Аналіз ефективності планувальників черги задач для суперкомп'ютера з кластерною архітектурою

Головинський Андрій Леонідович, Маленко Андрій Леонідович

Інститут кібернетики ім. В.М. Глушкова НАН України, просп. Глушкова, 40, Київ, Україна

golovinsky.andriy@gmail.com, exipilis@yandex.ru

Анотація. *У роботі пропонується програмний стенд, який дозволяє швидко оцінити ефективність того чи іншого планувальника та порівняти різні планувальники на певному наборі задач. Вводяться індекси якості надання обчислювальних сервісів, які порівнюються на модельованих даних.*

Ключові слова

Паралельні обчислення, Менеджер ресурсів, SLURM, Torque.

1 Вступ

При проектуванні високопродуктивної обчислювальної системи постає питання вибору оптимального алгоритму планування черги обчислювальних задач, що найбільш ефективно розподіляє ресурси між задачами, які запускаються на даному кластері.

Зазвичай планувальник обирається розробником чи адміністратором кластера, виходячи з власного досвіду, досвіду користувачів або шляхом натурних випробувань різних планувальників на кластері. Щоб отримати статистично значимий результат натурного випробування, необхідно проводити його тривалий період часу. Нна кластері, який використовується реальними користувачами, відсутня можливість порівняти роботу планувальника на одному наборі задач, а тому складно застосувати формальний критерій порівняння, який залежить від конкретного набору задач. Крім того, результати порівнянь планувальників, отримані в ході натурних випробувань на одному кластері, можуть бути незастосовними на іншому.

Для розв'язання цієї проблеми авторами був створений стенд для моделювання поведінки суперкомп'ютера з кластерною архітектурою. На одному потужному обчислювальному вузлі стенд дозволяє моделювати до 1000 обчислювальних вузлів, а також пришвидшувати час виконання віртуальних задач в будь-яку кількість разів без суттєвої втрати точності. Стенд створений на основі технології віртуалізації, яка використовує незначну кількість процесорних ресурсів для роботи кожного віртуального вузла.

Вхідними даними для стенду є набір з n задач, які описуються трійками чисел $\{(q_i, r_i, c_i),\ i = 1, \ldots, n\}$, де q_i – час постановки задачі в чергу, r_i – час виконання, c_i – кількість вузлів, необхідних задачі для виконання. Такі дані на реальному кластері отримуються після аналізу статистики виконання задач, які надає планувальник кластера. У результаті роботи стенду отримуються моменти початку роботи кожної задачі $\{s_i,\ i = 1, \ldots, n\}$. Ці дані в подальшому використовуються для аналізу якості роботи планувальника.

Авторам вдалось змоделювати поведінку планувальників на кластері з піковою продуктивністю до 380 ТФлопс, використовуючи лише один потужний вузол (4 восьмиядерних процесора з частотою 3 ГГц), що еквівалентно продуктивності кластерів з 50-60 місць списку TOP-500 станом на червень 2011 року [1]. Для аналізу результатів експериментів створено спеціальне ПЗ.

Основні характеристики стенду:

- віртуалізація до 1000 віртуальних вузлів на одному реальному обчислювальному вузлі;
- додаткові реальні вузли дозволяють масштабувати стенд до 10000-50000 віртуальних вузлів;

- масштабування часу в 20-50 разів;
- можливість встановлення різних планувальників та їх порівняння на одному наборі задач;
- автоматичний збір та обробка результатів запусків віртуальних задач.

2 Індекси оцінювання якості сервісу

Якість надання сервісу з високопродуктивних обчислень можна визначати різними шляхами. Це може бути наявність різноманітного прикладного ПЗ, зручність керування власною чергою задач та файлами, швидкість обміну даними між кластером та терміналом користувача, надійність роботи кластера вцілому тощо. У даній роботі ми зосередимось лише на одному аспекті якості кластера, який вимірюється чисельно, – періоді очікування ресурсів.

Кластер з необмеженою кількістю вузлів повинен миттєво обробляти запит користувача на отримання ресурсів і одразу запускати задачу на виконання. Зрозуміло, що в реальній ситуації так не відбувається. Реальний кластер має систему черг задач та спеціальні алгоритми для керування чергами. Кожен алгоритм по-своєму визначає пріорітетність запуску кожної задачі.

