УДК 621.518

О.О. СУДАКОВ, Ю.В. БОЙКО, Т.В. НИЧИПОРУК, Т.П. КОРОТКОВА

ОПТИМІЗАЦІЯ ПРОДУКТИВНОСТІ ОБЧИСЛЮВАЛЬНОГО КЛАСТЕРА НА БАЗІ РОЗПОДІЛЕНИХ СЛАБКОЗВ'ЯЗАНИХ КОМПОНЕНТІВ

Abstract: *For solution of different time-consuming problems highy performance computing clusters are very popular. Each cluster is a set of several relatively low-speed computing nodes interconnected with relatively fast links for data transfer. Therefore performance of optimization of such clusters on is actual. In this work the task of global performance optimization of computing cluster is consitared. The algorythm for cluster optimization on the base of theoretical analysis is proposed. The proposed algorythm is confirmed by the results of experimantal measuremen.*
Key words: *kluster, optimization, process.*

Анотація: *Для вирішення різного роду задач, що вимагають великої кількості обчислювальних ресурсів і, зокрема, потребують тривалого часу обчислення, останнім часом використовуються високопродуктивні обчислювальні кластери. Кожен такий кластер являє собою набір з великої кількості (від десятка до декількох тисяч) відносно дешевих обчислювальних блоків, як правило, на базі одно- або багатопроцесорних PC, що з'єднані за допомогою відносно швидких каналів передачі даних. Тому важливою є задача оптимізації швидкості обрахунку з використанням кластерів на основі розподілених компонент. У даній роботі розглядається задача глобальної оптимізації продуктивності кластера з широкодоступних компонент при використанні програмного забезпечення, що вільно поширюється на основі теоретичного аналізу. Запропоновано алгоритм оптимізації кластера та проведено експериментальні вимірювання, які підтвердили результати теоретичного аналізу.*
Ключові слова: *кластер, оптимізація, процес.*

Аннотация: *Для решения разного рода задач, которые требуют большого количества вычислительных ресурсов и длительного времени вычисления, в последнее время используют высокопродуктивные вычислительные кластеры. Каждый такой кластер представляет собой набор большого количества (от десятка до нескольких тысяч) вычислительных блоков, как правило, на базе одно- или многопроцессорных PC, которые соединены при помощи относительно быстрых каналов передачи данных. Поэтому важной задачей является оптимизация скорости вычисления с использованием кластеров на основе распределенных компонент. В данной работе рассматривается задача глобальной оптимизации продуктивности кластера с широкодоступными компонентами при использовании программного обеспечения, которое свободно расширяется на основе теоретического анализа. Предложен алгоритм оптимизации кластера и проведены экспериментальные измерения, подтверждающие результаты теоретического анализа.*
Ключевые слова: *кластер, оптимизация, процесс.*

1. Вступ

Для вирішення задач, що вимагають великої кількості обчислювальних ресурсів і, зокрема, потребують тривалого часу обчислення, останнім часом використовуються високопродуктивні обчислювальні кластери на базі широкодоступних компонент [1, 2]. Кожен такий кластер являє собою набір з великої кількості (від десятка до декількох тисяч) відносно дешевих обчислювальних блоків, як правило, на базі одно- або багатопроцесорних PC, що з'єднані за допомогою відносно швидких каналів передачі даних (FastEthernet, Gigabit Ethernet, Myrinet, SCI та ін.). Звичайно, обчислювальна ефективність подібних систем (при співрозмірних сумарних потужностях процесорів) поступається векторно-конвеєрним суперкомп'ютерам (CRAY T90) чи системам з масовим паралелізмом (CRAY-T3E, IBM SP2). Проте співвідношення ефективність-ціна, доступність, можливість масштабування у таких систем надзвичайно високі. Тому важливою є задача оптимізації швидкості обрахунку з використанням кластерів на основі розподілених компонент. Існує цілий ряд робіт, присвячених оптимізації швидкості обчислень в подібних системах. Однак в більшості з них оптимізуються лише окремі параметри або системи заданої конфігурації. В даній роботі

розглядається задача глобальної оптимізації продуктивності кластера з широкодоступних компонент при використанні програмного забезпечення, що вільно поширюється.

