРСАМИ

Ідея GRID не нова, вона з'явилася ще в минулому столітті, коли стало можливим об’єднувати різні обчислювальні ресурси в групи за допомогою спеціального програмного забезпечення, або, як такі групи називають, кластери. Але час не стоїть на місці, зростає кількість завдань, що потребують вирішення, також зростає і кількість ресурсів, які можна використовувати для таких систем.

Об’єм завдань також зростає, і кластера стає недостатньо для вирішення всіх завдань, тоді виникає ідея поєднати між собою не тільки кластери, а й персональні комп’ютери (ПК), не тільки ті, що знаходяться в одному місці, але і ті, які географічно віддалені один від одного, навіть у різних країнах. Як результат, у середині 1990-х з'явилася концепція GRID [1].

Отже, GRID – це географічно розподілені обчислювальні ресурси (комп’ютери, кластери, суперкомп’ютери та інші) інтегровані в єдину обчислювальну систему. Такі системи створюються для вирішення завдань, що вимагають великої обчислювальної потужності ресурсів і не можуть бути вирішені протягом розумного періоду часу на одному або кількох кластерах, комп’ютерах, суперкомп’ютерах.

GRID-системи можуть включати в себе будь-які обчислювальні ресурси. Будь то персональний комп’ютер, ноутбук, сервер або цілі програмно-апаратні комплекси. Їх можна назвати загальним поняттям – вузлами системи. У цьому випадку кожен вузол може працювати під керівництвом різних операційних систем та програмного забезпечення. GRID-система може об’єднувати різні організації, наукові установи, університети по всьому світу, які потребують великої обчислювальної

потужності ресурсів для вирішення своїх прикладних чи наукових завдань та не можуть бути вирішеними за допомогою власних ресурсів.

На рис. 1.1 зображена схема, що, для кращого розуміння, демонструє описане вище поняття GRID-системи. Зображення позначені підписами PC, Server, Cluster символізують обчислювальні вузли, що знаходяться в різних куточках світу. Зображення, обведене лінією з підписом GRID, означає, що дані ресурси об’єднані між собою спеціальним програмним забезпеченням, і символізує GRID-систему.

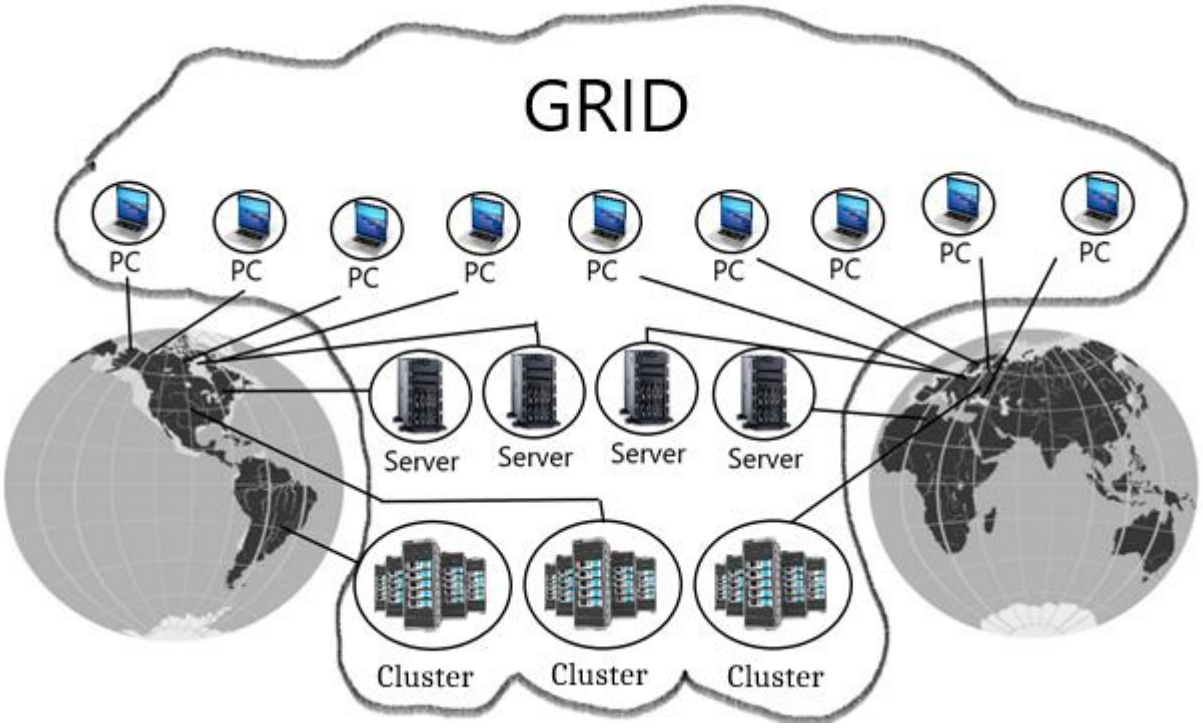


Рисунок 1.1 – Схематичне зображення GRID-системи

1.1 Характеристики GRID-систем

Для того, щоб система могла називатись GRID-системою, вона повинна відповідати певним критеріям та характеристикам. Інформації у відкритих джерелах на цю тему достатньо, тому нижче розглядаються тільки найважливіші, на думку автора, характеристики та принципи побудови GRID-систем. Відповідно до [1] опишемо основні характеристики GRID.

Різнорідність. Система може використовувати обчислювальні вузли (ПК) різної потужності, архітектури, з різним програмним та апаратним забезпеченням (крос-платформні). В ідеалі можливість підключення будь-якого пристрою, який має мікропроцесор, пам’ять і канал зв'язку.

Масштабованість. Можливість розширення до необмежених розмірів, наявність підсистеми, яка могла б контролювати та управляти великою кількістю географічно віддалених ресурсів. Має бути реалізована можливість виявлення та підключення нових ресурсів до існуючої системи.

Динамічність. Можливість контролю за станом завдань, що виконуються. Обчислення на GRID-системах часто не вдаються через те, що ресурси географічно віддалені, канали зв'язку, як правило, досить надійні, але все ж таки, з тих чи інших причин трапляються збої в роботі мережі. Тому потрібна підсистема, яка може відновити роботу вузла або перерозподілити завдання з вузла, в якому відбувся збій, на інші ресурси.

Єдиний інтерфейс. Хоча GRID-система – це сукупність географічно віддалених обчислювальних ресурсів, користувач розглядає її як окремий персональний комп’ютер або як віртуальну машину. Дана можливість реалізується завдяки єдиному інтерфейсу для всіх користувачів.

Безпечність. Система повинна реалізовувати надійний механізм передачі та зберігання даних, це необхідно для запобігання втрати результатів обчислень. А на випадок втрати, можливість відновлення з бази даних, диску, архіву, дзеркала або будь-якого іншого засобу збереження інформації. Головне, щоб вхідні дані та результати обчислень не втрачалися, а у випадку втрати була можливість відновити втрачену інформацію.

Географічна відстань вузлів. Ресурси повинні бути географічно віддалені один від одного, як описано вище, навіть у різних країнах або континентах.

Кілька адміністративних доменів. Оскільки ресурси в GRID-системах належать різним організаціям, для різних власників має бути кілька пунктів моніторингу. Різні власники мають власну політику спільного використання

та моделі доступу. Механізм повинен мати можливість справлятися з децентралізованим моніторингом та синхронізацією розрахунків різних організацій.

Обмін ресурсами. Кожен член GRID-системи повинен мати можливість працювати з ресурсами інших учасників відповідно до графіку, пріоритету чи будь-якої іншої схеми.

Отже, у наведеному вище списку перераховані лише найважливіші особливості GRID-системи, хоча він набагато ширший. Більш детальний опис можна знайти тут [1 – 6].

1.2 Проблеми та задачі при побудові GRID-систем

GRID-система – це досить складна система, яку не так просто реалізувати, існує ряд проблем та завдань з якими стикаються розробники таких систем [1, 7, 8]. Далі наведено найбільш важливі.

Нестандартизованість. Не існує єдиного стандарту архітектури системи та програмного забезпечення, яке б керувало нею. З одного боку, це «розв'язує руки» розробникам, оскільки вони можуть будувати систему так, як вважають за потрібне, а з іншого боку, в цьому випадку виникають проблеми синхронізації з іншими системами та обміну програмним забезпеченням.

Наявність програмного забезпечення. В даний час програмного забезпечення для створення GRID дуже мало. Крім того, в основному користуватися ним можна лише після отримання ліцензії. Також потрібно залучати висококваліфікований персонал, що є досить затратним фактором.

Багатозадачність. Важко реалізувати використання однакових ресурсів для різних типів завдань, особливо при їх географічному розподілі.

Складний процес розробки. Важко розробити програмне забезпечення, враховуючи різну архітектуру та різні платформи, на яких працюють

системні ресурси. Крім того, для цього потрібні висококваліфіковані розробники.

Вузький обсяг виконуваних задач. Як правило, програмне забезпечення розробляється для конкретних завдань, оскільки проблематично реалізувати використання однакових ресурсів для різних цілей.

Важка керованість. Враховуючи географічний розподіл та децентралізацію, важко керувати системою.

Проблема з обміном ресурсами. Надаючи ресурси для певного виду послуг, проблематично використовувати їх для інших.

Питання довіри та безпеки. Питання складне і, скоріше, не технічне, а питання людської довіри, хоча впровадження надійних механізмів шифрування, захисту інформації та автентифікації є непростим завданням.

Неоднозначна бізнес-модель. Для систем GRID взагалі не існує конкретної бізнес-моделі.

Диспетчеризація завдань. Задача складна і вимагає індивідуального підходу до її вирішення, залежно від типу завдань, типу та різноманітності ресурсів. Власне, це питання буде детально розглянуто в цій роботі.

Надійність. Як зазначалося вище, GRID – це досить складна система, яка включає географічно віддалені та неоднорідні ресурси, а тому підтримувати її в робочому стані складно, важко відновлювати роботу після збоїв, що потребує реалізації додаткових ефективних механізмів відновлення втраченої інформації.

У будь-якій GRID-системі повинен бути механізм диспетчеризації завдань, який вирішує питання про те, які завдання надсилати на який обчислювальний вузол, щоб мінімізувати час виконання обчислювального процесу. Це одна з головних задач, що виникає перед розробниками при побудові GRID-системи. Особливості цього процесу (його типи, цілі, методи тощо) будуть розглянуті нижче.

1.3 Диспетчеризація завдань в GRID-системах

Диспетчеризація (від англ. *Dispatch* – швидко виконувати) – процес централізованого оперативного управління та дистанційного контролю, використовуючи оперативну передачу інформації між об'єктами диспетчерської та контрольної точки [9].

Як зазначалося вище, в системах GRID повинен бути впроваджений механізм диспетчеризації завдань або, як його ще називають, планування обчислень. Це необхідно для розподілу завдань на виконання між вузлами системи, щоб мінімізувати час виконання та збалансувати навантаження системи, а також для запобігання ситуації, коли один вузол був би перевантажений, поки інший не працює.

Диспетчеризація завдань у GRID-системах є досить складною задачею, і в даний час не існує чіткого і однозначного її вирішення [10]. Ця особливість надихає дослідників з усього світу на пошук нових рішень для таких систем. На дану тему пропонується багато покращень існуючих методів, а також розроблено ряд нових. Ознайомитись з роботами на тему диспетчеризації завдань у GRID-системах можна за посиланнями [11 – 25].

Як і будь-який механізм, диспетчер працює по заздалегідь визначеному алгоритму, що базується на певному методі. Для різних систем використовуються різні алгоритми, залежно від різних особливостей, як самого алгоритму, так і системи, на якій він повинен працювати.

В даний час існує два основних підходи до розробки диспетчера завдань для GRID-систем [26, 27]:

статичне планування;

динамічне планування.

Нижче розглянемо кожен з них більш детально.

1.3.1 Статичне планування

Статичні алгоритми використовуються в системах, де заздалегідь відомо скільки роботи буде виконуватися та які пристрої будуть задіяні для її виконання, тобто система не змінюється. У цьому випадку диспетчер складає план або графік, після чого система починає виконання завдань.

Цей підхід має ряд переваг і недоліків. У цьому випадку система є цілком передбачуваною і не вимагає великих витрат на планування. Але в той же час система не є гнучкою і прихід нових завдань вимагає перепланування. Для GRID-систем з невідчужуваними ресурсами ця проблема додає того факту, що не можливо не тільки збільшити кількість завдань, а й змінити задіяні ресурси. У будь-який час зв’язок із використовуваним ресурсом може бути розірваний або відбудеться надходження нових завдань, що вимагає негайної зміни плану розподілу та, як описано в особливостях таких систем, перерозподілу ресурсів.

Крім того, такий підхід вимагає наявності всієї інформації про завдання, що виконуються (час, терміни, навантаження тощо), а вона не завжди доступна.

1.3.2 Динамічне планування

Динамічні методи застосовуються в системах де заздалегідь невідомо, скільки роботи і на яких пристроях потрібно буде її виконати, тобто система постійно змінюється. У цьому випадку скласти план або графік роботи можна лише на стартовому етапі роботи системи, після чого, коли умови змінюються, графік також повинен змінюватися. Перевагами цього підходу є гнучкість системи та пристосованість поведінки диспетчера. А до недоліків можна віднести високу вартість та непередбачуваність поведінки на тому чи іншому етапі.

У реальних системах часто використовують змішані або, як їх ще називають, гібридні методи диспетчеризації, вони використовують як статичний так і динамічний підходи, залежно від ситуації. Одним з таких підходів є, наприклад, планувальник в синхронних системах [28].

1.3.3 Методи статичного планування

Нидче наведено найпопулярніші алгоритми, що використовуються у статичному плануванні:

FCFS (First Come First Serve) – найпростіший метод планування "першим прийшов – першим обслуговується". Цей алгоритм заснований на принципі масового обслуговування в черзі (FIFO). Завдання ставляться в чергу до виконання і виконуються в системі без перерви до повного завершення, після виконання одного завдання, наступне по порядку береться з черги, процес триває до завершення всіх завдань. У GRID-системах цей алгоритм часто використовується, оскільки він зручний, надійний та простий у реалізації. Хоча він має ряд недоліків, а саме: високий середній час очікування в черзі та високий середній і повний час виконання [29].

SJF, SJN (Shortest Job First, Shortest Job Next) – найкоротша робота першою, або найкоротша робота наступна, це алгоритм короткострокового планування. Може бути як з переміщенням, так і без переміщення [30].

PP (Priority planning) – метод пріоритетного планування формує чергу завдань, що виконуються відповідно до пріоритетів важливості або терміновості, пріоритети встановлюються користувачем [31].

RR (Round Robin) – це метод, який стосується планування з переміщенням. Завдання, як і в FCFS, ставляться в чергу і виконуються один за одним, але лише за певний, заздалегідь визначений квант часу. Якщо завдання не виконано за цей квант часу, воно переходить до кінця черги і знову чекає, коли йому дозволено буде виконатися. Якщо завдання виконано до кінця кванта часу, воно звільняє систему і починає виконуватися наступне

завдання, і відлік кванта починається знову. Цей алгоритм має нижчу середню затримку, ніж FCFS, але має значний недолік: загальний час виконання істотно залежить від розміру кванта часу, який може бути різним для різних систем та завдань. Наприклад, якщо він дуже великий і всі завдання виконуються до завершення часу квантування, алгоритм вироджується в FCFS, але якщо квант часу малий, перемикання буде частим і в GRID-системах може зайняти багато часу, що суттєво вплине на продуктивність усієї системи [32].

1.3.4 Методи динамічного планування

Як зазначалося вище, динамічне планування відрізняється від статичного планування тим, що процес розподілу завдань відбувається постійно, залежно від того, як змінюється система. Тому ці методи часто називають методами балансування навантаження, в силу специфіки їх роботи. У наведеному нижче списку представлені лише найпопулярніші та найвідоміші, хоча їх набагато більше.

MET (Minimum Execution Time) – мінімальний час виконання. MET призначає кожне завдання ресурсу, який може виконати його найшвидше, незалежно від того, чи доступний ресурс на даний момент часу. Очевидним недоліком цього підходу є те, що важко оцінити, скільки часу знадобиться для виконання завдання [33]. Цей метод може спричинити значний дисбаланс навантаження в системі, але все-таки це один із методів, що використовуються в системі SmartNet [34].

Min-Min – метод планування, який працює за такою схемою: із набору завдань вибирається завдання з мінімальним передбачуваним часом виконання, після чого воно призначається ресурсу з мінімальним загальним часом виконання всіх завдань. Потім наступне завдання з мінімальним часом завершення береться з черги і призначається наступному вільному ресурсу з

мінімальним загальним часом виконання всіх завдань. Процес триває до призначення всіх завдань [35].

Max-Min – цей метод схожий на метод Min-Min, лише завдання вибираються з максимальним, а не мінімальним, як у Min-Min, передбачуваним часом завершення. Ресурси підбираються аналогічно методу Min-Min [36].

1.4 Невідчуженість ресурсів для GRID-систем

Список характеристик GRID-системи, що наведений у 1.1, описує систему GRID в цілому, але він не має такої важливої характеристики, як невідчужуваність. Що мається на увазі? Справа в тому, що обчислення, як зазначалося вище, було б непогано виконати на будь-якому пристрої, який має процесор, пам’ять і канал зв'язку. Наприклад, приєднати до GRID-системи не тільки персональний комп’ютер, суперкомп’ютер та кластер, а й мобільний телефон, планшет тощо. Але користувачі будуть не дуже задоволені, коли їх пристрій завантажується для роботи в той час, коли їм потрібно працювати з ним.

Тому хотілося б розглянути та додати до переліку характеристик таку властивість GRID-системи як невідчужуваність. Хоча це одна з властивостей GRID-систем, і було б логічно описати її в попередньому розділі, але оскільки робота пов’язана саме з цією характеристикою, було вирішено виділити це питання в окремий пункт розділу. А також для того, щоб зосередити увагу саме на цій властивості.

Отже, невідчужуваність – це можливість використання ресурсів, що підключені до GRID-системи, паралельно з власником цих ресурсів. При цьому кількість ресурсів, які можна використовувати для GRID-системи, має бути встановлена програмно на стороні клієнта. Тобто користувач повинен вирішити, скільки своїх ресурсів він може виділити і коли вони можуть бути

використані в GRID-системі. Має бути можливість змінювати цю характеристику.

Наприклад, є користувач у якого є звичайний ноутбук або настільний ПК, якому він може дозволити працювати в GRID-системі, поки він ним не користується, скажімо, з 18-00 до 9-00 у будні та цілий день у вихідні. Або він може дозволити використовувати 10% потужності його пристрою для GRID-системи в будь-який час дня і ночі. Але це повинен вирішувати лише користувач і тому в GRID-системі з невідчужуваними ресурсами повинен бути механізм, який реалізував би принцип невідчужуваності.

Крім того, було б дуже добре, якби користувач міг вказати, які типи завдань він дозволяє вирішувати на своєму пристрої. Оскільки завдання можуть бути різними за своїми характеристиками, для деяких може знадобитися велика пропускна здатність каналу зв'язку, а для інших потрібні лише процесор та пам’ять і не потрібна передача великої кількості інформації.

Для більш детального пояснення розглянемо наступну ситуацію – припустимо, у нас є такі персональні комп’ютери:

1) ПК1 на базі Intel Core i5 (3.7-4.6 GHz) / RAM 8 GB, канал зв'язку зі швидкістю передачі даних 256 Kbps;

2) ПК2 на базі Intel Celeron N4000 (1.1-2.6 GHz) / RAM 4 GB, канал зв'язку зі швидкістю передачі даних 100 Mbps.

І є таке завдання: розпакувати інформацію з архіву WinRar, тоді як розмір архіву становить 5000 МБ.

Оскільки ПК1 є потужнішим за тактовою частотою процесора та об'ємом оперативної пам'яті, можна припустити, що він буде краще справлятися з такими типами завдань, але він має досить повільний канал зв'язку, тому час передачі даних та результатів буде набагато довшим, ніж час роботи. Крім того, і без того повільний канал зв'язку буде завантажений на 100%, що спричинить незручності для користувача ПК. Можливо, було б краще вирішити таку проблему на ПК2?

Вищезазначене завдання є суто теоретичним і максимально простим для розуміння. Сформовано лише для того, щоб виробити правильне розуміння цієї проблеми. Хоча на GRID-системах можна вирішувати такі завдання.

Ці особливості слід враховувати, розподіляти завдання на типи та давати користувачеві можливість вирішити, які типи завдань він готовий прийняти та виконати.

Або застосувати механізм, який міг би визначити, які типи завдань найкраще вирішувати на певному ПК і розділити вузли за типом. Тим самим полегшити життя користувачеві, який може просто не бути компетентним у цьому питанні.

Далі в цьому ж розділі буде розглянуто один з видів GRID-систем з невідчужуваними ресурсами, які в різній літературі називають ще Desktop GRID.

1.5 Desktop GRID: поняття, переваги, недоліки та види

Desktop GRID – це GRID-система, що об'єднує через Інтернет неспеціалізовані обчислювальні вузли (персональні комп'ютери, ноутбуки, смартфони та ін.) і використовує їх вільні обчислювальні ресурси для виконання обчислювальних розрахунків. Як обчислювальний інструмент Desktop GRID веде свою історію з 1999 року, коли був запущений перший масштабний проєкт розподілених добровільних обчислень SETI@home. На сьогодні Desktop GRID є важливою частиною галузі високопродуктивних обчислень поряд з обчислювальними кластерами і обчислювальними GRID. Однак, на відміну від зазначених систем, фундаментальні основи функціонування Desktop GRID значно менше вивчені. При цьому успіх існуючих і поява нових наукових проєктів на основі Desktop GRID призводять до виникнення нових наукових задач [37].

Desktop GRID мають цілий ряд переваг у порівнянні з класичними GRID-системами, що робить їх перспективними та конкурентними:

– простота розгортання, підтримки та обслуговування;
– низька вартість розробки;
– велика масштабованість;
– висока потенційна пікова продуктивність;
– простота розробки додатків;
– можливість використання успадкованих додатків.

Але у той же час, такі системи мають багато недоліків, що створюють певні труднощі в розробці та створенні Desktop GRID. Тому є цілий ряд проблем та задач, що потребують вирішення, а також потребують наукових досліджень. Зокрема до недоліків Desktop GRID можливо віднести:

– ненадійність обчислювальних вузлів;
– невисока швидкість передачі даних;
– складно прогнозувати присутність обчислювальних вузлів;
– велика апаратна та програмна різнорідність;
– низька обчислювальна потужність вузлів;
– висока ймовірність збоїв при виконанні обчислень.

Описані недоліки суттєво впливають на продуктивність усієї системи. Враховуючи вище зазначене, можна зробити висновок, що політика розподілу завдань (диспетчеризація) може суттєво впливати на загальну продуктивність Desktop GRID.

У літературі виділяють п’ять основних видів Desktop GRID: добровільна, корпоративна, гібридна, ієрархічна, пірингова. Нижче наведено короткий опис кожної з них.

Добровільна (volunteer computing) – Desktop GRID у якій: обчислювальні вузли є комп'ютерами добровольців (волонтерів) і з'єднані з сервером мережею Інтернет.

Корпоративна – Desktop GRID у якій обчислювальні вузли є робочими станціями певної організації і з'єднані з сервером по локальній мережі, у

такій системі доступність і надійність вузлів значно вище, в порівнянні з добровільною, і знімає ряд недоліків, описаних вище, але продуктивність такої системи, як правило, не висока.

Гібридна – Desktop GRID у якій обчислювальні вузли є суперкомп'ютерами і обчислювальними кластерами [38-40], деякий огляд таких систем та методів планування в них наведено в роботі [41].

Ієрархічна – Desktop GRID у якій існує ієрархія серверів обчислювальної мережі, в якій кожен сервер нижчого рівня ієрархії отримує набір завдань від сервера більш високого рівня і повинен розподілити ці завдання між своїми підлеглими серверами або обчислювальними вузлами (на самому низькому рівні ієрархії), кожен сервер вирішує свою приватну задачу планування завдань [42].

Пірингова – Desktop GRID у якій обчислювальні вузли пов'язані один з одним і можуть обмінюватися даними. Планувальник завдань може брати до уваги граф зв’язків між завданнями, для прискорення обчислень, зниження навантаження на сервер, зменшення часу комунікацій та інші [42,43].

1.6 Характеристики Desktop GRID

Відповідно до [44-47], нижче описано характеристики та критерії, які повинна мати та яким відповідати будь-яка Desktop GRID.

Безпечність. Системи Desktop GRID повинні обмежувати виконуваним завданням доступ до файлів або даних, що розміщені на обчислювальних вузлах. Безпека є фундаментальною властивістю, адже якщо волонтери не будуть впевнені, що обчислення, які відбуваються на їх пристроях безпечні, то навряд чи вони будуть підключатися до таких систем.

Надійність. Має бути гарантоване надійне виконання. Desktop GRID повинні мати механізми, що передбачають можливі збої, мати інструкції по запобіганню таких випадків, відновленню інформації.

Довіра. Важливо гарантувати правильність результатів. Desktop GRID не повинні допускати помилкові результати.

Мотивація. Desktop GRID повинні мати механізм стимулювання для заохочення волонтерів. Він оцінює та класифікує волонтерів відповідно до їх поведінки. Відповідно до оцінки та рейтингу встановлюються пільги для волонтерів (тобто винагороди). Крім того, системи Desktop GRID можуть забезпечити планування на основі стимулів, репутації.

Ненав'язливість та опортунізм. Desktop GRID повинен поважати автономію добровольців. Волонтери можуть підключатися і відключатися за власним бажанням. Як тільки власники ПК починають виконувати власні задачі, запущене програмне забезпечення Desktop GRID слід негайно зупинити і ресурси слід звільнити. Коли настільні комп'ютери доступні, системи Desktop GRID повинні використовувати незайняті ресурси якомога швидше.

Планування. Desktop GRID повинна мати ефективний механізм планування. Зокрема, він повинен адаптуватися до динамічного, неоднорідного та ненадійного середовища.

Масштабованість. Системи Desktop GRID повинні бути масштабованими і мати можливість керувати добровольцями, не погіршуючи продуктивність, навіть якщо кількість добровольців зростає або якщо волонтери поширюються через Інтернет.

Простота використання та розгортання. Desktop GRID базується на добровільних користувачах, які не мають професійних навичок. Якщо потрібні спеціальні навички під час встановлення та розгортання програмного забезпечення Desktop GRID, то це буде заважати поширенню такої системи. Щоб більше користувачів приєдналися до Desktop GRID, таке програмне забезпечення має бути якомога простим, а встановлення та розгортання бажано щоб проходило в автоматичному режимі з мінімальною участю користувача.

1.7 Архітектура Desktop GRID

Опишемо архітектуру Desktop GRID на прикладі BOINC [48]. BOINC (Berkeley Open Infrastructure for Network Computing – це вільна програмна платформа для проведення розподілених обчислень. Система BOINC була розроблена в Каліфорнійському університеті в Берклі під керівництвом Девіда Андерсона (David Anderson) командою, яка створила проєкт SETI@home.

Дане програмне забезпечення складається з наступних компонентів:

– *база даних*, яка розміщена на веб-сервері – це основа всього проєкту BOINC, у базі даних зберігається вся використовувана в проєкті інформація: відомості про зареєстровані команди, відомості про зареєстрованих користувачів і пов'язаних з ними комп'ютерів, відомості про наявні додатки та їх версії, відомості про використовувані користувачами клієнти BOINC і інформація про поточні підзадачі та результати обчислень;

– *служба обробки* призначена для обробки стану обчислювальних підзадач і результатів їх вирішення, для перевірки поточного стану підзадачі в базі даних і оновлення відповідних полів, коли підзадача готова перейти в новий стан;

– *служба подачі* є допоміжною, вона завантажує необроблені підзадачі (ті підзадачі, для яких ще не отримано канонічний результат) з бази даних в сегмент пам'яті. Ця попередня робота виконується для підвищення продуктивності системи BOINC в цілому, за допомогою обмеження числа запитів до бази даних;

– *служба перевірки результатів* призначена для організації перевірки отриманих результатів на коректність.

– *служба освоєння* періодично перевіряє наявність завершених підзадач, адміністратор проєкту повинен створити функцію, яка визначає, що необхідно зробити з канонічними результатами, підзадача позначається, як завершена тільки після проходження через службу освоєння;

– *служба видалення* – це «збирач сміття» проєкту BOINC, вона перевіряє наявність завершених і освоєних підзадач, а при знаходженні таких, очищає вхідні і вихідні файли сервера, пов'язані з цими підзадачами;

– *міст* – служба, яка забезпечує зв'язок і спільну роботу над проєктом інфраструктури BOINC і стандартної GRID, додатки BOINC спеціально розробляються під архітектуру BOINC. Вони викликають функції BOINC через інтерфейси, реалізовані в клієнті і виконують таку специфічну роботу як, наприклад, дозвіл імен файлів, як наслідок, підзадачі проєкту BOINC не можуть бути безпосередньо запущені для розрахунку на інфраструктурі GRID, з іншого боку, GRID не може, подібно клієнту BOINC, без посередника з'єднатися з планувальником проєкту і запросити підзадачі для розрахунку;

– *планувальник* – це програма, яка запускається, коли до сервера проєкту приєднується клієнт, замість запиту до БД планувальник отримує завдання з сегмента пам'яті, в яку завдання завантажує служба подачі, планувальник має можливість самостійного призначення підзадач клієнтам, так як не всі клієнти мають однакові настройки і комп'ютери.

1.8 Диспетчеризація завдань в Desktop GRID

Диспетчеризація завдань в Desktop GRID – це складна задача, навіть для випадків з повною інформацією [49–52], в реальних умовах вона стає ще складнішою. По-перше, це пов'язано з тим, що повний набір завдань рідко сформований заздалегідь і доповнюється динамічно; по-друге, набір доступних обчислювальних вузлів змінюється з часом: вузли стають недоступні (тимчасово або назовсім), повертають неправильні результати до проєкту, підключаються нові обчислювальні вузли (що особливо актуально для добровільних обчислень) [37].

В роботі [37] авторами проведений комплексний аналіз методів планування завдань в Desktop GRID та критеріїв оптимізації відповідно до

яких ведуться розробки нових та модифікації відомих методів диспетчеризації завдань. Виділяються такі основні критерії оптимізації: пропускна здатність, час роботи, час завершення, надійність, доступність, рівень завантаження, накладні витрати.

Враховуючи, що наразі існує дуже багато нових методів диспетчеризації завдань та цілий ряд модифікацій загально відомих, що пропонуються різним авторами, то важливо було б визначити, які ж методи використовуються в реальних програмних засобах та діючих проєктах і чому. Тому було вирішено провести в даній роботі аналіз методів диспетчеризації завдань, що використовуються в реальних програмних засобах для побудови Desktop GRID.

1.9 Аналіз методів диспетчеризації завдань в Desktop GRID

Для вирішення поставленої задачі було проаналізовано ряд публікацій за останні роки на предмет використання методів диспетчеризації завдань в Desktop GRID. Зокрема встановлено, що методи планування розділяють на три категорії: прості, модельні та евристичні.

У *простому підході* завдання або ресурси вибираються за допомогою FCFS (First Come First Served) або випадковим чином.

Модельний підхід поділяється на детерміновану, економічну та математичну моделі. Детермінована модель базується на структурі або топології. Наприклад, черга, стек, дерево або кільце. Завдання або ресурси детерміновано підбираються відповідно до властивостей структури або топології. Наприклад, у дереві завдання розподіляються з батьківських вузлів на дочірні вузли. В економічній моделі рішення про планування базується на ринку (тобто ціні та бюджеті) [53-55]. У математичній моделі ресурси є обрані в математичній манері (генетичний алгоритм, теорія ігор та техніка машинного навчання, інші).

У *евристичному підході* завдання або ресурси вибираються методами

ранжування, узгодження та виключення, на основі репутації ресурсу або стану. Репутація пов'язана з шаблоном виконання та історією, тоді як стан відображає можливості машин (апаратні можливості, продуктивність, вага зв'язку тощо). Метод ранжування класифікує ресурси або завдання відповідно до критеріїв та потім обирає найбільший або найгірший. Метод відповідності вибирає найбільш відповідні завдання та ресурси відповідно до функцій оцінки (наприклад, Min-Min, Max-Min, виборче право тощо [56-58]). Метод виключення виключає ресурси відповідно до критеріїв, а потім вибирає найбільш підходящий серед тих, хто залишився. Рейтинг, методи узгодження та виключення можна використовувати разом або окремо.[59].

Серед існуючого програмного забезпечення для побудови Desktop GRID можна виділити наступні:

Alchemi [60-62] – GRID на базі NET, що агрегує обчислювальну потужність мережевих настільних комп’ютерів. Це забезпечує API .NET та інструменти для розробки GRID-додатків на основі .NET. Буває Alchemi застосовується в таких додатках, як широкомасштабна обробка документів, CSIRO Australia, гідрологія та обробка електронних таблиць Microsoft Excel. Планування роботи базується на пріоритеті та першочерговому обслуговуванні (FCFS).

Bayanihan [63-65] – це обчислювальна веб-система з використанням Java. Система Bayanihan складається з клієнта та сервера. Клієнт виконує код Java у веб-браузері або Java-програму на проміжному програмному забезпеченні. Він має робочий механізм, який виконує обчислення, або механізм спостерігача, який показує результати та статистику. Сервер складається з сервера HTTP, менеджера роботи, диспетчера часу та пула даних. Bayanihan – це фреймворк з відкритим кодом, розроблений на MIT, і вважається першим веб-інтерфейсом настільних мережевих систем [66]. Політика планування – FCFS.

BOINC [67] – це система проміжного програмного забезпечення для добровільних обчислень (або обчислень із загальнодоступними ресурсами).

BOINC розроблена для створення та управління обчислювальними проєктами з державними ресурсами. Є безліч проєктів на базі BOINC: SETI@Home, Predictor@Home, Folding@Home, Climatepredication.net, Climate@Home, LHC@Home, Einstein@Home, BBC Climate тощо [68-70].

CCOF (Cluster Computing on the Fly) [71,72] – провів всебічне дослідження загальних методів пошуку у високодинамічному середовищі P2P для визначення простоїв комп'ютерних циклів в Інтернеті. Більш недавня робота дослідників CCOF над рівноправною Desktop GRID, яка називається WaveGRID [73], побудувала мережу накладання часового поясу на основі Content-Addressable-Network [74] для географічного використання простоїв нічних циклів по всьому світу [75]. Однак модель доступності хоста в цій роботі не базується на потребах ресурсів, і ця робота не враховує балансування навантаження між доступними системними ресурсами [76]. Політика планування – плануванням хвиль, коли завдання рухаються по хвилі простоїв.

