УДК 004.7

*О.П. Ігнатенко, П.А. Іваненко, О.Б. Синецький, О.В. Ніколенко*

# ІГРОВА МОДЕЛЬ ВЗАЄМОДІЇ КОРИСТУВАЧІВ У ГЕТЕРОГЕННИХ РОЗПОДІЛЕНИХ СЕРЕДОВИЩАХ

В даній роботі досліджується ігрова модель взаємодії користувачів, що виконують паралельні обчислення у гетерогенній багатопроцесорній системі. Запропонований підхід моделювання застосовується до задачі множення матриць з планувальником мін-мін. Дією користувачів у даному випадку є розмір блоку на яку розрізається матриця. Експериментально отримані характеристики системи були використані для налаштування імітаційної моделі, що дозволило виміряти оцінку часу завершення роботи для всіх можливих комбінацій розбиття задач по процесорам та побудувати поверхню залежності часу закінчення роботи для кожного користувача. Отримані результати були обґрунтовані і узагальнені на базі ігрового підходу, зокрема, показано існування точки рівноваги Неша для взаємодії двох користувачів та знайдені умови її Парето неефективності.

## Вступ

В даній роботі проведені дослідження з застосування методів теорії ігор для аналізу процесів планування у складних гетерогенних обчислювальних системах. На таких системах побудована, зокрема, технологія хмарних обчислень. Будемо розглядати проблему планування ресурсів за умов конкуренції між користувачами [1]. Хмарна система надає послуги великій кількості користувачів, при цьому її ресурси (пропускна здатність, буфери, потужність вузлів) обмежені. Користувачі (люди, програми, сервіси) є неоднорідними у своїх запитах і можуть діяти непередбачуваним чином. Необхідно розділити ресурси між конкуруючими користувачами ефективним та справедливим (максимально задовольняючи користувачів) чином.

Сучасні прикладні наукові задачі вимагають значних обчислювальних ресурсів, тому задача оптимізації виконання обчислювальних задач у багатопроцесорних середовищах займає чи не найперше місце в процесі розробки високопродуктивних програм. Ефективне виконання обчислень вимагає використання ефективних паралельних алгоритмів, швидкодія яких у свою чергу залежить від конкретного середовища виконання. Необхідність оптимізації виникає тому, що на етапі проектування застосування частіше за все інформація про середовище виконання відсутня. І навіть якщо конфігурація відома, то майже неможливо визначити таку конфігурацію програми, за якої обчислення будуть виконуватися з максимально ефективним використанням ресурсів системи. Також слід пам’ятати, що програма, оптимізована під конкретну обчислювальну платформу, не зможе показати максимальну швидкодію на будь-якій іншій відмінній платформі. Саме тому зазвичай усі релевантні для швидкодії параметри виносяться в деяку зовнішню конфігурацію, значення якої змінюють під час оптимізації. Складність такої оптимізації полягає у тому, що зв’язок між цими параметрами не є тривіальним, тому оптимізація майже завжди зводиться до повного перебору всіх можливих конфігурацій. За виключенням тривіальних задач кількість та область значень цих параметрів визначають множину всіх можливих конфігурацій значної потужності, що робить такий підхід неможливим без автоматизації. Також слід зауважити, що часто оптимальна конфігурація не є інтуїтивно очевидною. Можна відзначити, наприклад, роботи [2–5].

Відносно нова парадигма програмної автоматичної оптимізації (для зручності будемо використовувати скорочений термін «автотюнінг») вважається перспективним програмним підходом, який дозволяє науковцям та інженерам ефективно використовувати безперервно зростаючі обчислювальні потужності суперкомп’ютерів з точки зору витраченого часу й енергії [6]. Сфера її застосування простягається від наукових до загально-

цільових обчислень й не обмежується суперкомп'ютерами. Підклас застосувань, оптимізація яких може виконуватися автотюнерами, підпадають ресурсоємні прикладні програми, швидкість й ефективність виконання яких залежить від конфігурації обчислювального середовища (кількість процесорів, обсяг оперативної пам'яті, розмір кешу процесора та пристроїв зчитування інформації, пропускна здатність мережі тощо) й які мають механізми адаптації чи дозволяють легко їх інтегрувати. Фактично цей клас задач включає усі нетривіальні застосування для будь-яких середовищ – від платформ для розподілених обчислень до мобільних пристроїв. Автотюнери успішно розв'язують описану вище проблему, а саме: дозволяють створювати максимально ефективні в будь-яких обчислювальних середовищах застосування з мінімальним людським втручанням у процес оптимізації.

З розвитком та ускладненням хмарних сервісів виникає проблема побудови ефективних алгоритмів забезпечення їх безперешкодної роботи. Одним з перспективних напрямків розвитку є теоретико-ігровий підхід [4, 7, 8], який дозволяє побудувати аналітичні моделі та дослідити динаміку системи з багатьма конфліктуючими за ресурси користувачами.