2. Фактори обмеження продуктивності

Розглянемо просту модель. Нехай ми маємо програму, що повинна обробити кількість даних S. Також існує N процесорів (вузлів), що можуть працювати паралельно і обробити в одиницю часу дані в об'ємі ν_{cpu}. Нехай α – частина програми, яка може бути розпаралелена, тобто виконана одночасно на всіх N процесорах, тоді $1-\alpha$ – та частина програми, а це вимагає виконання на одному процесорі. Для паралельної обробки дані необхідно передавати між процесорами, а це вимагає також певного часу. В результаті кожного паралельного виконання $N-1$ процесор передає дані іншому фіксованому процесору. Прийом даних не може йти паралельно. Тому, оскільки об'єм даних, що передається з кожного вузла, рівний в середньому SIN, час передачі буде рівний $S(N-1)/(\nu_{net}N)$, де ν_{net} – кількість даних, що передається по мережі в одиницю часу. Перед початком прийому даних проходить деякий час, що визначається латентністю мережі. Також у процесі роботи програми можуть виконувати неперіодичні звертання до дискових накопичувачів, може відбуватись підкачка S-терму сторінок пам'яті з диску або на диск. Останні параметри важко виразити в термінах кількості даних або швидкості обробки, оскільки вони не передбачувані. Тому врахуємо їх за допомогою параметра t_{io}, що визначає час, затрачений на введення-виведення даних та латентність мережі. Таким чином, загальний час обрахунку T:

$$T = \frac{(1-\alpha)S}{\nu_{cpu}} + \alpha\frac{S}{N\nu_{cpu}} + \frac{S(N-1)}{N\nu_{net}} + t_{io}.$$

Розглянемо індекс продуктивності P, що розраховується за формулою $P = \frac{1}{T}$

$$P = \frac{N\nu_{cpu}/S}{(1-\alpha)N + t_{io}N\nu_{cpu}/S + (N-1)\nu_{cpu}/\nu_{net} + \alpha}. \qquad (1)$$

Таким чином, згідно з (1), найбільший вклад у зменшення продуктивності дають фактори, що ростуть пропорційно до кількості процесорів, оскільки при цьому значення знаменника буде збільшуватись найбільш швидко.

Розглянемо ідеальну систему, тобто таку, що має нескінченну швидкість взаємодії між процесорами і нульовий час затримки. Найсуттєвішим фактором, згідно з (1), є неефективність паралельного алгоритму. При нехтуванні всіма іншими факторами, крім даного, ми отримаємо так званий закон Амдала, тобто $P \sim \frac{1}{1-\alpha+\alpha/N}$, що відповідає значному зменшенню ефективності паралельної машини порівняно з послідовною, хоча при великих кількостях процесорів ефективність все ж таки буде більше 1. Таким чином, збільшення ефективності паралельного алгоритму завжди збільшує швидкість обрахунку.

Розглянемо ідеальний паралельний алгоритм, тобто

$$P = \frac{1}{t_{io} + S/\nu_{net} + S/(1/\nu_{cpu} - 1/\nu_{net})N}. \qquad (2)$$

З (2) видно, що при великій кількості процесорів продуктивність системи визначається тільки латентністю мережі передачі, швидкістю передачі даних та часом на введення – виведення даних. Слід зауважити, що мала швидкість передачі суттєво зменшує ефективність системи з великими об'ємами передачі даних. Навпаки, при малих об'ємах передачі даних потрібно мінімізувати затримку.

Суттєве збільшення t_{io} відповідає ситуаціям відсутності властивості пам'яті, коли відбувається інтенсивна підкачка сторінок з диску та на диск. При цьому значно падає продуктивність. Цікавим є факт, що у разі нестачі пам'яті ситуацію можна покращити збільшенням швидкості передачі мережею та передачею даних на інші процесори замість підкачки сторінок.

Швидкість процесорів має суттєвий вплив у випадку малого часу передачі по мережі та малих затримок, тобто для задач, що мало обмінюються даними по каналах зв'язку.

3. Шляхи підвищення швидкості

Розглянемо можливі підходи до підвищення швидкості обрахунку в рамках нашої моделі.