Condor [77, 78] є пакетною системою для високопродуктивних обчислень на розподілених ресурсах. Condor забезпечує управління робочими місцями, планування роботи, моніторинг ресурсів та управління ресурсами тощо. Зокрема, Condor націлений на високопродуктивні обчислювальні та опортуністичні обчислення [79]. Політика планування – це вибір пар завдання-виконавець за пріоритетами, вимогами та рангом.

XtremWeb [80, 81] – це програмне забезпечення з відкритим кодом для створення полегшеної настільної сітки шляхом збору невикористаних ресурсів настільних комп'ютерів (процесор, сховище, мережа). Його основні функції дозволяють розгортання для кількох користувачів, декількох додатків та між-доменів. XtremWeb перетворює набір мінливих ресурсів, що поширюються по локальній мережі або Інтернету у середовище виконання, що виконує дуже паралельні програми. XtremWeb – це дослідницький проєкт, що належить до невеликої за розміром GRID-системи [82], він став основою для нового проєкту XtremWeb-HEP. XtremWeb-HEP складається з

трьох ролей: клієнтів, які подають завдання та дані, диспетчера, який планує завдання на обчислювальні ресурси, та працівників, які виконують завдання та надсилають результати назад клієнтам. Завдяки простоті цей підхід виявився надзвичайно масштабованим, надійним та безпечним [83]. Політика планування – FCFS.

HTCondor (раніше відомий як просто Condor) [84, 85] – це обчислювальний планувальник, розроблений в Університеті штату Вісконсін-Медісон. Він дозволяє користувачам запускати свої двійкові файли на робочих станціях Aalto Linux [86, 87] без явного входу на настільні машини. Condor піклується про вибір правильної робочої станції, встановлення правильного пріоритету роботи та подбає про очищення виходу. Condor розподіляє, планує, виконує та повертає результати [88].

В даному аналізі наведені найбільш популярні та широко використовувані системи Desktop GRID, хоча їх значно більше. Зокрема існують ще SZTAKI [89], Charlotte [90], CPM [91], Entropia [92], PVM [93], distributed.net [94], Organic GRID [95, 96], Messor [97, 98], WebCom [99] та інші. Деякі з них вже не підтримуються, інші ж активно розвиваються та удосконалюються. Що ж до політик планування, то в наведених системах в основному використовують метод планування завдань FCFS.

Отже, проведений аналіз існуючих методів диспетчеризації показав, що наразі в реальних системах зазвичай використовується переважно метод FCFS.

Чому саме FCFS використовується найчастіше? Це пов’язано з тим, на думку автора, що даний метод є дуже простим та надійним як в розробці так і в роботі. Використання інших методів значно ускладнює систему, що робить її менш надійною. Враховуючи, що такі системи і так є досить не стабільними, в міру багатьох факторів, то зрозуміло, чому розробники відмовляються від складних методів і віддають перевагу FCFS. З цього випливає висновок, що нові методи потрібні, але однією з ключових характеристик, якими вони повинні володіти, це простота, та, звісно, краща

продуктивність, в порівнянні з FCFS. Також на основі аналізу сучасних підходів до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами (Desktop GRID), що використовуються в реально діючих системах можна зробити висновок, що розроблення та дослідження методів для вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами (Desktop GRID) є актуальним науковим завданням, що потребує вирішення.

1.10 Висновки до розділу 1

1) Проведено огляд сучасного стану розвитку GRID-систем з невідчужуваними ресурсами. Наведено поняття таких систем, їх основні характеристики. Виділено проблеми та задачі, що виникають при розробці та побудові таких систем, однією з яких є задача диспетчеризації завдань. Проведено огляд методів, що використовуються для її вирішення.

2) Описано поняття невідчужуваності ресурсів та встановлено, що GRID-системи з невідчужуваними ресурсами можуть ще називатися Desktop GRID.

3) Проведений огляд Desktop GRID як одного з видів GRID-систем, наведено переваги, недоліки та види таких систем. На прикладі загальновідомого проєкту BOINC описана архітектура Desktop GRID.

4) Проведено аналіз існуючих проєктів та програмних засобів для побудови GRID-систем з невідчужуваними ресурсами / Desktop GRID на предмет використання в них методів планування. Аналіз показав, що в основному в таких проєктах використовується метод FCFS.

5) Отже, нові методи потрібні, але однією з ключових характеристик, якими вони повинні володіти, це простота, та, звісно, краща продуктивність в порівнянні з FCFS. Тому розроблення методів диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами є актуальним науковим завданням, що потребує вирішення.

РОЗДІЛ 2
РОЗРОБЛЕННЯ МЕТОДІВ ДИСПЕТЧЕРИЗАЦІЇ ЗАВДАНЬ В GRID-СИСТЕМАХ З НЕВІДЧУЖУВАНИМИ РЕСУРСАМИ

Як зазначено в попередньому розділі, GRID-система з невідчужуваними ресурсами / Desktop GRID – це сукупність різнорідних обчислювальних вузлів на різних програмних платформах та з різними за швидкістю передачі інформації каналами зв’язку, що об’єднуються спільним програмним забезпеченням для вирішення трудомістких, з точки зору величини обчислень, задач.

На GRID-системах з невідчужуваними ресурсами / Desktop GRID можуть виконуватися самі різноманітні задачі, з різними властивостями та характеристиками, такі, що потребують великої кількості, скажімо, оперативної пам’яті обчислювального вузла або ж високої тактової частоти процесора чи великого об’єму дискового місця або ж, наприклад, швидкісного каналу зв’язку, так як має бути передана велика кількість інформації для їх обробки.

При цьому обчислювальні вузли можуть бути також з різноманітними характеристиками. Одні, наприклад, мають високу тактову частоту процесора чи кількість процесорів, інші можуть мати великий об’єм оперативної пам'ять чи велику кількість вільного місця на диску, або ж швидкісний диск типу SSD. При цьому підключені до системи вузли можуть використовувати будь-яке основне програмне забезпечення, тобто різну операційну систему, що керує ними. Також у всіх вузлів можуть бути різні за своїми характеристиками канали зв’язку, основні з яких, в даному випадку, – це швидкість передачі інформації та тип з’єднання.

Планування обчислень різних за властивостями задач на таких різнорідних вузлах є досить трудомісткою задачею і, на сьогоднішній день,

не існує єдиного чіткого рішення для її виконання. Тому необхідні нові методи, які б дозволили проводити обчислення якомога ефективніше.

Для кращого розуміння сформулюємо постановку задачі диспетчеризації завдань у GRID-системах з невідчужуваними ресурсами / Desktop GRID та зобразимо її схематично.

2.1 Постановка задачі

Припустимо, у нас є багато обчислювальних вузлів, які відрізняються між собою за потужністю, програмним забезпеченням та апаратною архітектурою – персональних комп’ютерів (ПК). Ці ПК можуть включати будь-які пристрої, будь-то ноутбуки, настільні комп'ютери, сервери або кластери. Дані ПК пов'язані між собою різними каналами зв'язку з різною швидкістю передачі даних.

З іншого боку, у нас є множина завдань, які нам потрібно виконати. Завдання можуть бути різними за своїми властивостями і вимагати різного часу виконання та різної обчислювальної потужності ПК.

Між ними знаходиться планувальник або диспетчер, який вирішує, яке завдання якому вузлу виконувати. Призначення цього планувальника – розподілити завдання так, щоб загальний час на виконання всіх завдань був мінімальним. Він повинен працювати за спеціальним алгоритмом, що має бути побудований на методі.

Задача: визначити метод, відповідно до якого буде відбуватися розподіл завдань між обчислювальними вузлами GRID-системи з невідчужуваними ресурсами / Desktop GRID.

На рис. 2.1 схематично показано процес планування обчислень для GRID-системи з невідчужуваними ресурсами / Desktop GRID відповідно до описаної вище постановки задачі диспетчеризації завдань для GRID-системи з невідчужуваними ресурсами. Пояснення до рис. 2.1: зображені прямокутники *Task 1, Task 2,…, Task n* символізують множину завдань з

різними властивостями, які потребують вирішення; прямокутники *PC 1, PC 2,…, PC m* символізують множину обчислювальних вузлів з різними характеристиками, на яких можливо проводити виконання завдань; прямокутник *Диспетчер* символізує програму-планувальник, задача якої розподіляти завдання між обчислювальними вузлами системи. Потрібно визначити метод, яким буде керуватися дана програма.

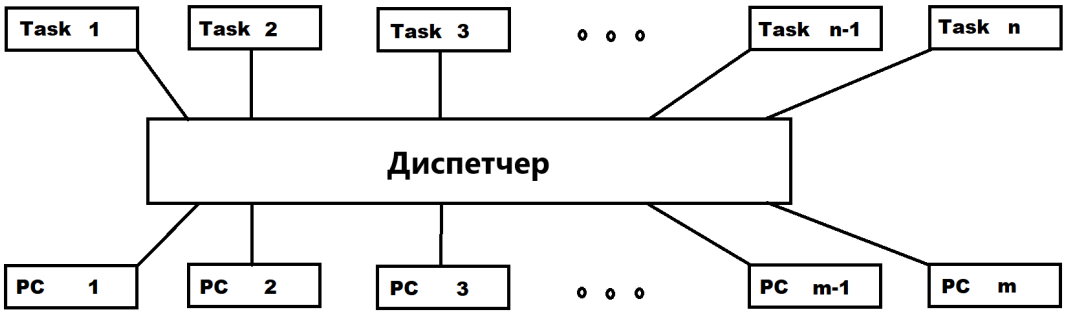


Рисунок 2.1 – Схематичне зображення постановки задачі диспетчеризації

2.2 Новий підхід до планування завдань у GRID-системах

Як згадувалося в попередньому розділі, нам потрібно визначити метод, відповідно до якого буде проводитись розподіл завдань між обчислювальними вузлами в GRID-системі з невідчужуваними ресурсами.

Після тривалих пошуків, автором був знайдений новий підхід, який міг би виконати таке завдання. Цей підхід базується на законі балансу сил, або принципу рівноваги. Суть цього закону полягає в наступному: для кожної сили існує інша сила, яка врівноважує її. Обидві сили приблизно однакові і вони постійно змінюють свої потенціали. Коли потенціал першої сили збільшується, регулятор автоматично включається і відбувається процес вирівнювання потенціалів. Цей закон є універсальним, і розуміння його дозволить, на думку автора, використовувати не лише для розв’язку даної задачі, а й в інших сферах життя.

Описаний вище закон є більш високого рівня ніж третій закон Ньютона, який звучить так: «Сили взаємодії двох матеріальних точок рівні за величиною, протилежно спрямовані, і діють вздовж прямої, що з'єднує ці матеріальні точки» [100], так як він передбачає наявність ще третьої сили в даній системі сил, третя сила виконує роль регулювальника (диспетчера), задача якого є збереження системи протидіючих сил.

На рис. 2.2 схематично показано основну суть запропонованого підходу до розв’язку задачі диспетчеризації завдань у GRID-системах з невідчужуваними ресурсами / Desktop GRID, що базується на даному законі. Ідея полягає в тому, щоб розглянути завдання, які потребують вирішення, як одну силу, а обчислювальні вузли, на яких вони повинні виконуватися, як іншу силу, які протидіють одна одній. Роль диспетчера має виконувати програма, що працює за певним алгоритмом, і розподіляє завдання таким чином, щоб підтримувати баланс сил.

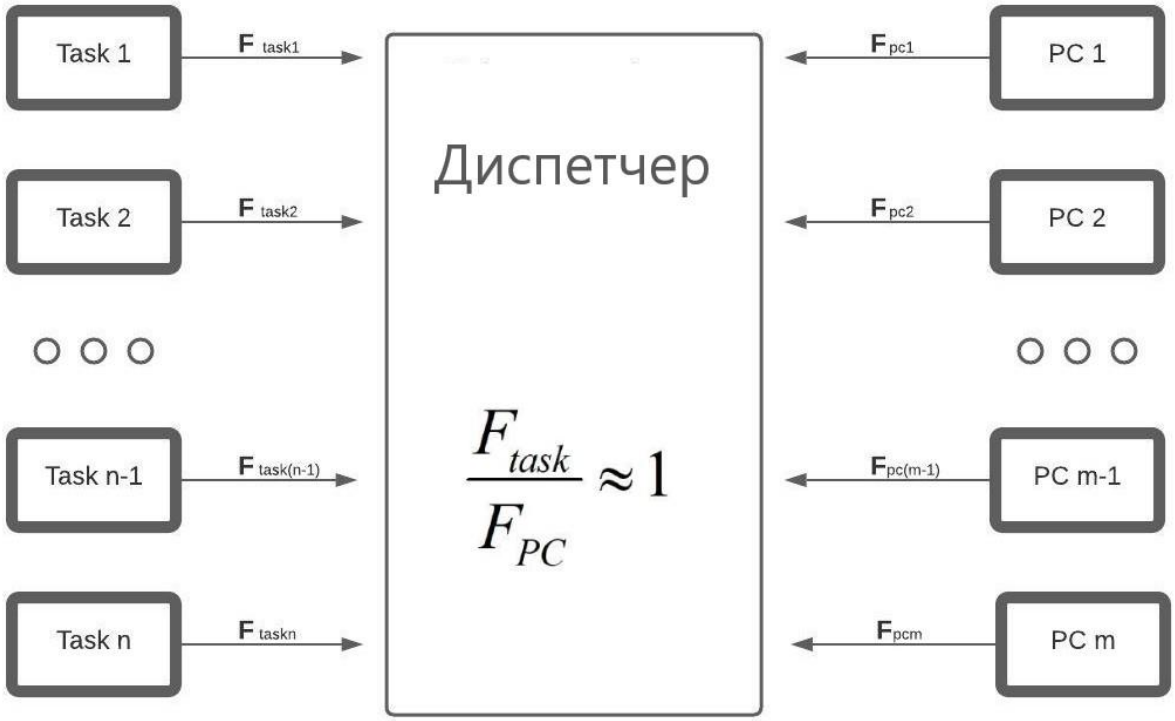


Рисунок 2.2 – Схематичне зображення нового підходу

Враховуючи, що такі поняття, як «сила обчислювального завдання» та «сила обчислювального вузла» досить абстрактні поняття, пропонується перейти на більш нижчий рівень абстракції та замінити їх на поняття відповідно "потужність завдання" та "потужність вузла". І здійснювати розподіл вже згідно балансу цих потужностей [100].

Такий вибір можна пояснити наступним чином: з елементарної фізики відомо, що потужність *P* дорівнює відношенню роботи до часу [101]:

$$P = \frac{A}{t}, \tag{2.1}$$

де *A* – робота;

t – час.

Робота *A*, в свою чергу, дорівнює силі помноженій на шлях, який потрібно пройти [101]:

$$A = F \times s, \tag{2.2}$$

де *F* – сила, яку треба прикласти;

s – шлях, що потрібно пройти.

Враховуючи формулу (2.2) можемо привести формулу (2.1) до наступного вигляду:

$$P = \frac{F \times s}{t}.$$

Зробивши деякі перетворення отримаємо формулу для визначення сили:

$$F = \frac{P \times t}{s}.$$

Маючи формулу для визначення сили в загальному вигляді, можемо конкретизувати її для обчислення сили обчислювального вузла (позначимо її F_{PC}):

$$F_{PC} = \frac{P_{PC} \times t_{PC}}{s_{PC}}, \tag{2.3}$$

де t_{PC} – час роботи обчислювального вузла;

s_{PC} – шлях, який має пройти обчислювальний вузол.

Аналогічною буде формула для визначення сили обчислювального завдання (позначимо її F_{task}):

$$F_{task} = \frac{P_{task} \times t_{task}}{s_{task}} \tag{2.4}$$

де t_{task} – час виконання обчислювального завдання;

s_{task} – шлях, який має пройти обчислювальне завдання.

Відповідно до запропонованого підходу, розподіляти завдання по обчислювальним вузлам потрібно таким чином, щоб сила завдання була рівна (або ж якнайближча) до сили обчислювального вузла:

$$F_{task} \approx F_{PC} \tag{2.5}$$

Згідно формул (2.3) та (2.4) перепишемо вираз (2.5):

$$\frac{P_{task} \times t_{task}}{s_{task}} \approx \frac{P_{PC} \times t_{PC}}{s_{PC}} \tag{2.6}$$

Враховуючи, що при виконанні завдання на обчислювальному вузлі, час виконання завдання t_{task} буде еквівалентний часу завантаженості вузла t_{PC}, а шлях s_{task}, який має пройти завдання (перейти від стану «невиконане» до стану «виконане»), дорівнює шляху s_{PC}, який повинен пройти обчислювальний вузол (перевести обчислювальне завдання від стану «невиконане» до стану «виконане»), то величини t_{task}, t_{PC}, s_{task}, s_{PC} можемо скоротити, і тоді формула (2.6) приймає вигляд:

$$P_{task} \approx P_{PC}.$$

Отже, розподіл завдань у GRID-системах з невідчужуваними ресурсами / Desktop GRID відповідно до закону балансу сил, не втрачаючи чинності, можна звести до такого, що використовує поняття потужності обчислюваного завдання P_{task} та поняття потужності обчислювального вузла P_{PC}, за умови балансу кількісних величин, що відповідають даним поняттям.

Дані величини є величинами відносними і на кожній ітерації можуть бути перерахованими в залежності від зміни стану GRID-системи [102].

2.3 Метод диспетчеризації завдань FSA

На основі описаного вище підходу автором розроблено метод диспетчеризації завдань у GRID-системах з невідчужуваними ресурсами / Desktop GRID, який отримав назву FSA (*Flows Scheduling Algorithm*).

Для реалізації цього методу нам потрібно визначити, що буде розумітися під поняттями потужності обчислювального вузла та потужності обчислювального завдання.

Потужність обчислювального завдання – відносна величина, яка є сукупністю всіх або ж тільки деяких (можливе використання лише однієї) характеристик обчислювального завдання, які вона може мати, визначених певним методом. Дана величина є змінною і залежить від набору завдань, що знаходяться в черзі.

Потужність обчислювального вузла – відносна величина, яка є сукупністю всіх або ж тільки деяких (можливе використання лише однієї) характеристик обчислювального вузла, які він може мати, визначених певним методом. Дана величина є змінною і залежить від набору задіяних обчислювальних вузлів.

На рис. 2.3 схематично зображено описані вище поняття потужностей обчислювальної задачі та обчислювального вузла. Дані поняття є відносними величинами і тому можуть обчислюватися різними способами, залежно від різних характеристик як завдань так і вузлів.

Зокрема, у обчислювальної задачі можуть бути такі характеристики:

– обчислювальна складність;
– алгоритмічна складність;
– об’єм даних;
– розмір файлів та ін.

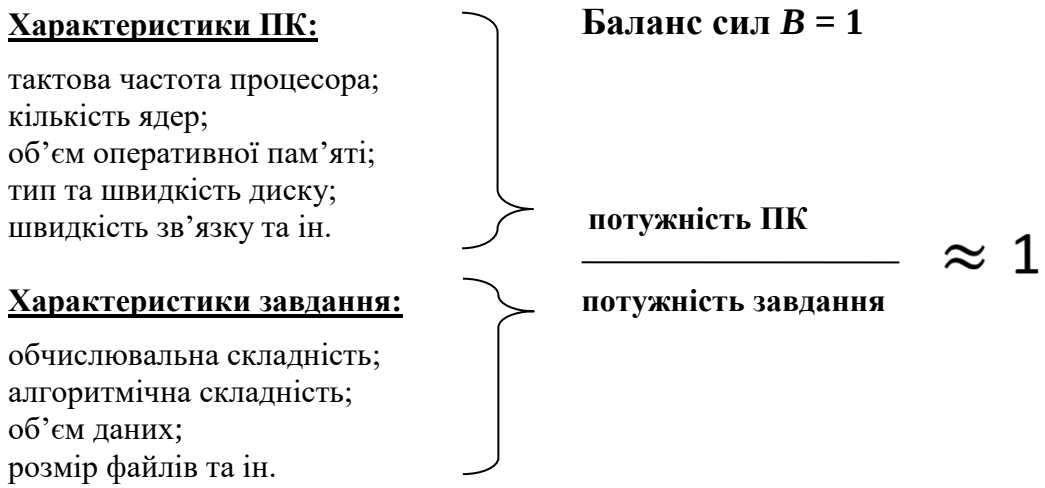


Рисунок 2.3 – Схематичне зображення понять потужності обчислюваної задачі та потужності обчислювального вузла

У обчислювального вузла можуть бути такі характеристики, кожна з яких впливає на продуктивність та швидкодію обчислень:

– тактова частота процесора;
– кількість ядр процесора;
– об’єм оперативної пам’яті;
– тип та швидкість диску;
– швидкість каналу зв’язку;
– наявне програмне забезпечення та ін.

Перелічені вище характеристики обчислювальної задачі та обчислювального вузла можуть бити використані для визначення потужностей завдань, що потребують вирішення, та вузлів, на яких відбуваються обчислення. Використовувати можна як одну з них, так і сукупність всіх, зведених певним чином до однієї величини.

Необхідно зазначити, що зведення такої кількості характеристик до однієї величини – задача досить складна і для обчислюваного завдання, і для обчислювального вузла. Більше того, якщо для ПК все ще можна використовувати якусь загальну формулу для обчислення його потужності, то для потужності завдання така формула буде змінюватися кожного разу,

залежно від типу завдань, що потребують вирішення. Але все ж таки існує цілий ряд прикладних завдань, для яких це досить легко вирішувати.

Нижче в даній роботі ще буде висвітлюватись дане питання, а зараз опишемо сам метод диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами / Desktop GRID, що пропонується.

Метод FSA (загальна форма):

1) баланс сил = 1;
2) існує *n* завдань і *m* вузлів;
3) вибрати *перше* завдання;
4) знайти такий вузол з множини *m*, при якому відношення *першого* завдання до нього було б якомога ближче до 1;
5) відправити на виконання *перше* завдання до знайденого вузла;
6) вибрати *друге* завдання;
7) знайти такий вузол з множини *(m-1)*, при якому відношення *другого* завдання до нього було б якомога ближче до 1;
8) відправити на виконання *друге* завдання до знайденого вузла;
9) вибрати *n-не* завдання;
10) знайти такий вузол з множини *(m-n)*, при якому відношення *n-го* завдання до нього було б якомога ближче до 1;
11) відправити на виконання *n-не* завдання до знайденого вузла.

Описаний вище метод та постановка задачі є досить високого абстрактного рівня і тому сформулюємо її ж більш чітко, з математичної точки зору.

Постановка задачі. Нехай є GRID-система з невідчужуваними ресурсами / Desktop GRID з *n* обчислювальними завданнями та *m* обчислювальними вузлами. Позначимо потужність обчислюваного завдання як P_n, та потужність обчислювального вузла як R_m.

Введемо поняття балансу сил системи *B* в сенсі описаного вище підходу як стан системи, коли потужність *i-го* обчислюваного завдання

наближається до потужності *j-го* обчислювального вузла:

$$B = \frac{P_i}{R_j} \approx 1.$$

Для ідеального випадку дані потужності рівні:

$$\frac{P_i}{R_j} = 1.$$

Отже, маємо потужності обчислювальних завдань $P = \{P_1, P_2, P_3, .., P_n\}$ та потужності обчислювальних вузлів $R = \{R_1, R_2, R_3, .., R_m\}$. Потрібно розподілити завдання по вузлах таким чином, щоб для і-го завдання та j-го вузла мінімізувалось би значення виразу:

$$\left| \frac{P_i}{R_j} - 1 \right|. \qquad (2.7)$$

Алгоритм FSA:

1) встановлюємо: $B = 1$;
2) існує *n* завдань і *m* вузлів;
3) обчислюємо: $P = \{P_1, P_2, P_3, .., P_n\}$, $R = \{R_1, R_2, R_3, .., R_m\}$;
4) обираємо *i-те* завдання, де $i \in \{1, .., n\}$;
5) знаходимо пару *i–j*, де $j \in \{1, .., m\}$, для якої мінімізується вираз:

$$\left| \frac{P_i}{R_j} - 1 \right|;$$

6) відправляємо *i-те* завдання на *j-ий* вузол;
7) якщо є невідправлені завдання, то переходимо до п. 2;
8) завершуємо роботу.

Враховуючи, що баланс сил повинен бути близьким до 1, а потужності завдань та вузлів вимірюються однаковими одиницями, можна провести деякі перетворення над формулою (2.7). В ідеальному випадку вираз $\frac{P_i}{R_j} - 1$

повинен бути рівним нулю: $\frac{P_i}{R_j} - 1 = 0$, тоді вираз (2.7), який потрібно мінімізувати, зводиться до вигляду:

$$\left| P_i - R_j \right|. \tag{2.8}$$

Таке перетворення трохи спростить сам алгоритм та дасть незначний приріст його швидкодії, адже зменшить кількість операцій процесора при обчисленнях. На перший погляд може здатися, що такі перетворення не мають особливого сенсу, але з практики програмування відомо, що оптимізація ніколи не буває зайвою і при роботі алгоритму на сильно завантажених системах таке покращення дасть свої результати.

Після спрощення алгоритм прийме наступний вигляд:

Алгоритм FSA:

1) встановлюємо: $B = 1$;
2) існує *n* завдань і *m* вузлів;
3) обчислюємо: $P = \{P_1, P_2, P_3, .., P_n\}$, $R = \{R_1, R_2, R_3, .., R_m\}$;
4) обираємо *i-те* завдання, де $i \in \{1, ., n\}$;
5) знаходимо пару *i–j*, де $j \in \{1, ., m\}$, для якої мінімізується вираз:

$$\left| P_i - R_j \right|;$$

6) відправляємо *i-те* завдання на *j-ий* вузол;
7) якщо є невідправлені завдання, то переходимо до п. 2;
8) завершуємо роботу.

На рис. 2.4 зображений псевдокод для даного алгоритму, який був використаний при побудові програмного комплексу для симуляції роботи GRID-системи з невідчужуваними ресурсами та дослідження алгоритмів диспетчеризації завдань, що буде описаний в наступному розділі. Даний псевдокод є лише орієнтовним і не може бути використаний буквально, адже системи можуть будуватися на різних платформах та з використанням різних

мов програмування. Від демонструє лише основні кроки, які повинен виконати алгоритм.

```
while tasks in meta-tasks
        for all tasks in meta-tasks
                calculate Ptask;
        endfor;
        for all pcs in meta-pc
                calculate Ppc;
        endfor;
        find minimum delta Ptask – Ppc;
        send Ptask to Ppc;
endwhile;
```

Рисунок 2.4 – Програмний псевдокод до алгоритму FSA

На рис. 2.5 зображена блок-схема алгоритму FSA, що демонструє основні кроки, які повинен виконати диспетчер для здійснення розподілу завдань по обчислювальним вузлам відповідно до запропонованого методу.

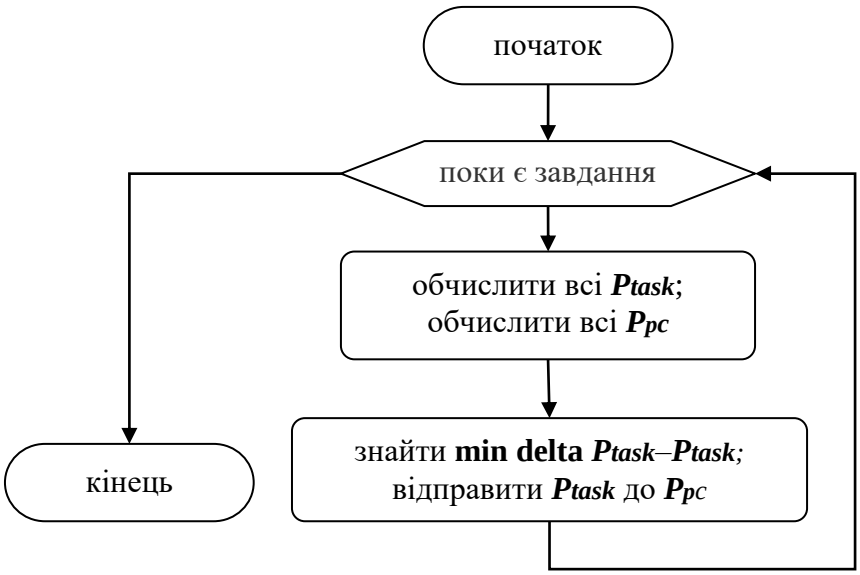


Рисунок 2.5 – Блок-схема до алгоритму FSA

Для вирішення задачі диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами / Desktop GRID згідно запропонованого автором методу FSA та побудованого на його основі алгоритму, потрібно визначити, як знайти взаємозв'язок між завданнями та вузлами, для того, щоб можна було зрівнювати їх між собою. Сам алгоритм не передбачає такого вирішення, і покладає дане питання на розробників, що будуть використовувати його. Адже рішення даної задачі буде залежати від того, які завдання планується обчислювати на GRID-системі з невідчужуваними ресурсами / Desktop GRID, а також які обчислювальні вузли мають бути задіяні.

Описаний вище метод FSA розроблений для послідовних завдань, що не можуть бути розділені на декілька частин та бути виконаними на декількох обчислювальних вузлах. Тобто, метод FSA не дозволяє розпаралелювати завдання.

Для завдань, що можуть бути розпаралелені на декілька обчислювальних вузлів чи процесорів розроблений окремий метод, який буде описано нижче.

2.4 Метод диспетчеризації завдань FSA_P

Як написано вище, розроблений метод FSA диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами / Desktop GRID не припускає, що в черзі можуть бути завдання, які можна розділити на ряд частин та виконати ці частини окремо одна від одної на всіх обчислювальних вузлах системи. Після чого, отримані під час виконання результати об’єднати і визначити загальний результат для кожного конкретного завдання. Тобто, розпаралелити завдання.

Якщо розглянути одне велике завдання, яке можна розподілити між обчислювальними вузлами на ряд частин та виконати ці частини паралельно, з одного боку системи сил, а саму GRID-систему з невідчужуваними

ресурсами з різними за своїми характеристиками обчислювальними вузлами – з іншого, тоді можливо застосувати запропонований підхід балансу сил. Де обчислювальне завдання – перша сила, а GRID-система – друга сила.

Нам потрібно розпаралелювати це завдання таким чином, щоб воно виконалось якомога швидше.

Використовуючи закон балансу сил, це можна зробити наступним чином: треба визначити потужність кожного обчислювального вузла системи та загальну потужність всієї системи, після чого розділити своє завдання на набір підзадач, кількість яких має відповідати кількості вузлів, потужність кожної підзадачі повинна бути пропорційна потужності відповідного обчислювального вузла. Потім отримані підзадачі відправити на відповідні вузли.

В даному випадку виконання завдань відбувається послідовно, при цьому кожне завдання розпаралелюється на низку частин, кількість яких дорівнює кількості обчислювальних вузлів системи. Вважається, що кожне завдання має якусь відому числову характеристику (наприклад, кількість операцій, розмір файлу тощо). Дана характеристика є фізичною величиною кожного з завдань, і не може змінюватися. При цьому, використовуючи дану характеристику, можливо здійснювати поділ завдання на безліч менших за розміром завдань.

Для прикладу, розглянемо таке завдання: скажімо, що у нас є один великий файл Excel, у якому заповнена таблиця розміром 10000×1000 якимись даними, які потрібно програмно опрацювати, при цьому дані не зв’язні між собою і їх можна оброблювати паралельно на багатьох обчислювальних вузлах. І є 15 різнорідних обчислювальних вузлів, що можуть бути задіяні для обчислень. Тоді ми можемо відправити кожному з вузлів даний файл, а також задати параметри комірок для кожного з вузлів – з якої по яку виконувати обробку файлу. Ці параметри можна розраховувати вже з використанням описаного вище поняття потужності обчислювального вузла.

Враховуючи, що обчислювальні завдання можуть бути різними за своїми характеристиками, в методі використовується абстрактне поняття V – *об'єм завдання,* яке для різного завдання може означати різні властивості, головне, щоб по ньому можливо було ділити завдання.

Постановка задачі. Нехай є множина n завдань, що потребують вирішення та можуть бути розпаралелені на довільну кількість вузлів, $V=\{V_1,V_2,V_3,.,V_n\}$, де V_i — об'єм *i-го* завдання та GRID-система з невідчужуваними ресурсами з множиною m обчислювальних вузлів $R=\{R_1,R_2,R_3,.,R_m\}$, де R_j – потужність *j-го* обчислювального вузла з даної множини. Необхідно розподілити завдання по вузлам таким чином, щоб час виконання всієї черги був би мінімальним.

Для диспетчеризації завдань, які можна легко розпаралелювати, автором розроблений метод, що отримав назву FSA_P (*Flow Scheduling Algorithm Parallel*). На базі даного методу розроблено алгоритм, що наведений нижче.

Алгоритм FSA_P:

1) встановлюємо $B=1$;
2) існує n завдань з об'ємом $V=\{V_1,V_2,V_3,.,V_n\}$ і m вузлів;
3) обчислюємо потужності вузлів $R=\{R_1,R_2,R_3,.,R_m\}$ та сумарну потужність системи: $\sum R$;
4) обираємо V_i, де V_i — об'єм *i-го* завдання з множини V, відповідно до порядку в черзі;
5) для всіх вузлів знаходимо: $V_{ij}=\frac{V_i\times R_j}{\sum R}$, де $j\in\{1,.,m\}$;
6) відправляємо завдання з об'ємом V_{ij} на виконання *j-му* вузлу;
7) якщо в системі є не розподілені завдання, переходимо до п. 2;
8) завершуємо роботу.

В описаному вище методі, баланс сил B зберігається завдяки

пропорційному розподілу завдання між вузлами. Кожному вузлу відправляється частина завдання відповідно до його потужності. Наприклад, якщо є завдання з об’ємом 10 і три вузли з потужністю 0.2, 0.3, 0.5, тоді завдання буде розділене на три частини з об’ємом 2, 3, 5, відповідно до величин потужностей. Даний приклад досить простий і наведений суто для кращого розуміння.

На рис. 2.6 зображений псевдокод для даного алгоритму.

```
for all pcs in meta-pc
        calculate Ppc;
endfor;
calculate sumP;
for all tasks in meta-tasks
        for all pcs in meta-pc
                send task * Ppc / sumP;
        endfor
endfor
```

Рисунок 2.6 – Програмний псевдокод до алгоритму FSA_P

На рис. 2.7 зображена блок-схема запропонованого алгоритму FSA_P, що досить добре демонструє основні кроки, які повинен виконати диспетчер для здійснення розподілу завдань по обчислювальним вузлам відповідно до запропонованого підходу та розробленому на його основі методу диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами / Desktop GRID.

Цей метод дуже ефективний і дозволяє майже ідеально розпаралелювати завдання. У цьому випадку система буде працювати максимально ефективно, і час виконання буде швидким. Крім того, як показують результати експерименту, вузли завершують свою роботу майже

одночасно. Що, безумовно, дуже добре, оскільки час очікування, поки всі вузли не звільняться, у цьому випадку буде мінімальним.