Хмарні обчислення стали новим напрямком розвитку інформаційних технологій, об'єднавши у собі ідеї комп'ютерних мереж та обчислювальних дата центрів. Власне хмара – це можливість надавати сервіси користувачам через мережу Інтернет.

Таким чином, ключовим елементом роботи хмарної системи є ефективні алгоритми надання сервісів користувачам. Сучасна хмарна система існує у середовищі постійних змін. Змінюються технології, протоколи та програмне забезпечення. Змінюються користувачі, їх пріоритети, задачі і поведінка. Всі ці зміни є непередбачуваними. Тому теорія ігор, яка вже має історію успішного застосування до розв'язання задач маршрутизації, планування, керування потоками даних і перевантаженнями, є головним напрямком аналітичного моделювання хмарних систем.

Аналітичне моделювання дозволяє формалізувати роботу системи, поведінку користувачів і залучити їх у побудовану модель. Кожен гравець тут є автономним агентом, що прагне отримати певну частину обчислювального ресурсу. Припускаючи, що гравці діють раціонально – тобто максимізують свою функцію корисності можна здійснити аналіз отриманої гри. Ключовим елементом аналізу є пошук точки рівноваги та її характеристика. Рівновага – це одна з головних ідей теорії ігор. Буквально – рівноважний стан відповідає ситуації, коли гравці досягли максимуму своїх функцій корисності. В залежності від зовнішніх факторів можливі стійкі і нестійкі рівноваги. Взагалі, концепцій рівноваги в іграх існує більш ніж два десятки.

Найбільш відомою є рівновага за Нешем: множина стратегій гравців утворюють рівновагу за Нешем, якщо ніхто з них не може покращити свою функцію корисності змінюючи окремо свою стратегію. Іншими словами кожен гравець використовує найкращу можливу стратегію у відповідь на стратегії своїх опонентів.

## Задача множення матриць

В роботах [2, 5] була побудована аналітична модель алгоритму множення матриць у залежності від конкретної конфігурації обчислювальної системи з метою пошуку оптимального розв'язку.

В даній роботі будемо вважати, що обчислювальні вузли не зв'язані один з одним і не можуть обмінюватись напряму даними. Кожен вирішує свою задачу незалежно і повертає користувачу результат по закінченню. Нехай система складається з $m$ обчислювальних елементів та кожен з них характеризується швидкістю роботи $p_i$, $i = 1,...,m$ – тобто кількістю операцій з плаваючою точкою за секунду, які він може здійснити. Далі у статті будемо вважати, що процесори впорядковані за зменшенням потужності. Обчислювачі з'єднані лініями зв'язку з планувальником, який передає задачі та приймає від них результат. Будемо вважати, що лінії зв'язку ідентичні та мають швидкість передачі даних $q$.

В роботі [2] описаний паралельний алгоритм, який виконує множення матриць за найкоротший час для довільної кількості неоднорідних процесорів. Нагадаємо основні результати.

Будемо вважати, що виконуються наступні припущення:

1) всі процесори починають роботу одночасно;

2) планувальник здійснює призначення миттєво.

Нехай задані дві квадратні матриці розмірністю $N \times N$, результат множення яких необхідно обчислити. При використанні блочного алгоритму користувач задає розмір блоку $n$, який визначає елементарні задачі, які будуть виконуватися в обчислювальній системі.

Таким чином, алгоритм сформує $k = \frac{N^2}{n^2}$ задач, кожна з яких буде мати складність $O(n^2)$. Припустимо, що планувальник забезпечує пересилку повідомлень на вузли за певним фіксованим детерміністичним алгоритмом, який завершує обчислення за час $T(N.n)$. Тоді задача користувача полягає у пошуку мінімуму функції:

$$T(N,n) \to \min .$$

Функція $T(N,n)$, взагалі кажучи, може мати багато локальних мінімумів (в залежності від обчислювальної системи та планувальника) оскільки існує мінімальна фіксована величина задачі. При перепланування блоку з одного процесора на інший відбувається дискретна зміна часу виконання. Тому задача визначення глобального мінімуму є складною.

Одним з розповсюджених підходів до аналізу таких задач полягає у дослідженні *потокової* моделі даного процесу.

Припустимо, що користувач вибрав вектор $x \in R^m$ з компонентами $x_i$, де $x_i \ge 0$, $x_i \le k$, $i = 1,...,m$, $\sum_{i=1}^{k} x_i = k$. Всі такі вектори утворюють множину $X(n)$. Кожен компонент вектора $x$ описує відсоток задач, призначених для виконання на $i$-му процесорі. Будемо брати до уваги тільки операції множення. Таке спрощення дозволяє у явному вигляді виписати функції часу. Загальний час закінчення залежить від $x$, та дорівнює

$$T(x, X(n)) = \max_{i=1,...,m} \left\{ \frac{x_i N n^2}{p_i} \right\}.$$

Потокова модель передбачає можливість розділення задачі на «шматочки» розміру $\varepsilon = Nn^2$, компонування з них підходящих підзадач та визначення загального часу при $\varepsilon \to 0$.