Нехай кожен процесор може обробляти дані порціями об'ємом S/M, де M – ціле число, запускати результат обробки на передачу мережею і приступати до обробки наступної порції даних. Тоді процес обробки, очевидно, може йти паралельно до процесу передачі, тобто існує можливість *конвеєризації*. Оцінимо час обрахунку. Час обрахунку паралельної частини програми буде складатись з обрахунку однієї порції даних, часу обрахунку другої частини даних паралельно до передачі та передачі частини даних, що лишились не переданими після закінчення обробки. Враховуючи, що швидкість обробки не менша, ніж швидкість передачі, отримаємо

$$T = \frac{(1-\alpha)S}{\nu_{cpu}} + \left\{\alpha\frac{S}{MN\nu_{cpu}}\right\} + \left\{\alpha\frac{S(M-1)}{MN\nu_{cpu}}\right\} + \left\{\frac{S}{N} - \alpha\frac{\nu_{cpu}}{\nu_{net}}\frac{S(M-1)}{MN\nu_{cpu}(N-1)}\right\}\frac{N-1}{N\nu_{net}} + t_{io} =$$
$$= \frac{(1-\alpha)S}{\nu_{cpu}} + \alpha\frac{S}{MN\nu_{cpu}} + \alpha\frac{S(N-1)}{N\nu_{net}} + t_{io}.$$

Індекс продуктивності в даному випадку буде збільшуватись при збільшенні M. Особливо актуальним збільшення M буде у випадку тривалих часів обрахунку і малих часів передачі даних по мережі. Практично дана ситуація відповідає збільшенню кількості процесів на паралельних процесорах, що веде до підвищення продуктивності при виконанні вказаних умов.

Розглянемо ще один випадок ідеально паралельного алгоритму та дві системи, які мають однакову сумарну швидкість процесорів, тобто $N_1\nu_{cpu1}/S = N_2\nu_{cpu2}/S = P_\Sigma$, всі інші параметри

також однакові. Знайдемо різницю коефіцієнтів продуктивності цих систем

$$P_1 - P_2 = \frac{Nv_{cpu1}/S}{t_{io}Nv_{cpu1}/S + (N-1)v_{cpu1}/v_{net} + 1} - \frac{Nv_{cpu2}/S}{t_{io}Nv_{cpu2}/S + (N-1)v_{cpu2}/v_{net} + 1} = \\ = P_\Sigma \frac{(Nt_{io}/S + (N-1)/v_{net})(v_{cpu2} - v_{cpu1})}{(t_{io}Nv_{cpu2}/S + (N-1)v_{cpu2}/v_{net} + 1)(t_{io}Nv_{cpu1}/S + (N-1)v_{cpu1}/v_{net} + 1)}. \quad (3)$$

Знаменник (3) – додатній, а знак чисельника залежить тільки від знаку різниць швидкостей процесорів. Тобто, при однаковій сумарній продуктивності процесорів система, яка має процесори *меншої* швидкості, буде продуктивнішою у випадку інтенсивного обміну даних мережею або при інтенсивному читанні даних з диску, як, наприклад, при підкачці сторінок. Даний, досить не звичний результат, відомий як закон Дедкова–Ідлайна [3]. Практично це означає, що коли при роботі програми відбувається інтенсивна передача даних мережею, інтенсивний обмін даних з диском, як наприклад при підкачці сторінок, то ефективним буде зменшення швидкості роботи процесорів при збільшенні їх кількості, що може бути реалізовано, наприклад, зменшенням пріоритету виконання таких програм у системі.

4. Розподіл та балансування навантаження

У розглянутій вище моделі процесори, що паралельно виконують задачу, вважались однаковими, швидкість каналів передачі та доступу до пам'яті також вважалась однаковою. Реальні системи можуть мати різну швидкість процесорів, каналів зв'язку та ін. за рахунок неоднорідності параметрів системи та неоднакової завантаженості вузлів, каналів, пам'яті. У цьому випадку швидкість роботи системи буде визначатись найповільнішими її компонентами.

Розглянемо задачу оптимізації подібної системи. Видно, що задачу можна звести до попередньої, якщо пронормувати кількість даних, що передаються на процесорний вузол, на швидкість процесорів відповідних вузлів та швидкість відповідних каналів зв'язку. Дана процедура називається розподілом навантаження. Сумарна продуктивність такої системи буде дорівнювати середній продуктивності вузла помноженій на кількість вузлів. Усі інші міркування лишаються аналогічними. При наявності процесорів та каналів зв'язку різної швидкості потрібно забезпечувати відношення v_{cpu}/v_{net}, по можливості однаковими для всіх вузлів кластера.