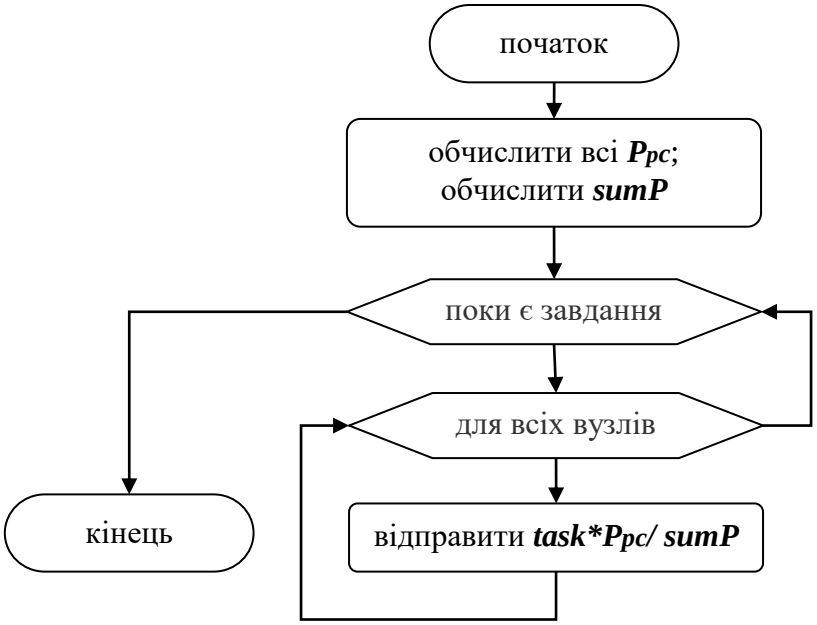


Рисунок 2.7 – Блок-схема до алгоритму FSA_P

Під час проведення досліджень розроблених методів диспетчеризації завдань FSA та FSA_P було визначено ряд випадків, коли дані методи дають гірші результати ніж прогнозувалось. Такі випадки стали причиною для пошуку можливих варіантів покращень та модифікацій для них. В результаті яких автором було розроблено ще два методи (FSA Min, FSA Max) диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами / Desktop GRID: .

Нижче буде детально описано дані методи. А в розділі, що присвячений тестуванню, описано випадки, при яких дані методи дають кращі результати.

2.5 Метод диспетчеризації завдань FSA Min

Даний метод диспетчеризації завдань отримав свою назву не випадково. Справа в тому, що це модифікований метод FSA, який поєднується з загальновідомим методом балансування навантаження Min-Min. Про метод

Min-Min вже писалось в першому розділі, коли проводився огляд методів диспетчеризації завдань для динамічного планування.

Відповідно до методу Min-Min, розподіл здійснюється таким чином: завдання з мінімальним передбачуваним часом виконання призначається ресурсу з мінімальним загальним часом виконання всіх завдань.

З методу Min-Min взята ідея розподілу за принципом мінімальне завдання на виконання до мінімального вузла. Але використовується не прогнозований час виконання, а потужність завдання та потужність обчислюваного вузла. І такий розподіл здійснюється тільки на початковій стадії розподілу, потім же розподіл продовжується відповідно до методу FSA.

Визначений він експериментальним шляхом. Коли в системі кількість завдань дорівняє або ж менше ніж кількість обчислювальних вузлів він дає кращий результати у порівнянні з методом FSA. І хоча такі стани системи виникають у GRID-системах з невідчужуваними ресурсами / Desktop GRID дуже рідко, все ж таки використання даного методу дасть певну перевагу.

Нижче наведено постановку задачі для таких випадків.

Постановка задачі. Нехай є множина *n* завдань, що потребують вирішення $P=\{P_1,P_2,P_3,.,P_n\}$, де P_i — потужність *i-го* завдання та GRID-система з невідчужуваними ресурсами з множиною *m* обчислювальних вузлів $R=\{R_1,R_2,R_3,.,R_m\}$, де R_j – потужність *j-го* обчислювального вузла з даної множини. При цьому $n\le m$. Необхідно розподілити завдання по вузлам таким чином, щоб час виконання всієї черги був би мінімальним.

Нижче наведено даний алгоритм:

Алгоритм FSA Min:

1) встановлюємо: $B=1$;
2) існує *n* завдань і *m* вузлів, $n\le m$;
3) обчислюємо: $P=\{P_1,P_2,P_3,.,P_n\}$, $R=\{R_1,R_2,R_3,.,R_m\}$;
4) обираємо *i-me* завдання, де $i\in\{1,.,n\}$;

5) знаходимо пару *i*–*j*, де $i \in \{1,.,n\}$, $j \in \{1,.,m\}$, для якої виконуються умови: $P_i = \min P$, $R_j = \min R$;

6) відправляємо *i-те* завдання на *j-ий* вузол;

7) якщо є невідправлені завдання, то переходимо до п. 2;

8) завершуємо роботу.

На рис. 2.8 зображений псевдокод для алгоритму FSA Min. Даний псевдокод був використаний при побудові програмного комплексу для симуляції роботи GRID-системи з невідчужуваними ресурсами та дослідження алгоритмів диспетчеризації завдань.

```
while tasks in meta-tasks
        for all tasks in meta-tasks
                calculate Ptask
        endfor
        for all pcs in meta-pc
                calculate Ppc
        endfor
        find Ptask = min meta-tasks;
        find Ppc = min  meta- pc;
        send Ptask to Ppc;
endwhile;
```

Рисунок 2.8 – Програмний псевдокод до алгоритму FSA Min

На рис. 2.9 зображена блок-схема запропонованого алгоритму FSA Min, що демонструє основні кроки, які повинен виконати диспетчер для здійснення розподілу завдань по обчислювальним вузлам відповідно до запропонованого підходу та розробленого на його основі методу диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами / Desktop GRID.

Описаний вище метод FSA Min розроблений для послідовних завдань, що не можуть бути розпаралелені.

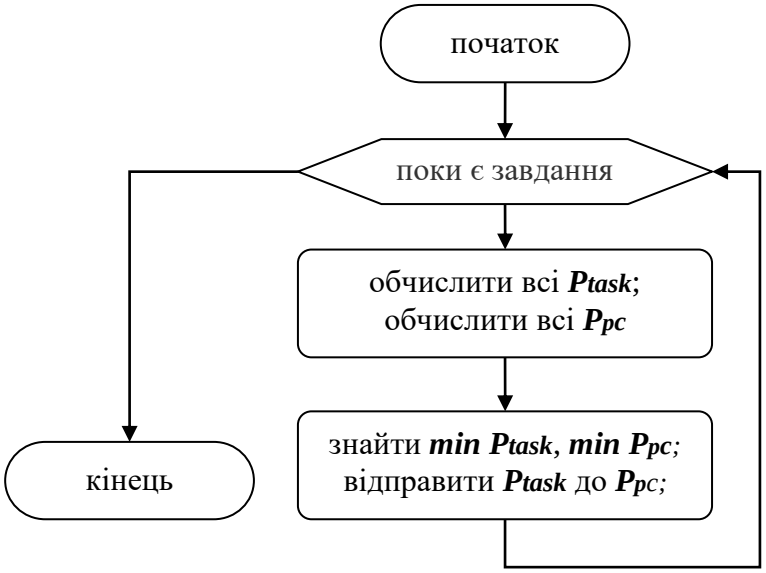


Рисунок 2.9 – Блок-схема до алгоритму FSA Min

Цей метод дає кращі результати ніж FSA, при умові початкового стану системи, коли кількість завдань, що потребують вирішення менше або ж рівна кількості обчислювальних вузлів у системі.

2.6 Метод диспетчеризації завдань FSA Max

FSA Max – модифікований метод FSA, аналогічно методу FSA Min, який поєднується з загальновідомим методом балансування навантаження Max-Min. Про даний метод також вже писалось в першому розділі, коли проводився огляд методів диспетчеризації завдань для динамічного планування.

Відповідно до методу Max-Min, розподіл здійснюється подібно методу Min-Min тільки завдання вибираються не з мінімальним прогнозованим часом завершення, а з максимальним. Вузли ж підбираються з мінімальним загальним часом виконання всіх завдань.

З методу Max-Min взята лише ідея розподілу за принципом максимальне завдання на виконання до максимального вузла. Але використовується не прогнозований час виконання, а потужність завдання та

потужність обчислювального вузла. І такий розподіл здійснюється тільки на початковій стадії розподілу, потім же розподіл продовжується відповідно до методу FSA.

Визначений він експериментальним шляхом. Коли в системі кількість завдань дорівняє або ж менше ніж кількість обчислювальних вузлів він дає кращий результат, у порівнянні з методом FSA, близький до методу FSA Min. Але у випадку, коли кількість завдань менше кількості вузлів та при середній потужності системи від 0,15 до 0,35 дає кращі результати ніж метод FSA Min. І хоча такі стани системи дуже рідко виникають, все ж таки використання даного методу при таких випадках дасть певну перевагу та більшу ефективність. Нижче наведено постановку задачі для таких випадків.

Постановка задачі. Нехай є множина *n* завдань, що потребують вирішення $P=\{P_1,P_2,P_3,..,P_n\}$, де P_i — потужність *i-го* завдання та GRID-система з невідчужуваними ресурсами з множиною *m* обчислювальних вузлів $R=\{R_1,R_2,R_3,..,R_m\}$, де R_j – потужність *j-го* обчислювального вузла з даної множини. При цьому $n\le m$ та $0.15<P_{сер}<0.35$, де $P_{сер}$ – зважена нерівномірність всіх вузлів системи. Необхідно розподілити завдання по вузлам таким чином, щоб час виконання всієї черги був би мінімальним. Нижче наведено даний алгоритм:

Алгоритм FSA Max:

1) встановлюємо: $B=1$;
2) існує *n* завдань і *m* вузлів, $n\le m$, $0.15<P_{сер}<0.35$;
3) обчислюємо: $P=\{P_1,P_2,P_3,..,P_n\}$, $R=\{R_1,R_2,R_3,..,R_m\}$;
4) обираємо *i-те* завдання, де $i\in\{1,..,n\}$;
5) знаходимо пару *i*–*j*, де $i\in\{1,..,n\}$, $j\in\{1,..,m\}$, для якої виконуються умови: $P_i=\max P$, $R_j=\max R$;
6) відправляємо *i-те* завдання на *j-ий* вузол;
7) якщо є невідправлені завдання, то переходимо до п. 2;

8) завершуємо роботу.

На рис. 2.10 зображений псевдокод для алгоритму FSA Max, що використовувався при побудові програмного комплексу для симуляції роботи GRID-системи з невідчужуваними ресурсами та дослідження алгоритмів диспетчеризації завдань.

```
while tasks in meta-tasks
        for all tasks in meta-tasks
                calculate Ptask;
        endfor;
        for all pcs in meta-pc
                calculate Ppc;
        endfor;
        find Ptask = max meta-tasks;
        find Ppc = max meta- pc;
        send Ptask to Ppc;
endwhile
```

Рисунок 2.10 – Програмний псевдокод до алгоритму FSA Max

На рис. 2.11 зображена блок-схема алгоритму FSA Max, що демонструє основні кроки, які повинен виконати диспетчер відповідно до запропонованого підходу та розробленого на його основі методу.

Описаний вище метод FSA Max диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами / Desktop GRID розроблений автором для послідовних завдань, що не можуть бути розпаралелені.

Цей метод дає кращі результати ніж FSA, при умові початкового стану системи, коли кількість завдань, що потребують вирішення, менше або ж рівна кількості обчислювальних вузлів та середній потужності системи від 0,15 до 0,35.

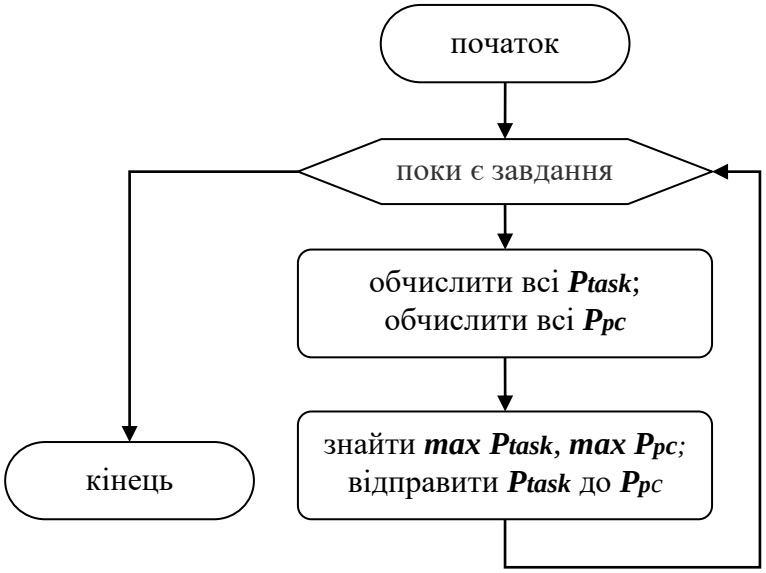


Рисунок 2.11 – Блок-схема до алгоритму FSA Max

2.7 Висновки до розділу 2

1) Описано постановку задачі диспетчеризації завдань для GRID-системи з невідчужуваними ресурсами.

2) Автором запропоновано новий підхід до вирішення задачі диспетчеризації завдань для GRID-системах з невідчужуваними ресурсами / Desktop GRID, який побудований на основі закону балансу сил. Відповідно до даного підходу кожне завдання, що потребує вирішення, розглядається як перша сила, а кожен обчислювальний вузол системи розглядається як друга сила, що протидіє першій, а диспетчер задач розглядається як третя сила, задача якого є збереження балансу сил всієї системи. Запропонований підхід є універсальним і дозволяє планувати завдання як для статичних так і для динамічних систем.

3) На основі запропонованого підходу автором розроблено метод диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами FSA. Даний метод розроблено для послідовних завдань, що не можуть бути розділені на частини та виконуватися на декількох вузлах одночасно, а потребують послідовного вирішення. В основу даного методу закладено

поняття потужності завдання та потужності обчислювального вузла, які випливають з поняття сили завдання та сили вузла, що описані в запропонованому підході. До даного методу автором розроблено алгоритм, наведено програмний псевдокод, що використовується при розробці програмного комплексу для симуляції роботи GRID-системи з невідчужуваними ресурсами, приведено блок-схему для даного алгоритму.

4) На основі запропонованого підходу автором розроблено метод диспетчеризації завдань у GRID-системі з невідчужуваними ресурсами FSA_P. Даний метод розроблено для завдань, що можуть бути розділені на частини та виконуватися на декількох вузлах одночасно. До даного методу розроблено алгоритм, наведено програмний псевдокод та блок-схему.

5) На основі запропонованого підходу автором розроблено ще два методи FSA Min та FSA Max, що є комбінаціями загальновідомих методів Min-Min та Max-Min з методом FSA. Дані методи були визначені експериментально та в деяких випадках дають кращі результати ніж FSA. Зокрема при станах системи коли кількість завдань в черзі менше кількості вузлів. Наведено їх алгоритми, псевдокод та блок-схеми.

РОЗДІЛ 3
ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ ДОСЛІДЖЕННЯ МЕТОДІВ ДИСПЕТЧЕРИЗАЦІЇ ЗАВДАНЬ ДЛЯ GRID-СИСТЕМ З НЕВІДЧУЖУВАНИМИ РЕСУРСАМИ

Для проведення подальших досліджень методів, що описані в розділі 2, потрібно провести порівняння їх ефективності з методом FCFS. Для виконання цього завдання потрібні програмні засоби, що могли б надати таку можливість.

Тому було вирішено провести аналіз існуючих програмних засобів, що дозволяють моделювати GRID-системи та системи Desktop GRID, а також досліджувати та порівнювати між собою різні методи планування завдань. При цьому основною задачею аналізу було визначення, чи дозволяє той чи інший інструмент враховувати потужності завдань та потужності вузлів. А також можливість запуску даних програм не лише на одному персональному комп’ютері, а й на цілому ряді. Також хотілося б, щоб обчислювальні вузли могли б передавати дані по мережі Інтернет.

3.1 Аналіз існуючих засобів моделювання GRID-систем

Аналіз існуючого програмного забезпечення для моделювання GRID-систем показав, що наразі існує чимало програмних засобів, які створені для симуляції роботи GRID-систем та великих комп’ютерних мереж, а також для дослідження методів планування завдань в них.

В [103] проведено огляд симуляторів, серед яких виділено: OMNET++ [104], Bricks [105], MicroGRID [106, 107], SimGRID [108, 109], GRIDSim [110, 111]. Нижче наведено короткий опис кожного з виділених симуляторів.

OMNET++ — програмний пакет для моделювання роботи великих комп’ютерних мереж, здебільшого орієнтований на імітаційний комп'ютер

мережі та інші розподілені системи. OMNET++ повністю програмований і модульний. Він був розроблений для моделювання дуже великих мереж, має можливість легко відстежувати та налагоджувати імітаційну модель, має потужний графічний інтерфейс. Це робить внутрішні елементи імітаційної моделі повністю видимими для людини, яка виконує симуляцію: відображається мережева графіка і потік повідомлень, користувачеві дозволяється зазирнути в об'єкти та змінити їх в рамках моделі. Ці особливості роблять OMNET++ хорошим об'єктом як для досліджень, так і для навчальних цілей. Модуль моделювання OMNET++ може бути вбудованим у великі програми. OMNET++ є відкритим джерелом, безкоштовним для некомерційного використання [112].

Bricks – система оцінки продуктивності мережі, яка дозволяє аналізувати та порівнювати різні схеми планування на типових високопродуктивних глобальних обчислювальних установках. Призначена для моделювання глобальної мережі та різних вузлів в системі. Може імітувати різноманітну поведінку глобальних обчислювальних систем, особливо поведінку мереж та алгоритмів планування ресурсів. Bricks компонується так, що її складові можуть бути замінені для імітації різних системних алгоритмів, вона також дозволяє включити існуючі глобальні обчислювальні компоненти через зовнішній інтерфейс [113].

MicroGRID – симулятор**,** основною задачею якого є моделювання GRID експериментів у віртуальному середовищі. Це є віртуальне середовище, яке імітує роботу GRID-системи та підтримує GRID-програми, що використовують проміжне програмне забезпечення Globus GRID [114].

SimGRID – фреймворк, написаний мовою програмування C/C++, задуманий як інструмент для вивчення розподілених алгоритмів. Його сучасний інтерфейс S4U дозволяє легко оцінювати хмарні, P2P, HPC, IoT і подібні налаштування. Типове моделювання SimGRID складається з декількох елементів — «акторів», які виконують функції, що надаються користувачем. Ці елементи повинні виконувати свої обчислення, передавати

дані, використовувати диск чи виконувати інші дії, чітко слідуючи за інтерфейсом S4U, що відображається в симуляторі. Ця діяльність відбувається на ресурсах (хости, посилання, диски). SimGRID передбачає час, який забирає кожна активність, і відповідно організовує «акторів», які чекають завершення цієї активності [115].

GRIDSim – це зріла система моделювання GRID з відкритим кодом, яка існує понад вісім років. Він був реалізований на мові програмування Java і неодноразово використовувався для моделювання алгоритмів у сітці. Окрім базових функцій, таких як моделювання задач та моделювання мережевих об'єктів (користувачів, додатків, ресурсів, посередників ресурсів тощо), він пропонує розширені функції, такі як модель аукціону, мережа даних або розширення мережі [116].

Системи Bricks та MicroGRID створені для моделювання та оцінки ефективності великих комп’ютерних мереж, а для дослідження алгоритмів диспетчеризації завдань їм не достатньо функціональних можливостей і потребують значного доопрацювання. Крім того, система MicroGRID працює у віртуальному середовищі, що потребує великої апаратної потужності системи, тобто кластер чи хоча б суперкомп’ютер, а не звичайного ПК.

Найбільш потужний інструмент серед зазначених вище є саме SimGRID або його наступне покоління SimGRID 2. Даний фреймворк, розроблений саме для дослідження алгоритмів диспетчеризації в навантажених системах та мережах, постійно підтримується і розвивається. Тому він найчастіше зустрічається в наукових працях дослідників з усього світу, як інструмент для дослідження нових алгоритмів та порівняння їх з існуючими.

Окрім позитивних властивостей SimGRID має і ряд недоліків, які стали причиною відхилення можливості використання його в даному випадку. У запропонованому новому підході до вирішення задачі диспетчеризації для GRID-систем з невідчужуваними ресурсами даний фреймворк не може надати такої можливості без додаткового доопрацювання та розробки нових бібліотек. Не зовсім зрозуміло, як можна враховувати потужності окремих

вузлів та потужності завдань при розрахунках в алгоритмах. Також не зовсім зручна процедура запуску завдання на виконання при симуляції постійної зміни кількості вузлів та їхньої потужності. Для цього потрібно кожен раз конфігурувати систему та компілювати програму, що в даному випадку не дуже зручно. Крім того, є потреба в експериментах відкинути все зайве в системі та зосередитись саме на алгоритмі в чистому вигляді.

3.2 Аналіз існуючих засобів моделювання Desktop GRID

Аналіз існуючого програмного забезпечення для моделювання Desktop GRID показав, що наразі існує чимало різних рішень, які створені для симуляції роботи Desktop GRID, а також для дослідження методів планування завдань в них.

Серед існуючих симуляторів було виділено наступні: SimBOINC [117-119], EmBOINC [120-123], SimBA [124], ComBoS [125], Alea [126]. Також було розглянуто ряд публікацій, що частково присвячені огляду та опису програм-симуляторів для Desktop GRID. Зокрема, частина книги [119], в якій розглядаються SimBA, SimBOINC та EmBOINC [127]. Нижче наведено короткий опис кожного з виділених симуляторів.

SimBOINC – симулятор для неоднорідних та змінних Desktop GRID та волонтерських обчислювальних систем, заснованих на структурі SimGRID. Він був створений для вивчення різних алгоритмів планування для клієнтського планувальника процесора та політик вибору роботи BOINC. Однак проєкт був заморожений через зміни в головному проєкті BOINC та самому SimGRID [117].

EmBOINC – інструмент для вивчення політики планування серверів у добровільних обчислювальних проєктах на базі BOINC. EmBOINC використовує гібридний підхід, при якому частина досліджуваної системи (сервер) не імітується, тоді як решта системи (популяція добровільних хостів) імітується. EmBOINC простий в установці та використанні. Вибір

сукупності та параметрів хоста та аналіз результатів можна здійснити за допомогою простих веб-інтерфейсів. Не менш важливо, що EmBOINC самообслуговується. EmBOINC інтегрований безпосередньо в код BOINC і може використовуватися з останньою версією серверного програмного забезпечення BOINC [121].

SimBA – послідовний, дискретний симулятор подій, написаний на Python [128], який моделює поведінку сервера BOINC під час роботи з нестабільною, схильною до помилок і дуже неоднорідними працівниками системою. SimBA генерує робочі одиниці та створює для кожної робочої одиниці ряд екземплярів або копій відповідно до політики реплікації проєкту, розподіляє екземпляри відповідно до запитів працівників та обраної політики планування, моделює нестабільність та неоднорідність працівників за допомогою характеристик працівника, отриманих з файлу трасування, визначає статус повернутих результатів працівника (успішним чи невдалим, тобто помилковим) з використанням характеристики коефіцієнта помилок працівника, визначає дійсність успішно завершених результатів за допомогою кворуму, тобто необхідну кількість результатів за погодженням, встановлену політикою валідації проєкту, і обчислює ефективність змодельованого проєкту з точки зору пропускної здатності [124].

ComBoS – повний симулятор інфраструктури BOINC, що імітує поведінку всіх задіяних компонентів: проєкти, сервери, мережа, резервні обчислення та волонтерські вузли. Основною метою цього симулятора є керівництво розробкою проєктів BOINC. ComBoS був реалізований за допомогою SimGRID, заснованої на моделюванні основи для оцінки кластерних, сіткових та P2P-алгоритмів та евристики. ComBoS може керувати розробкою проєктів BOINC, використовуючи симулятор з різними навантаженнями та платформами, що вимагають великих даних, щоб проаналізувати можливі вузькі місця та обмеження, які представляє така архітектура, як BOINC [125].

Симулятор дозволяє аналізувати широкий діапазон характеристик: пропускну здатність кожного проєкту, кількість завдань, виконаних клієнтами, загальний обсяг наданих кредитів та середнє зайняття серверів BOINC [129].

Alea – незалежний від платформи симулятор планування, який є продовженням популярного набору інструментів GRIDSim. Мета симулятора – запропонувати дослідникам простий спосіб вивчення, модифікації та розширення різних алгоритмів планування. Alea залишається повністю сумісним з оригінальним GRIDSim. Надана функціональність охоплює вимоги типових дослідників, що включають симулятор “готовий до використання”, який включає в себе повні реалізації загальних алгоритмів планування та підтримує загальні цільові функції. Більше того, інтерфейс візуалізації дозволяє швидше налагоджувати та налаштовувати вивчені алгоритми, а також здійснювати прямий експорт результатів моделювання у растрове зображення та файли csv [126].

Враховуючи, що описані вище симулятори, без додаткового доопрацювання та розробки нових бібліотек до них, не можуть враховувати потужності завдань та потужності вузлів таким чином, як це потрібно відповідно до запропонованого підходу, а також великі затрати на таку розробку, то було вирішено, що вони не підходять для продовження цієї роботи. Крім того всі описані симулятори Desktop GRID розроблені для дослідження обчислень, з урахуванням, що система буде створюватися на основі BOINC, а в даному випадку це не так, то це стало ще однією причиною для прийняття рішення про необхідність розробки нового симулятора. Що ж до доопрацювання якогось існуючого симулятора, відомо, що інколи на вивчення та налаштування чужого програмного коду можна витратити набагато більше часу, ніж на написання своєї бібліотеки чи програми. Враховуючи сукупність всіх вище описаних причин було прийнято рішення створити нову програму [130], яка б могла симулювати роботу GRID-системи з невідчужуваними ресурсами, використовуючи різні

алгоритми, а головне, щоб можна було показати особливість нового підходу.

3.3 Симулятор GRID-системи з невідчужуваними ресурсами

Розробку програми було вирішено проводити на платформі Windows засобами Visual Studio 2018 [131] та з використанням мови програмування C#. Для створення обміну даними між компонентами системи було вирішено використовувати Windows Communication Foundation (WCF) [132]. WCF – це платформа для створення сервіс-орієнтованих додатків. Використовуючи WCF, можливо відправляти дані в вигляді асинхронних повідомлень від однієї кінцевої точки до іншої. Кінцева точка служби може бути частиною постійно доступної служби, розміщеної у IIS [133], або може бути службою, розміщеною в додатку. Кінцева точка може бути клієнтом служби, яка запитує дані від кінцевої точки. Повідомлення можуть бути як простими, як один символ або слово, що відправляються як XML [134], так і складними, як потік двійкових даних. Ось кілька прикладів сценаріїв:

– безпечний сервіс для обробки бізнес-транзакцій;

– служба, яка надає поточні дані іншим, наприклад звіт про трафік або іншу службу моніторингу;

– сервіс чату, який дозволяє двом людям спілкуватися або обмінюватися даними в режимі реального часу;

– додаток панелі моніторингу, який опитує одну або кілька служб на предмет даних і представляє їх в логічному поданні;

– подання робочого процесу, реалізованого з використанням Windows Workflow Foundation в якості служби WCF;

– додаток Silverlight [135] для опитування служби про останні потоки даних.

Хоча створення таких програм було можливо до існування WCF, WCF робить розробку кінцевих точок простіше, ніж коли-небудь. Таким чином, WCF розроблений, щоб запропонувати керований підхід до створення веб-

служб і клієнтів веб-служб [136]. Відповідно до документації WCF, для створення даної служби необхідно реалізувати контракт даних.

Контракт даних є офіційною угодою між сервісом і клієнтом, який абстрактно описує дані, якими будуть обмінюватись. Тобто для взаємодії клієнт і служба не повинні використовувати одні і ті ж типи, тільки одні і ті ж контракти даних. Контракт даних точно визначає для кожного параметра або типу значення, що повертається, які дані серіалізуються (перетворюються в XML) для обміну [136].

Отже, з наведеного опису WCF можливо зробити висновок, що даний механізм цілком підходить для вирішення задачі побудови симулятора GRID-системи з невідчужуваними ресурсами.

На рис. 3.1 зображена архітектурна модель симулятора GRID-системи з невідчужуваними ресурсами, яка була розроблена автором перед написанням програмного коду. Дана архітектурна модель призначена не тілки для створення програми-симулятора, також може бути використана для створення програмного забезпечення для Desktop GRID безпосередньо для вирішення практичних задач, що потребують великої кількості обчислювальних ресурсів. Вона описує основні компоненти та зв’язки між ними, що можуть бути реалізованими в Desktop GRID.

На рис. 3.2 зображена схема компонентів, що належать до розробленої програми-симулятора GRID-системи з невідчужуваними ресурсами та основні функції, які вони виконують. Даний програмний комплекс отримав назву SGRIDAR-1 (*Simulator GRID with non-Alienable Resources, version 1.0*). Дана схема – це спрощений варіант архітектурної моделі, що зображена на рис. 3.1. В ній функції користувача, веб-сервера, диспетчера завдань та сервера даних виконує лише одна програма – Server.

SGRIDAR-1 складається з трьох компонентів: Client, Server, Host. Нижче наведено короткий опис кожного з них та їх основні функції, подалі в даному розділі кожна з програм буде розглянута детально.

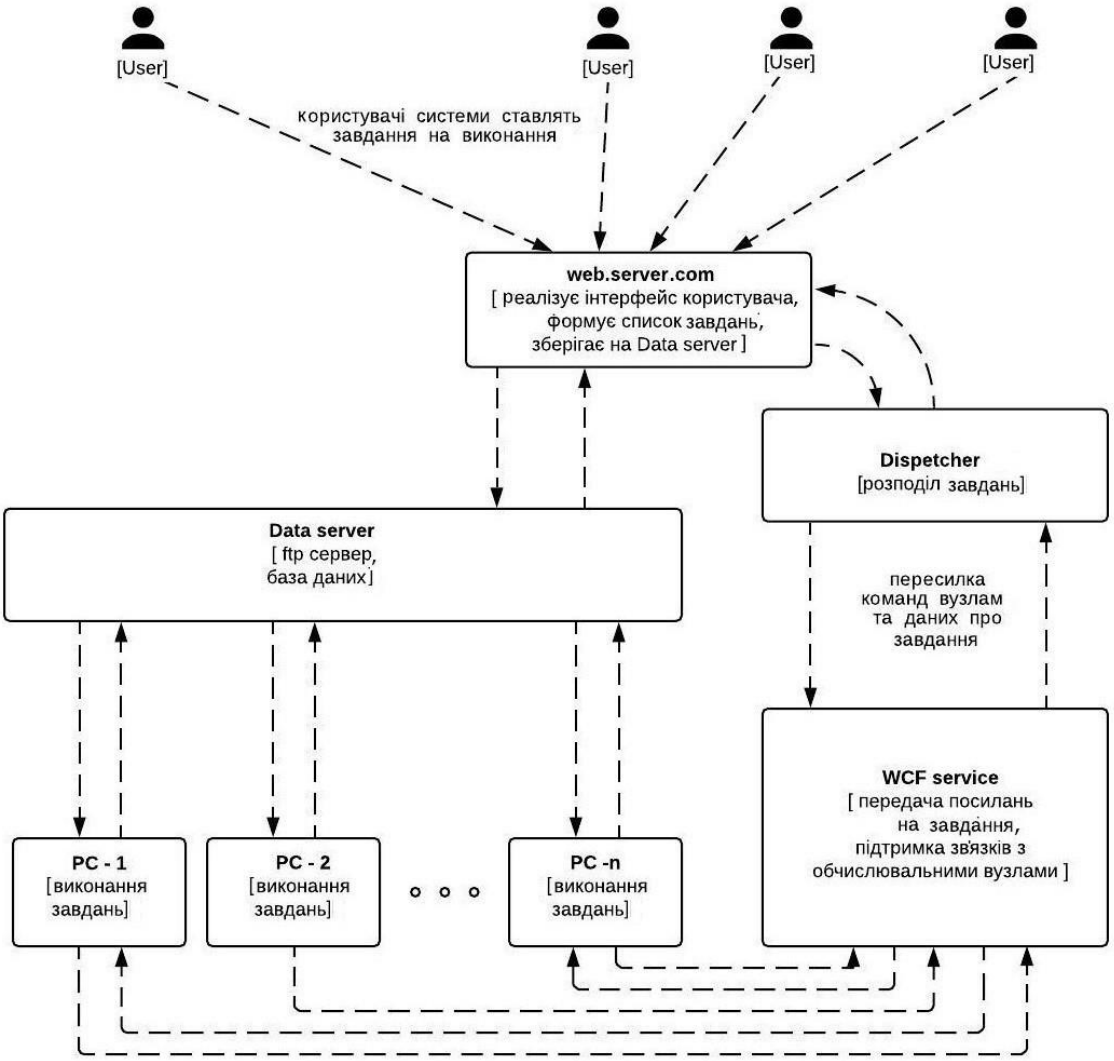


Рисунок 3.1 – Архітектурна модель симулятора GRID-системи з невідчужуваними ресурсами

Рисунок 3.2 – Схема компонентів SGRIDAR-1 та їх основні функції

На рис. 3.3 зображена схема [137] взаємодії компонентів SGRIDAR-1:

програми-клієнта (Client), програми-хоста (Host), програми-сервера (Server).