**Твердження 1** [2]**.** Мінімальний час закінчення обчислень (без пересилок) для потокової моделі з одним користувачем дорівнює

$$T = N^3 \left( \sum_{i=1}^{m} p_i \right)^{-1}.$$

Відповідно, користувач, для досягнення оптимального часу, має розділити свою задачу таким чином, щоб вузол $i$ отримав $p_i T$ обчислень.

Розглянемо в даній роботі більш загальний підхід, який може бути корисним при перенесенні результатів на більш складні системи.

Функція Мінковського для множини $X$ та вектора $p \in R^m$ визначається наступним чином:

$$\mu_X(p) = \inf\{\lambda > 0 : p \in \lambda X\}.$$

Відомо, що ця функція опукла для опуклої $X$.

Визначимо множину потужностей системи $R = \{r \in R^m : r_i \in [0, p_i]\}$ та масштабуємо її наступним чином:

$$R(n) = \frac{1}{Nn^2} R.$$

Тоді $T(x, X(n)) = \mu_{R(n)}(x)$.

*Доведення.* Розглянемо праву частину: $\mu_{R(n)}(x) = \inf\{\lambda > 0 : x \in \lambda R(n)\}$. Умова належності вектора $x$ множині $R$ записується у вигляді

$$\max_{i=1,\dots,m}\left\{\frac{x_i}{p_i}\right\}=1,$$

отже

$$\mu_{R(n)}(x)=\inf\left\{\lambda>0:\max_{i=1,\dots,m}\left\{\frac{x_i}{p_i}\right\}=\frac{\lambda}{Nn^2}\right\}.$$

У іншому вигляді

$$\lambda=\max_{i=1,\dots,m}\left\{\frac{x_i Nn^2}{p_i}\right\}.$$

З властивостей функції $\mu_{R(n)}(x)$ випливає, що мінімальний час

$$T_{\min}=\min_{x\in X(n)}\mu_{R(n)}(x)$$

існує і єдиний.

Для врахування пересилок потрібно зазначити, що алгоритм надсилає $2x_iNn$ елементів на відповідний вузол та приймає $x_in^2$ елементів. Отже, сумарний час закінчення з урахуванням пересилок дорівнює

$$T_s(x,X(n))=\max_{i=1,\dots,m}\left\{\frac{x_i Nn^2}{p_i}+\frac{x_i(n^2+2Nn)}{q}\right\}.$$

**Твердження 2.** Існує мінімум часу по $x$ – $\min\limits_{x\in X(n)} T_s(x,X(n))$.

*Доведення.* Оскільки $x$ належить компактній множині, а функція $T_s(x)$ неперервна та опукла (максимум лінійних функцій), то мінімум існує.

**Дискретна модель обчислення множення матриць.** Нехай розмір блоку $n$ може бути тільки цілим числом, причому таким, щоб $k=\dfrac{N^2}{n^2}$ теж було цілим. Тоді мінімізація часу буде здійснюватись за скінченною множиною точок

$$Y(n)=\left\{\begin{matrix} y\in R^m : y_i\in\{0,1,..,k\},\\ \sum_i y_i=k, i=1,..,m \end{matrix}\right\}.$$

Зрозуміло, що $Y(n)\subset X(n)$.

Введемо поняття планувальника. Спочатку користувач вибирає розмір блоку $n$, в результаті чого формується множина $Y(n)$. Ця множина описує всі можливі розташування задач на процесорах.

Планувальник відповідає за вибір конкретного $y^*\in Y(n)$. В даній роботі ми розглянемо прості планувальники типу $extrextr$. Один з широко вживаних планувальників такого типу називається $\min\min$ і він вибирає розподіл за наступним алгоритмом:

1) формується черга з $k$ задач, кожна обсягом обчислень $Nn^2$;

2) вибирається задача з найменшим обсягом обчислень (в даній роботі розглядається випадок однакових задач, тому вибирається довільна);

3) задача надсилається на вільний процесор з найбільшою потужністю (тобто мінімізується час виконання), якщо вільних процесорів немає, чекаємо поки з'явиться;

4) якщо залишились задачі у черзі, то повертаємось до п. 2.