Часто у користувачів виникає потреба контролювати хід виконання задач, особливо на початку запуску, щоб переконатись у відсутності помилок. У таких випадках час очікування задачі у черзі, з точки зору користувача, є втраченим марно. У даній роботі ми сформулювали індекси, які кількісно виражають середній час очікування ресурсів обчислювальними задачами. Ці індекси можна обчислювати та порівнювати для різних кластерів та планувальників на одному наборі задач.

При розгляді індексів ми виходили з наступних неформальних міркувань. "Зручність" сервісу в нашому розумінні полягає у деякому компромісі між часом, який прийнятно зачекати перед тим, як отримати дозвіл на виконання задачі, та самим обсягом отриманих ресурсів (часом виконання задачі та кількості вузлів). Важко вказати точну формулу, скільки має чекати користувач, і яка межа часу чекання є прийнятною. Наступні міркування є спробою отримати такий компроміс.

Розглянемо набір задач $\{(q_i, s_i, r_i, c_i),\ i = 1, \ldots, n\}$ за досліджуваний період $[0, T]$, які запускаються на N-вузловому кластері. Вважаємо, що час виконання задачі більший за одну часову одиницю. Це логічне припущення, оскільки на практиці основну масу ресурсів споживають задачі з великим часом виконання.

Введемо наступні індекси. Середній час очікування ресурсів:

$$W = \frac{1}{n}\sum_{i=1}^{n}(s_i - q_i),$$

середній час очікування, зважений на час виконання задачі:

$$W_1 = \frac{1}{n}\sum_{i=1}^{n}\frac{s_i - q_i}{r_i},$$

середній час очікування, зважений на порядок тривалості виконання:

$$W_2 = \frac{1}{n}\sum_{i=1}^{n}\frac{s_i - q_i}{\ln r_i}.$$

Усі індекси тою чи іншою мірою описують час очікування ресурсів. Менше значення індексу означає кращий сервіс.

Розглянемо спочатку індекс W – звичайне середнє арифметичне часу чекання. Недоліком цього індексу є те, що він не враховує, що ймовірність виділення ресурсів спадає при їх збільшенні. З іншого боку, для "довгих" задач, які обчислюються, наприклад, тиждень, очікування у одну або дві години не впливає на комфортність сприйняття сервісу.

Для більш коректного порівняння ми вводимо індекси W_1 та W_2, які показують кількість часу, яку очікує задача, нормовану часом її виконання або порядком тривалості.

Ідея введення індексів W_1 та W_2 проста. Вони позбавлені одного з недоліків індексу W, а саме: враховується довжина задачі. Але W_1 та W_2 не враховують розмір задачі. Наступними міркуваннями вводиться індекс, який буде містити і цей фактор. Для його обчислення потрібно отримати оцінку ймовірності звільнення необхідної кількості ресурсів.

Позначимо $\xi(t)$ – випадковий процес, що означає кількість зайнятих ресурсів у момент t, $\xi(t) \in \{0, \ldots, N\}$, де N – загальна кількість вузлів кластера. Процес $\xi(t)$ є кусково сталим з випадковими моментами стрибків. У момент запуску задачі процес збільшується на її розмір, у момент закінчення задачі – відповідно зменшується.

Для спрощення будемо вважати, що розподіл вільних ресурсів в кожен момент часу t не залежить від t. Позначимо $F(x)$ – функція розподілу зайнятих вузлів $\xi(t)$. Тоді ймовірність негайного запуску задачі на c вузлів дорівнює ймовірності, що кількість зайнятих вузлів не перевищує $N - c$, що в свою чергу рівне $F(N - c + 1)$. Ми не робили жодних припущень щодо F, а лише обчислювали емпіричну функцію розподілу $\hat{F}(x)$ на основі спостережень наступним чином.