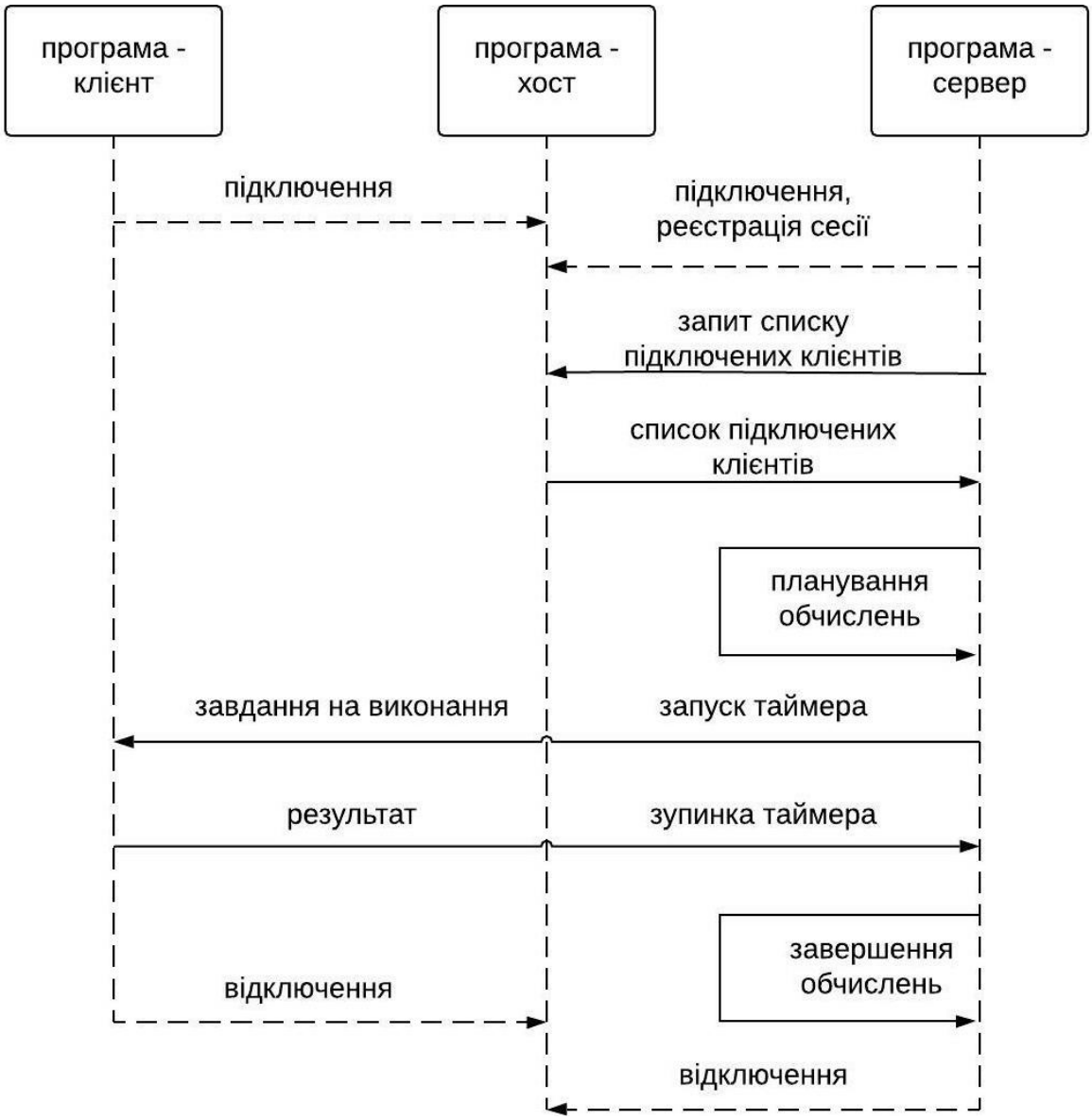


Рисунок 3.3 – Схема взаємодії компонентів SGRIDAR-1

Опишемо дану схему:

– програма-клієнт підключається до програми-хоста та реєструється ним в черзі як обчислювальний вузол;

– програма-сервер підключається до програма-хоста та реєструє початок сесії обчислень;

– далі програма-сервер запитує у програми-хоста список наявних

обчислювальних вузлів, який програма-хост їй надає;

– програма-сервер планує обчислення використовуючи певний метод диспетчеризації завдань;

– програма-сервер надсилає завдання на виконання певним вузлам, пересилка здійснюється через програму-хост, при цьому запускається таймер;

– після отримання завдання програма-клієнт проводить виконання обчислень та відправку їх результатів, після отримання результатів програма-сервер зупиняє таймер та обчислює час виконання завдання;

– після виконання всіх обчислень відбувається процес завершення обчислень, під яким розуміється збереження результатів;

– потім здійснюється відключення програми-сервера та програми-клієнта від програми-хост.

3.3.1 Опис компонента Client

Client – проста Windows програма, створена на основі платформи WPF [138], яка симулює роботу ПК, або вузла в GRID-системі: виконує функції підключення (відключення) до Host, приймає завдання від Server, виконує його, відправляє результат. Додатково реалізована можливість спостерігати за її роботою, тобто користувач може бачити, коли Client працює. В системі може бути задіяно безліч клієнтів, на одному ПК також може бути декілька клієнтів. Кількість задіяних клієнтів на одному ПК обмежується лише характеристиками ПК, здебільшого об’ємом оперативної пам’яті. Наприклад, на ПК з характеристиками: Intel Core i7-7500U (2.7 – 3.5 ГГц), RAM 24 ГБ, SSD 512 МБ, NVidia GeForce 940MX 2 ГБ, вдавалось запустити одночасно більше 100 клієнтів.

На рис. 3.4 зображений візуальний інтерфейс програми Client. Як видно з рисунка, схематично зображено лише три клієнти, хоча в даному випадку їх підключено 20. Даний інтерфейс є простим та зрозумілим. Реалізовано

кнопку підключення (відключення) до сервісу, механізм вибору потужності вузла, інформаційна область для виводу службової інформації та таймер зворотного відліку, що працює, коли вузол проводить обчислення та показує скільки часу залишилось до завершення виконання роботи, що надійшла від Server.

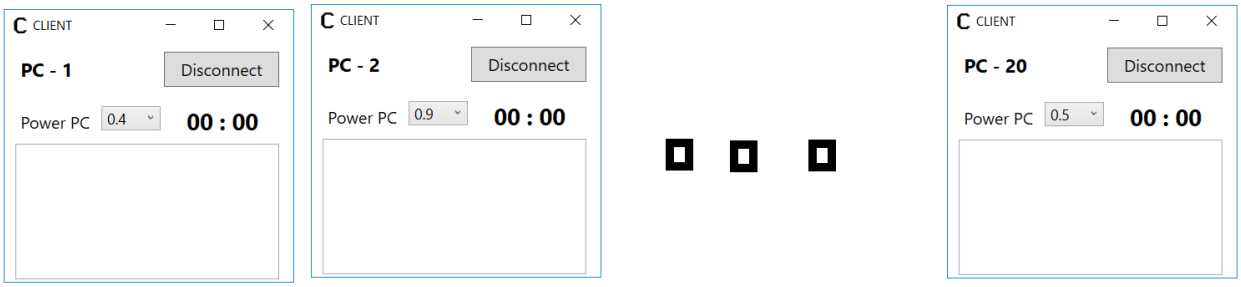


Рисунок 3.4 – Візуальний інтерфейс Client

Програмна реалізація Client. Дана програма реалізована за допомогою Windows Presentation Foundation (WPF) – це платформа для користувача інтерфейсу, на основі якої створюються настільні клієнтські програми. Платформа WPF підтримує широкий набір функцій розробки додатків, включаючи моделі: додатки, ресурси, елементи управління, графіку, макет, прив'язку даних, документи і безпеку. Платформа є частиною .NET [139]. WPF використовує розширювану мову розмітки додатків (XAML) [140], щоб надати декларативну модель для програмування додатків [138].

На рис. 3.5 зображена UML-діаграма класів програми Client. Це досить проста програма, написана для операційної системи Windows. Задача у неї також проста – симулювати роботу обчислювального вузла. А саме, підключитись до сервісу, отримувати завдання, виконувати його, відправляти результат на сервер.

Для її реалізації був створений лише один клас та підключена служба WCF, що буде детально описана в наступному підрозділі.

Опишемо призначення полів та методів, що використовуються в даній програмі, з програмним кодом якої можна ознайомитись в додатку А.

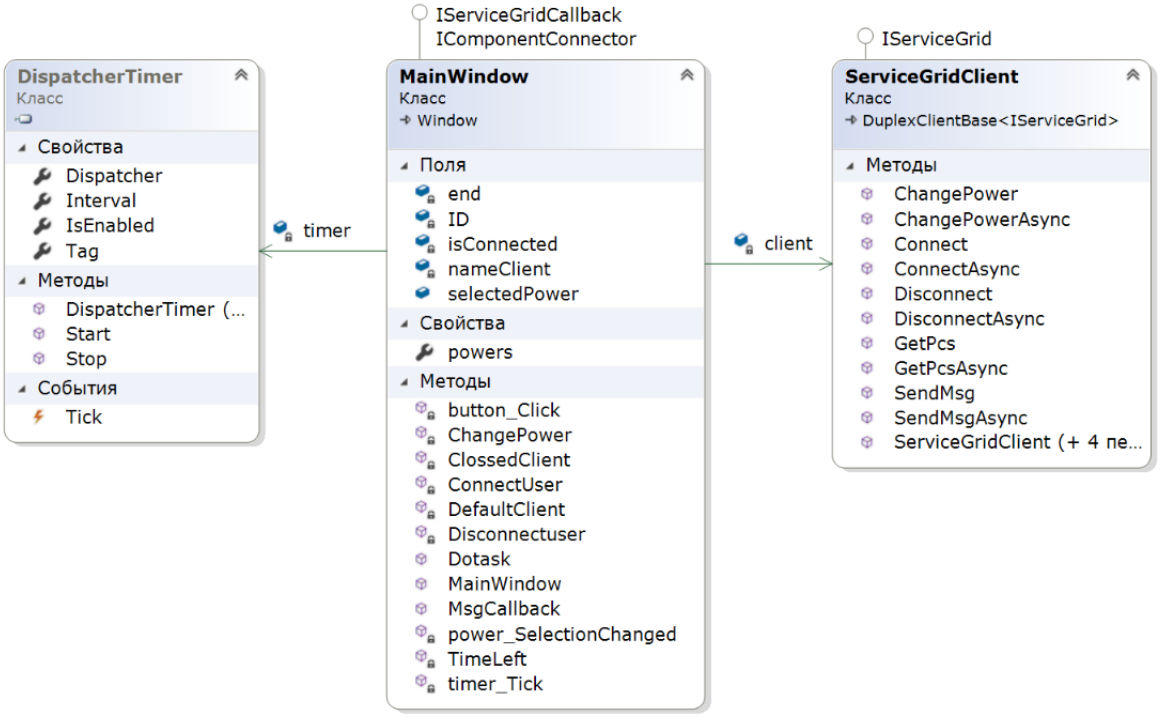


Рисунок 3.5 – UML-діаграма класів програми Client

Поле *timer* типу *DispatcherTimer* визначає об’єм роботи, що потрібно виконати. Так як це симулятор, то використання таймеру зворотного відліку є доцільно, адже на виконання будь-якої роботи потрібно затратити певний час. Клієнту приходить об’єм роботи у вигляді кількості часу, який потрібно відрахувати. Після завершення цього часу, клієнт відправляє повідомлення, що робота виконана.

Поле *end* – допоміжне поле типу *DateTime*, призначене для роботи таймера.

Масив *powers* типу числа з плаваючою крапкою – масив значень потужностей, що може приймати обчислювальний вузол. Визначається набором чисел з плаваючою крапкою в діапазоні від 0,1 до 1 з інтервалом 0,1.

Поле *isConnected* визначає, чи підключений наразі вузол до сервісу.

Поле *client* типу *ServiceGRIDClient* створюється після підключення клієнта до сервісу.

ID – унікальний ідентифікатор, який присвоюється сервісом при підключенні.

Поле *nameClient* – ім'я клієнта, що виводиться в інтерфейс програми.

Поле *selectedPower* – початкове значення потужності обчислювального вузла.

MainWindow() – це метод, що задає початкові дані для програми, формує масив потужностей вузла, щоб користувач міг змінювати.

ConnectUser() – це метод, що перевіряє, чи підключений до сервісу сервер, якщо так, то здійснює підключення клієнта до сервісу.

DefaultClient(string msg) – це метод, що повертає налаштування до початкового стану, зупиняє таймер, якщо він працює.

Disconnectuser() – це метод, що здійснює відключення від сервісу, викликає метод *DefaultClient().*

Метод *button_Click(object sender, RoutedEventArgs e)* оброблює подію натискання кнопки «Connect/Disconnect» і викликає, в залежності від стану вузла, методи *ConnectUser()* або ж *Disconnectuser()*.

Для обробки події зміни потужності вузла реалізовано метод *power_SelectionChanged(object sender, SelectionChangedEventArgs e)*.

ClossedClient(object sender, EventArgs e) – метод, що оброблює подію закриття користувачем вікна програми.

MsgCallback(int id, int code, int timeWork, PC[] pcs) – метод, що реалізовує механізм зворотного зв'язку, відповідно до інтерфейсу *IServiceGRIDCallback.*

ChangePower(int index) – допоміжний метод для методу *power_SelectionChanged().*

Dotask(int timeWork) – метод, що симулює виконання роботи обчислювального вузла, запускаючи таймер зворотного відліку на заданий сервером інтервал часу.

TimeSpan TimeLeft() – допоміжний метод для роботи таймера.

Метод *timer_Tick(object sender, EventArgs e)* – метод, що реалізовує роботу таймера, виводить лічильник у вікно програми через заданий інтервал часу.

Дана програма є простою та ефективною і цілком справляється з тими функціями, що на неї покладено. Таймер зворотного відліку, що реалізований в ній, надає можливість користувачу спостерігати за процесом здійснення обчислень, що значно полегшує процес моніторингу під час тестування, адже користувач може в будь-який момент часу перевірити працює вузол чи ні.

Враховуючи, що користувач має можливість спостерігати за процесом виконання завдань, це робить дану програму ще більш корисною і дозволяє використовувати програмний комплекс в освітньому процесі, для наглядної демонстрації роботи GRID-системи з невідчужуваними ресурсами / Desktop GRID.

3.3.2 Опис компонента Host

Host — проста консольна програма [141], що реалізує WCF контракти на обслуговування для Client та Server, а також запускає програму Host, в якій реалізовано механізм зв’язку між вузлами системи та сервером, слідкує за підключенням (відключенням) вузлів до системи та інформує сервер про зміну стану того чи іншого вузла. Також Host реалізує механізм передачі даних між вузлами та сервером.

В даній програмі описується контракт на обслуговування для WCF сервісу, а також контракт даних для обміну між компонентами програмного комплексу. Також описаний контракт зворотного зв’язку, що повинен бути реалізований в кожному клієнті, що підключається до сервісу. Дана програма реєструє обчислювальні вузли, передає інформацію про підключені вузли на сервер. Візуальний інтерфейс у даної програми (рисунок 3.6) – як у консольної програми Windows. На екран виводиться лише одне повідомлення, про успішний запуск хоста або про помилку, якщо запуск не

відбувся.

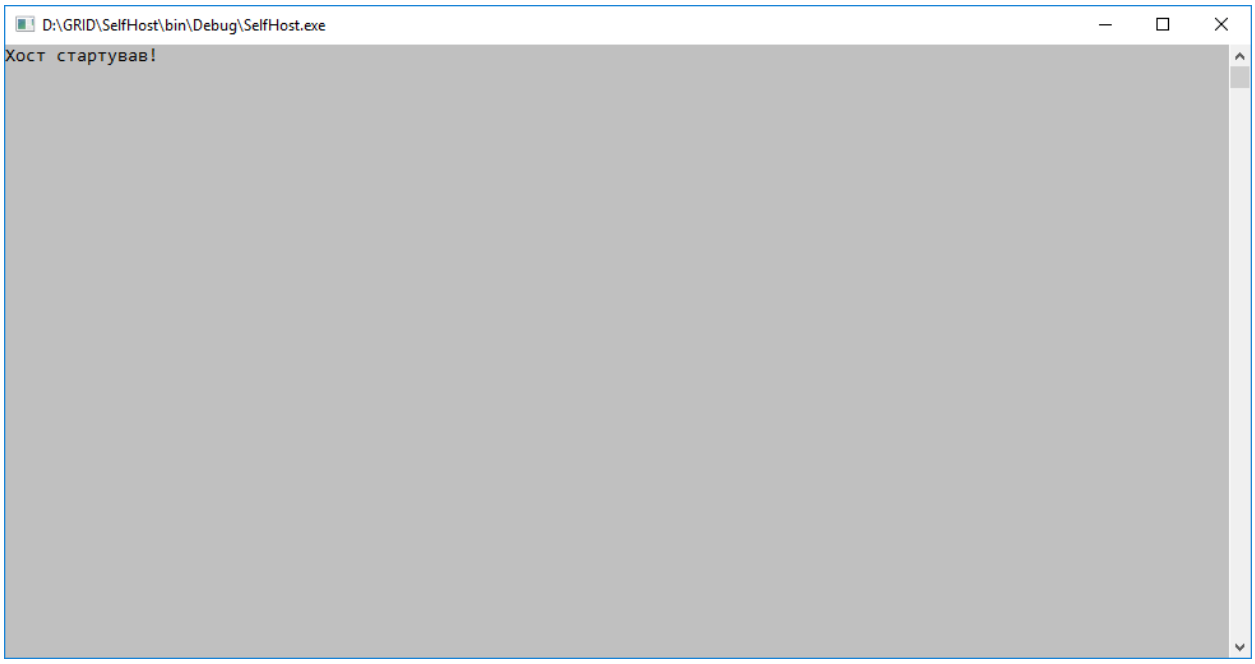


Рисунок 3.6 – Візуальний інтерфейс компонента Host

Host реалізований як служба, що сама себе запускає на локальному хості на одному ПК, але може бути запущена і на розміщеному в мережі Internet сервері. Для цього потрібно лише змінити деякі налаштування в конфігураційному файлі. Тоді стане можливим запускати клієнти на різних ПК, що зробить даний програмний комплекс уже не програмою-симулятором, а повноцінним програмним забезпеченням для Desktop GRID. Дана особливість є значною перевагою перед іншими програмами-симуляторами, що розглядалися в попередніх підрозділах даного розділу.

Програмна реалізація Host. Опишемо основні класи, поля, методи, що використовуються для побудови програми Host. На рисунку 3.7 наведена UML-діаграма класів програми. Як писалось вище, ця програма є простою консольною програмою, написаною мовою програмування C#. Її задача також досить проста – це реалізувати механізм зв’язку компонентів між

собою, що моделюють роботу обчислювального вузла (Client) та головного сервера обчислювальної системи (Server).

Як видно з рис. 3.7, в програмі Host реалізовані два інтерфейси, що описують контракти даних, якими будуть обмінюватись між собою компоненти системи. Нижче наведено призначення інтерфейсів, класів, полів та методів, що використовуються в даній програмі, з кодом яких можна ознайомитись у додатку Б:

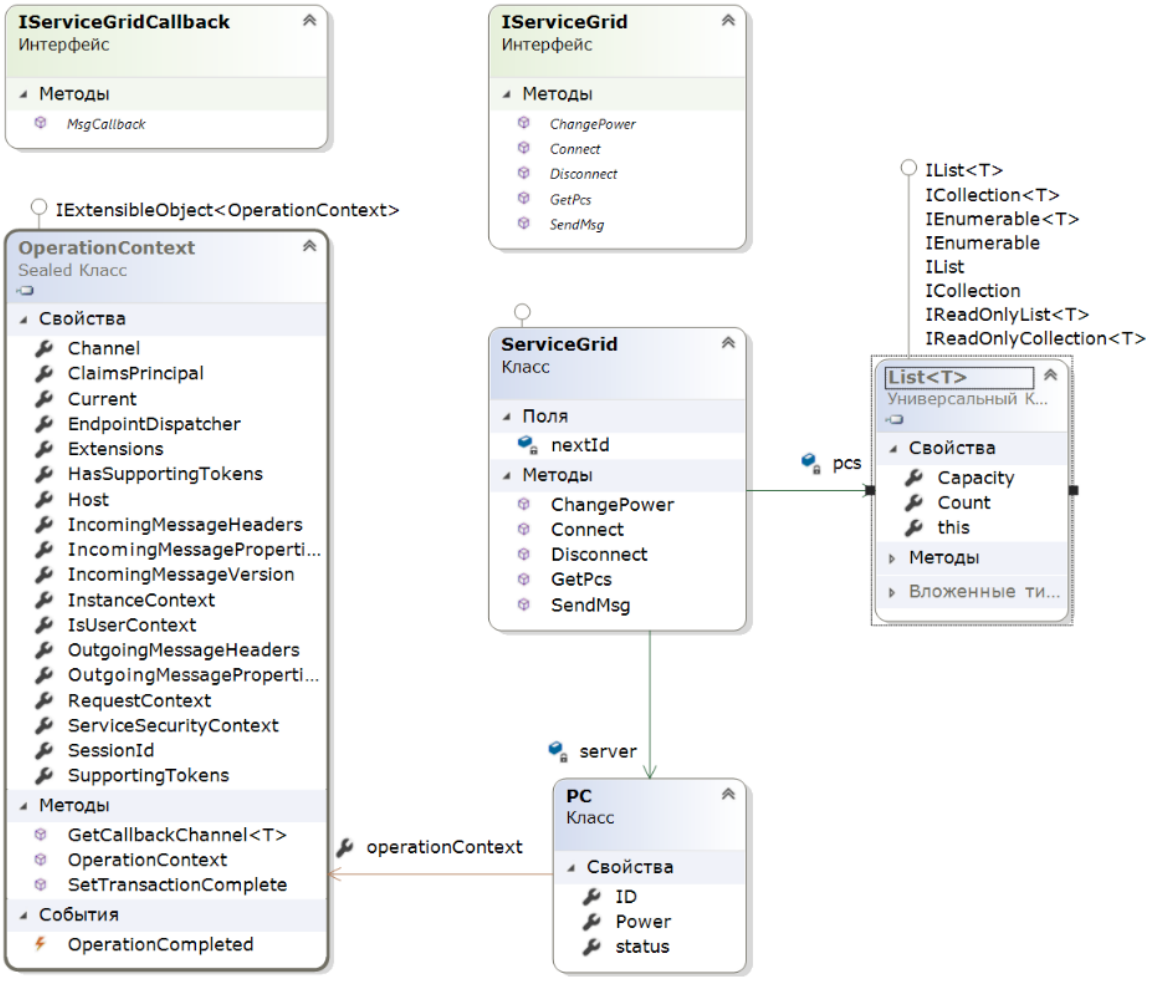


Рисунок 3.7 – UML-діаграма класів програми Host

Інтерфейс IServiceGRID описує методи, які є частиною контракту служби в програмі Windows Communication Foundation (WCF), що будуть

реалізовані та до яких матимуть доступ клієнти та сервер. Нижче наведено короткий опис даних методів.

Connect() – метод, за допомогою якого здійснюється підключення клієнта або сервера до WCF сервісу. Даний метод повертає клієнту унікальний ідентифікатор та створює об'єкт класу *PC,* добавляє його в список обчислювальних вузлів. Для сервера створює об'єкт *server*.

Disconnect() – метод, за допомогою якого здійснюється відключення клієнта або сервера від WCF сервісу. У випадку відключення клієнта, об'єкт, що йому відповідає, видаляється зі списку вузлів, та відправляється повідомлення на сервер про цю подію. У випадку відключення сервера, автоматично відключаються і клієнти.

ChangePower() – метод, що інформує сервер про зміну потужності вузла.

SendMsg (int timeWork, int id, int code) – метод, що реалізовує механізм пересилки повідомлень між клієнтами та сервером. В параметри даному методу передається інформація про об'єм роботи, ідентифікатор вузла, код ситуації. В залежності від даних параметрів, відправляються різні повідомлення різним вузлам.

GetPcs () – метод, що повертає серверу список підключених до WCF сервісу клієнтів.

Інтерфейс IServiceGRID описує односторонній метод зворотного зв'язку, що дозволяє службі передавати дані клієнту, не чекаючи відповіді. Реалізований для підтримки розподіленого клієнта, керованого подіями.

MsgCallback(int id, int code, int timeWork, List<PC> pcs) – метод зворотного зв'язку, що повинен бути реалізований як на стороні клієнта так і на стороні сервера.

Клас PC – це один з основних класів, що реалізовано в даній програмі. Його задача описати властивості обчислювального вузла та визначити контракт даних для служби WCF. Для цього використовуються три члени контракту даних, що будуть передаватись та одна властивість для зворотного

зв’язку.

ID – унікальне ціле число, по якому проводиться ідентифікація вузла.

Power – потужність вузла, яка є відносною величиною та визначається числом з плаваючою крапкою. Дана характеристика є узагальнюючою та включає в себе всі можливі властивості обчислювального вузла. В симуляторі вона задається довільним чином при створенні об’єкта.

Властивість *status* типу *bool*, яка може мати два значення: «так» чи «ні» і визначає чи зайнятий роботою обчислювальний вузол.

Властивість *operationContext* типу *OperationContext*, задача якої зберігати інформацію про канал зворотного зв’язку з вузлом системи.

Клас ServiceGRID – клас, що реалізує описаний вище інтерфейс *IServiceGRID,* визначає поведінку служби. При цьому тільки один об'єкт *InstanceContext* використовується для всіх вхідних викликів і не використовується повторно після викликів. Якщо об'єкт служби не існує, він створюється [142].

Клас описує такі поля та методи.

Поле *pcs* – список всіх обчислювальних вузлів, кожен з яких є об’єктом типу *PC*.

Поле *server* – поле типу *PC*, описує сервер.

Поле *nextId* – ціле число, що визначає унікальний ідентифікатор при підключенні обчислювального вузла, при запуску програми приймає значення 1, та з кожним новим підключенням збільшується на 1. Таким чином ідентифікатор буде завжди унікальним, що дозволяє ефективно оперувати інформацією.

Connect() – метод, за допомогою якого здійснюється підключення клієнта або сервера до WCF сервісу, повертає унікальний ідентифікатор для клієнта, створює об’єкт класу *PC* та добавляє його в список обчислювальних вузлів. Для сервера створює об’єкт *server* типу *PC*.

Disconnect() – метод, за допомогою якого здійснюється відключення клієнта або сервера від WCF сервісу. У випадку відключення клієнта, об’єкт,

що йому відповідає, видаляється зі списку вузлів та відправляється повідомлення на сервер. У випадку відключення сервера, автоматично відключаються і клієнти.

ChangePower() – метод, що інформує сервер про зміну потужності вузла.

SendMsg() – метод, що реалізовує механізм пересилки повідомленнями між клієнтами та сервером. В параметри даному методу передається інформація про об’єм роботи, ідентифікатор вузла, код ситуації. В залежності від даних параметрів, відправляються різні повідомлення різним вузлам.

GetPcs() – метод, що повертає серверу список підключених до WCF сервісу клієнтів.

Частина програмного коду *App.config:*

```
<host>
    <baseAddresses>
        <add baseAddress="http://localhost:8301" />
        <add baseAddress="net.tcp://localhost:8302" />
    </baseAddresses>
</host>
```

В пакеті програми є конфігураційний файл *App.config*, частина програмного коду якого наведена вище, що дозволяє змінювати налаштування для хоста, що буде запущений. Наприклад, IP адресу в мережі та номер порту, по якому буде здійснюватися підключення клієнтів. Можливі і інші налаштування, з повним описом яких можна ознайомитись за посиланням [143].

3.3.3 Опис компонента Server

Server – Windows програма, як і Client, створена на основі платформи WPF, що симулює роботу сервера. Це є головна програма і в системі може бути запущений тільки один сервер.

На рис. 3.8 зображений візуальний інтерфейс компонентів програми Server. Даний інтерфейс є простим, інтуїтивно зрозумілим та досить інформативним, що надає можливість користувачу задати різні налаштування при проведенні тестування методів планування.

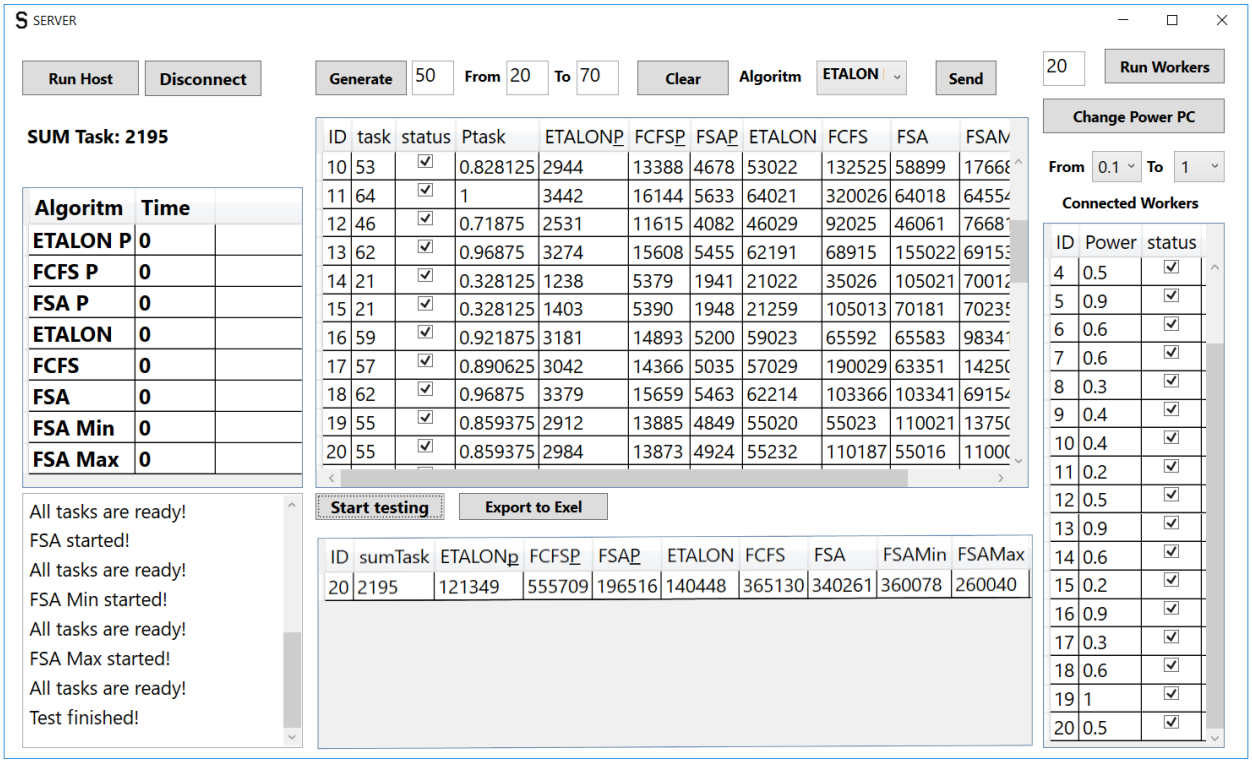


Рисунок 3.8 – Візуальний інтерфейс Server

Програма Server виконує наступні функції:

– запускає Host, до якого буде здійснюватися підключення (відключення) клієнтів, та через який буде передаватись інформація;

– здійснює підключення (відключення) до Host, у випадку відключення всі клієнти також автоматично відключаються;

– генерує завдання на виконання, при цьому можна задавати кількість завдань, а також їх мінімальний та максимальний об’єм;

– відображає список завдань, згенерованих системою автоматично, а також їх статус на кожному етапі обчислень та час виконання відповідно до кожного алгоритму;

– веде обрахунок часу виконання кожного завдання та обраховує загальний час виконання всіх завдань при розподілі відповідно до кожного з алгоритмів;

– відображає список підключених до Host клієнтів, яких можна використовувати для обчислень, а також статус цих вузлів (зайнятий або вільний);

– для зручності роботи, реалізовано можливість запуску Client, попередньо задавши кількість, яку потрібно запустити;

– довільно змінює потужність вузлів, реалізуючи тим самим властивість невідчужуваності та різнорідності GRID-системи, при цьому можна задавати мінімальну та максимальну потужності вузлів;

– реалізує механізм вибору алгоритму, за яким будуть проводитись індивідуальні обчислення;

– реалізує механізм тестування, при якому всі згенеровані завдання обчислюються на вузлах кожним з алгоритмів, що пропонуються системою, після чого результати можна зберегти в Excel файлі для подальшого аналізу та досліджень;

– надає можливість користувачу системи спостерігати за змінами, що відбуваються в інтерактивному режимі.

На рис. 3.9 виділено дев'ять областей у вікні програми Server, у яких, для проведення тестування, задаються різні налаштування та відображається інформація про його результати.

Нижче наведено опис цих областей, відповідно до номеру на рис. 3.9:

1) в даній області відображається загальний час (в секундах), що потрібно витратити на виконання всіх завдань, що згенерувала система, або об’єм всієї роботи, яку необхідно виконати;

2) в дану область виводиться інформація про результати тестування, а саме: загальний час виконання всіх завдань для кожного алгоритму;

3) область, в якій відображаються повідомлення про початок і кінець розрахунків, а також про те, який алгоритм працює на даний момент часу;

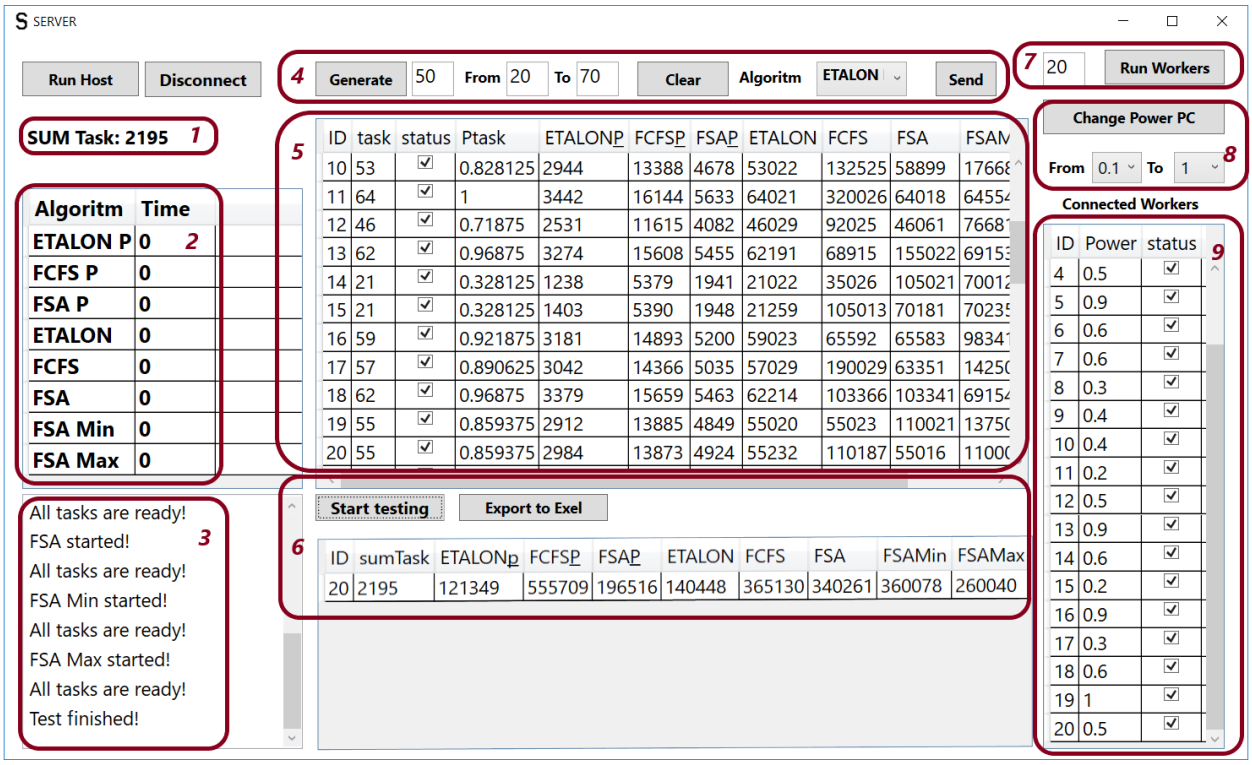


Рисунок 3.9 – Інформативні області програми Server

4) у цій області визначаються умови для генерації завдань, що будуть виконуватися: кількість завдань, їх мінімальне і максимальне значення, вибір алгоритму, за яким буде проводитись обчислення;

5) в дану область виводиться інформація про перелік усіх згенерованих завдань, їх обсяг і потужність, стан (виконується чи ні), час виконання відповідно до алгоритму;

6) після завершення тесту ця область отримує зведену інформацію про час роботи кожного алгоритму, після чого в області 2 інформація оновлюється для виконання нового тесту, а кнопка "Export to Excel" стає активною;

7) в даній області задається кількість програм Client, що потрібно запустити для проведення обчислень;

8) в даній області задається максимальне та мінімальне значення потужності обчислювальних вузлів, після натискання кнопки "Change PowerPC" система автоматично змінюватиме потужність на всіх вузлах у

межах від мінімального до максимального значень, використовуючи механізм випадкового вибору;

9) інформація про вузли, підключені до системи, їх потужність та стан (вільні чи зайняті).