В результаті роботи алгоритму по-перше буде визначено вектор $y$, який описує яка кількість задач потрапила на який обчислювальний вузол, по-друге, буде визначено час закінчення останньої задачі. Для порівняння опишемо, також, планувальник $\max\max$:

1) формується черга з $k$ задач, кожна обсягом обчислень $Nn^2$;

2) вибирається задача з найбільшим обсягом обчислень. (в даній роботі розглядається випадок однакових задач, тому вибирається довільна);

3) задача надсилається на вільний процесор з найменшою потужністю;

4) якщо залишились задачі у черзі, то повертаємось до п. 2.

Для дослідження часу закінчення у дискретній моделі зафіксуємо розмір блоку $n$.

**Твердження 3.** Для будь-якого $n$ виконуються нерівності:

$$T_d(n)=\min_{y\in Y(n)}T(y,X(n))\ge\min_{x\in X(n)}T(x,X(n)),$$

$$T_{\min\min}(n)\ge T_d(n).$$

*Доведення.* Зрозуміло, що мінімум існує. Легко представити приклад неєди-

ності мінімуму $T_d(n)$ (два рівних процесори і розбиття, що не ділиться навпіл). Оскільки $Y(n) \subset X(n)$, то перша нерівність виконується.

Друга нерівність виконується, оскільки $T_d(n)$ – мінімум.

## Некооперативна ігрова модель обчислення множення матриць

Некооперативна гра описує ситуацію прийняття рішень гравцями за умов конфлікту інтересів. Під терміном гравець тут ми розуміємо користувача, який впливає на систему шляхом вибору стратегій – дій, які він може застосовувати. В залежності від дій його та інших учасників відбувається визначення виграшів гравців. Говорять, що гравець раціональний, якщо його дії спрямовані на максимізацію власного виграшу.

Якщо у грі приймають участь хоча б двоє учасників, можлива ситуація, коли перший гравець може покращити свій виграш за рахунок зменшення виграшу інших. В такому випадку кажуть про конфліктну взаємодію. Частинним випадком конфлікту є ситуація повністю протилежних інтересів – коли виграш одного є програшем іншого. Такі ігри називають антагоністичними або іграми з нульовою сумою.

Зауважимо, що некооперативність не означає, що гравці взагалі не кооперуються, а тільки те, що немає зовнішніх причин, які б спричиняли координацію або узгодження їх стратегії. Будь-яка кооперація що може виникнути має виходити зі структури гри і визначатися функціями корисності учасників.

Гру називають статичною, якщо гравці приймають свої рішення одноразово, незалежно один від одного. В певному розумінні, статична гра не залежить від часу. Навіть якщо гравці приймають рішення протягом певного часового інтервалу, якщо вони не володіють інформацією щодо дій інших гравців, така гра є статичною.

У динамічних іграх гравці отримують певну інформацію щодо рішень інших учасників і можуть змінювати свою стратегію у часі, тобто приймають рішення більше одного разу. Динамічні ігри є найбільш складним для аналізу і відіграють важливу роль у дослідженні процесів у мережах.

Опишемо задачу множення матриць у вигляді статичної некооперативної гри.

## Основні визначення

При визначенні статичних ігор загальновживаною є стратегічна або нормальна форма, яка включає опис трьох компонентів: множини гравців, їх стратегій і виграшів.

Гравцями в даному випадку являються користувачі $\{u_i\}_{i \in L}$, $L$ – множина індексів, які мають доступ до спільного ресурсу з $m$ обчислювальних елементів з швидкістю роботи $p_i$, $i = 1,...,m$. Будемо вважати, що для кожного користувача задані квадратні матриці розмірності $n_l$, $l \in L$. Тоді стратегіями користувачів є допустиме розбиття матриць на блоки $k_l \in K_l$, $l \in L$. Після розбиття матриць, блоки потрапляють у планувальник, який надсилає їх на процесори згідно з певним визначеним алгоритмом роботи. Часом закінчення обчислень $T_l$, $l \in L$ будемо називати час закінчення останньої задачі користувача $u_l$. Кожен користувач намагається зменшити свій час закінчення і через обмеженість спільних ресурсів виграш одного користувача спричинить програш інших.

Будемо вважати, що виконуються наступні припущення:

1) використовується планувальник Мін-мін;

2) вибрана множина можливих розмірів $\{n_j\}_{j=1,...,s}$ – стратегій користувачів, впорядкована за збільшенням;

3) якщо користувачі вибрали однакові розміри блоків, то їх час завершення однаковий і дорівнює подвоєному індивідуальному часу для даного розміру блоку;

4) існує єдиний мінімум $T_d(n_j)$, $j = 1,...,s$. Позначимо індекс стратегії, на якій він досягається $j^*$.

Побудуємо платіжну матрицю гри за наступними правилами. Нехай гравці вибрали стратегії $n_1, n_2$ відповідно. Позначимо їх виграші $T_1(n_1,n_2)$, $T_2(n_1,n_2)$.