У процесі роботи системи задача розподілу завантаження вирішується запуском процесів на різних машинах за умови мінімуму суми квадратів завантажень всіх машин або простим рівномірним розподілом задач між вузлами кластера (системи MPI [4], PVM [5]).

Невиконання вказаних вище вимог щодо розподілу навантаження призводить до зменшення продуктивності системи. Крім того, завантаженість вузлів може змінюватись у процесі роботи системи за рахунок внутрішніх та зовнішніх факторів. У зв'язку з цим виникає проста ідея: для максимальної продуктивності роботи системи розподіл задач між вузлами потрібно змінювати динамічно – *балансування навантаження.* Розв'язанню цієї задачі присвячено ряд робіт, наприклад [6, 7].

Основна ідея розв'язку полягає в мінімізації цільових функцій, які залежать від значення ресурсів окремих вузлів кластера (об'єм пам'яті, швидкість процесорів і т.д.) та від їх використання.

Найбільш успішним практичним вирішенням даної задачі можна вважати систему MOSIX [8]. В даній системі всі вузла кластера періодично посилають інформацію про свої ресурси та їх використання випадково вибраним іншим вузлам. Для балансування використовується евристичний алгоритм, за яким кожен вузол кластера визначає необхідність балансування навантаження у випадку, коли існує вузол з меншим навантаженням. Для міграції вибирається процес, міграція якого призведе до зменшення часу обрахунку всіх задач у системі. Аналогічний алгоритм використовується для балансування використання пам'яті та збільшення ефективності вводу-виводу. Існують також інші більш ефективні алгоритми розподілу ресурсів, наприклад [6] .

Таким чином, балансування навантаження призводить до більш ефективного використання ресурсів системи, особливо у випадку сильного завантаження та задач, що вимагають тривалого часу обрахунку. Отже, систему з балансуванням навантаження в останньому випадку можна розглядати як кластер з однаковими компонентами, швидкість процесорів та швидкості каналів передачі якого є зваженим середнім швидкостей процесорів та каналів передачі відповідно.

5. Алгоритм оптимізації кластера

Враховуючи описані вище міркування, можна запропонувати такий підхід до оптимізації обчислювального кластера:

1. Настройка апаратного та програмного забезпечення вузлів кластера.

Даний пункт включає такі моменти, як підбір оптимальних параметрів частоти процесора, частоти шини, оптимізація ядра операційної системи під певний тип апаратури та ін.

2. Настройка каналів передачі.

Сюди можна віднести раціональне планування мережі передачі кластера, наприклад, приєднання кожного вузла кластера за допомогою каналу передачі, швидкість якого приблизно пропорційна продуктивності процесорів даного вузла; підбір конфігурації апаратного забезпечення кожного вузла та каналів передачі, а також підбір конфігурації програмного забезпечення операційної системи для забезпечення мінімальної латентності та максимальної швидкості передачі.

3. Настройка системи балансування навантаження.

При використанні системи балансування навантаження вона обов'язково повинна бути настроєна з урахуванням швидкостей процесорів окремих вузлів, швидкостей каналів передачі між вузлами та ін.

4. Оптимізація розрахункового програмного забезпечення для роботи на даному кластері.

Даний пункт включає застосування оптимальних параметрів програмного забезпечення, таких як розміри повідомлень, що передаються по мережі, оптимізація програмного забезпечення до типу процесора, параметрів операційної системи та ін.

5. Оптимізація параметрів запуску розрахункових задач.

До цього пункту належить визначення оптимальної кількості процесів виконання програми та пріоритетів виконання з урахуванням завантаженості кластера.

6. Тестування продуктивності системи.

6. Результати оптимізації продуктивності

Експериментальні результати отримано, використовуючи обчислювальний кластер Київського національного університету імені Тараса Шевченка, який на момент написання даної статті

складався з основної частини, що має 8 двохпроцесорних вузлів на базі процесорів Intel Pentium III з частотою 933 МГц (4 вузли) та Intel Pentium III з частотою 1ГГц (4 вузли) та додаткової частини, що має 48 вузлів на базі Intel Celeron з частотою 600 МГц. Вузли основної частини з'єднані з використанням мережі Gigabit Ethernet, вузли додаткової частини з'єднані між собою та з вузлами основної частини мережею Fast Ethernet. Вузли неосновної частини приєднуються до кластера динамічно. Кластер працює під операційною системою Linux на базі RedHat Linux – 7.1 з використанням ядра 2.4.18 та MOSIX – 1.6.0. На першому етапі проведено настройки вузлів кластера.