Програмна реалізація Server. Програма Server реалізована, аналогічно програмі Client, за допомогою WPF. Це головна в системі програма, написана для операційної системи Windows. Вона повинна бути запущена лише одна. На рис. 3.10 зображена UML-діаграма класів програми Server.

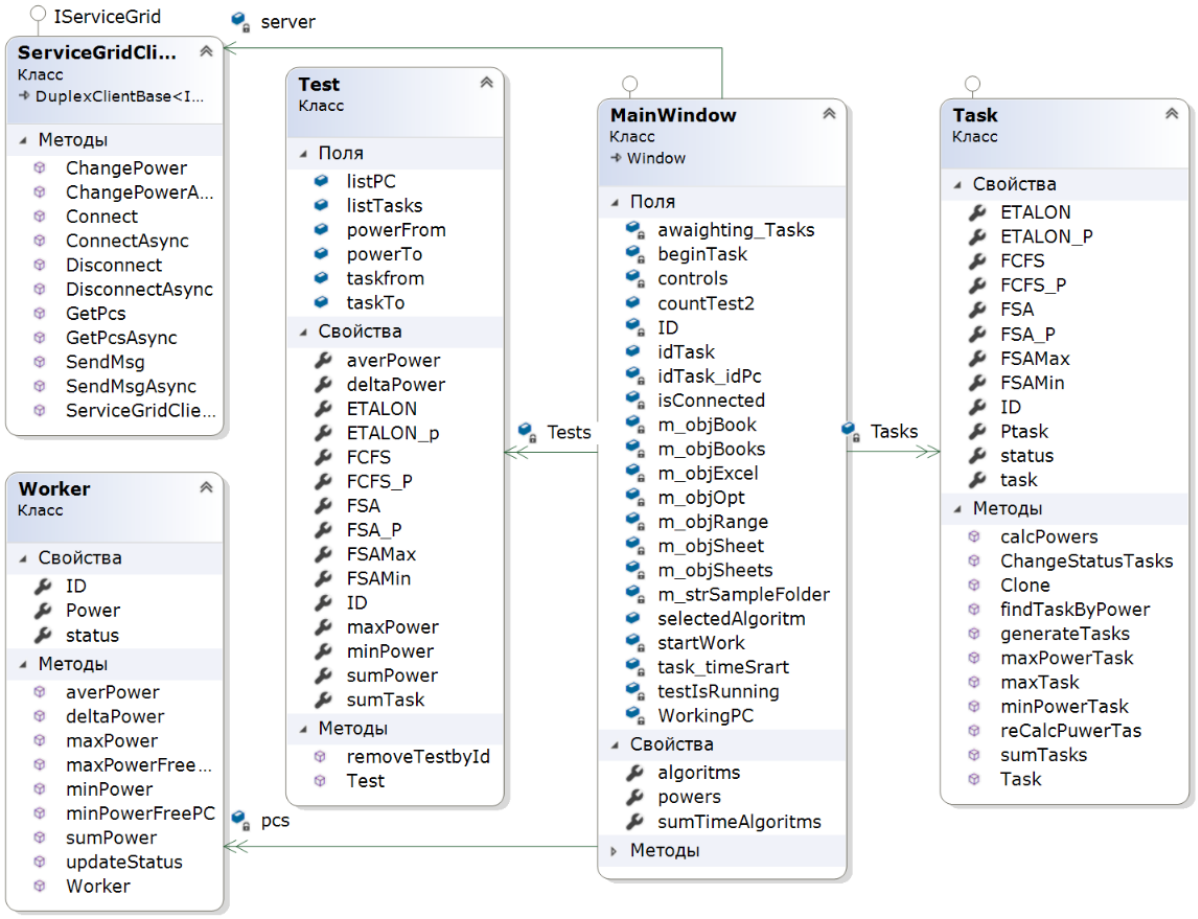


Рисунок 3.10 – UML-діаграма класів програми Server

Як видно з рис. 3.10, для реалізації даної програми було написано декілька класів, що описують вузол, задачі, тести. Нижче наведено опис

призначення класів, полів та методів, що використовуються в даній програмі. Частина програмного коду самої програми наведено в додатку В.

Клас Worker описує сутність обчислювального вузла. Це простий клас, що має схожий інтерфейс на клас *PC* у програмі Host. Нижче описані властивості та методи, що він має.

ID – унікальне ціле число, по якому проводиться ідентифікація вузла.

Power – потужність вузла, яка є відносною величиною та визначається числом з плаваючою крапкою.

status – властивість типу *bool*, яка визначає чи зайнятий роботою обчислювальний вузол чи ні.

Worker() – метод (конструктор), що визивається при надходженні інформації до сервера про підключені вузли до сервісу WCF, на основі інформації, що прийшла, створює об’єкти даного класу.

Статичний метод *minPower()* шукає у списку обчислювальних вузлів вузол з найменшою потужністю та повертає її величину.

Статичний метод *minPowerFreePC()* шукає у списку обчислювальних вузлів, які наразі не зайняті роботою, вузол з найменшою потужністю та повертає її величину.

Статичний метод *maxPower()* шукає у списку обчислювальних вузлів вузол з найбільшою потужністю та повертає її величину.

Статичний метод *maxPowerFreePC()* шукає у списку обчислювальних вузлів, які наразі не зайняті роботою, вузол з найбільшою потужністю та повертає її величину.

Статичний метод *deltaPower()* шукає у списку обчислювальних вузлів вузли з найбільшою та найменшою потужностями та повертає їх різницю.

Статичний метод *sumPower()* обчислює у отриманому списку обчислювальних вузлів сумарну потужність системи.

Статичний метод *averPower()* обчислює у отриманому списку обчислювальних вузлів середню потужність обчислювального вузла.

Метод *updateStatus()* змінює статус вузла, в залежності від отриманого параметра, на вільний чи зайнятий.

Клас Task описує сутність обчислювальної задачі. Нижче наведено основні властивості та методи.

ID – поле типу «ціле число», унікальний ідентифікатор завдання.

Поле *task* типу «ціле число», означає об’єм завдання, що задається при генерації сервером завдань на виконання.

Поле *status* приймає значення «так» чи «ні», в залежності від того чи виконане завдання.

Ptask – поле типу «число з плаваючою крапкою», що означає потужність завдання, його величина є відносною, в залежності від об’єму завдання в згенерованому сервером списку.

ETALON_P, FCFS_P, FSA_P, ETALON, FCFS, FSA, FSAMin, FSAMax – властивості типу «ціле число», що відповідають часу виконання завдання, в залежності від методу планування. Дані значення при створенні заповнюються нулями, а після тестування заповнюються значеннями часу, що обчислив таймер і зберігаються в мілісекундах.

Task() – метод-конструктор, що призначений для створення об’єктів завдання під час генерації сервером нових завдань.

Clone() – метод, що призначений для повного копіювання об’єктів даного класу.

Статичний метод *minPowerTask()* призначений для пошуку в отриманому списку завдань завдання з найменшою потужністю, повертає знайдене значення.

Статичний метод *maxPowerTask()* призначений для пошуку в отриманому списку завдань завдання з найбільшою потужністю, повертає знайдене значення.

Статичний метод *findTaskByPower()* призначений для пошуку в отриманому списку завдань завдання з найближчою по величині потужністю до отриманої потужності вузла, повертає знайдене значення. Даний метод

потрібен для реалізації алгоритмів диспетчеризації завдань FSA, FSA_Min, FSA_Max.

ChangeStatusTasks() – статичний метод, що призначений для зміни статусів всіх завдань в отриманому списку завдань на «не виконані». Визивається даний метод при запуску тестування, для реалізації можливості спостерігати за ходом виконання роботи.

Статичний метод *calcPowers()* призначений для обчислення потужностей всіх завдань в отриманому списку.

Статичний метод *reCalcPuwerTas()* призначений для перерахунку потужностей всіх завдань в отриманому списку, в залежності від величини об’єму отриманого завдання.

Статичний метод *maxTask()* призначений для пошуку в отриманому списку завдань завдання з найбільшим об’ємом, повертає знайдене значення.

Статичний метод *sumTasks()* призначений для обчислення суми об’ємів всіх завдань в отриманому списку завдань, повертає знайдене значення.

Статичний метод *generateTasks()* призначений для генерації заданої кількості завдань, відповідно до отриманих мінімального та максимального значень об’ємів та добавляє згенеровані завдання в отриманий список завдань.

Клас Test описує сутність тесту. Створений для реалізації механізму зберігання всієї інформації про результати тестування, що потрібна для подальшого аналізу та дослідження ефективності розроблених методів, а також виявлення закономірностей та ситуацій при яких той чи інший метод дає кращі результати, в порівнянні з іншими. Нижче описані поля, властивості та методи.

ID – унікальне ціле число, по якому проводиться ідентифікація номеру тесту.

Поле sumTask – ціле число, що дорівнює сумі всіх завдань, що потрібно виконати.

Поле *listTasks* – список всіх завдань на виконання, що згенерувалися системою.

Поле *listPC* – список всіх обчислювальних вузлів, що підключилися до сервісу.

Поле *powerFrom* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про мінімальну потужність вузла.

Поле *powerTo* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про максимальну потужність вузла.

Поле *taskfrom* – змінна типу «ціле число», в якій зберігається інформація про мінімальну величину згенерованих завдань.

Поле *taskTo* – змінна типу «ціле число», в якій зберігається інформація про максимальну величину згенерованих завдань.

ETALON_P, FCFS_P, FSA_P, ETALON, FCFS, FSA, FSAMin, FSAMax – змінні типу «ціле число», в яких зберігається інформація про загальний час виконання всіх завдань, що згенерувала система, на обчислювальних вузлах, відповідно до кожного з методів планування.

Поле *sumPower* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про сумарну потужність всіх обчислювальних вузлів.

Поле *averPower* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про середню потужність всіх обчислювальних вузлів.

Поле *minPower* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про мінімальну величину потужності обчислювальних вузлів, що виконували завдання.

Поле *maxPower* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про максимальну величину потужності обчислювальних вузлів, що виконували завдання.

Поле *deltaPower* – змінна типу «число з плаваючою крапкою», в якій зберігається інформація про різницю між максимальною та мінімальною величинами потужностей обчислювальних вузлів, що виконували завдання.

Test() – метод-конструктор, що призначений для створення об’єктів тесту при запуску нового тестування.

Статичний метод *removeTestbyId()* призначений для видалення тесту з отриманого списку тестів, відповідно до заданого ідентифікатору тесту.

Клас MainWindow – це головний клас в даній програмі, всі основні змінні, методи, події описані в ньому. Він є зв’язуючим та керуючим елементом в програмі. Крім того він є наслідуючим класом для інтерфейсу *IServiceGRIDCallback*, і в ньому повинен бути реалізований метод зворотного зв’язку для нашої служби.

Опишемо основні поля, властивості та методи даного класу.

Об’єкт *server* типу *ServiceGRIDClient*, що створюється при підключенні сервера до сервісу, потрібен для реалізації механізму обміну повідомленнями з вузлами за допомогою WCF служби.

Поле *isConnected* – змінна типу «істина чи ні», яка зберігає інформацію, чи підключений сервер до сервісу чи ні.

ID – унікальне ціле число, по якому проводиться ідентифікація сервера на стороні сервісу, при здійснені підключення сервіс видає дане значення.

Масив *powers* – масив значень «чисел з плаваючою крапкою», що відповідають можливим потужностями вузлів, що можуть виконувати завдання. Дані значення використовуються для симуляції різнорідних обчислювальних вузлів у GRID-системі з невідчужуваними ресурсами.

Список *pcs* зберігає інформацію про всі обчислювальні вузли. Це є об’єкти типу *Worker*.

WorkingPC – поле типу «ціле число», що містить інформацію про кількість працюючих обчислювальних вузлів на конкретний момент часу. Потрібне для реалізації механізму тестування та дослідження різних методів планування обчислень.

Test – список всіх об’єктів типу *Test*. Даний список заповнюється при створенні тесту та потрібен для реалізації механізму збереження всієї інформації про результати проведених тестів.

Поле *testIsRunning* типу «істина чи ні» визначає на конкретний момент часу чи запущений режим тестування. Потрібне для унеможливлювання одночасного запуску декількох тестів.

Поле *countTest2* типу «ціле число» визначає кількість тестів, що потрібно провести з одним і тим самим набором даних, для отримання більш точного результату.

Tasks – список всіх об'єктів типу *Task*. Даний список заповнюється при генерації завдань.

Поле *idTask* типу «ціле число», що містить інформацію про унікальний ідентифікатор виконуваного завдання. Це поле виконує допоміжну функцію для реалізації різних механізмів, що працюють в програмі.

DateTime beginTask, startWork – поля типу *DateTime*, що виконують допоміжні функції при обрахунку часу виконання конкретного завдання та загального часу роботи методу диспетчеризації завдань.

Властивість *algoritms* типу «масив рядків», в якому зберігається інформація про закладені в систему методи планування. Потрібна для реалізації механізму вибору користувачем методу диспетчеризації за яким він хоче виконати всі згенеровані завдання. В систему заведено наступні алгоритми: ETALON_P, FCFS_P, FSA_P, ETALON, FSA, FSA_Min, FSA_Max.

Поле *selectedAlgoritm* типу «ціле число», що містить інформацію про вибраний користувачем з списку *algoritms* алгоритм диспетчеризації.

Список *awaighting_Tasks* – список всіх об'єктів типу *Test*, що знаходяться в стані очікування виконання. Даний список заповнюється на початку тесту методом копіювання всіх згенерованих системою завдань та по мірі відправки на виконання з нього видаляються завдання, що були відправлені. Це допоміжний список, що потрібний для реалізації можливості проведення циклічного тестування.

Поле *controls* типу «масив об'єктив *Control*», в якому зберігаються посилання на всі елементи управління, що реалізовані в інтерфейсі програми.

Даний список необхідний для реалізації відключення активності даних елементів під час проведення тестування.

Враховуючи, що в мові програмування C#, не має можливості реалізації масивів з доступом до елементів по ключу, і для цих цілей використовують словники, а при написанні програми така необхідність виникла, тому було використано даний механізм та реалізовано ряд словників [144].

Поле *idTask_idPc* типу «*Dictionary* <ціле число, ціле число>», в якому зберігається інформація: яке завдання на якому вузлі виконувалось.

Поле *task_timeSrart* типу «*Dictionary* <ціле число, *DateTime*>», в якому зберігається інформація: скільки часу було затрачено на виконання того чи іншого завдання.

Поле *sumTimeAlgoritms* типу «*Dictionary* <рядок, ціле число>», в якому зберігається інформація: скільки часу було затрачено на виконання списку всіх завдань в залежності від методу диспетчеризації. Дана інформація виводиться в інтерфейс програми, а також зберігається в об’єкт *Test*.

Поля *m_objExcel, m_objBooks m_objBook, m_objSheets, m_objSheet, m_objRange, m_objOpt, m_strSampleFolder* – поля, що створені для реалізації механізму збереження результатів тестування в окремий файл Excel і визначають: програму Excel, робочу книгу, робочі листи, робочий лист, налаштування, ранг, шлях до папки де знаходиться файл.

Для реалізації всього функціоналу класу було реалізовано 45 методів. Описувати всі не доцільно, тому було вирішено описати лише найбільш важливі механізми програми та методи, що їх реалізовують.

Як писалось вище, в програмі реалізований механізм генерації та керування завданнями. Описаний даний механізм за допомогою методів *Generate_Tasks(), Clear_Tasks(), Send_Tasks(), StartWork()*. Користувач задає в інтерфейсі програми кількість завдань, що потрібно згенерувати, максимальний та мінімальний об’єм завдань в секундах та натискає кнопку «Generate», після чого спрацьовує метод *Generate_Tasks()*.

Механізм моніторингу або спостереження за процесом роботи реалізовується за рахунок методів *UpdateGRIDPC(), UpdateSumAverPower(), UpdateGRIDTask(), UpdateSumTime()*, задача яких оновлювати інформацію в інтерфейсі програми в разі, якщо відбулися зміни в стані системи. Наприклад, підключився чи відключився обчислювальний вузол, або ж змінив свою потужність, що вплинуло на зміну сумарної та середньої потужності, оновився список завдань, або ж змінився статус завдання на виконане та інші випадки.

Механізм підключення-відключення до сервісу реалізований за допомогою методів *ConnectUser(), ConnectDisconnect_Click(), Disconnectuser()*. Для підключення-відключення до сервісу користувач має натиснути кнопку «Connect/Disconnect», після чого спрацює метод *ConnectDisconnect_Click()*, який, в залежності від стану, здійснить підключення-відключення за допомогою методів *ConnectUser(), Disconnectuser()*.

Механізм диспетчеризації завдань між вузлами системи реалізований за допомогою методів *BeginWork(), ProceedWork(), EndWork()* (для паралельних методів ETALON_P, FCFS_P, FSA_P), *BeginWorkFCFS(), ProceedWorkFCFS(), EndWorkFCFS()* (для послідовних методів ETALON та FCFS), *BeginWorkFSA(), BeginWorkFSAMinMax(), ProceedWorkFSA(), EndWorkFSA()* (для послідовних методів FSA, FSA Min, FSA Max). Вибір методів залежить від того, який метод вибере користувач у списку вибору методів для тестування «Algorithm» та натисне кнопку «Send». Після чого визветься метод *StartWork()* та почнеться тестування. Це алгоритм роботи при індивідуальному тестуванні, при циклічному ж вибір методу змінюється програмно та використовуються всі методи по черзі.

Для реалізації функції обліку часу виконання завдань використовуються методи *UpdateSumTime(), GetTimeWork()* та поля *beginTask, startWork*. Як писалось вище, в системі фіксується час виконання кожного з завдань, в залежності від методу диспетчеризації, а також

загальний час виконання всіх згенерованих системою завдань по кожному з методів диспетчеризації окремо.

Для роботи механізму тестування відповідно до вибраного методу або ж циклічного тестування реалізовано методи *ActiveDeactivControls(), TaskIsready(), BeginTest(), ProccedTest(), EndTest(), testCount(), StartTest()*. Після задання всіх необхідних налаштувань, користувач має натиснути кнопку «Start testing», що визве метод *BeginTest()* та деактивує всі керуючі елементи за допомогою метода *ActiveDeactivControls()* і процес тестування почнеться. Коли всі завдання будуть виконані на системі, буде визвано метод *EndTest()*, який створить об'єкт *Test* та збереже в нього всю інформацію про завдання, вузли, результати та добавить створений об'єкт в список. При цьому для створення об'єкту буде визвано метод *GetTestID(),* що зчитає з файлу Excel номер попереднього тесту, після чого збільшить його на один та результат присвоїть, як унікальний ідентифікатор для тесту. Після чого перевірить кількість тестів, які потрібно провести, якщо проведено всі тести, то завершить роботу і деактивує керуючі елементи, а зведену інформацію про час роботи виведе у інформаційне вікно.

В системі реалізовано механізм зміни потужностей вузлів, що підключені до сервісу, використовуючи механізм випадкового вибору. Після натискання користувачем кнопки «Change Power PC» визивається метод ChangePower(), який випадковим чином, в залежності від заданих мінімальної та максимальної потужностей вузлів, змінить потужності всіх підключених до сервісу вузлів.

Для зручності обробки результатів роботи методів було реалізовано механізм збереження інформації про результати тестів, який описано за допомогою методів *Export_to_Exel(), Export_to_Exel_2()*. Для цього користувач має натиснути кнопку «Export to Exel», в результаті чого будуть викликані обидва методи, які і збережуть всі дані в 2 різні файли. В один файл буде збережена вся інформація про тестування, а в другий зведена.

В програмному комплексі було реалізовано ряд додаткових функцій, які значно полегшують роботу з програмою.

Run_PC() – після вводу інформації в поле *Count_Workers* та натискання кнопки «Run Workers» відбудеться запуск вказаної кількості клієнтів, які, в свою чергу, автоматично підключяться до сервісу та з’являться в списку обчислювальних вузлів на сервері, щоб не запускати кожен окремо.

Run_Host() – метод, що запускає програму Host після натискання користувачем кнопки «Run Host».

Remove_Test_by_ID() – метод, призначений для видалення тесту зі списку тестів, спрацьовує після вводу в поле ідентифікатору тесту, що потрібно видалити, та натисканню кнопки «Remove».

MsgCallback() – метод, що реалізує інтерфейс *IServiceGRIDCallback* та забезпечує зворотній зв'язок з сервісом. Даний метод визивається програмою Host коли зі сторони клієнта приходить повідомлення для сервера і передає серверу це повідомлення, а сервер вже, в залежності від отриманих параметрів, виконує різні дії. У повідомленні може прийти оновлений список вузлів, який оновлюється на стороні сервера за допомогою метода *UpdatteWorwers()*, чи про завершення виконання завдання тим чи іншим вузлом, тоді, в залежності від вибраного на даний момент часу методу диспетчеризації, визивається один з методів: *EndWork(), EndWorkFCFS(), EndWorkFSA()*.

GetKeyByValueFromDictionary() – метод, що реалізовує пошук в словнику ідентифікатора завдання, в залежності від отриманого параметру ідентифікатора вузла, та повертає знайдене значення. Даний метод є допоміжним для реалізації механізму диспетчеризації.

NumberValidationTextBox() – метод, що здійснює перевірку значень, що вводить користувач, з метою упередження програмних збоїв.

Метод *algoritm_change()* визивається коли користувач в вікні програми змінює вибір алгоритму за яким буде виконуватись розподіл завдань по обчислювальним вузлам.

3.3.4 Механізм тестування методів планування на SGRIDAR-1

Як написано вище, в SGRIDAR-1 реалізовано механізм тестування. Нижче опишемо покроково дії, які потрібно виконати для отримання результатів:

1) запустити програму Server з правами адміністратора (права адміністратора потрібні, щоб запустився Host);

2) у відкритому вікні (рисунок 3.9) натиснути кнопку «Run Host», в результаті чого буде здійснено запуск служби WCF та Host;

3) здійснити підключення до Host, натиснувши кнопку «Connect»;

4) в області 1 рис. 3.9 необхідно задати кількість завдань, які будуть згенеровані програмою, а також їх мінімальний та максимальний об’єм (задається в секундах) та натиснути кнопку «Generate», також є можливість очистити завдання натисканням кнопки «Clear»;

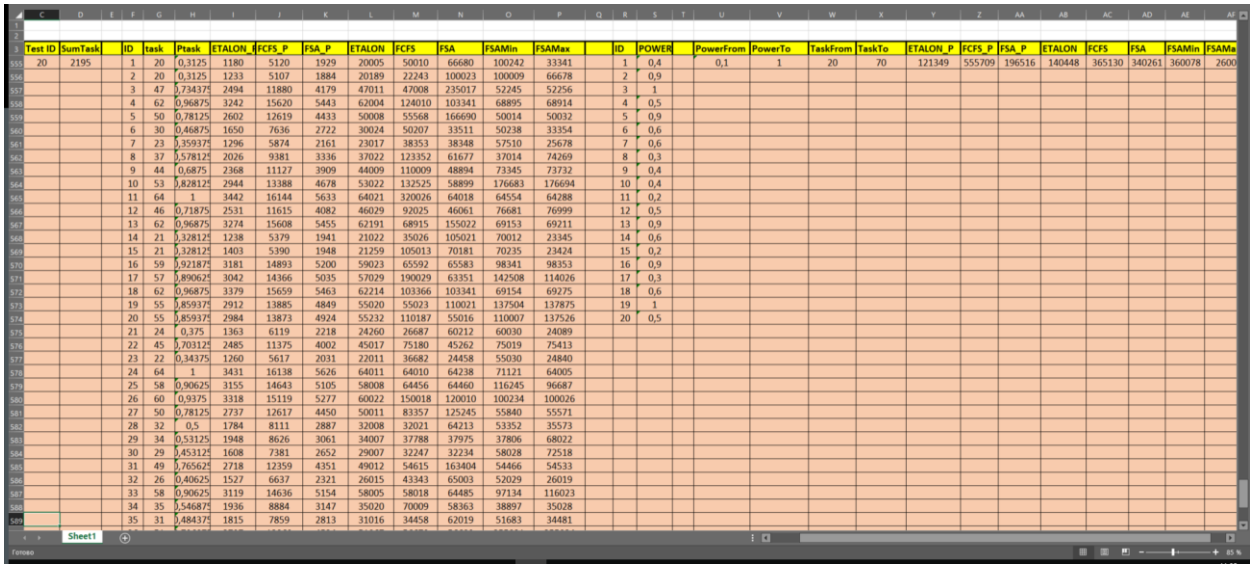
5) в області 2 рис. 3.9 потрібно задати кількість Client, які необхідно запустити, після цього можна довільно змінити потужності запущених Client, перед цим задавши максимальне та мінімальне значення в області 3 рис. 3.9, при цьому потужність 1 відповідає найбільш потужному ПК в GRID-системі, а далі вона зменшується на 10% на кожному кроці — до 0,1, що на даному етапі цілком достатньо, щоб побачити зміну поведінки алгоритмів при зміні потужності вузлів;

6) після проведення всіх налаштувань необхідно запустити тест, натиснувши кнопку «Start testing».

Система почне автоматично виконувати завдання, використовуючи для їх розподілу по обчислювальним вузлам різні алгоритми по черзі. Спочатку виконуються всі завдання при розподілі завдань відповідно до першого алгоритму диспетчеризації. При цьому обліковується час виконання кожного завдання, після чого підраховується загальний час роботи, від початку запуску до виконання всіх завдань. Потім ті ж самі завдання знову подаються на виконання, тільки розподіл здійснюється за іншим алгоритмом

диспетчеризації. При цьому також виконується облік часу. Така процедура продовжується до виконання тестом розподілу за останнім алгоритмом диспетчеризації. В програмі реалізована можливість запуску роботи для виконання лише якогось конкретного алгоритму диспетчеризації. Для цього необхідно вибрати алгоритм, який цікавить, і натиснути кнопку «Send».

На рис. 3.11 наведено вигляд таблиці результатів тестування. Структура таблиці дозволяє далі працювати з нею. При проведенні нового тесту інформація додається до вже існуючої, а тесту присвоюється ідентифікатор, що не потребує створення нового файлу.



Test ID	SumTask		ID	task	Ptask	ETALON_P	FCFS_P	FSA_P	ETALON	FCFS	FSA	FSAMin	FSAMax
20	2195		1	20	0,3125	1180	5120	1929	20005	50010	66680	100242	33341
			2	20	0,3125	1233	5107	1884	20189	22243	100023	100009	66678
			3	47	0,734375	2494	11880	4179	47011	47008	235017	52245	52256
			4	62	0,96875	3242	15620	5443	62004	124010	103341	68895	68914
			5	50	0,78125	2602	12619	4433	50008	55568	166690	50014	50032
			6	30	0,46875	1650	7636	2722	30024	50207	33511	50238	33354
			7	23	0,359375	1296	5874	2161	23017	38353	38348	57510	25678
			8	37	0,578125	2026	9381	3336	37022	123352	61677	37014	74269
			9	44	0,6875	2368	11127	3909	44009	110009	48894	73345	73732
			10	53	0,828125	2944	13388	4678	53022	132525	58899	176683	176694
			11	64	1	3442	16144	5633	64021	320026	64018	64554	64288
			12	46	0,71875	2531	11615	4082	46029	92025	46061	76681	76999
			13	62	0,96875	3274	15608	5455	62191	68915	155022	69153	69211
			14	21	0,328125	1238	5379	1941	21022	35026	105021	70012	23345
			15	21	0,328125	1403	5390	1948	21259	105013	70181	70235	23424
			16	59	0,921875	3181	14893	5200	59023	65592	65583	98341	98353
			17	57	0,890625	3042	14366	5035	57029	190029	63351	142508	114026
			18	62	0,96875	3379	15659	5463	62214	103366	103341	69154	69275
			19	55	0,859375	2912	13885	4849	55020	55023	110021	137504	137875
			20	55	0,859375	2984	13873	4924	55232	110187	55016	110007	137526
			21	24	0,375	1363	6119	2218	24260	26687	60212	60030	24089
			22	45	0,703125	2485	11375	4002	45017	75180	45262	75019	75413
			23	22	0,34375	1260	5617	2031	22011	36682	24458	55030	24840
			24	64	1	3431	16138	5626	64011	64010	64238	71121	64005
			25	58	0,90625	3155	14643	5105	58008	64456	64460	116245	96687
			26	60	0,9375	3318	15119	5277	60022	150018	120010	100234	100026
			27	50	0,78125	2737	12617	4450	50011	83357	125245	55840	55571
			28	32	0,5	1784	8111	2887	32008	32021	64213	53352	35573
			29	34	0,53125	1948	8626	3061	34007	37788	37975	37806	68022
			30	29	0,453125	1608	7381	2652	29007	32247	32234	58028	72518
			31	49	0,765625	2718	12359	4351	49012	54615	163404	54466	54533
			32	26	0,40625	1527	6637	2321	26015	43343	65003	52029	26019
			33	58	0,90625	3119	14636	5154	58005	58018	64485	97134	116023
			34	35	0,546875	1936	8884	3147	35020	70009	58363	38897	35028
			35	31	0,484375	1815	7859	2813	31016	34458	62019	51683	34481

ID	POWER
1	0,4
2	0,9
3	1
4	0,5
5	0,9
6	0,6
7	0,6
8	0,3
9	0,4
10	0,4
11	0,2
12	0,5
13	0,9
14	0,6
15	0,2
16	0,9
17	0,3
18	0,6
19	1
20	0,5

PowerFrom	PowerTo	TaskFrom	TaskTo	ETALON_P	FCFS_P	FSA_P	ETALON	FCFS	FSA	FSAMin	FSAMa
0,1	1	20	70	121349	555709	196516	140448	365130	340261	360078	2600

Рисунок 3.11 – Загальний вигляд результатів тестування в файлі MS Excel

Після запуску тестування всі кнопки та елементи вибору налаштувань стають неактивними, до моменту завершення тестування. Результати тесту можна зберегти в файлі MS Excel, натиснувши кнопку «Export to Excel». В даному файлі зберігається вся інформація про виконаний тест, а саме: кількість вузлів та їх потужність, кількість завдань та їх потужність, сумарний час виконання всіх завдань та час виконання кожного завдання відповідно до кожного з алгоритмів диспетчеризації, задані настройки для мінімальної та максимальної потужностей обчислювальних вузлів, заданий

мінімальний та максимальний об’єм завдань, а також сумарні результати обчислень всіх завдань відповідно до кожного алгоритму диспетчеризації.

На рис. 3.12 показано вигляд запущеного симулятора на одному ПК з підключеним до нього монітором, тобто на двох екранах з розширенням 1920×1080. В даному випадку без особливих труднощів розміщено 15 Client та один Server.

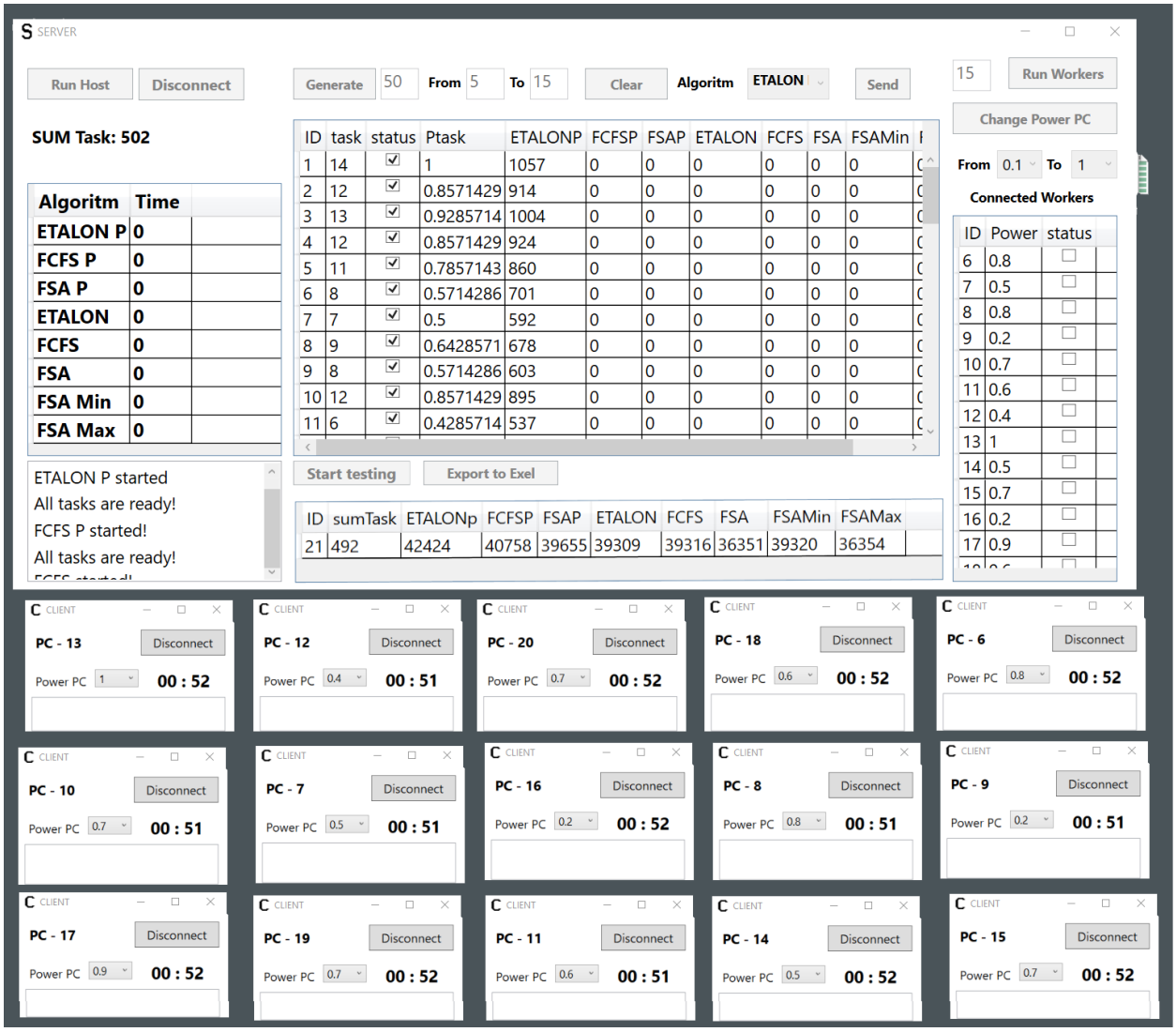


Рисунок 3.12 – Вигляд SGRIDAR-1 на двох екранах

Однією з позитивних якостей даного симулятора є можливість спостерігати за роботою системи в реальному часі, за рахунок чого

користувачеві добре видно, як вузли приймають завдання та виконують їх. Цю функцію реалізовано за допомогою таймера зворотного відліку часу. Ця особливість робить зручним даний комплекс програм не тільки для досліджень методів диспетчеризації завдань, але і в освітньому процесі, для кращого розуміння суті розподілених обчислень та GRID-систем в цілому.