1. Якщо $n_1 < n_2$, то виграш першого користувача дорівнює $T_d(n_1)$, другого $T_d(n_1)+T_d(n_2)$.

2. Якщо $n_1 > n_2$, то виграш першого користувача дорівнює $T_d(n_1)+T_d(n_2)$, другого $T_d(n_2)$.

3. Якщо $n_1 = n_2$, то виграш першого і другого користувача дорівнює $2T_d(n_1)$.

**Твердження 3.** Нехай виконуються нерівність $2T_d(n_{j^*}) \le T_d(n_{j^*-1})$, тоді пара стратегій $(n_{j^*}, n_{j^*})$ – рівновага Неша.

*Доведення.* Запишемо визначення рівноваги Неша в точці $(n_{j^*}, n_{j^*})$:

$$T_1(n_{j^*}, n_{j^*}) \le T_1(n_{j^*-1}, n_{j^*}),$$

$$T_2(n_{j^*}, n_{j^*}) \le T_2(n_{j^*}, n_{j^*-1}).$$

Застосуємо до цих нерівностей правила визначення виграшів:

$$T_1(n_{j^*}, n_{j^*}) = 2T_d(n_{j^*}),$$

$$T_1(n_{j-1}, n_{j^*}) = T_d(n_{j^*-1}).$$

Аналогічно для другого користувача. Виконання нерівності $2T_d(n_{j^*}) \le$ $\le T_d(n_{j^*-1})$, таким чином, гарантує рівновагу Неша.

**Твердження 4.** Нехай виконуються нерівність $2T_d(n_{j^*}) > T_d(n_{j^*-1})$, тоді рівновагою Неша будуть пара стратегій $(n_{j^*-1}, n_{j^*-1})$.

*Доведення.* Запишемо визначення рівноваги Неша в точці $(n_{j^*-1}, n_{j^*-1})$:

$$T_1(n_{j^*}, n_{j^*-1}) > T_1(n_{j^*-1}, n_{j^*-1}),$$

$$T_2(n_{j^*-1}, n_{j^*-1}) < T_2(n_{j^*-1}, n_{j^*}).$$

Запишемо виграші для першої нерівності:

$$T_1(n_{j^*}, n_{j^*-1}) = T_d(n_{j^*-1}) + T_d(n_{j^*}),$$

$$T_1(n_{j^*-1}, n_{j^*-1}) = 2T_d(n_{j^*-1}).$$

Оскільки $2T_d(n_{j^*}) > T_d(n_{j^*-1})$, то $T_1(n_{j^*}, n_{j^*-1}) > T_1(n_{j^*-1}, n_{j^*-1})$.

Для іншого користувача – аналогічно.

*Наслідок.* Отримана рівновага Парето неефективна.

$$T_1(n_{j^*}, n_{j^*}) < T_1(n_{j^*-1}, n_{j^*-1}),$$

$$T_2(n_{j^*}, n_{j^*}) < T_2(n_{j^*-1}, n_{j^*-1}).$$

Це досить характерна ситуація для ігор такого типу [4].

## Імітаційна модель

Імітаційне моделювання розбиття користувачами задач та їх виконання в розподіленому середовищі було проведено за допомогою програмного пакету CloudSim, призначеному для моделювання хмарних обчислень.

Фреймворк CloudSim було розроблено в Університеті Мельбурна як допоміжний засіб аналізу розподілених та хмарних середовищ, що міг би бути застосований як дослідниками, так і галузевими фахівцями. Можливості програмного пакету дозволяють визначати та задавати властивості таких сутностей як центр обробки даних (ЦОД, datacenter); вузол, що знаходиться в ЦОД, з визначенням параметрів швидкодії процесора та обсягу оперативної пам'яті; віртуальна машина, що запущена на вузлі ЦОД і використовує певну частину його ресурсів; та, нарешті, задачі для виконання (cloudlet), що характеризується обчислювальною складністю і обсягом даних, що пересилаються мережею.

В межах цього дослідження було розглянуто таку логічну схему взаємодії користувачів з хмарним середовищем (рис. 1).

Користувачі незалежно один від одного надсилають до хмарного середо-

вища задачі, що потребують обчислення (в даному дослідженні в якості таких задач було вибрано задачі множення матриць).

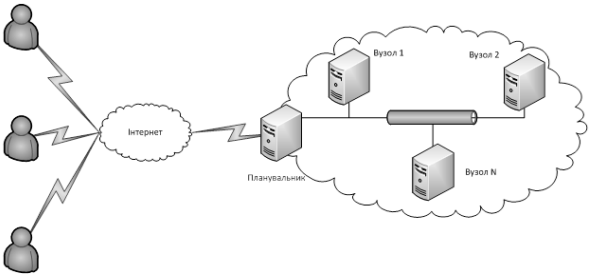


Рис. 1. Схема взаємодії користувачів і хмарної платформи

Кожен користувач перед відсилкою може розбити свою задачу на довільну кількість менших підзадач, що дозволить виконати обчислення паралельно. Всі користувацькі задачі надходять на зовнішній інтерфейс хмарного ЦОД і потрапляють на спеціальний вузол-планувальник, задачею якого є розподіл отриманих завдань між вузлами хмари відповідно до заданої політики планування. Отримавши свої підзадачі, вузли виконують обчислення і надсилають результати назад користувачам, які виконують об'єднання результатів обчислень.