Другим етапом оптимізації системи було вимірювання та оптимізація продуктивності каналів передачі. Для цього використано програмне забезпечення NetPipe [9] для вимірювання швидкості передачі по мережі. При різних конфігураціях програмного забезпечення було виміряне швидкість передачі з використанням протоколу TCP. Найбільш суттєву залежність швидкість передачі має від розміру блока даних та параметрів затримки при конфігурації програмного забезпечення, поданих на рис. 1.

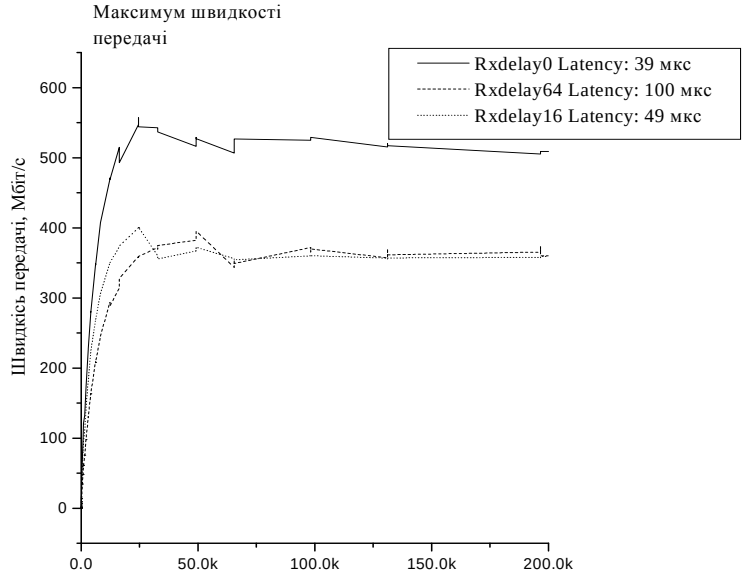

Рис. 1. Залежність максимуму швидкості передачі мережею кластера від розміру блока даних для різних значень параметра RxInitDelay, що визначає затримку посилання переривання після надходження пакета. Також показано латентність каналу передачі та залежність швидкості передачі від розміру блока в широких межах зміни розміру блока

Видно, що найбільшій швидкості передачі та найменшій латентності мережі відповідає найменша затримка переривання. При цьому, звичайно, ефективність використання процесора дещо знижується. При великих розмірах блока швидкість передачі мало залежить від розміру блока, проте при малих розмірах блока існує оптимальне значення. Максимальній швидкості передачі відповідає блок розміром 24756 байт, що обумовлюється апаратними особливостями системи. Максимальна швидкість передачі при виконанні даного тесту сягала 557 Мбіт/с для протоколу TCP, інші тести показали швидкість 609 Мбіт/с для протоколу TCP та 896 Мбіт/с для протоколу UDP.

NetPipe також дозволяє оцінити латентність каналу передачі шляхом екстраполяції залежності затримки від розміру блока при зменшенні розміру блока до нуля. Оптимізація латентності проводилась зміною розміру буфера пам'яті для сокета. Мінімальна латентність каналу передачі – 34 мкс, що відповідає розміру буфера сокета 131070 байт.

На третьому етапі було проведено настроювання системи міграції процесів та балансування навантаження MOSIX. Вузла кластера було розбито на 3 блоки, кожен з яких має схожі параметри апаратури, такі як частоти роботи процесорів та швидкість каналів передачі. Перший блок містить вузли, що мають процесори з частотою 1ГГц, другий блок – процесори з частотою 933 МГц, третій блок – усі вузли неосновної частини кластера. Для кожного блока даної топології за допомогою стандартних утиліт настроювання системи MOSIX вимірювались оптимальні зважувальні коефіцієнти, що використовуються для балансування навантаження.

Четвертий етап плану оптимізації стосується конкретних розрахункових програм, які мають свої особливості реалізації. Тому даний пункт може бути предметом для іншої роботи.

П'ятий пункт плану має на меті визначення оптимальної кількості процесів для вирішення даної задачі, а також їх пріоритету виконання. Звичайно даний пункт також є специфічним для різних задач.