3.4 Висновки до розділу 3

1) Проведено аналіз існуючого програмного забезпечення для моделювання GRID-систем, яке дозволяє симулювати її роботу, а також досліджувати різні методи планування завдань при умові різнорідності обчислювальних вузлів системи та неоднорідності каналів зв’язку між ними. Аналіз показав, що наразі існує чимало програмних засобів, які створені для симуляції роботи GRID-систем та великих комп’ютерних мереж, а також для дослідження методів планування завдань в них. Серед існуючих симуляторів було виділено наступні: OMNET++, Bricks, MicroGRID, SimGRID, GRIDSim. Наведено їх короткий опис та функціональні можливості.

2) Проведено аналіз існуючого програмного забезпечення, яке дозволяє симулювати роботу Desktop GRID, а також досліджувати різні методи планування завдань. Аналіз показав, що наразі існує ряд програмних засобів, які створені для симуляції роботи Desktop GRID та дослідження методів планування завдань в них. Серед існуючих симуляторів було виділено наступні: SimBOINC, EmBOINC, SimBA, ComBoS, Alea. Наведено короткий опис кожного з виділених симуляторів.

3) В результаті проведених аналізів встановлено, що описані вище симулятори, без додаткового доопрацювання та розробки нових бібліотек до них, не можуть враховувати потужності завдань та потужності вузлів таким чином, як це потрібно відповідно до запропонованого підходу та розроблених на його основі методів планування завдань, а також потребують великих затрат на таку розробку. Крім того всі описані симулятори Desktop

GRID розроблені для дослідження обчислень, з урахуванням, що система буде створюватися на основі BOINC, а в даному випадку це не так.

4) Обґрунтовано чому було вирішено, що вони не підходять для продовження цієї роботи і потрібно створити нову програму, яка б могла симулювати роботу GRID-системи з невідчужуваними ресурсами, використовуючи різні алгоритми, а головне, щоб можна було дослідити новий підхід та запропоновані методи.

5) Розроблено програмний комплекс, що дозволяє симулювати роботу GRID-системи з невідчужуваними ресурсами, а також спостерігати та досліджувати роботу різних методів планування при різних умовах. Описано архітектурну модель розробленого симулятора, який складається з трьох програм: Client, Host, Server, а також їх основні функції. Розроблений програмний комплекс дозволяє показати роботу запропонованого методу та порівняти результати його роботи з іншими, загальновідомими методами диспетчеризації. За допомогою даного програмного комплексу можна проводити експерименти та досліджувати роботу алгоритмів, змінюючи кількість та розмір завдань, кількість та потужність вузлів. Реалізований візуальний інтерфейс наглядно показує, як працює GRID-система. Даний комплекс програм можна використовувати не тільки для наукових досліджень, але і для освітнього процесу завдяки наявності інтуїтивно зрозумілого та інтерактивного графічного інтерфейсу, що дозволяє спостерігати за процесом розподілених обчислень на вузлах системи.

РОЗДІЛ 4
ТЕСТУВАННЯ МЕТОДІВ FSA, FSA_P, FSA Min, FSA Max НА ПРОГРАМНОМУ КОМПЛЕКСІ SGRIDAR-1

Для проведення аналізу ефективності запропонованих методів потрібно визначити, з яким загальновідомим методом доцільно порівнювати запропоновані методи та які умови необхідно задати для тестування на програмному комплексі SGRIDAR-1.

4.1 Обґрунтування вибору методу для дослідження ефективності методів FSA, FSA_P, FSA Min, FSA Max

На прийняття рішення про вибір загальновідомого методу диспетчеризації завдань для проведення порівняння ефективності запропонованих методів FSA, FSA_P, FSA Min, FSA Max вплинули ряд причин, що описані нижче.

По-перше, проведений в 1.9 аналіз існуючих методів диспетчеризації завдань показав, що наразі в реальних системах зазвичай використовується переважно метод FCFS.

По-друге, було проаналізовано цілий ряд публікацій за останні роки на дану тему, в яких розробниками пропонуються нові методи та різні модифікації відомих, при цьому порівнюються нові методи переважно з FCFS та з SJF (shorts job first). З прикладами таких публікацій можна ознайомитись за посиланнями [145-147]. Враховуючи, що метод SJF відрізняється тим, що береться найкоротша робота, а не та, що перша прийшла, як у FCFS, то результати роботи будуть різними, що вплине на таку характеристику, як середній час очікування в черзі. Але наразі вона нас не цікавить, а цікавить лише загальний час виконання всіх завдань. Крім того, кількість операцій для роботи SJF більша, хоча це сильно не впливає на

роботу, ніж для FCFS, тому доцільно використовувати в цьому дослідженні саме FCFS.

Крім того, хотілося б зазначити, що інші методи недоцільно використовувати через те, що вони не передбачають врахування потужності вузлів та потужності завдань, як того вимагають запропоновані в даній роботі методи диспетчеризації завдань, а інші критерії, такі, наприклад, як квант часу, пріоритети виконання, прогнозований час виконання та інші, що вимагають інші відомі методи диспетчеризації завдань, не враховуються запропонованими методами FSA, FSA_P, FSA Min, FSA Max.

Сукупність вище описаних причин дало підстави зробити висновок про необхідність розробки нових методів, основними характеристиками яких повинні бути простота і краща продуктивність у порівнянні з FCFS.

Нижче буде наведено результати тестування.

4.2 Тестування

Описаний вище підхід та розроблені на його основі методи диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами / Desktop GRID є досить простими у використанні та програмній реалізації, але він фактично розділяє задачу диспетчеризації завдань у GRID-системах з невідчужуваними ресурсами / Desktop GRID на три підзадачі:

1) обчислення потужностей завдань;
2) обчислення потужностей вузлів;
3) розподіл (метод FSA або ж FSA_P).

Як перша так і друга підзадачі є досить складними і на сьогодні не існує однозначного та універсального їх вирішення. Справа в тому, що будь-яке завдання має цілий ряд характеристик, про які вже писалось вище, і порівнювати їх та якимось чином зводити до однієї величини досить складно. Для цього потрібно розроблювати додаткові методи, які б надавали таку можливість.

З іншого боку, задача обчислення потужностей вузлів є не менш складною і також потребує окремого вивчення та вирішення, хоча і є значно простішою ніж задача обчислення потужності завдання.

Але в той же час є цілий ряд завдань, для яких обчислення потужностей не визве особливих складнощів. Це добре демонструє приклад простої практичної задачі, що буде описаний в 4.3.

В даному розділі показані результати обчислень, що проводились в симуляторі SGRIDAR-1. В даній програмі був реалізований механізм генерації завдань, який генерує завдання на виконання довільним чином, задавши час, який потрібно завданню виконуватись. Потужності завдань розраховуються пропорційно даному часу. Потужності ж вузлів також генеруються довільним чином. Та, в залежності від їх величини, сповільнюють роботу таймера.

Тобто, потужність завдання розглядається як відносна величина і залежить від набору завдань в множині всіх завдань на виконання, при цьому максимальна потужність завдання завжди буде дорівнювати одиниці і перераховуватись в залежності від змін у черзі. Наприклад, коли з черги буде видалено завдання з максимальною потужністю, то тоді потужність всіх решти завдань автоматично буде перерахована. Знову буде виділено максимальне за об’ємом завдання, якому буде присвоєно значення потужності рівної одиниці, а решта потужностей завдань буде перерахована вже в залежності від нового максимального завдання.

Аналогічним чином мають перераховуватися і потужності вузлів. Це також відносна величина і завжди залежить від максимальної потужності вузла.

Будь-яка обчислювальна GRID-система має початковий стан. Для задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами / Desktop GRID такий стан визначається початковим набором завдань, що потрібно обчислити чи виконати та набором обчислювальних

вузлів чи ПК, які на даний момент часу доступні та можуть прийняти завдання та виконати його.

Далі будемо розглядати такі основні випадки початкового стану системи та дії, які виконуються в такому випадку:

– кількість завдань перевищує кількість вузлів ($N > M$): відправляємо N завдань на виконання доступним M вузлам, а решта ($N – M$) завдань чекають своєї черги, доки не звільниться якийсь вузол, після чого вибираємо для нього нове завдання з тих, що чекають на виконання;

– кількість завдань дорівнює кількості вузлів ($N = M$): розподіляємо всі завдання по всім обчислювальним вузлам;

– кількість завдань менше кількості вузлів ($N < M$): обираємо N обчислювальних вузлів та відправляємо їм завдання на виконання, а решта ($M – N$) вузлів чекають, доки на вхід не поступить нове завдання.

З використанням розробленого програмного комплексу для симуляції роботи GRID-системи з невідчужуваними ресурсами / Desktop GRID SGRIDAR-1 було проведено ряд тестів, враховуючи початкові стани системи, що описані вище.

Механізм тестування передбачає виконання всіх завдань, що згенерувала система, використовуючи різні алгоритми розподілу завдань між вузлами. Тобто, спочатку обчислюється час виконання всіх згенерованих завдань при розподілі на виконання з використанням першого методу диспетчеризації, що заведено в систему, потім завдання позначаються як не виконані та знову обчислюється час виконання всіх згенерованих завдань при розподілі на виконання з використанням другого методу диспетчеризації, що заведено в систему, і такі дії проводяться для всіх методів диспетчеризації, що заведені в систему. Після чого виводиться зведений результат та з’являється можливість зберегти результати в файл Excel. В систему заведено наступні методи диспетчеризації: ETALON_P, FCFS_P, FSA_P, ETALON, FSA, FSA Min, FSA Max.

Враховуючи, що запропоновані методи були розроблені як для

послідовних завдань (що не можна розпаралелювати) так і для паралельних (завдання можна розпаралелювати між декількома вузлами), то було вирішено розділити всі методи на дві умовні групи: паралельні методи та послідовні методи.

Послідовні методи:

– ETALON – метод взято за еталонний. Це метод FCFS, в якому не враховуються потужності обчислювальних вузлів і завдання не можна розподіляти на декілька вузлів;

– FCFS – метод, в якому враховано потужності обчислювальних вузлів і завдання не можна розподіляти між вузлами, моделюється система, в якій всі вузли однакові за потужністю;

– FSA – запропонований в роботі метод, в якому враховано потужності обчислювальних вузлів і завдання не можна розподіляти між вузлами;

– FSA Min – запропонований в роботі метод, в якому враховано потужності обчислювальних вузлів і завдання не можна розподіляти між вузлами, даний метод є модифікацією методу FSA з методом Min-Min, який був визначений експериментальним шляхом і дає кращі результати за метод FSA лише у деяких випадках, що буде показано нижче;

– FSA Max – запропонований в роботі метод, в якому враховано потужності обчислювальних вузлів і завдання не можна розподіляти між вузлами, даний метод є модифікацією методу FSA з методом Max-Min, який також був визначений експериментальним шляхом і тільки при певних обставинах дає кращі результати за метод FSA.

Паралельні методи:

– ETALON_P – метод взято за еталонний, це метод FCFS, в якому не враховуються потужності обчислювальних вузлів (ПК) та потужності завдань, моделюється система, в якій всі вузли однакові за потужністю і всі завдання можна розпаралелити на декілька вузлів;

– FCFS_P – метод FCFS, в якому враховано, що обчислювальний

вузол може мати різну потужність, всі завдання можливо розподіляти на всі вузли рівними частинами;

– FSA_P – запропонований в роботі метод, в якому враховано, що обчислювальні вузли можуть бути різної потужності і є можливість розподіляти одне завдання на декілька обчислювальних вузлів.

4.2.1 Результати тестування за тестом 1

В першому тесті розглядається стан системи, коли кількість завдань перевищує кількість вузлів ($N > M$). З отриманими в процесі тестування числовими результатами даного тесту можна ознайомитись в таблицях 4.1, 4.2.

Таблиця 4.1 – Результати тестування за тестом 1 (послідовні методи)

Середня потужність системи	Метод				
	ETALON	FCFS	FSA	FSAMin	FSAMax
0,1	9069	90066	90035	90076	90058
0,15	9048	90057	70180	70200	70214
0,2	9045	90055	70178	70182	70046
0,25	9061	70067	30197	30177	30168
0,3	9062	80029	60074	60049	60037
0,35	9051	70060	30074	30081	30063
0,4	9061	70030	30193	30224	30295
0,45	9052	30045	23382	23533	23505
0,5	9024	70034	30207	30162	30204
0,55	9078	70051	30040	30071	30068
0,6	9057	40090	30058	30056	30073
0,65	9059	26714	23411	23393	23377
0,7	9024	70048	30062	30067	30073
0,75	9051	35049	15156	15147	15191
0,8	9061	35052	15183	15190	15160
0,85	9042	35043	15062	15036	15057
0,9	9066	70039	30064	30057	30113
0,95	9068	10058	10063	9079	9059
1	9057	9072	9062	9081	9068

Таблиця 4.2 – Результати тестування за тестом 1 (паралельні методи)

Середня потужність системи	Метод		
	ETALON_P	FCFS_P	FSA_P
0,1	7638	67238	67454
0,15	7813	67012	45530
0,2	8007	67107	34301
0,25	7987	66945	27957
0,3	8078	66879	23508
0,35	7828	66591	20208
0,4	7990	67129	17781
0,45	8045	22977	16113
0,5	8046	66828	14495
0,55	7730	66541	13420
0,6	7897	33893	12296
0,65	7919	22923	11550
0,7	7909	66570	10816
0,75	7987	33998	10171
0,8	7913	33962	9613
0,85	7883	33593	9185
0,9	7988	66499	8639
0,95	8037	8410	8626
1	7934	7988	7834

В загальному було проведено 19 тестів для різної середньої потужності системи для кожного з методів диспетчеризації завдань, що заведені в систему. Такі дослідження проводились для того, щоб показати, як впливає середня потужність системи на ефективність роботи методів диспетчеризації завдань.

Зокрема, в таблиці 4.1 наведені дані для порівняння послідовних методів диспетчеризації завдань, що заведені в розроблений симулятор SGRIDAR-1(ETALON, FCFS, FSA, FSA Min, FSA Max). Розглядається залежність загального часу виконання всіх завдань, що обчислюється системою в мілісекундах, за допомогою таймера відліку часу, від середньої потужності системи. На рис. 4.1, 4.2 зображено діаграми, які побудовані на основі даних, що наведені в таблиці 4.1.

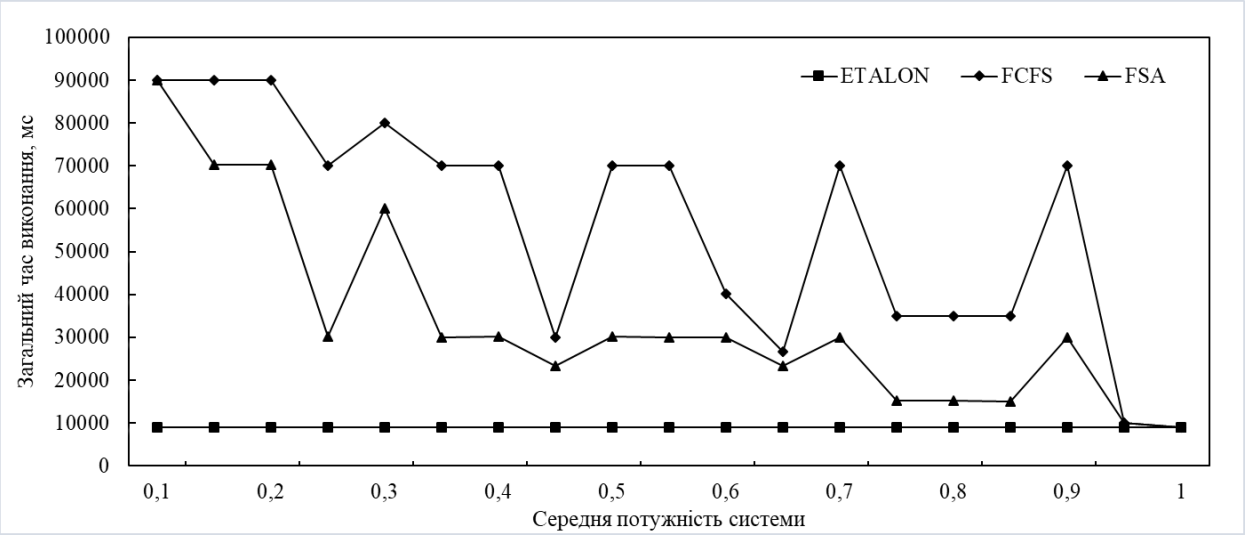


Рисунок 4.1 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 1 для методів ETALON, FCFS, FSA

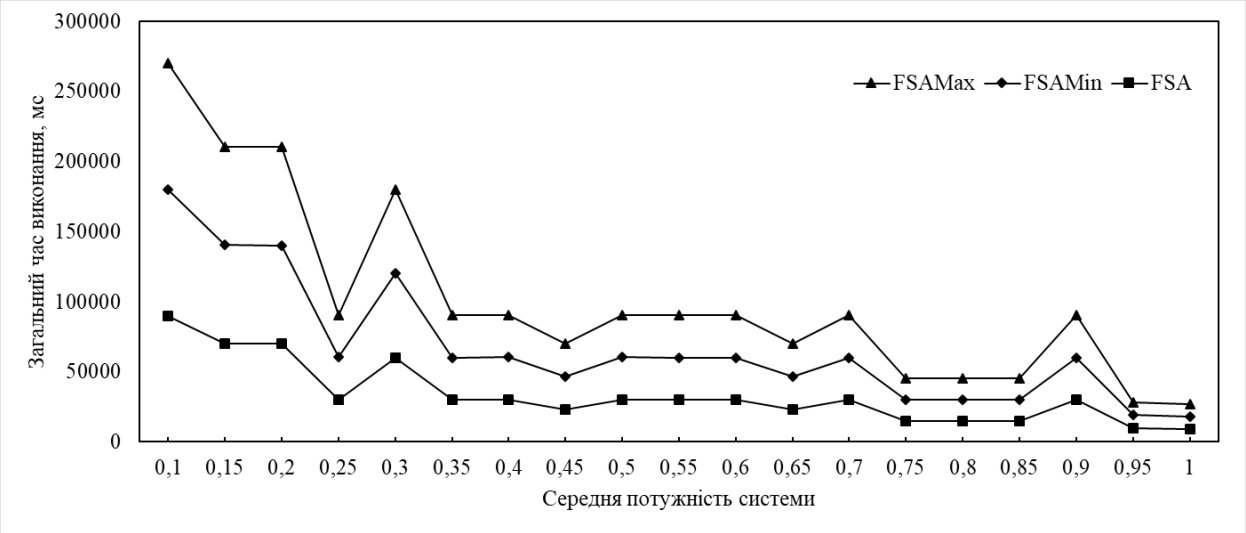


Рисунок 4.2 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 1для методів FSA, FSA Min, FSA Max

На основі отриманих результатів можна зробити висновок про те, що метод FSA дає кращі результати ніж метод FCFS.

На рис. 4.1 видно, що послідовний метод FSA має більш гладку криву, ніж метод FCFS, що свідчить про кращу його прогнозованість. На рис. 4.2 порівнюються розроблені модифікації методу FSA: FSA Min та FSA Max з

ним же, криві методів показують незначні відмінності в результатах. При цьому метод FSA, для випадку, коли кількість завдань перевищує кількість вузлів, дає кращі результати ніж методи FSA Min та FSA Max.

В таблиці 4.2 наведені дані для порівняння послідовних методів диспетчеризації завдань (ETALON_P, FCFS_P, FSA_P), що заведені в розроблений симулятор SGRIDAR-1. Як і у попередньому випадку, розглядається залежність загального часу виконання (відповідно обраного методу диспетчеризації) всіх завдань від середньої потужності системи. На основі цих результатів побудовано діаграму, що зображена на рис. 4.3.

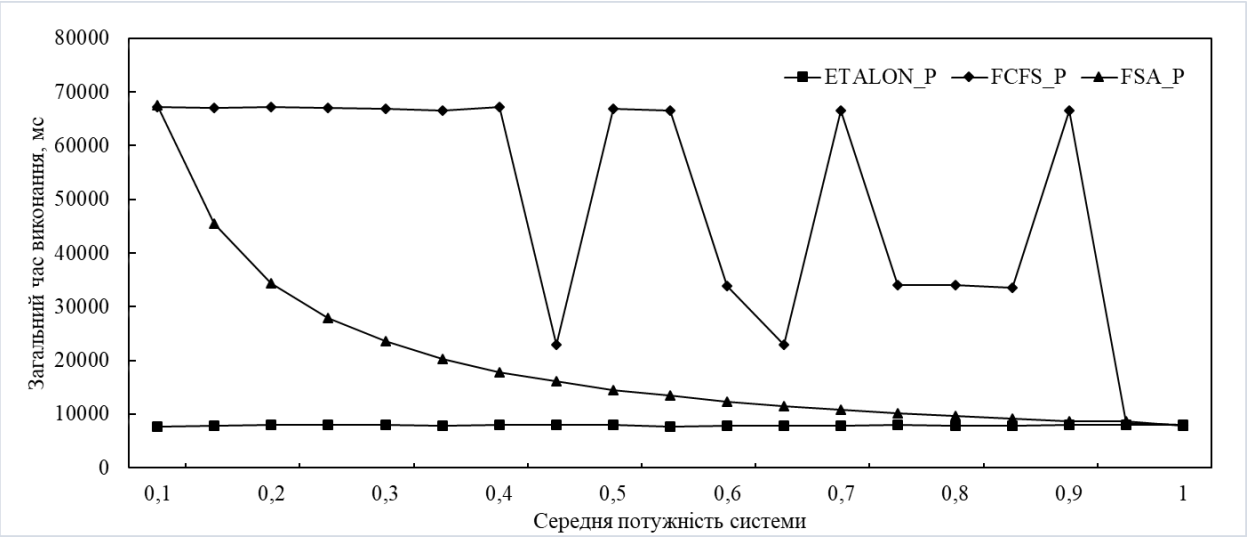


Рисунок 4.3 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 1 для методів ETALON_P, FCFS_P, FSA_P

На основі отриманих результатів можна зробити висновок про те, що метод FSA_P дає кращі результати (виконує всі завдання, що згенерувала система, значно швидше) ніж метод FCFS.

Як видно на рис. 4.3, крива методу FSA_P має плавну форму, а крива методу FCFS_P є ламаною. Це дозволяє зробити висновок, що метод FSA_P є добре прогнозованим і, знаючи потужність системи та об’єм завдань, можна складати прогноз завершення обчислень. А при розподілі завдань методом

FCFS_P, досить складно прогнозувати завершення обчислень, адже багато залежить від того, на який вузол, яке завдання прийде на виконання.

Як видно з діаграм (див. рис. 4.1, 4.2, 4.3), в систему також заведено методи ETALON та ETALON_P, і на перший погляд може здатися, що ефективність їх роботи значно краще, ніж методи FSA та FSA_P. Справа в тому, що методи ETALON та ETALON_P зображені не для порівняння з іншими, а для демонстрації еталонного стану системи, коли в ній всі вузли мають потужність 100%. При збільшенні середньої потужності до 1, час виконання за різними методами наближається до максимально можливого еталонного значення. Дані методи заводилися в систему для перевірки правильності думки щодо впливу різнорідності обчислювальних вузлів, що в даній системі позиціонується, як різна потужність обчислювальних вузлів.

На рис. 4.1, 4.3 видно, що поведінка методу FCFS та FCFS_P дуже нестабільна, великі стрибки у результатах. Це пов’язано з тим, що розподіл за методом FCFS дуже сильно залежить від випадкового фактору, а також від того, на який вузол припаде яке завдання. Це цілком логічно, адже якщо, наприклад, на вузол з потужністю 0,1 прийде завдання з потужністю 1, то час виконання у такому випадку значно виросте, і може скластися ситуація, що решта вузлів відпрацюють та будуть надто довго очікувати завершення цього завдання.

4.2.2 Результати тестування за тестом 2

В другому тесті розглядається стан системи коли кількість завдань дорівнює кількісті вузлів ($N = M$). Хоча такий стан системи, як правило, рідкою буває, адже GRID-системи з невідчужуваними ресурсами / Desktop GRID створюються переважно коли кількість завдань, що потрібно обчислити, значно перевищує кількість вузлів, розглянути його потрібно, бо він теоретично можливий і хотілося б знати як в такому випадку поведуть себе запропоновані в роботі методи диспетчеризації завдань.

Для проведення порівняльних експериментів було згенеровано 30 завдань з часом виконання від 1 до 10 секунд та запущено 30 вузлів, на яких вони повинні виконуватись. Тестування відбувалося аналогічно до тестування за тестом 1, а саме: почергове виконання всіх завдань при розподілі кожним з заведених в систему методів диспетчеризації завдань для кожної величини середньої потужності. Числові результати описаного тесту наведені в таблицях 4.3, 4.4.

Таблиця 4.3 – Результати тестування за тестом 2 (послідовні методи)

Середня потужність системи	Метод				
	ETALON	FCFS	FSA	FSAMin	FSAMax
0,1	9201	90205	90232	90329	90230
0,15	9215	90244	70264	50245	50213
0,2	9266	90206	70217	50221	50225
0,25	9235	80083	70115	30311	30228
0,3	9217	80092	70274	30241	30186
0,35	9244	70068	60047	22747	22728
0,4	9225	80066	23435	20205	20263
0,45	9201	30205	26746	20243	20246
0,5	9221	80047	20226	20218	20243
0,55	9200	70067	20095	20083	20079
0,6	9221	40030	15232	15235	15252
0,65	9212	30216	16732	16738	16772
0,7	9200	70095	20076	20076	20069
0,75	9212	40079	10202	10212	10224
0,8	9213	40090	10224	10203	10207
0,85	9225	35118	10199	10062	10082
0,9	9201	70077	20087	20059	20102
0,95	9233	10190	10248	9242	9202
1	9222	9248	9228	9252	9221

Зокрема, в таблиці 4.3 наведені дані для порівняння послідовних методів диспетчеризації завдань (ETALON, FCFS, FSA, FSA Min, FSA Max),

що заведені в розроблений симулятор SGRIDAR-1. Аналогічно першому тесту, розглядається залежність загального часу виконання всіх завдань від середньої потужності системи.

Таблиця 4.4 – Результати тестування за тестом 2 (паралельні методи)

Середня потужність системи	Метод		
	ETALON_P	FCFS_P	FSA_P
0,1	15596	60829	61130
0,15	16329	57489	43400
0,2	15507	57884	34756
0,25	16383	56309	29947
0,3	12518	56219	26654
0,35	15357	56024	23429
0,4	16304	56245	21471
0,45	11473	21865	18226
0,5	11440	56219	17046
0,55	12348	55744	16305
0,6	11967	29690	15525
0,65	11594	21639	14833
0,7	11351	55989	14455
0,75	12017	29165	13529
0,8	11469	29021	12857
0,85	11952	29236	12578
0,9	11482	55985	12488
0,95	11766	11332	11454
1	10938	12826	12859

В таблиці 4.4 наведені дані для порівняння методів диспетчеризації завдань, що можуть розпаралелювати завдання (ETALON_P, FCFS_P, FSA_P) на всі обчислювальні вузли. Для кожного з зазначених методів було обчислено загальний час виконання всіх завдань при різній середній потужності системи.

На рис. 4.4, 4.5 наведено діаграми, що побудовані за результатами

тестування 2 для послідовних методів (див. табл. 4.3). На основі отриманих результатів можна зробити висновок про те, що метод FSA дає кращі результати ніж метод FCFS.

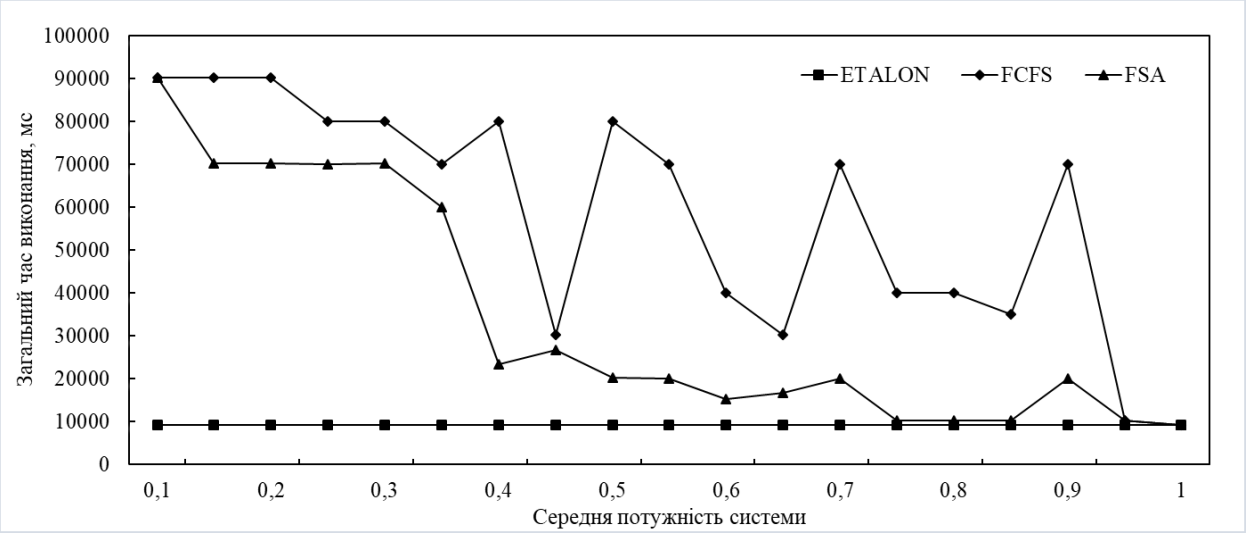


Рисунок 4.4 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 2 для методів ETALON, FCFS, FSA

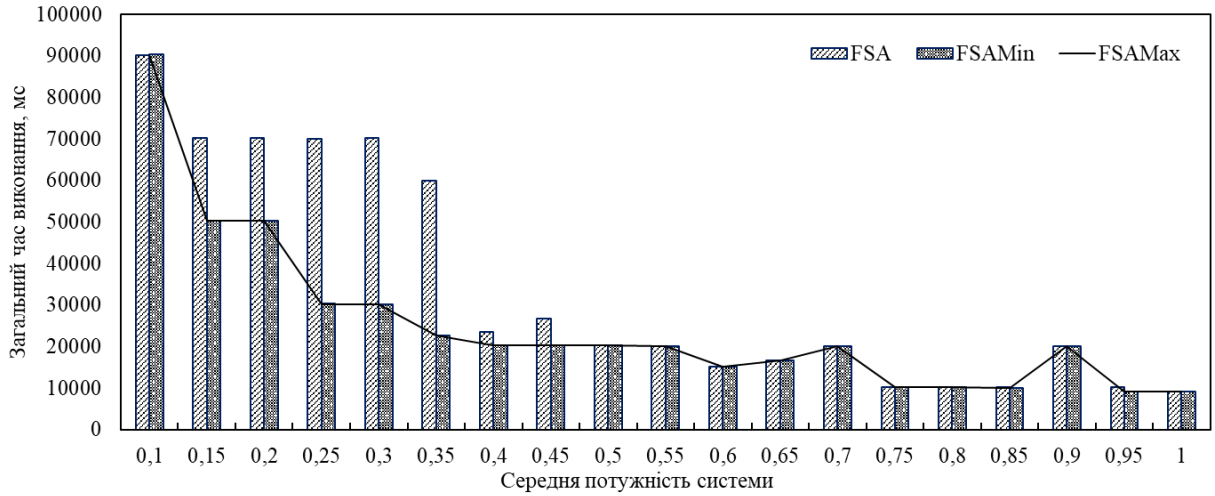


Рисунок 4.5 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 2 для методів FSA, FSA_Min, FSA_Max

На рис. 4.6 зображені діаграми, побудовані за результатами тестування

2 для паралельних методів ETALON_P, FCFS_P, FSA_P (див. табл. 4.4). На основі отриманих результатів можна зробити висновок про те, що метод FSA_P дає кращі результати ніж метод FCFS_P.

Як видно на рис. 4.6, крива методу FSA_P має плавну форму, а крива методу FCFS_P є ламаною. Це дозволяє зробити висновок, що метод FSA_P є добре прогнозованим і, знаючи потужність системи та об’єм завдань, можна складати прогноз завершення обчислень. А розподіляючи завдання методом FCFS_P, досить складно прогнозувати завершення обчислень, адже багато залежить від того, на який вузол, яке завдання прийде на виконання.