Для побудови імітаційної моделі в програмному пакеті CloudSim було розроблено клас Scheduler, що представляє планувальник. Він містить дані про наявні задачі, що потребують обчислень, та вільні обчислювальні ресурси. Оперуючи такою інформацією, планувальник розподіляє задачі між вузлами відповідно до однієї з чотирьох змодельованих політик планування – Min-Min, Min-Max, Max-Min та Max-Max. Крім того, для коректного обчислення затримок, пов'язаних з пересилкою даних мережею було доопрацьовано механізм мережевої взаємодії, наявний в CloudSim таким чином, щоб він враховував будь-яку топологію локальної мережі хмарного ЦОД.

На побудованій імітаційній моделі було проведено серію експериментів для випадку одного користувача, що намагається мінімізувати свій час, змінюючи розбиття задачі множення двох матриць розмірністю 1200x1200. Отримана залежність часу виконання від кількості підзадач у розбитті показана на рис. 2.


Рис. 2. Графік залежності загального часу обчислень від кількості підзадач в розбитті

Крім того, на імітаційній моделі було проведено ряд експериментів для двох гравців з аналогічними умовами (матриці 1200x1200, змінні розбиття) та різними політиками планування для підтвердження отриманих теоретично результатів. Графік для політики Min-Min показано на рис. 3, і він демонструє добру узгодженість з теоретичними результатами.

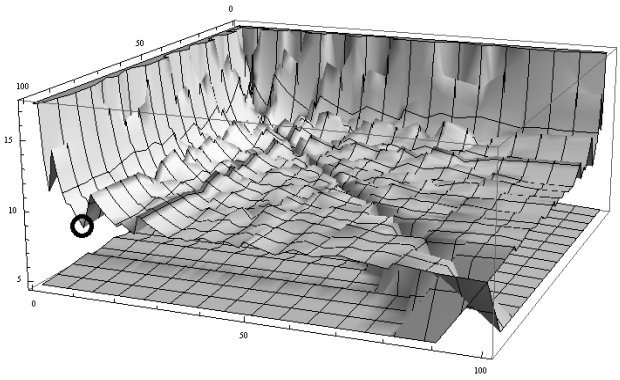

Рис. 3. Графік залежності часу від розбиття для двох гравців при політиці Min-Min (виділено точку мінімуму)

## Експериментальні дані

Для перевірки імітаційної моделі було створено сервісно-орієнтовану систему розподіленого виконання задач (рис. 4). Вона представляє собою декілька класичних REST [9] сервісів, які взаємодіють між собою через HTTP протокол. Розпочнемо її опис з розгляду діаграми компонентів.

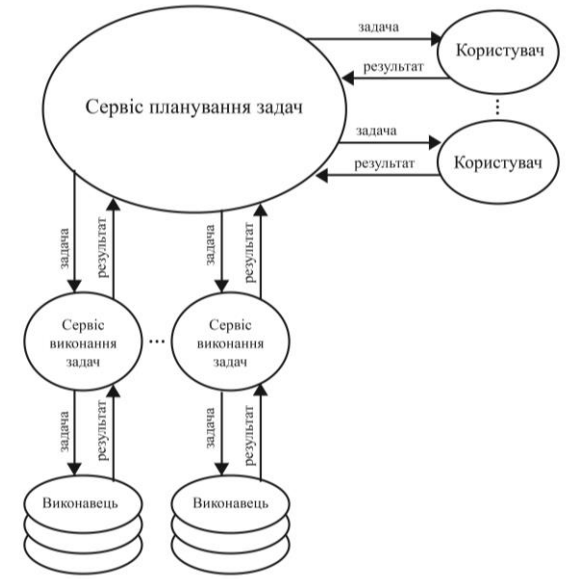


Рис. 4. Компоненти системи виконання задач

Користувачі надсилають свої задачі до сервісу планування (далі СП). В системі є лише один планувальник який упорядковує задачі згідно з обраною стратегією. Для експерименту було реалізовано чотири статичні стратегії планування: min-min, min-max, max-min та max-max [10]. Оскільки для даного дослідження було обрано задачу множення матриць, а саме її блочний алгоритм множення, то складність/вага задачі визначалася розміром блоку. У випадку коли обидва користувачі надсилали блоки однакової розмірності – порядок визначався випадково.