Для тестування продуктивності системи було використано програмне забезпечення NAS Parallel Benchmarks [10], яке доступне в Інтернет за адресою http://www.nas.nasa.gov/NAS/NPB/. Дане програмне забезпечення містить 8 тестів: FT- тривимірне перетворення Фур'є, MG – вирішення тривимірного рівняння Пуассона методом сіток, LU, BT, SP – вирішення тривимірних рівнянь Нав'є-Стокера різними методами факторизації, CG – вирішення системи лінійних рівнянь методом спряжених градієнтів, EP – генерація псевдо-випадкових чисел з гаусовим розподілом, IS – сортування цілих чисел. У даній роботі використано тести класу A, які мають малий об'єм. Для тестів класів B та C, які мають відповідно середній та великий розмір, отримані аналогічні результати. Проте, очевидно, що час обрахунку цих задач буде більшим.

На рис. 2 показані результати вимірів для даного кластера при використанні різної кількості процесів виконання. Видно, що для більшості задач максимальна продуктивність отримана при використанні балансування навантаження, що підтверджує теоретичні міркування. Для тесту LU отримано найбільшу продуктивність при кількості процесів, вдвічі більшій, ніж кількість процесорів, що підтверджує можливість підвищення продуктивності за рахунок конвеєризації обрахунку та передачі даних по мережі. При значному збільшенні кількості процесів виконання продуктивність кластера падає.

Дані задачі вимагають інтенсивної передачі даних по мережі. У зв'язку з цим у результаті дії закону Дедкова–Ідлайна існує можливість прискорення роботи за рахунок зменшення пріоритету виконання процесів у системі.

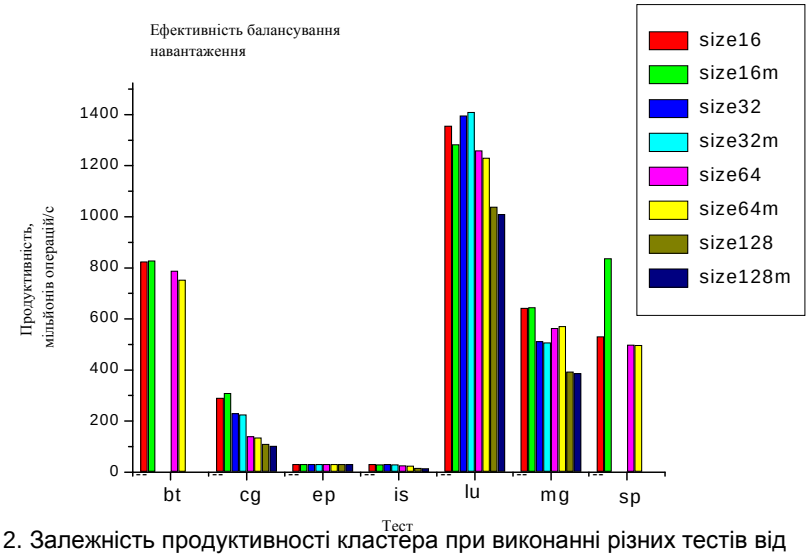

Рис. 2. Залежність продуктивності кластера при виконанні різних тестів від кількості запущених процесів (параметр size)

Приведені порівняльні результати при використанні системи MOSIX (параметри з індексом m) та без неї.

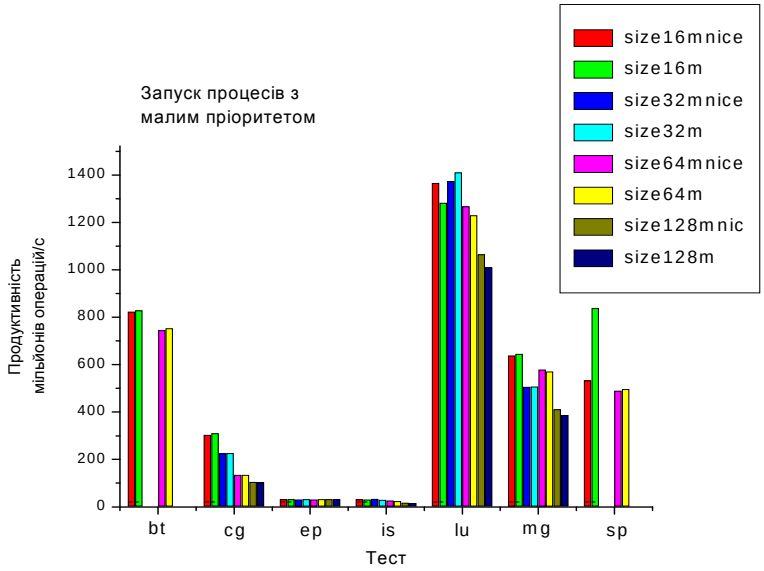

Рис. 3. Ефективність зниження пріоритетів виконання процесів (nice) для різних тестів при різній кількості процесів