На практиці час постановки задачі в чергу, час початку роботи та час закінчення дискретні, обчислюються в секундах. На інтервалі $[0, T]$ щосекунди виміряємо кількість зайнятих ресурсів. Це нескладно зробити на основі даних $\{(c_i, s_i, r_i),\ i = 1, \ldots, n\}$:

$$\xi(t) = \sum_{i=1}^{n} c_i \mathbb{I}_{t \in [s_i; s_i + r_i)}, \quad t \in 0, \ldots, T,$$

тут $\mathbb{I}$ означає індикатор. На основі вибірки $\{\xi(t),\ t \in \{0, \ldots, T\}\}$ отримуємо емпіричну функцію розподілу:

$$\hat{F}(x) = \frac{1}{T+1} \sum_{t=0}^{T} \mathbb{I}_{\xi(t) < x}.$$

Індекс

$$W_3 = \frac{1}{n} \sum_{i=1}^{n} \frac{(s_i - q_i)}{r_i} \hat{F}(N - c + 1)$$

означає середній час очікування, зважений на час виконання та ймовірність миттєвого отримання ресурсів для задачі даного розміру.

На практиці обчислення $\hat{F}(x)$ для великої кількості задач вимагає значних обчислень. Спрощено можна вважати ймовірність завантаження ресурсів рівномірною на множині $\{0, \ldots, N\}$ та розглядати індекс

$$W_4 = \frac{1}{n(N+1)} \sum_{i=1}^{n} \frac{(s_i - q_i)(N + 1 - c_i)}{r_i},$$

який простіше обчислюється.

Очевидно, що усі наведені вище індекси залежать від кластера, на якому виконуються задачі, набору цих задач та планувальника, який їх розподіляє. Тому коректне порівняння планувальника можливе лише на одному кластері для одного набору задач, який має бути досить великим та репрезентативним. В тестовий набір мають потрапляти основні типи задач за кількістю вузлів та часом виконання, які найчастіше запускаються користувачами даного кластера. Гарні результати порівняння також дають набори реальних задач, які запускались на кластері протягом значного періоду часу (кілька років).

3 Тестові результати

В ході експерименту ми порівнювали два популярних менеджери ресурсів Torque [2] та Slurm [3], причому Torque працював в режимі FIFO, а Slurm – в режимах Backfill та FIFO. Стенд моделював роботу кластера з 166 вузлів та 1328 ядер.

Алгоритм планування FIFO (First In First Out) забезпечує принцип простої черги, коли задачі отримують ресурси строго в тій послідовності, в якій вони з'явилися в черзі. Цей алгоритм підтримується планувальниками Torque та Slurm.

Алгоритм планування Backfill є розвитком FIFO і підтримується менеджером ресурсів Slurm. Особливість Slurm Backfill полягає в тому, що він може пропускати позачергово задачі з низьким пріоритетом за умови, що це не затримує виконання задач з високим пріоритетом. Цей спосіб ефективний у тому випадку, якщо користувачі точно вказують обмеження за часом виконання задачі.

В якості тестової послідовності задач була обрана статистика роботи кластера СКІТ-3 Інституту кібернетики імені В.М. Глушкова НАНУ за 10 місяців 2010 року. Для прискорення роботи стенду використовувався коефіцієнт