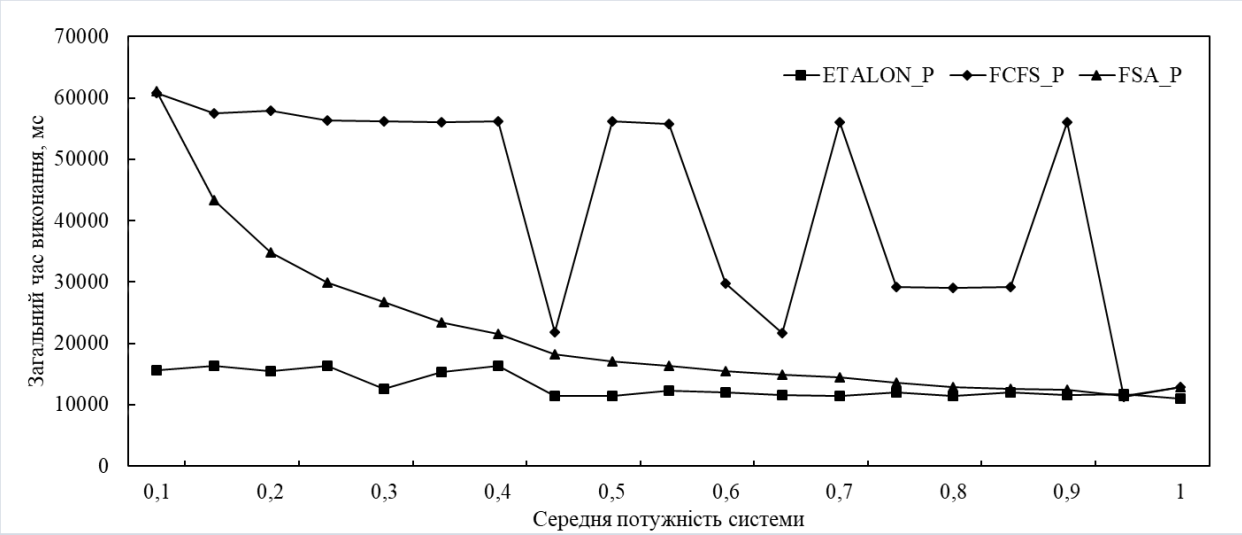


Рисунок 4.6 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 2 для методів ETALON_P, FCFS_P, FSA_P

На рис. 4.5 зображені результати роботи методів FSA Min та FSA Max в порівнянні з методом FSA, видно, що при середній потужності системи від 0,15 до 0,35 час виконання черги завдань при розподілі методами FSA Min та FSA Max кращий ніж час виконання тої самої черги завдань методом FSA. При цьому, результати для методів FSA Min та FSA Max майже не відрізняються.

4.2.3 Результати тестування за тестом 3

В третьому тесті розглядається стан системи коли кількість завдань менше за кількість вузлів ($N < M$). Хоча такий стан для GRID-системи з невідчужуваними ресурсами / Desktop GRID, як і у випадку рівної кількості завдань та вузлів (тест 2), рідкою буває, розглянути його потрібно, бо він теоретично можливий і хотілося б знати, як в такому випадку покажуть запропоновані в роботі методи диспетчеризації завдань.

Результати проведених тестів представлені в таблицях 4.5, 4.6.

Таблиця 4.5 – Результати тестування за тестом 3 (послідовні методи)

Середня потужність системи	Метод				
	ETALON	FCFS	FSA	FSAMin	FSAMax
0,1	9201	90205	90232	90329	90230
0,15	9215	90244	70264	50245	50213
0,2	9266	90206	70217	50221	50225
0,25	9235	80083	70115	30311	30228
0,3	9217	80092	70274	30241	30186
0,35	9244	70068	60047	22747	22728
0,4	9225	80066	23435	20205	20263
0,45	9201	30205	26746	20243	20246
0,5	9221	80047	20226	20218	20243
0,55	9200	70067	20095	20083	20079
0,6	9221	40030	15232	15235	15252
0,65	9212	30216	16732	16738	16772
0,7	9200	70095	20076	20076	20069
0,75	9212	40079	10202	10212	10224
0,8	9213	40090	10224	10203	10207
0,85	9225	35118	10199	10062	10082
0,9	9201	70077	20087	20059	20102
0,95	9233	10190	10248	9242	9202
1	9222	9248	9228	9252	9221

Зокрема, в таблиці 4.5 наведені дані для порівняння послідовних методів диспетчеризації завдань (ETALON, FCFS, FSA, FSA Min, FSA Max).

Розглядається залежність загального часу виконання всіх завдань від середньої потужності системи.

В таблиці 4.6 наведені дані для порівняння паралельних методів. Як і у попередньому випадку, розглядається залежність загального часу виконання (відповідно обраного методу диспетчеризації: FCFS_P, FSA_P) всіх завдань від середньої потужності системи.

Таблиця 4.6 – Результати тестування за тестом 3 (паралельні методи)

Середня потужність системи	Метод		
	ETALON_P	FCFS_P	FSA_P
0,1	15596	60829	61130
0,15	16329	57489	43400
0,2	15507	57884	34756
0,25	16383	56309	29947
0,3	12518	56219	26654
0,35	15357	56024	23429
0,4	16304	56245	21471
0,45	11473	21865	18226
0,5	11440	56219	17046
0,55	12348	55744	16305
0,6	11967	29690	15525
0,65	11594	21639	14833
0,7	11351	55989	14455
0,75	12017	29165	13529
0,8	11469	29021	12857
0,85	11952	29236	12578
0,9	11482	55985	12488
0,95	11766	11332	11454
1	10938	12826	12859

Для проведення порівняльних експериментів було згенеровано 25 завдань з часом виконання від 1 до 10 секунд та запущено 30 вузлів, на яких вони повинні виконуватись. Тестування відбувалося аналогічно до

тестування за тестом 1 та тестом 2 – почергове виконання всіх завдань при розподілі кожним з методів диспетчеризації для кожної величини середньої потужності.

На рис. 4.7, 4.8, 4.9 наведено діаграми, що побудовані за результатами тестування 3 (див. табл. 4.3, 4.4).

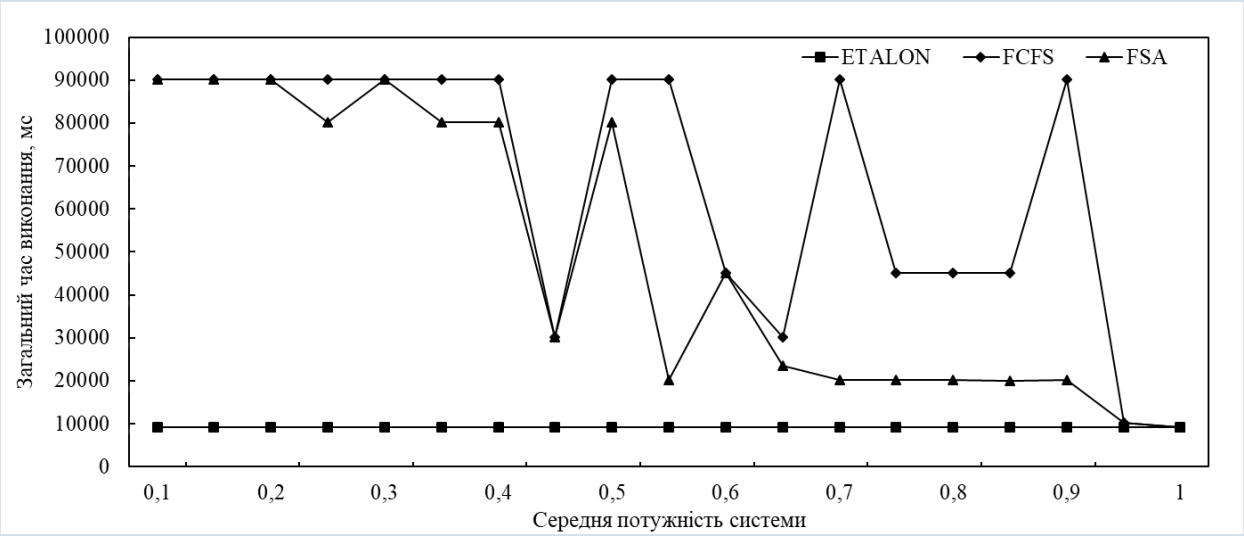


Рисунок 4.7 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 3 для методів ETALON, FCFS, FSA

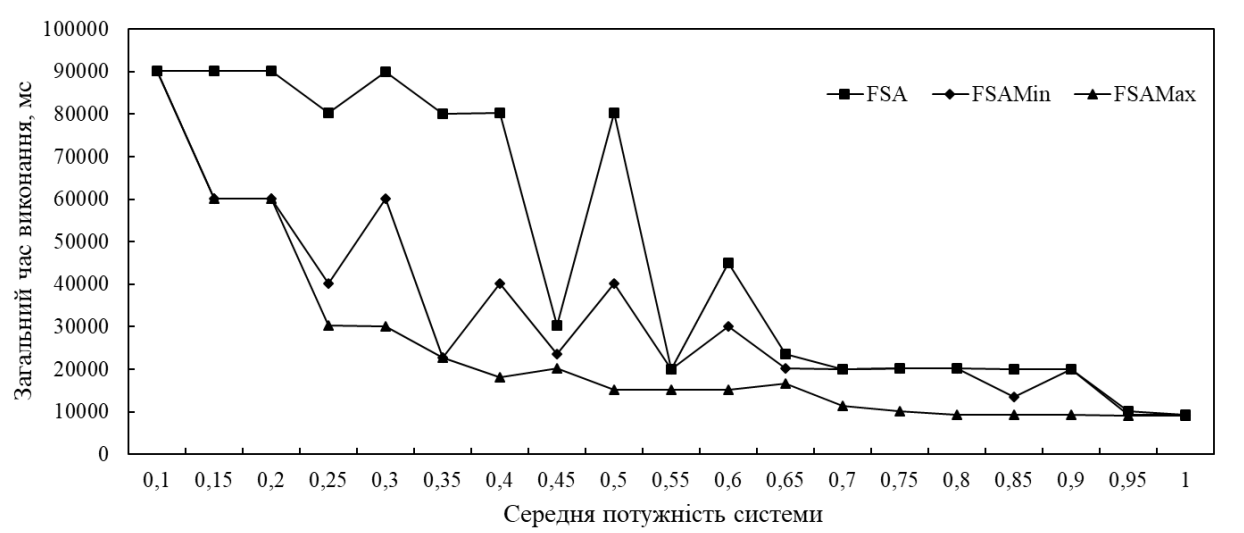


Рисунок 4.8 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 3 для методів FSA, FSA_Min, FSA_Max

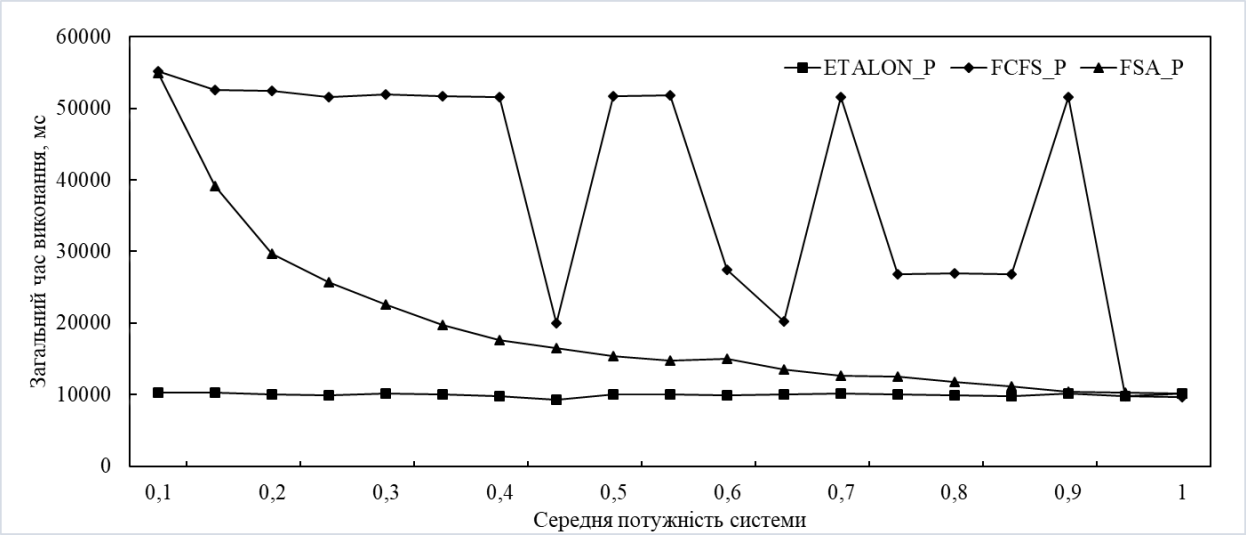


Рисунок 4.9 – Діаграми залежності часу виконання черги завдань від середньої потужності системи за тестом 3 для методів ETALON_P, FCFS_P, FSA_P

На основі отриманих результатів можна зробити висновок про те, що методи FSA і FSA_P, як і у попередніх тестах, дають кращі результати ніж методи FCFS і FCFS_P.

На рис. 4.7, 4.9 видно, що поведінка методу FCFS, як і для тесту 1 та тесту 2, дуже нестабільна, великі стрибки у результатах.

На рис. 4.8 зображені діаграми для додаткових методів FSA Min та FSA Max в порівнянні з методом FSA, видно, що майже при будь-якій середній потужності системи час виконання черги завдань при розподілі методами FSA Min та FSA Max кращий ніж час виконання тої самої черги завдань методом FSA, і найкращий з них належить методу FSA Max.

Як видно на рисунку 4.9, крива методу FSA_P має плавну форму, а крива методу FCFS_P є ламаною. Що також говорить про кращу прогнозованість методу FSA_P у порівняні з методом FCFS_P і для тесту 3.

4.3 Приклад практичної задачі оптимізації розміру зображень

В даному розділі пропонується розглянути таку просту задачу, як приклад використання запропонованих методів диспетчеризації завдань у

GRID-системах з невідчужуваними ресурсами / Desktop GRID.

Якщо, наприклад, ми маємо веб-ресурс із великою кількістю зображень (каталог зображень, соціальна мережа, портфоліо, інтернет-магазин тощо) і ми хочемо просувати його в пошукових системах для того, щоб залучити більше користувачів і збільшити дохід, нам потрібно виконати оптимізацію сторінок нашого ресурсу відповідно до правил пошукової системи (наприклад, за рекомендаціями Google PageSpeed [148]), що включає в себе і оптимізацію зображень на сайті.

В якості прикладу використання запропонованих методів у GRID-системах з невідчужуваними ресурсами / Desktop GRID в роботі розглянуто задачу, схематично наведену на рис. 4.10. Скажімо, у нас є 1000 зображень, які потрібно оптимізувати відповідно до технології оптимізації Google PageSpeed. Загальний розмір файлів – 5 Гб. Якщо виконувати цю задачу на одному ПК, то це займе багато часу. Можливо, навіть цілий день. Все залежить від швидкості роботи ПК, на якому її запустити. Тому, щоб пришвидшити виконання цієї роботи, було б доцільно поділити її між різними ПК, які ми маємо.

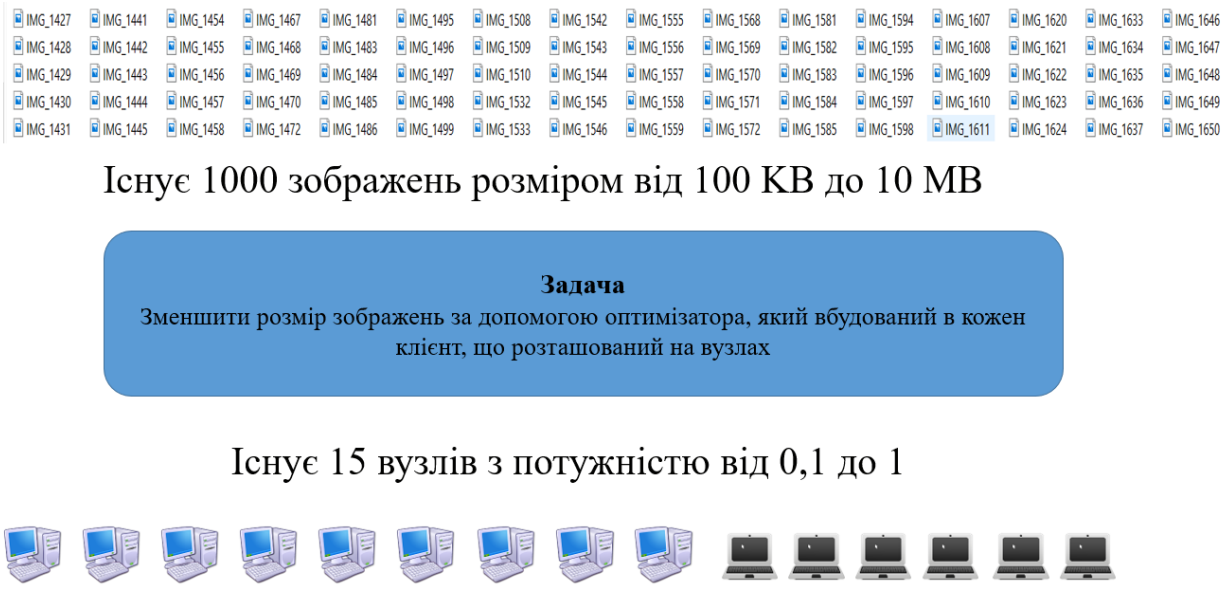


Рисунок 4.10 – Схематичне зображення задачі оптимізації розміру зображень

Припустимо, що у нас є ряд обчислювальних вузлів (ПК) в одному приміщенні чи навіть в різних, які ми можемо використовувати для вирішення таких завдань. І всі обчислювальні вузли пов’язані між собою за допомогою Desktop GRID програмного забезпечення.

Для цього Desktop GRID нам потрібно буде визначити, який метод буде використаний для розподілу окремих файлів зображень між обчислювальними вузлами системи.

Насправді, як зазначалося вище, існує новий метод FSA, заснований на описаному підході.

Основний недолік запропонованого методу полягає в складності розрахунку потужності завдань і вузлів. Але для цього практичного завдання ця проблема вирішується досить просто. Враховуючи, що всі зображення відрізняються одне від одного лише за розміром, найбільшому зображенню ($S_{\max}$) присвоюється потужність $P_{\max}=1$, а решта $(n-1)$ потужностей завдань обчислюється за формулою:

$$P_i = \frac{S_i \times P_{\max}}{S_{\max}}. \tag{4.1}$$

Отже, нехай ми маємо *n* зображень та *m* обчислювальних вузлів.

Враховуючи, що $P_{\max}=1$ формулу (4.1) можемо привести до наступного вигляду:

$$P_i = \frac{S_i}{S_{\max}} \tag{4.2}$$

З іншого боку, у нас є *m* ПК (обчислювальних вузлів), які відрізняються один від одного, наприклад, лише тактовою частотою процесора. Тоді, подібно до розрахунку потужності завдань, ми можемо пропорційно знайти потужність вузлів.

Вузлу з найбільшою тактовою частотою ($F_{\max}$) присвоюється потужність $R_{\max}=1$, а решта $(m-1)$ потужностей вузлів обчислюється за формулою:

$$R_j = \frac{F_j \times R_{\max}}{F_{\max}} \quad (4.3)$$

Враховуючи, що $R_{\max} = 1$ ми можемо скоротити вираз:

$$R_j = \frac{F_j}{F_{\max}}. \quad (4.4)$$

Формули (4.2) та (4.3) використовуються на третьому кроці алгоритму планування завдань FSA згідно його визначення.

Таким чином показано, що розрахувати потужність завдань і потужність вузлів не є такою складною задачею, коли потрібно розподілити зображення між вузлами, щоб оптимізувати розмір. Якщо ж використовуються вузли з різною швидкістю з'єднання з мережею, то їх потужності необхідно обчислювати якимось іншим способом, оскільки не завжди вузол з процесором з найбільшою тактовою частотою матиме найбільшу потужність. І якщо цей момент не враховано, то розподіл буде не таким ефективним, як у першому випадку.

Як зазначалося вище, метод FSA універсальний, і, розподіляючи завдання відповідно до нього, можливо значно пришвидшити виконання черги завдань і тим самим підвищити ефективність роботи GRID-системи з невідчужуваними ресурсами / Desktop GRID, ніж використовуючи для розподілу завдань метод FCFS.

4.4 Висновки до розділу 4

1) Обґрунтовано доцільність використання методу FCFS як простого загальновідомого та широко використовуваного методу диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами / Desktop GRID для його порівняння з запропонованими у даній роботі методами FSA, FSA Min, FSA Max. Зокрема, порівнюється час виконання черги завдань.

2) Проведено тестування запропонованих методів диспетчеризації завдань FSA, FSA Min, FSA Max, а також методу FCFS. Тестування

проводилось на програмному комплексі SGridAR-1. Було проведено ряд тестів для різних станів системи. Зокрема, розглядалися наступні випадки: коли кількість завдань вища за кількість обчислювальних вузлів, кількість завдань рівна кількості обчислювальних вузлів та кількість завдань менша за кількість обчислювальних вузлів. Наведено числові результати тестування у вигляді таблиць, на основі яких побудовано графіки залежності загального часу виконання черги завдань від середньої потужності системи.

3) Результати дослідження ефективності запропонованих методів засвідчили, що їх використання для розподілу завдань у GRID-системах з невідчужуваними ресурсами забезпечує зменшення часу виконання черги завдань до 2,3 разів, у порівнянні з методом планування FCFS, за умови, що кількість завдань перевищує кількість вузлів. Водночас, у початковому стані системи, коли кількість завдань дорівнює кількості вузлів, а середня потужність системи складає від 0,15 до 0,35, використання методів FSA Min та FSA Max забезпечує зменшення часу виконання черги завдань (в порівнянні з FSA) приблизно у два рази.

4) Всі запропоновані методи досить стабільні та добре прогнозовані, а це означає, що використання їх в GRID-системах дасть переваги не тільки в часі та продуктивності, а й дозволить більш ефективно планувати роботу системи. Метод FCFS працює добре, але для GRID-систем, які мають різнорідні за потужністю ресурси, його продуктивність суттєво залежить від випадкового фактору, що є значним недоліком.

5) Встановлено, що основна складність запропонованих методів FSA, FSA Min, FSA Max полягає в обчисленні потужностей вузлів та потужностей завдань. Але існує цілий ряд таких завдань, при вирішенні яких, це цілком легко можливо вирішувати. Наведено приклад простої практичної задачі оптимізації розміру зображень, який показав, як досить просто розрахувати як потужність завдань, так і потужність вузлів.

ВИСНОВКИ

У дисертаційній роботі наведено нове вирішення актуальної науково-практичної задачі, що пов'язана з диспетчеризацією завдань в GRID-системах з невідчужуваними ресурсами. Дана задача полягає в подоланні низької ефективності простих і високій вартості та ненадійності складних підходів до створення методів диспетчеризації, зокрема у системах з різнорідними ресурсами. Вирішується дана задача шляхом розроблення методів диспетчеризації завдань, що враховують різнорідність ресурсів.

У роботі отримано наступні наукові та практичні результати:

1) проведено порівняльний аналіз сучасних підходів до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами, проаналізовано існуючі програмні засоби для дослідження таких систем;

2) вперше запропоновано підхід до вирішення задачі диспетчеризації завдань в GRID-системах з невідчужуваними ресурсами, який, на відміну від існуючих, пропонує здійснювати розподіл завдань відповідно до закону балансу сил, що забезпечує розроблення простих та ефективних методів диспетчеризації;

3) розроблено метод диспетчеризації завдань (що легко розпаралелюються) в GRID-системах з невідчужуваними ресурсами (FSA_P), який, на відміну від існуючих, враховує різну обчислювальну потужність вузлів, що забезпечує близьку до максимально можливої продуктивності системи;

4) розроблено метод диспетчеризації завдань (що не розпаралелюються) в GRID-системах з невідчужуваними ресурсами (FSA), який, на відміну від існуючих, враховує різну обчислювальну потужність вузлів та різні властивості завдань, що забезпечує зменшення часу виконання черги завдань;

5) запропоновано дві модифікації методу FSA – FSA Min та FSA Max, в яких для початкового розподілу завдань використовуються ідеї методів Min-Min та Max-Min відповідно, що забезпечує зменшення часу виконання черги завдань в порівнянні з методом FSA при середній потужності системи від 0,15 до 0,35 та початковому стані системи, коли кількість завдань дорівнює кількості вузлів. Таким чином, встановлено, що запропонований метод FSA можливо ефективно комбінувати із загальновідомими методами;

6) розроблено програмний комплекс «Симулятор GRID-системи з невідчужуваними ресурсами» (SGridAR-1), який, на відміну від існуючих, побудований з використанням WCF служби, що дозволяє проводити дослідження методів диспетчеризації, як на одному ПК, так і на багатьох (за умови їх підключення до локальної або глобальної комп'ютерної мережі). Як свідчать експерименти, ПК з характеристиками: Intel Core i7-7500U (2.7-3.5 ГГц) / RAM 24 Гб / SSD 512 МБ / NVidia GeForce 940MX, 2 Гб дозволяє симулювати GRID-систему розміром до 100 вузлів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] M. Singh, “An Overview of Grid Computing,” *Proc. - 2019 Int. Conf. Comput. Commun. Intell. Syst. ICCCIS 2019*, vol. 2019-January, pp. 194–198, 2019, doi: 10.1109/ICCCIS48478.2019.8974490.

[2] “Research data management simplified.” https://www.globus.org/ (accessed 12.02, 2021).

[3] О. М. Данильченко, Т. А. Узденов, “Процес диспетчеризації для GRID-системи з невідчужуваними ресурсами,” *Вісник ЖДТУ*, vol.2, no. 61, pp. 147–154, 2012.

[4] M. U. Bokhari, Q. Makki, and Y. K. Tamandani, “A survey on cloud computing,” *Adv. Intell. Syst. Comput.*, vol. 654, no. 16, pp. 149–164, 2018, doi: 10.1007/978-981-10-6620-7_16.

[5] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the grid: Enabling scalable virtual organizations,” *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 3, pp. 200–222, 2001, doi: 10.1177/109434200101500302.

[6] I. Foster, C. Kesselman, and S. Tuecke, “The Open Grid Services Architecture,” *Grid 2*, pp. 215–257, 2004, doi: 10.1016/b978-155860933-4/50022-5.

[7] H. Kaur, K. Gupta, “Challenges in Grid computing,” *International Journal of Scientific Research Engineering & Technology (IJSRET)*, ISSN 2278 – 0882, vol. 2, no. 3, pp. 141–144, 2013.

[8] О. М. Данильченко, Т. А. Узденов, “Проблеми розробки методів керування паралельними завданнями та їх алгоритмічної підтримки для актуальної форми GRID,” *Тези XXXVI науково-практичної міжвузівської конференції, присвяченої Дню науки, 12-13 травня 2011 року*, vol. 1, pp. 51–52, 2011.

[9] “Task Scheduler for developers - Win32 apps.” https://docs.microsoft.com /en-us/windows/win32/taskschd/task-scheduler-start-page (accessed 12.02, 2021).

[10] V. B. Kropyvnytska, B. V. Klim, A. G. Romanchuk, M. O. Slabinoga, “Investigation of scheduling algorithms in computer systems,” *Exploration and development of oil and gas fields*, UDC 004.942, ISSN 1993—9973, vol. 2, no. 39, pp. 93–105, 2013.

[11] S. Sahana, “Evolutionary based hybrid GA for solving multi-objective grid scheduling problem,” *Microsystem Technologies*, vol. 26, no. 5, pp. 1405-1416, Springer Berlin Heidelberg, 2019, doi: 10.1007/s00542-019-04673-z.

[12] D. Carastan-santos, R. Y. De Camargo, and D. Trystram, “One can only gain by replacing EASY Backfilling: A simple scheduling policies case study,” *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, pp. 1–10, 2019, doi: 10.1109/CCGRID.2019.00010.

[13] K. Dheenadayalan, V. N. Muralidhara and G. Srinivasaraghavan, “Storage Load Control Through Meta-Scheduler,” *Springer International Publishing Switzerland 2016 N. Bjørner et al. (Eds.): ICDCIT 2016*, pp. 75–86, 2016, doi: 10.1007/978-3-319-28034-9_9.

[14] A. A. Haruna, L. T. Jung, and N. Zakaria, “Design and Development of Hybrid Integrated Thermal Aware Job Scheduling on Computational Grid Environment,” *2015 International Symposium on Mathematical Sciences and Computing Research*, IEEE, pp. 13–17, 2015, doi: 10.1109/ISMSC.2015.7594020.

[15] Y. Thet, H. Hlaing, and T. T. Yee, “Static Independent Task Scheduling on Virtualized Servers in Cloud Computing Environment,” *2019 International Conference on Advanced Information Technologies (ICAIT)*, IEEE, pp. 55–59, 2019, doi: 10.1109/AITC.2019.8920865.

[16] О. М. Данильченко, Д. Г. Літвинчук, Т. А. Узденов, “Моделі системи масового обслуговування для розрахунку навантаженості

багатоагентних систем,” *Вісник ЖДТУ*, vol.4, no. 55 – ЖДТУ, pp. 86–93, 2010.

[17] О. М. Данильченко, Д. Г. Літвинчук, Т. А. Узденов, “Моделі розрахунку трудомісткості операцій комунікації,” *Моделювання та інформаційні технології*, vol. 59, pp.72–76, 2011.

[18] P. S. Kumar, P. C. College, V. Jegatheeswari, “Privacy and security issues in cloud computing using idyllic approach Latha Parthiban,” *International Journal of Networking and Virtual Organisations*, vol. 21, no. 1, pp. 30–42, 2019, doi: 10.1504/IJNVO. 2019.101146.

[19] Т. А. Узденов, В. О. Артемчук, “Модель процесу диспетчеризації для організації розподілених обчислень на GRID-системах з невідчужуваними ресурсами,” *IV Международная научная конференция «МОДЕЛИРОВАНИЕ-2012»*, pp. 71–74, 2012.

[20] A. K. Mathur, S. Charan Teja, P. K. Yemula, “Optimal Charging Schedule for Electric Vehicles in Parking Lot with Solar Power Generation,” *Int. Conf. Innov. Smart Grid Technol. ISGT Asia 2018*, pp. 611–615, 2018, doi: 10.1109/ISGT-Asia.2018.8467916.

[21] P. Naithani, “Genetic Algorithm Based Scheduling To Reduce Energy Consumption In Cloud,” *2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, IEEE, pp. 616–620, 2018, doi: 10.1109/PDGC.2018.8745801.

[22] A. Pujiyanta, L. Nugroho, E. Widyawan, “Advance reservation for parametric job on grid computing,” *2019 4th International Conference on Informatics and Computing (ICIC 2019)*, IEEE, pp. 61–67, 2019, doi: 10.1109/ICIC47613.2019.8985978.

[23] A. Pujiyanta, L. Nugroho, E. Widyawan, “Planning and scheduling jobs on grid computing,” *2018 International Symposium on Advanced Intelligent Informatics: Revolutionize Intelligent Informatics Spectrum for Humanity*, SAIN 2018, pp.162-166, 2019, doi: 10.1109/SAIN.2018.8673372.

[24] D. Ramyachitra and P. P. Kumar, “Frog leap algorithm for homology modelling in grid environment,” *International Journal of Grid and Utility Computing*, vol. 7, no. 1, pp, 29–40, 2016, doi: 10.1504/IJGUC.2016.073775.

[25] L. Sant’Ana, D. Carastan-Santos, D. Cordeiro, and R. De Camargo, “Real-time scheduling policy selection from queue and machine states,” *Proc. - 19th IEEE/ACM Int. Symp. Clust. Cloud Grid Comput. CCGrid 2019*, pp. 381–390, 2019, doi: 10.1109/CCGRID.2019.00052.

[26] K. Tindell and H. Hansson, “Real Time Systems by Fixed Priority Scheduling DoCS,” *Processing*, Uppsala University, 1997.

[27] N. C. Audsley, A. Bums, “Scheduling Real-Time Systems,” *YCS 134*, Department of Computer Science, University of York, 1990.

[28] M.V. Danilov, “Methods for scheduling tasks in real-time systems,” Programs and systems, vol. 4, pp. 28-35, 2019.

[29] “First-Come, First-Served (FCFS) — Вікі ЦДПУ.” https://wiki.cuspu.edu.ua/index.php/1._First-Come,_First-Served_(FCFS) (accessed 12.02, 2021).

[30] “Самая короткая работа сначала (SJF) - CoderLessons.com.” https://coderlessons.com/tutorials/akademicheskii/osnovy-operatsionnykh-sistem/27-samaia-korotkaia-rabota-snachala-sjf (accessed 12.02, 2021).

[31] “Алгоритм приоритетного планирования - CoderLessons.com.” https://coderlessons.com/tutorials/akademicheskii/osnovy-operatsionnykh-sistem/25-algoritm-prioritetnogo-planirovaniia (accessed 12.02, 2021).

[32] “Round Robin (rr).” https://studfile.net/preview/920333/page:2/ (accessed 12.02, 2021).

[33] “Scheduling algorithms - operating systems.” https://studme.org/329864/informatika/algoritmy_planirovaniya (accessed 12.02, 2021).

[34] R. F. Freund *et al.*, “Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet,” *Proc. - 7th Heterog. Comput.*

Work. HCW 1998, vol. 1998-March, pp. 184–199, 1998, doi: 10.1109/HCW.1998.666558.

[35] K. Etminani and M. Naghibzadeh, “A min-min max-min selective algorihtm for grid task scheduling,” *2007 3rd IEEE/IFIP Int. Conf. Cent. Asia Internet, ICI 2007*, 2007, doi: 10.1109/canet.2007.4401694.

[36] T. Kokilavani and D. I. G. Amalarethinam, “Load Balanced Min-Min Algorithm for Static Meta-Task Scheduling in Grid Computing,” *International Journal of Computer Applications*, vol. 20, no. 2, pp. 43–49, 2011.

[37] И. А. Чернов, Е. Е. Ивашко, Н. Н. Никитина, “Обзор методов планирования заданий в Desktop Grid,” *Програмные системы: теория и приложения*, ISSN 2079-3316, vol.3, no. 34, pp. 3–29, 2017.

[38] N. M. Durrani, J. A. Shamsi, “Volunteer computing: requirements, challenges, and solutions,” *Journal of Network and Computer Applications*, vol. 39, pp. 369–380, 2014.

[39] А. П. Афанасьев, И. В. Бычков, О. С. Заикин, М. О. Манзюк, М. А. Посыпкин, А. А. Семенов, “Концепция многозадачной грид-системы с гибким распределением свободных вычислительных ресурсов суперкомпьютеров,” *Известия Российской академии наук. Теория и системы управления*, no. 4, pp. 133–139, 2017.

[40] A. Afanasiev, Y. Evtushenko, M. Posypkin, “The layered software infrastructure for solving large-scale optimization problems on the grid,” *International Journal of Computer Research*, vol. 18, no. ¾, pp. 307, 2011.

[41] D. Thain, T. Tannenbaum, M. Livny, “Distributed computing in practice:the Condor experience,” *Concurrency-Practice and Experience*, vol. 17, no. 2–4, pp. 323–356, 2005.