Результати порівняння продуктивності при використанні процесів, що виконуються з нормальним пріоритетом та з малим пріоритетом (nice=20), наведено на рис. 3. Видно, що у випадку великої кількості процесів, які виконуються в системі зниження пріоритету, призводить до підвищення продуктивності системи для більшості тестів. Даний факт доцільно використовувати, коли паралельно обраховується велика кількість задач і при значних завантаженнях кластера.

Незважаючи на те, що підвищення продуктивності при зміні пріоритету чи зміні кількості процесів незначне, для задач, що вимагають тривалого часу обчислення, пришвидшення буде суттєвим.

6. Висновки

Проста модель, розглянута в даній роботі дозволяє запропонувати шляхи оптимізації обчислювального кластера, побудованого на базі розподіленого компонента, що підтверджено експериментально. Звичайно, дана модель не дає можливості знайти оптимальні параметри для будь-якої обчислювальної задачі, оскільки не враховує особливостей алгоритму та реалізації. У зв'язку з цим, для кожної окремої задачі доцільно проводити вимірювання, які з запропонованих шляхів дозволяють отримати максимальну продуктивність. Результати отримані з використанням кластера інформаційного обчислювального центру Київського національного університету імені Тараса Шевченка, за підтримки корпорації INTEL. В подальшому планується розробити модель, яка буде враховувати особливості алгоритму та реалізації для будь-якої задачі.

СПИСОК ЛІТЕРАТУРИ

1. Anderson T.E., Culler D.E., Patterson D.A. A case for NOW (Networks of Workstations) // IEEE Micro. – 1995. – Vol. 15, N 1. – P. 54 – 64.
2. Ridge D., Becker D., Merkey P., Sterling T. Beowulf: harnessing the power of parallelism in a pile-of-PCs // Proc. IEEE Aerospace. – 1997. – P. 67 – 73.
3. Dedkov A.F., Eadline D.J. Performance Considerations for I/O-Dominant Application Parallel Computers. ftp://ftp.plogic. com/plogic/papers/exs-pap6.ps
4. Pacheco P. Parallel Programming with MPI. Morgan Kaufmann Pub. Inc. – 1996. – P. 15 – 24.
5. PVM: Parallel Virtual Machine / Geist A., Beguelin A., Dongarra J., Jiang W., Manchek R., Sunderam V. – MIT – Press, 1994. – P. 24 – 32.
6. An Opportunity Cost Approach for Job Assignment in a Scalable .Computing Cluster / Amir Y., Awerbuch B., Barak A., Borgstrom R.S., Keren A. // IEEE Tran. Parallel and Distributed Systems. – 2000. – Vol. 11, N. 7. – P. 760 – 768.
7. On-Line MachineScheduling with Applications to Load Balancing and Virtual Circuit Routing / Aspnes J., Azar Y., Fiat A., Plotkin S., Waarts O. // Proc. of the ACM Symposium on Theory Of Computing (STOC). – 1993. – P. 32 – 41.
8. Barak A., La'adan O. The MOSIX Multicomputer Operating System for High Performance Cluster Computing // Journal of Future Generation Computer Systems. – 1998. – Vol. 13, N. 4–5. – P. 361 – 372.
9. Quinn O., Snell, Armin R. Mikler, John L. Gustafson. NetPIPE: A Network Protocol Independent Performance Evaluator. http://www.scl.ameslab.gov/netpipe/paper/full.html
10. RNR Technical Report RNR-94-007 / Bailey D., Barszcz E., Barton J., Browning D., Carter R., Dagum L., Fatoohi R., Fineberg S., Frederickson P., Lasinski T., Schreiber R., Simon H., Venkatakrishnan V., Weeratunga S. – 1994. – P. 47 – 56.