Далі задачі надсилаються до сервісів виконання (далі СВ). В системі таких сервісів може бути багато й кожен з них представляє один або декілька виконавців які знаходяться в одному середовищі й займаються безпосередньо обробкою задач. Усі виконавці в межах кожного середовища гомогенні й мають індекс продуктивності згідно з яким СП обирає якому виконавцю надіслати задачу. Відстеженням статусу виконавців займається сервіс планування. Під кожну задачу резервується окремий виконавець який вважається зайнятим доки СП не отримає результат задачі. Запити від СП обслуговуються асинхронно – СВ відповідає повідомленням зі статус-кодом 202 (Accepted) [11] одразу після пересилки задачі виконавцю. СП розсилає задачі тільки коли в системі є вільні виконавці. Якщо таких немає – задачі чекають на ресурсі в черзі СП.

СВ відповідає за надсилання результату обробки задачі виконавцем до СП. Як вже було вказано раніше – СП вивільняє виконавця тільки після того як отримає результат задачі від СВ. На цьому обробка задачі вважається завершеною й користувач може отримати її результат.

Усі сервіси були написані на Java 8 з використанням Spring Framework 4.1. Для повідомлень використовувався формат JSON.

До експерименту було залучено два середовища (рис. 5):

- один вузол кластера з двома процесорами Quad Core Intel Xeon E5405 2GHz й 16 GB DDR2 @667 Mhz RAM спільної пам’яті;
- ноутбук з процесором Intel Core i7 2.3 GHz (4 cores + HyperThreading) та 16 GB DDR3 @1600 MHz RAM.

Наступна діаграма пояснює розташування компонентів системи (для компактності на ній не зображено сервіси виконання).

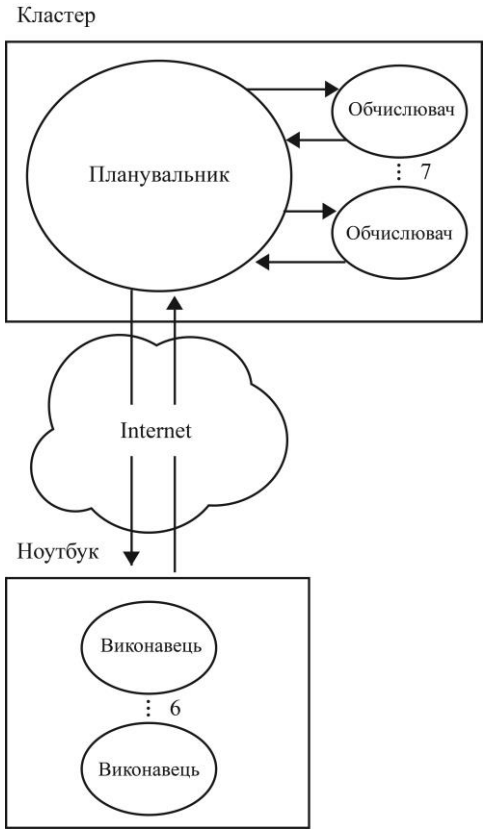


Рис. 5. Експериментальне середовище

Сервіс планування задач був розміщений на кластері разом з сервісом виконання з сімома виконавцями. На ноутбуці працював сервіс виконання й шість виконавців.

## Вплив декомпозиції задачі на час обчислень

Варто зауважити що розмір розбиття вихідної задачі користувачем впливає не лише на порядок виконання задач а й на час розрахунків. Разом з розміром задачі змінюються часові затрати на пересилання даних й загальний "чистий" час обчислень. Останнє твердження варто пояснити більш детально.

Для демонстрації розглянемо (рис. 6) залежність загального "чистого" часу множення двох квадратних матриць розміром 1200x1200 елементів від величини розбиття (вісь абсцис) у другому середовищі (ноутбук).