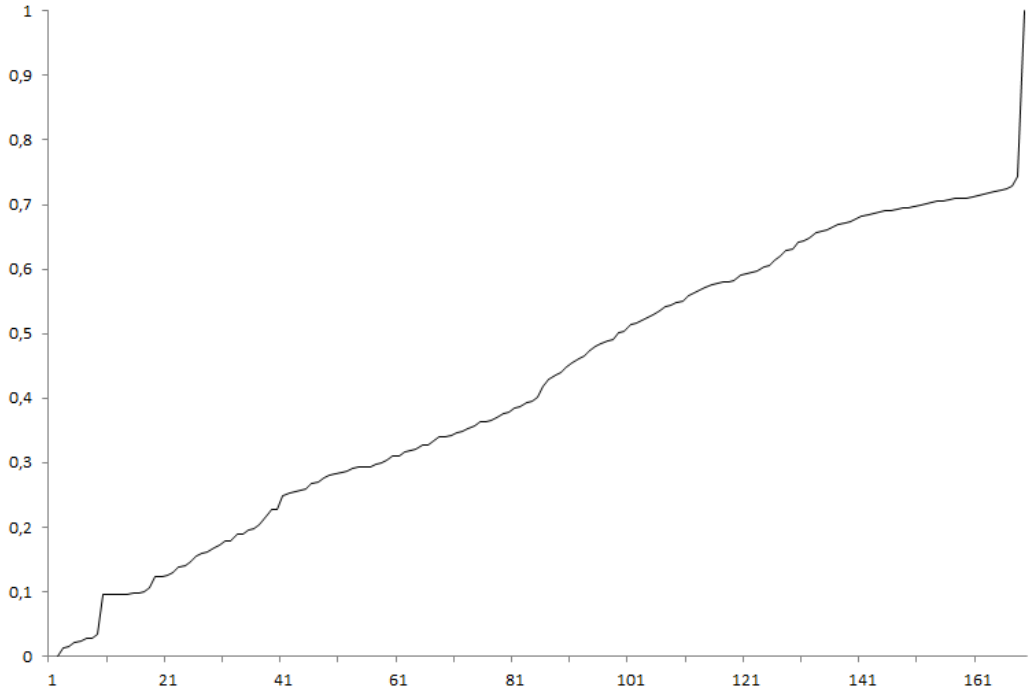

Рис. 1. Графік емпіричної функції розподілу зайнятих вузлів для Torque FIFO

часу рівний 20, що дозволило скоротити моделювання до двох тижнів. Моделювання зводилося до запуску на стенді задач, аналогічних оригінальним за кількістю зайнятих вузлів та часом виконання.

Тестова послідовність складалась з 32567 задач, реальна довжина яких перевищувала 20 секунд. Ця вимога пов'язана з округленням моментів запуску та виконання до найближчої 20-секундної поділки. Параметри тестової послідовності наведені в таблиці 1.

Середній час роботи задачі, годин	12
Найменша кількість вузлів	1
Найбільша кількість вузлів	100

Табл. 1. Параметри тестової послідовності

На стенд по черзі встановлювалися різні менеджери ресурсів і алгоритми планування: Torque FIFO, Slurm Backfill та Slurm FIFO.

Для кожного з трьох випадків на стенд завантажувалась тестова послідовність задач таким чином, щоб зімітувати для планувальника моменти появи задач та необхідну кількість ресурсів точно так само, як це відбувається на реальному кластері. Після запуску віртуальні задачі "виконувались" необхідну кількість часу, після чого припинялись. Стенд зупиняв свою роботу після завершення останньої задачі у тестовій послідовності.

Журнал запусків задач опрацьовувався, відновлювався масштаб часу та проводилось обчислення усіх вищезгаданих індексів.

На рис. 1 зображена емпірична функція розподілу зайнятих вузлів для планувальника Torque FIFO. Вона поводить себе подібно до функції розподілу для рівномірного розподілу крім останнього значення в точці $x = 166$, що відповідає повному завантаженню кластера.

Емпіричні функції розподілу зайнятих вузлів для Slurm FIFO та Slurm Backfill незначно відрізняються від емпіричної функції розподілу для Torque FIFO, (див. рис. 2). Відмітимо, що ймовірність завантаження не більше x вузлів, де $x \in [20; 40] \cup [110; 165]$, для Slurm Backfill вища, ніж у Torque FIFO. У випадку порівняння емпіричних функцій розподілу Torque FIFO та Slurm FIFO бачимо, що, за деякими виключеннями, відповідна ймовірність вища у Torque FIFO, ніж у Slurm FIFO.

Результати порівняння індексів для різних планувальників наведені в таблицях 2 та 3. Цифри в таблиці 2 округлені до цілого для полегшення сприйняття, цифри в дужках означають різницю з Torque FIFO. Бачимо явну пе-