[42] М. А. Посыпкин, В. А. Сухомлин, Н. П. Храпов, “Комбинированные распределенные инфраструктуры в науке и образовании,” *Современные информационные технологии и ИТ-образование*, vol. 1, no. 11, pp. 31–36, 2015.

[43] J. Rius, F. Cores, F. Solsona, “Cooperative scheduling mechanism for large-scale peer-to-peer computing systems,” *Journal of Network and Computer Applications*, vol. 36, no. 6, pp. 1620–1631, 2013.

[44] D. P. Anderson, “BOINC: A system for public-resource computing and storage,” *Proc. - IEEE/ACM Int. Work. Grid Comput.*, pp. 4–10, 2004, doi: 10.1109/GRID.2004.14.

[45] G. Fedak, C. Germain, V. Neri and F. Cappello, “Xtremweb: A generic global computing system,” *In Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2001)*, IEEE, pp. 582–587, 2001.

[46] A. Chien, B. Calder, S. Elbert K. and Bhatia, “Entropia: architecture and performance of an enterprise desktop grid system,” *Journal of Parallel and Distributed Computing*, vol. 63, no. 5, pp. 597–610, 2003.

[47] Т. А. Узденов, “Планування обчислень в Desktop GRID,” *Матеріали І міжнародної спеціалізованої наукової конференції «Актуальні питання механічної та електричної інженерії, транспортних технологій, електроніки, автоматизації та ІТ»*, МЦНД, pp. 35–36, 2021.

[48] “Calculations for Science.” https://boinc.berkeley.edu/ (accessed 12.02, 2021).

[49] F. Xhafa, A. Abraham, “Computational models and heuristic methods for grid scheduling problems,” *Future Generation Computer Systems*, vol. 26, no. 4, pp. 608–621, 2010.

[50] H. Casanova, F. Dufoss´e, Y. Robert, F. Vivien, “Scheduling parallel iterative applications on volatile resources,” *2011 IEEE International Parallel & Distributed Processing Symposium*, IEEE, pp. 1012–1023, 2011.

[51] M. Maheswaran, Sh. Ali, H. J. Siegel, D. Hensgen and R. F. Freund, “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, 1999.

[52] Т. А. Узденов, “Алгоритми диспетчеризації для GRID систем з невідчужуваними ресурсами,” *Збірник матеріалів VIII Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2020»*, pp. 186–189, 2020.

[53] J. Yu, R. Buyya, “A taxonomy of scientific workflow systems for grid computing,” *SIGMOD Record, Special Issue on Scientific Workflows*, vol. 34, no. 3, pp. 44–49, 2005.

[54] C. S. Yeo, R. Buyya, “A taxonomy of market-based resource management systems for utility-driven cluster computing,” *Software: Practice and Experience*, vol. 36, no. 13, pp. 1381–1419, 2006.

[55] K. Krauter, R. Buyya, M. Maheswaran, “A taxonomy and survey of grid resource management systems for distributed computing,” *Software: Practice and Experience*, vol. 32, no. 2, pp. 135–164, 2002.

[56] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, “Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems,” *In Heterogeneous Computing Workshop (HCW ’99) Proceedings. Eighth*, pp. 30–44, 1999.

[57] T. D. Braun *et al.*, “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,” *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001, doi: 10.1006/jpdc.2000.1714.

[58] H. Casanova, A. Legrand, D. Zagorodnov, F. Berman, “Heuristics for scheduling parameter sweep applications in grid environments,” *In Proceedings of the 9th Heterogeneous Computing Workshop*, IEEE CS Press, Cancun, Mexico, pp. 349–363, 2000.

[59] S. Choi, H. Kim, E. Byun, and C. Hwang, “A Taxonomy of Desktop Grid Systems Focusing on Scheduling,” *Technical Report: KU-CSE-2006-1120-02*, Department of Computer Science and Engeering, Korea University, 2006.

[60] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, “Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids,” [Online], http://arxiv.org/abs/cs/0402017 (accessed 12.02, 2021).

[61] H. Zhao, X. Liu, X. Li, “A taxonomy of peer-to-peer desktop grid paradigms,” *Cluster Computing*, vol. 14, no. 2, pp. 129–144, 2011, doi: 10.1007 /s10586-010-0138-3.

[62] A. Luther, R. Buyya, R. Ranjan, S. Venugopal, “Alchemi: A .net-based enterprise grid computing system,” *In Proceedings of the 6th International conference on Internet Computing (ICOMP’05)*, Las Vegas, USA, pp. 269–278, 2005.

[63] L. F. G. Sarmenta, S. Hirano, “Bayanihan: Building and studying web-based volunteer computing systems using Java,” *Future Generation Computer Systems*, vol. 15, no. 5, pp. 675-686, 1999.

[64] O. Nov, D. Anderson, and O. Arazy, “Volunteer computing,” p. 741, 2010, doi: 10.1145/1772690.1772766.

[65] L. F. Sarmenta, “Sabotage-tolerance mechanisms for volunteer computing systems,’ *Future Generation Computer Systems*, vol. 18, no. 4, pp. 561–572, 2002.

[66] L. F. G. Sarmenta *et al.*, “Bayanihan computing.NET: Grid computing with XML web services,” *2nd IEEE/ACM Int. Symp. Clust. Comput. Grid, CCGrid 2002*, pp. 1–2, 2002, doi: 10.1109/CCGRID.2002.1017182.

[67] “BOINC.” http://boinc.berkeley.edu/ (accessed 12.02, 2021).

[68] M. Taufer, D. Anderson, P. Cicotti, and C. L. Brooks, “Homogeneous redundancy: A technique to ensure integrity of molecular simulation results using public computing,” *Proc. - 19th IEEE Int. Parallel Distrib. Process. Symp. IPDPS 2005*, 2005, doi: 10.1109/IPDPS.2005.247.

[69] D. P. Anderson, E. Korpela, R. Walton, “High-performance task distribution for volunteer computing,” *In Proceedings of the First IEEE International Conference on e-Science and Grid Technologies (e-Science2005)*, IEEE CS Press, Melbourne, Australia, pp. 196–203, 2005.

[70] D. P. Anderson, “Boinc: A system for public-resource computing and storage,” *In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID’04)*, IEEE CS Press, Pittsburgh, USA, pp. 4–10, 2004.

[71] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao, “Cluster computing on the fly: P2P scheduling of idle cycles in the internet,” *Lect. Notes Comput. Sci.*, vol. 3279, pp. 227–236, 2004, doi: 10.1007/978-3-540-30183-7_22.

[72] D. Zhou and V. Lo, “Cluster Computing on the Fly: Resource discovery in a cycle sharing peer-to-peer system,” *2004 IEEE Int. Symp. Clust. Comput. Grid, CCGrid 2004*, pp. 66–73, 2004, doi: 10.1109/ccgrid.2004.1336550.

[73] “Welcome to WaveGrid.” https://www.wavegrid.net/ (accessed 12.02, 2021).

[74] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, “A scalable content addressable network,” *In Proceedings of the ACM SIGCOMM 2001 Technical Conference*, August 2001.

[75] D. Zhou and V. Lo, “Wave grid: A scalable fast-turnaround heterogeneous peer-based desktop,” *20th Int. Parallel Distrib. Process. Symp. IPDPS 2006*, vol. 2006, 2006, doi: 10.1109/IPDPS.2006.1639267.

[76] J. S. Kim *et al.*, “Creating a robust desktop grid using peer-to-peer services,” *Proc. - 21st Int. Parallel Distrib. Process. Symp. IPDPS 2007*, pp. 1–7, 2007, doi: 10.1109/IPDPS.2007.370505.

[77] W. Saad, H. Abbes, C. Cerin, and M. Jemni, “A data prefetching model for desktop grids and the condor use case,” *Proc. - 12th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2013*, pp. 1065–1072, 2013, doi: 10.1109/TrustCom.2013.130.

[78] D. Thain, T. Tannenbaum, M. Livny, “Condor and the grid. In Grid Computing: Making the Global Infrastructure a Reality,” *Wiley*, Chichester, West Susses, p.1060, 2003.

[79] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice the condor experience,” *Computer Sciences Department, University of*

Wisconsin-Madison 1210 West Dayton Street, Madison WI 53706, vol. 17, no. 2-4, pp. 323–356, 2004.

[80] "XtremWeb : the Open Source Platform for Desktop Grids." http://xtremweb.gforge.inria.fr/ (accessed 12.02, 2021).

[81] F. Cappello *et al.*, "Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid," *Futur. Gener. Comput. Syst.*, vol. 21, no. 3, pp. 417–437, 2005, doi: 10.1016/j.future.2004.04.011.

[82] H. He *et al.*, "Extending the EGEE Grid with XtremWeb-HEP Desktop Grids," *CCGrid 2010 - 10th IEEE/ACM Int. Conf. Clust. Cloud, Grid Comput.*, pp. 685–690, 2010, doi: 10.1109/CCGRID.2010.100.

[83] "Understanding the Technology Behind the iEx.ec Distributed Cloud." https://medium.com/iex-ec/understanding-the-technology-behind-the-iex-ec-distributed-cloud-d91965fff00a (accessed 12.02, 2021).

[84] "HTCondor – Home." https://research.cs.wisc.edu/htcondor/ (accessed 12.02, 2021).

[85] "HTCondor Version 8.9.13 Manual; HTCondor Manual 8.9.13 documentation." https://htcondor.readthedocs.io/en/latest/ (accessed 12.02, 2021).

[86] "Aalto Linux - Aalto scientific computing." https://scicomp.aalto.fi/aalto/linux/ (accessed 12.02, 2021).

[87] "Aalto Linux - Aalto Linux Guide." https://linux.aalto.fi/aalto/ (accessed 12.02, 2021).

[88] "HTCondor – Aalto scientific computing." https://scicomp.aalto.fi/aalto/htcondor/ (accessed 12.02, 2021).

[89] P. Kacsuk, J. Kovacs, Z. Farkas, A. C. Marosi, G. Gombas, and Z. Balaton, "SZTAKI Desktop Grid (SZDG): A flexible and scalable desktop grid system," *J. Grid Comput.*, vol. 7, no. 4, pp. 439–461, 2009, doi: 10.1007/s10723-009-9139-y.

[90] A. Baratloo, M. Karaul, Z. Kedem, P. Wijckoff, “Charlotte: Metacomputing on the web. Future Generation Computer Systems,” *Special Issue on Metacomputing*, vol. 15, no. 5-6, pp. 559–570, 1999.

[91] S. Vazhkudai, “Compute Power Market: Towards a Market-Oriented Grid,” *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*, IEEE, pp. 574–581, 2001, doi: 10.1109/CCGRID.2001.923245.

[92] B. Calder, A. A. Chien, “The Entropia Virtual Machine for Desktop Grids,” *VEE’05, Chicago, Illinois, USA, June 11-12*, pp. 186–196, 2005.

[93] “PVM: Parallel Virtual Machine.” https://www.csm.ornl.gov/pvm/ (accessed 12.02, 2021).

[94] “distributed.net: distributed.net.” https://www.distributed.net/Main_Page (accessed 12.02, 2021).

[95] A. J. Chakravarti, G. Baumgartner, M. Lauria., “The Organic Grid: SelfOrganizing Computation on a Peer-to-Peer Network,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 3, pp. 1-12, 2005.

[96] A. J. Chakravarti, G. Baumgartner, M. Lauria, “The Organic Grid: SelfOrganizing Computational Biology on Desktop Grids,” *Chapter 27 in Parallel Computing for Bioinformatics and Computational Biology: Models, Enabling Technologies, and Case Studies*, Wiley, 2006.

[97] O. Babaoglu, H. Meling, A. Montresor, “Anthill: a framework for the development of agent-based peer-to-peer systems,” *22nd International Conference on Distributed Computing Systems*, pp. 15-22, 2002.

[98] A. Montresor, H. Meling, O. Babaoglu, ”Messor: Load-Balancing through a Swarm of Autonomous Agents,” *International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2002)*, LNAI 2530, pp. 125-137, 2003.

[99] N. Cafferkey, P. D. Healy, D. A. Power, and J. P. Morrison, “Job Management in WebCom,” *Sixth International Symposium on Parallel and Distributed Computing (ISPDC’07)*, 2007, doi: 10.1109/ispdc.2007.34.

[100] T. Uzdenov, “A New Task Scheduling Algorithm for GRID Systems with Non-alienable Resources,” *Studies in Systems, Decision and Control*, vol 346, pp. 207–220, 2021, doi: 10.1007/978-3-030-69189-9_12.

[101] Д. В. Сивухин, “Общий курс физики,” *Наука*, vol. 1, pp. 78–88, 1979.0 с.

[102] T. A. Uzdenov, “Task scheduling in Desktop GRID by FSA method: a practical example,” *CEUR Workshop Proceedings*, vol. 2850, pp. 97–109, 2021, http://ceur-ws.org/Vol-2850.

[103] Т. А. Узденов, “Огляд програм-симуляторів для дослідження алгоритмів диспетчеризації для GRID-систем,” *Тези доповідей III Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення»*, Житомир: Житомирська політехніка, pp. 21–22, 2020.

[104] A. Varga, “A practical introduction to the OMNeT++ simulation framework,” *In EAI/Springer Innovations in Communication and Computing*, pp. 3-31, 2019, doi: 10.1007/978-3-030-12842-5_1.

[105] “Bricks.” http://ninf.is.titech.ac.jp/bricks/ (accessed 12.02, 2021).

[106] “GrADS - MicroGrid Toolkit.” http://www.hipersoft.rice.edu/grads /microgrid.htm (accessed 12.02, 2021).

[107] K. Taura and A. Chien, “Heuristic algorithm for mapping communicating tasks on heterogeneous resources,” *Proc. Heterog. Comput. Work. HCW*, pp. 102–115, 2000, doi: 10.1109/hcw.2000.843736.

[108] “SimGrid Home.” https://simgrid.org/ (accessed 12.02, 2021).

[109] “SimGrid: Getting Started: SimGrid Main Concepts.” http://simgrid.gforge.Inria.fr/simgrid/3.20/doc/getting_started.html (accessed 12.02, 2021).

[110] A. Gallardo, L. D. De Cerio, R. Messeguer, A. P. Isern-Deyà, and K. Sanjeevan, “GRID resource searching on the gridsim simulator,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes*

Bioinformatics), vol. 5544 LNCS, no. PART 1, pp. 357–366, 2009, doi: 10.1007/978-3-642-01970-8_35.

[111] "GridSim - REGATRON." https://www.regatron.com/product/overview/application-software/gridsim (accessed 12.02, 2021).

[112] A. Varga, "The OMNeT++ Discrete event simulation system," *In Proceedings of the European Simulation Multiconference (ESM'01)*, Prague, Czech Republic, pp. 6-9, June 2001.

[113] "Overview of a Performance Evaluation System for Global Computing Scheduling. Algorithms." http://ninf.apgrid.org/takefusa/bricks/contents/hpdc99. html (accessed 12.02, 2021).

[114] H. J. Song *et al.*, "The MicroGrid: a Scientific Tool for Modeling Computational Grids," vol. 00, no. c, pp. 53–53, 2015, doi: 10.1109/sc.2000.10028.

[115] "Simulating Algorithms — SimGrid documentation." https://simgrid.org/doc/latest/Tutorial_Algorithms.html (accessed 12.02, 2021).

[116] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurr. Comput. Pract. Exp.*, vol. 14, no. 13–15, pp. 1175–1220, 2002, doi: 10.1002/cpe.710.

[117] "SimBOINC." http://simboinc.gforge.inria.fr/ (accessed 12.02, 2021).

[118] Sebastio, M. Amoretti, and A. L. Lafuente, "AVOCLOUDY: A simulator of volunteer clouds," *Softw. - Pract. Exp.*, vol. 46, no. 1, pp. 3–30, 2016, doi: 10.1002/spe.2345.

[119] T. Estrada, M. Taufer, "Challenges in Designing Scheduling Policies in Volunteer Computing," I*n Desktop Grid Computing*, Chapman & Hall/CRC, pp. 167–190, 2012.

[120] T. Estrada, M. Taufer, K. Reed, and D. P. Anderson, "EmBOINC: An emulator for performance analysis of BOINC projects," *IPDPS 2009 - Proc. 2009 IEEE Int. Parallel Distrib. Process. Symp.*, no. Vc, 2009, doi: 10.1109/IPDPS.2009.5161135.

[121] T. Estrada, M. Taufer, and D. P. Anderson, “Performance prediction and analysis of BOINC projects: An empirical study with EmBOINC,” *J. Grid Comput.*, vol. 7, no. 4, pp. 537–554, 2009, doi: 10.1007/s10723-009-9126-3.

[122] K. Benson, T. Estrada, M. Taufer, J. Lawrence, and E. Cochran, “On the powerful use of simulations in the quake-catcher network to efficiently position low-cost earthquake sensors,” *Proc. - 2011 7th IEEE Int. Conf. eScience, eScience 2011*, pp. 77–84, 2011, doi: 10.1109/eScience.2011.19.

[123] “EmBoinc.” https://boinc.berkeley.edu/trac/wiki/EmBoinc (accessed 12.02, 2021).

[124] M. Taufer, A. Kerstens, T. Estrada, D. Flores, and P. Teller, “SimBA: A discrete event simulator for performance prediction of volunteer computing projects,” *in Principles of Advanced and Distributed Simulation, 21st International Workshop on*, vol. 7, pp. 189–197, 2007.

[125] S. Alonso-Monsalve, F. García-Carballeira, and A. Calderón, “ComBos: A complete simulator of Volunteer Computing and Desktop Grids,” *Simul. Model. Pract. Theory*, vol. 77, pp. 197–211, 2017, doi: 10.1016/j.simpat.2017.06.002.

[126] D. Klusacek, L. Matyska, H. Rudova, “Alea — Grid scheduling simulation environment,” *In 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*, vol. 4967, pp. 1029–1038, 2008.

[127] E. E. Ivashko, “Dynamic forecasting of the completion time of a computational experiment in a Desktop Grid,” *Proc. Inst. Syst. Program. RAS*, vol. 31, no. 5, pp. 183–190, 2019, doi: 10.15514/ispras-2019-31(5)-14.

[128] “About Python™ | Python.org.” https://www.python.org/ (accessed 12.02, 2021).

[129] E. Ivashko, I. Chernov, N. Nikitina, “A survey of desktop grid scheduling.,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2882–2895, 2018,. https://doi.org/10.1109/TPDS.2018.2850004

[130] Т. А. Узденов, “Симулятор процесу диспетчеризації задач в GRID-системах з невідчужуваними ресурсами,” *Електронне моделювання*, vol. 43, no. 1, pp. 87–97, 2021.

[131] “Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio.” https://visualstudio.microsoft.com/ (accessed 12.02, 2021).

[132] “What is WPF? — Visual Studio | Microsoft Docs.” https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019 (accessed 12.02, 2021).

[133] “Home: The Official Microsoft IIS Site.” https://www.iis.net/ (accessed 12.02, 2021).

[134] “Що таке XML / Хабр.” https://habr.com/ru/post/524288/ (accessed 12.02, 2021).

[135] “Microsoft Silverlight.” https://www.microsoft.com/silverlight/ (accessed 12.02, 2021).

[136] “What Is Windows Communication Foundation - WCF | Microsoft Docs.” https://docs.microsoft.com/dotnet/framework/wcf/whats-wcf (accessed 12.02, 2021).

[137] О. М. Данильченко, Т. А. Узденов, “Моделювання програмного комплексу для дослідження паралельних алгоритмів на кластерній системі,” *Тези науково-практичної міжвузівської конференції, 18-19 січня 2010 року*, Житомир: ЖДТУ, pp. 36-37, 2010.

[138] “Hello World app with WPF in C# - Visual Studio | Microsoft Docs.” https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-wpf?view=vs-2019 (accessed 12.02, 2021).

[139] “.NET | Free. Cross-platform. Open Source.” https://dotnet.microsoft.com/ (accessed 12.02, 2021).

[140] “XAML overview - WPF .NET | Microsoft Docs.” https://docs.microsoft.com/en-us/dotnet/desktop/wpf/fundamentals/xaml?view=netdesktop-5.0 (accessed 12.02, 2021).

[141] “Tutorial: Create a simple C# console app in Visual Studio.” https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-console?view=vs-2019 (accessed 12.02, 2021).

[142] “InstanceContextMode Enum (System.ServiceModel) | Microsoft Docs.” https://docs.microsoft.com/en-us/dotnet/api/system.servicemodel. instancecontextmode?view=netframework-4.8 (accessed 12.02, 2021).

[143] “Configuring Services Using Configuration Files - WCF | Microsoft Docs.” https://docs.microsoft.com/en-us/dotnet/framework/wcf/configuring-services-using-configuration-files (accessed 12.02, 2021).

[144] “How to initialize a dictionary with a collection initializer - C# Programming Guide | Microsoft Docs.” https://docs.microsoft.com/uk-ua/dotnet/csharp/ programming-guide/classes-and-structs/how-to-initialize-a-dictionary-with-a-collection-initializer (accessed 12.02, 2021).

[145] M. Kaur, “Multi-objective Evolution-Based Scheduling of Computational Intensive Applications in Grid Environment,” *Proceedings of the International Conference on Data Engineering and Communication Technology*, pp. 457–467, 2017, doi: 10.1007/978-981-10-1678-3_44.

[146] T. Aladwani, “Types of Task Scheduling Algorithms in Cloud Computing Environment,” *Sched. Probl. - New Appl. Trends*, pp. 1–12, 2020, doi: 10.5772/intechopen.86873.

[147] A. Kaur, “Different Task Scheduling Algorithms in Cloud Computing,” *Int. J. Latest Trends Eng. Technol.*, vol. 9, no. 3, pp. 217–223, 2018, doi: 10.21172/1.93.37.

[148] “PageSpeed Insights.” https://developers.google.com/speed/pagespeed/insights/ (accessed 12.02, 2021).

ДОДАТКИ

Додаток А. Програмний код компонента SGRIDAR-1 Client

```
public partial class MainWindow : Window, SelfHost.IServiceGRIDCallback
{
    private DispatcherTimer timer ;
    private DateTime end;
    public float[] powers { get; set; }
    bool isConnected = false;
    ServiceGRIDClient client;
    int ID;
    string nameClient = "PC";
    public float selectedPower = 1;
}
```

Додаток Б. Програмний код компонента SGRIDAR-1 Host

Інтерфейс IServiceGRID

```
[ServiceContract(CallbackContract = typeof(IServiceGRIDCallback))]
public interface IServiceGRID
{
    [OperationContract]
    int Connect(float power);
    [OperationContract]
    void Disconnect(int id);
    [OperationContract(IsOneWay = true)]
    void ChangePower(int id, float power);
    [OperationContract(IsOneWay = true)]
    void SendMsg(int timeWork, int id, int code);
    [OperationContract]
    List<PC> GetPcs();
}
```

Інтерфейс IServiceGRID

```
public interface IServiceGRIDCallback
{
    [OperationContract(IsOneWay = true)]
    void MsgCallback(int id, int code, int timeWork, List<PC> pcs);
}
```

Клас PC:

```
[DataContract] public class PC
{
    [DataMember]
    public int ID { get; set; }
    [DataMember]
```

```
        public float Power { get; set; }
        [DataMember]
        public bool status { get; set; }
        public OperationContext operationContext { get; set; }
}
```

Клас ServiceGRID

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
public class ServiceGRID : IServiceGRID
{
        List<PC> pcs = new List<PC>();
        PC server;
        int nextId = 1;
        public void ChangePower(int id, float power){…}
        public int Connect(float power) {…}
        public void Disconnect(int id) {…}
        public List<PC> GetPcs(){…}
        public void SendMsg(int timeWork, int id, int code) {…}
}
```

Додаток В. Програмний код компонента SGRIDAR-1 Server

Клас Worker:

```
public class Worker
{
    public int ID { get; set; }
    public float Power { get; set; }
    public bool status { get; set; }
    public Worker(int ID, float Power, bool status)
    {
        this.ID = ID;
        this.Power = Power;
        this.status = status;
    }
    public static float minPower(List<Worker> pcs) {…}
    public static int minPowerFreePC(List<Worker> pcs) {…}
    public static float maxPower(List<Worker> pcs) {…}
    public static int maxPowerFreePC(List<Worker> pcs) {…}
    public static float deltaPower(List<Worker> pcs) {…}
    public static float sumPower(List<Worker> pcs) {…}
    public static float averPower(List<Worker> pcs) {…}
    public void updateStatus(bool status) {…}
}
```

Клас Task:

```
public class Task : ICloneable
{
    public int ID { get; set; }
    public int task { get; set; }
```

```
        public bool status { get; set; }
        public float Ptask { get; set; }
        public int ETALON_P { get; set; }
        public int FCFS_P { get; set; }
        public int FSA_P { get; set; }
        public int ETALON { get; set; }
        public int FCFS { get; set; }
        public int FSA { get; set; }
        public int FSAMin { get; set; }
        public int FSAMax { get; set; }
        public Task(int ID, int task, bool status, float Ptask, int ETALON_P, int
FCFS_P, int FSA_P, int ETALON, int FCFS, int FSA, int FSAMin, int
FSAMax)
        {
            this.ID = ID;
            this.task = task;
            this.status = status;
            this.Ptask = Ptask;
            this.ETALON_P = ETALON_P;
            this.FCFS_P = FCFS_P;
            this.FSA_P = FSA_P;
            this.ETALON = ETALON;
            this.FCFS = FCFS; this.FSA= FSA;
            this.FSAMin = FSAMin;
            this.FSAMax = FSAMax;
        }
        public object Clone() {…}
        public static int minPowerTask(List<Task> tasks) {…}
        public static int maxPowerTask(List<Task> tasks) {…}
        public static int findTaskByPower(float power, List<Task> tasks) {…}
```

```
        public static void ChangeStatusTasks(List<Task> tasks) {…}
        public static void calcPowers(List<Task> tasks) {…}
        public static void reCalcPuwerTas(int task, List<Task> tasks) {…}
        public static int maxTask(List<Task> tasks) {…}
        public static int sumTasks(List<Task> tasks) {…}
        public static void generateTasks(int from, int to, int count, List<Task>
Tasks) {…}
 }
```

Клас Test:

```
 public class Test
 {
        public int ID { get; set;}
        public int sumTask { get; set; }
        public List<Task> listTasks;
        public List<Worker> listPC;
        public float powerFrom;
        public float powerTo;
        public int taskfrom;
        public int taskTo;
        public int ETALON_p { get; set; }
        public int FCFS_P { get; set; }
        public int FSA_P { get; set; }
        public int ETALON { get; set; }
        public int FCFS { get; set; }
        public int FSA { get; set; }
        public int FSAMin { get; set; }
        public int FSAMax { get; set; }
        public float sumPower { get; set; }
        public float averPower { get; set; }
```

```
        public float minPower { get; set; }
        public float maxPower { get; set; }
        public float deltaPower { get; set; }
        public Test(int ID, int sumTask, List<Task> listTasks, List<Worker>
listPC, float powerFrom, float powerTo, int taskfrom, int taskTo, int
ETALON_p, int FCFS_P, int FSA_P,int ETALON, int FCFS, int FSA, int
FSAMin, int FSAMax, float sumPower, float averPower, float minPower,
float maxPower, float deltaPower)
        {
        this.ID = ID;
        this.sumTask = sumTask;
        this.listTasks = listTasks;
        this.listPC = listPC;
        this.powerFrom = powerFrom;
        this.powerTo = powerTo;
        this.taskfrom = taskfrom;
        this.taskTo = taskTo;
        this.ETALON_p = ETALON_p;
        this.FCFS_P = FCFS_P;
        this.FSA_P = FSA_P;
        this.ETALON = ETALON;
        this.FCFS = FCFS;
        this.FSA = FSA;
        this.FSAMin = FSAMin;
        this.FSAMax = FSAMax;
        this.sumPower = sumPower;
        this.averPower = averPower;
        this.minPower = minPower;
        this.maxPower = maxPower;
        this.deltaPower = deltaPower;
```

```
        }
        public static void removeTestbyId(List<Test> tests, int id) {…}
    }
```

Клас MainWindow

```
    public partial class MainWindow : Window, SelfHost.IServiceGRIDCallback
    {
        ServiceGRIDClient server;
        bool isConnected = false;
        int ID;
        public float[] powers { get; set; }
         List<Worker> pcs = new List<Worker>();
        int WorkingPC;
        List<Test> Tests = new List<Test>();
        bool testIsRunning = false;
        public int countTest2 = 0;
        List<Task> Tasks = new List<Task>();
        public int idTask;
        private DateTime beginTask;
        private DateTime startWork;
        public string[] algoritms { get; set; }
        public int selectedAlgoritm;
        Dictionary<int, int> idTask_idPc = new Dictionary<int, int>();
        Dictionary<int, DateTime> task_timeSrart = new Dictionary<int, DateTime>();
        List<Task> awaighting_Tasks = new List<Task>();
        Control[] controls;
        public Dictionary<string, int> sumTimeAlgoritms { get; set; }
        private Excel.Application m_objExcel = null;
```

```
        private Excel.Workbooks m_objBooks = null;
        private Excel._Workbook m_objBook = null;
        private Excel.Sheets m_objSheets = null;
        private Excel._Worksheet m_objSheet = null;
        private Excel.Range m_objRange = null;
        private object m_objOpt = System.Reflection.Missing.Value;
        private object m_strSampleFolder = "D:\\GRID\\";
    }
```

Додаток Г. Довідка впровадження

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
Ministry of Education and Science of Ukraine, Zhytomyr Polytechnic State University

вул. Чуднівська,103, м. Житомир, 10005
103, Chudnivska Str., Zhytomyr, Ukraine, 10005
Phone/fax: (0412) 24-14-22, 24-14-23, e-mail: rector@ztu.edu.ua, https://ztu.edu.ua, код ЄДРПОУ 05407870

СИСТЕМА УПРАВЛІННЯ ЯКІСТЮ ВІДПОВІДАЄ ДСТУ ISO 9001:2015
QUALITY MANAGEMENT SYSTEM ISO 9001:2015

Від 16 березня 2021 № 44-01.00/736
На № _____ від _____

Довідка
про впровадження результатів дисертаційної роботи
Узденова Тараса Амуровича

Результати дисертаційної роботи Узденова Тараса Амуровича на тему: «Методи диспетчеризації завдань для GRID-систем з невідчужуваними ресурсами» впровадженні в освітній процес Державного університету «Житомирська політехніка» на кафедрі «Інженерії програмного забезпечення». Зокрема, результати дослідження використані для методичного матеріалу у навчальних дисциплінах: «Математичні методи дослідження операцій», «Розподілені та паралельні системи», а також при розробці навчальних планів і програм, при читанні лекцій, проведення семінарських та практичних занять.

Крім того, в освітній процес впроваджено програмний комплекс SGridAR-1, який розроблений Узденовим Т А. при виконанні дисертаційної роботи, як інструмент, що дозволяє наглядно демонструвати студентам роботу GRID-системи з невідчужуваними ресурсами.

Проректор з науково-педагогічної роботи
Державного університету
«Житомирська політехніка», к.т.н., доц. Андрій МОРОЗОВ

Додаток Д. Список публікацій здобувача за темою дисертації

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Uzdenov T. (2021) A New Task Scheduling Algorithm for GRID Systems with Non-alienable Resources. Studies in Systems, Decision and Control, vol 346, pp. 207-220. https://doi.org/10.1007/978-3-030-69189-9_12. (Scopus, ISSN 2198-4182)

2. Uzdenov T.A. (2021) Task scheduling in Desktop GRID by FSA method: a practical example. CEUR Workshop Proceedings, vol. 2850, pp. 97-109. http://ceur-ws.org/Vol-2850. (Scopus, ISSN 1613-0073)

3. Данильченко О.М. Моделі системи масового обслуговування для розрахунку навантаженості багатоагентних систем / О.М. Данильченко, Д.Г. Літвинчук, Т.А. Узденов // Вісник ЖДТУ №4(55) – ЖДТУ, 2010. – Т.ІІ. – С. 86-93.

4. Данильченко О.М. Моделі розрахунку трудомісткості операцій комунікації / О.М. Данильченко, Д.Г. Літвинчук, Т.А. Узденов // Моделювання та інформаційні технології, 2011. – Вип. 59. – С. 72-76.

5. Данильченко О.М. Процес диспетчеризації для GRID-системи з невідчужуваними ресурсами / О.М. Данильченко, Т.А. Узденов // Вісник ЖДТУ №2(61) – ЖДТУ, 2012. – Т.ІІ. – С. 147-154.

6. Узденов Т.А. Симулятор процесу диспетчеризації задач в GRID-системах з невідчужуваними ресурсами / Т.А. Узденов // Електронне моделювання. 2021. Т. 43. № 1, С. 87-97.

Наукові праці, які засвідчують апробацію матеріалів дисертації

7. Данильченко О.М. Моделювання програмного комплексу для дослідження паралельних алгоритмів на кластерній системі / О.М. Данильченко, Т.А. Узденов // Тези науково-практичної міжвузівської конференції, 18-19 січня 2010 року. Житомир: ЖДТУ, 2010. – С. 36-37.

8. Данильченко О.М. Проблеми розробки методів керування паралельними завданнями та їх алгоритмічної підтримки для актуальної форми GRID / О.М. Данильченко, Т.А. Узденов // Тези XXXVI науково-практичної міжвузівської конференції, присвяченої Дню науки, 12-13 травня 2011 року. Житомир: ЖДТУ, 2011. – Т.І. – С. 51-52.

9. Узденов Т.А. Модель процесу диспетчеризації для організації розподілених обчислень на GRID-системах з невідчужуваними ресурсами / Т.А. Узденов, В.О. Артемчук // IV Международная научная конференция «МОДЕЛИРОВАНИЕ-2012». Сборник трудов конференции, К., 2012. – С. 71-74.

10. Узденов Т.А. Алгоритми диспетчеризації для GRID систем з невідчужуваними ресурсами / Т.А. Узденов // Збірник матеріалів VIII Всеукраїнської науково-практичної конференції молодих вчених «Наукова молодь-2020» (Київ, 21 жовтня 2020 р.). – К.: ФОП Ямчинський О.В. – С. 186-189.

11. Узденов Т.А. Огляд програм-симуляторів для дослідження алгоритмів диспетчеризації для GRID-систем / Т.А. Узденов // Тези доповідей III Всеукраїнської науково-технічної конференції «Комп’ютерні технології: інновації, проблеми, рішення», м. Житомир, 26-27 листопада 2020 р. – Житомир: Житомирська політехніка, 2020. – С. 21-22.

12. Узденов Т.А. Планування обчислень в Desktop GRID / Т.А. Узденов // Матеріали І міжнародної спеціалізованої наукової конференції «Актуальні питання механічної та електричної інженерії, транспортних технологій, електроніки, автоматизації та ІТ». 5 березня 2021 р., Хмельницький – МЦНД – С. 35-36.