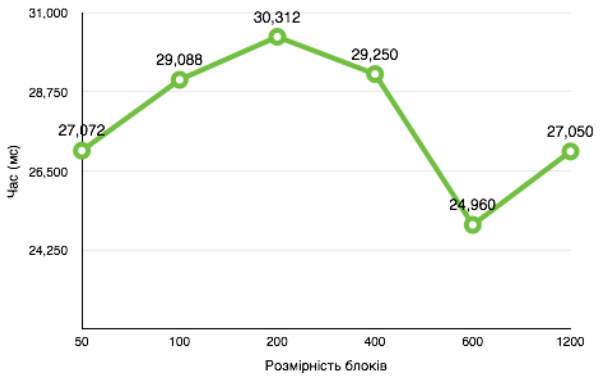


Рис. 6. Вплив декомпозиції

Для розбиття 400x400 загальна кількість блоків рівна дев'яти і тому "чистий" час розраховується як 9 * T400. Фактично це час, який потрібен на обрахування задачі на одному процесорі. Як видно з графіка – найкращий час маємо для розбиття на блоки розмірністю 600x600. За такого розбиття послідовне множення матриць буде виконуватися на 21% швидше ніж при розбитті на блоки по 200 елементів. Такий результат пояснюється тим що для обрахування такого блоку необхідно мати блок 1200x600 елементів першої матриці й 600x1200 з другої. Разом з результатом множення (600x600) це 1.8 млн. елементів. В експерименті множилися цілі числа типу Integer, який займає в Java 4 байта. Тому загальний обсяг необхідної пам'яті рівний ~6.86 МБ, що дуже близько до розміру L3 кеша (6 МБ). Тобто при множенні блоками 600x600 елементів усі данні майже повністю уміщуються в процесорний кеш, який значно швидший за оперативну пам'ять.

Слід зауважити що така декомпозиція не буде оптимальною за умови паралельного обчислення декількох блоків але ідея залишається незмінною – найкращий час буде у випадку з найбільш оптимальним використанням процесорного кешу.

## Висновки

В роботі розвивається новий підхід до аналізу хмарних обчислень, який полягає у теоретико-ігровому моделюванні системи планування та її взаємодії з користувачами. На прикладі задачі множення матриць була експериментально побудована гетерогенна кластерна система та реалізовано планувальник Мін-мін. Експериментально отримані характеристики системи були використані для налаштування імітаційної моделі, розробленої у середовищі моделювання CloudSim. Це дозволило виміряти оцінку часу завершення роботи для всіх можливих комбінацій розбиття задач по процесорам та побудувати поверхню залежності часу закінчення роботи для кожного користувача.

Отримані результати були обґрунтовані й узагальнені на базі теоретико-ігрового підходу, зокрема показано існування точки рівноваги Неша для взаємодії двох користувачів та знайдені умови її Парето неефективності.

1. *Wei, Guiyi, et al.* A game-theoretic method of fair resource allocation for cloud computing services // The journal of supercomputing – 2010, 54.2. – P. 252–269.
2. *Ігнатенко О.П., Парусімов Г.В., Синецький О.Б.* Одна модель виконання обчислень у гетерогенних розподілених середовищах // Проблеми програмування. – 2015. – № 1. – С. 15–24.
3. *Grosu D., Chronopoulos A.T.* A Game-Theoretic Model and Algorithm for Load Balancing in Distributed Systems,

Proceedings of IEEE IPDPS 2002, The 16th International Parallel and Distributed Processing Symposium, 4th Workshop on Advances in Parallel and Distributed Computational Models (APDCM'02), Fort Lauderdale, Florida, 15–19 April 2002. – P. 146–153.

4. *Kameda, Hisao, and Eitan Altman* Inefficient noncooperation in networking games of common-pool resources // Selected Areas in Communications, IEEE Journal on 26.7. – 2008. – P. 1260–1268.
5. *Дорошенко А.Ю., Ігнатенко О.П., Іваненко П.А.* Про одну модель оптимального розподілу ресурсів у багатопроцесорних середовищах // Проблеми програмування. – 2011. – № 1. – С. 21–28.
6. *Naono K., Teranishi K., Cavazos J., Suda R.* Software Automatic Tuning From Concepts to State-of-the-Art Results. Springer, 2010. 240 p.
7. *Андон Ф.И., Игнатенко А.П.* Моделирование конфликтных процессов в сети Интернет // Кибернетика и системный анализ. – 2013. – № 4. – С. 153–162.
8. *Ignatenko O., Synetskyi O.* Evolutionary Game of N Competing AIMD Connections // In Information and Communication Technologies in Education, Research, and Industrial Applications. Springer International Publishing. – 2014. – P. 325–342.
9. *Fielding R.T., Taylor R.N.* Principled Design of the Modern Web Architecture // In Proceedings of the 2000 International Conference on Software Engineering (ICSE 2000). Limerick, Ireland. – 2000. – С. 407–416.
10. *Magoules F., Pan J., Teng F.* Cloud Computing Data-Intensive Computing and Scheduling. Chapman & Hall/CRC. – 2012. – 231 p.
11. *Http* status code definitions: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Одержано 25.03.2015

***Про авторів:***

*Ігнатенко Олексій Петрович*,
кандидат фізико-математичних наук,
старший науковий співробітник,

*Іваненко Павло Андрійович*,
молодший науковий співробітник,

*Синецький Олександр Борисович*,
аспірант,

*Ніколенко Оксана Валеріївна,*
молодший науковий співробітник.

***Місце роботи авторів:***

Інститут програмних систем
НАН України,
03187, Київ-187,
Проспект Академіка Глушкова, 40.
Тел.: 526 6025.
E-mail: o.ignatenko@gmail.com