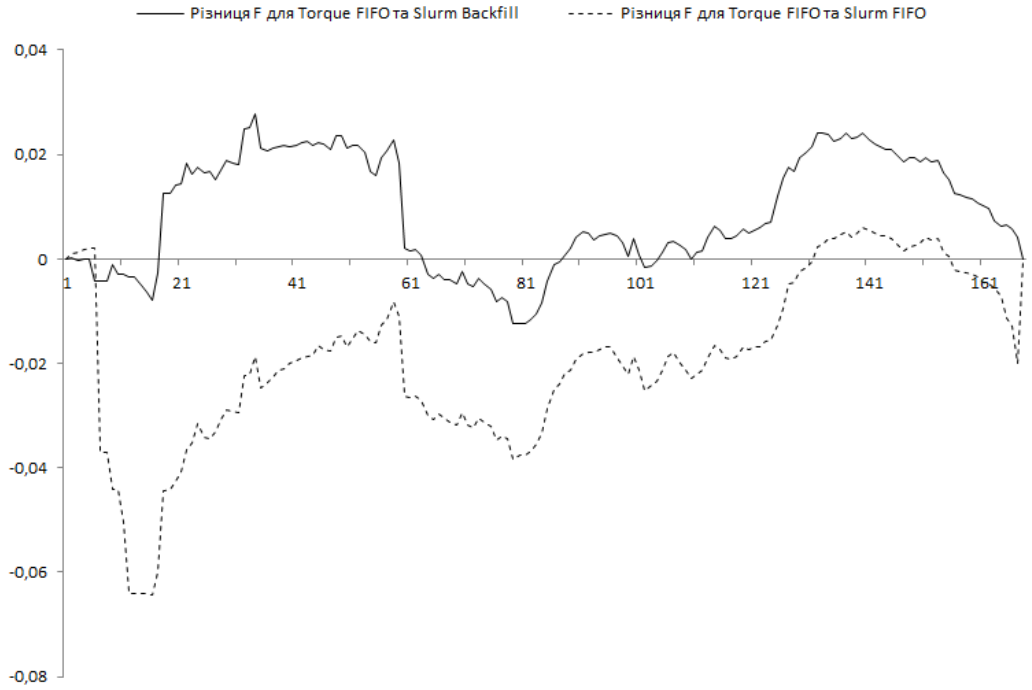

Рис. 2. Різниці емпіричних функцій розподілу

ревагу алгоритму Torque FIFO, який краще поводиться навіть у порівнянні зі Slurm FIFO, що може вказувати на кращу реалізацію алгоритму FIFO у планувальника Torque. У таблиці 3 бачимо, що Torque FIFO виділяє ресурси зовсім по-іншому, кількість задач, які отримують ресурси одразу вдвічі менша за той самий показник у обох версій Slurm. При цьому Torque FIFO має менший середній час очікування W.

	Torque FIFO	Slurm Backfill	Slurm FIFO
W	28597	31748 (+11%)	33035 (+16%)
W_1	83	101 (+21%)	105 (+26%)
W_2	351	417 (+19%)	434 (+23%)
W_3	62	74 (+21%)	80 (+29%)
W_4	83	100 (+21%)	104 (+26%)

Табл. 2. Порівняння індексів для різних планувальників

Torque FIFO	Slurm Backfill	Slurm FIFO
19,56%	41,5%	41,05%

Табл. 3. Відсоток задач, які отримали ресурси одразу

4 Висновки

Побудовано програмний стенд для моделювання роботи кластера та перевірки параметрів взаємодії програмних систем, встановлених на кластері, які не вимагають реальної обчислювальної потужності. Стенд застосований для практичної перевірки роботи двох популярних менеджерів ресурсів та трьох алгоритмів планування: Torque FIFO, Slurm FIFO та Slurm Backfill. Введено та обгрунтовано кілька чисельних індексів для порівняння ефективності роботи планувальників в сенсі часу очікування надання ресурсів. За результатами порівняння однозначну перевагу отримав алгоритм Torque FIFO, оскільки в середньому він забезпечив більш ефективний розподіл ресурсів між задачами.

Порівнюючи однакові алгоритми планування (FIFO), можна сказати, що в Torque FIFO реалізований краще, оскільки в середньому забезпечує на 11% менший час очікування задач у черзі.

Алгоритму Slurm Backfill вдається краще впоратись з потоком задач, що збільшується, він швидше надає вузли новим задачам, забезпечуючи найбільший відсоток задач з нульовим очікуванням.

Розроблена методика і побудований стенд дозволяють підібрати планувальник, найбільш ефективний для кожного кластера. Зокрема, правильно підібраний планувальник дозволяє підвищити енергоефективність кластера, ефективність проходження обчислювальних задач.

Література

[1] http://top500.org.

[2] TORQUE Resource Manager. http://www.clusterresources.com

[3] SLURM Resource Manager. https://computing.llnl.gov/linux/slurm/overview.html