Szeregowanie zadań na superkomputerach z uwzględnieniem buforów impulsowych

*Jan KOPAŃSKI**

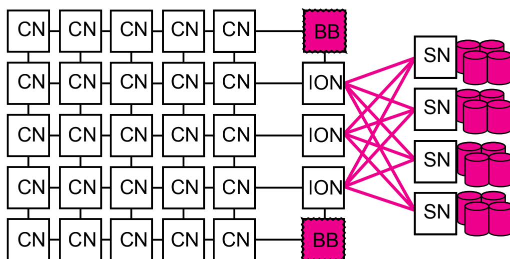
* Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

Praca magisterska autora pt. *Optimisation of job scheduling for supercomputers with burst buffers* otrzymała pierwszą nagrodę w XXXVIII Ogólnopolskim Konkursie Polskiego Towarzystwa Informatycznego na najlepsze prace magisterskie z Informatyki. Na jej podstawie powstał artykuł *Plan-Based Job Scheduling for Supercomputers with Shared Burst Buffers* opublikowany na konferencji Euro-Par 2021. Opiekunem pracy oraz współautorem publikacji był Krzysztof Rządca.

Jedną z istotnych decyzji do podjęcia w trakcie studiów jest wybór tematu pracy dyplomowej. Dla osób, które myślą o podjęciu pracy naukowej, stanowić to może okazję do napisania pierwszej publikacji naukowej. Podstawą do tego może stać się znalezienie półpraktycznego, półteoretycznego tematu, którego celem jest zbadanie jednego z otwartych obecnie problemów w nauce. Wdzięczną dziedziną takich zagadnień są problemy optymalizacyjne, mające swoje umocowanie w świecie rzeczywistym. Często praktyczne problemy optymalizacyjne mogą mieć wiele alternatywnych rozwiązań, w szczególności zależnych od przyjętych założeń modelujących świat rzeczywisty. Ponadto rozwój technologii wpływa na założenia problemów uznawanych już za rozwiązane, tworząc nowe zagadnienia wymagające kolejnych usprawnień.

Co to jest superkomputer?

W najprostszym ujęciu superkomputer to bardzo wiele komputerów połączonych ze sobą specjalistyczną siecią. Sieć ta charakteryzuje się dużą przepustowością danych, krótkim czasem dostępu oraz przede wszystkim wysoką niezawodnością. Poniższy rysunek prezentuje schemat superkomputera.



Źródło: https://glennklockwood.blogspot.com

Pojedynczy komputer w sieci nazywamy węzłem obliczeniowym (CN). Oprócz węzłów obliczeniowych istnieją także inne typy węzłów, np. węzły operacji wejścia/wyjścia (ION), które przekierowują ruch z sieci łączącej węzły obliczeniowe do macierzy dyskowych (SN) tworzących główną pamięć trwałą superkomputera.

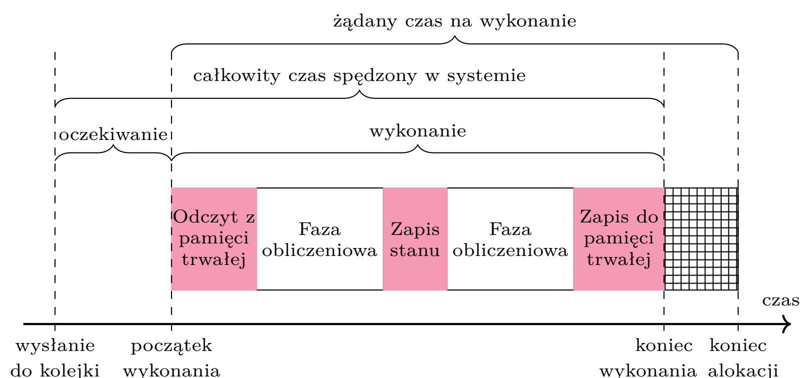
O superkomputerach pisaliśmy już w *Delcie* w 2017 roku („Superkomputery”, Δ^7_{17}). Od tego czasu powstały trzy superkomputery, które według rankingu TOP500 były w czasie pisania tego artykułu najszybsze na świecie: amerykańskie Summit i Sierra oraz szybszy od nich japoński Fugaku. Fugaku posiada 158 976 węzłów obliczeniowych, każdy z procesorem specjalnej konstrukcji w architekturze ARM wyposażonym w 48 rdzeni obliczeniowych. Węzły połączone są za pomocą sieci w topologii sześciowymiarowego torusa. Pojedynczy węzeł ma teoretyczną przepustowość 40,8 GB/s.

Bufory impulsowe

Problemem podkreślanym od wielu lat jest stale zwiększająca się różnica pomiędzy wydajnością jednostek obliczeniowych a przepustowością dostępu do pamięci trwałej. Wraz z rosnącą liczbą węzłów rośnie wydajność obliczeniowa superkomputerów. Za tym wzrostem nie nadąża jednak rozwój pamięci trwałych. Również przepustowość pomiędzy węzłami obliczeniowymi a główną pamięcią trwałą jest ograniczona. Próbą rozwiązania tego problemu są właśnie bufory impulsowe (*burst buffers*, oznaczone na diagramie BB). Bufory impulsowe to w uproszczeniu inteligentnie brzmiący neologizm odnoszący się do szybkich dysków SSD umieszczonych pomiędzy węzłami obliczeniowymi a główną pamięcią trwałą.

Programy uruchamiane na superkomputerach składają się zazwyczaj z naprzemiennie następujących po sobie faz obliczeniowych oraz faz zapisu i odczytu (patrz rysunek na dole strony).

Typowym scenariuszem w fazie zapisu i odczytu jest zapis stanu wykonania programu: programy periodycznie zapisują swój globalny stan po wykonaniu kolejnych faz obliczeń na wypadek wywłaszczenia (czyli po prostu ich przerwania)lub błędu. Taki globalny zapis stanu generuje nagły ogromny strumień danych, który może przewyższyć przepustowość zapisu do głównej pamięci trwałej. Bufory impulsowe umożliwiają szybkie zaabsorbowanie zapisu stanu, wznowienie obliczeń, a następnie w ograniczonym tempie przesłanie zapisu do głównej pamięci trwałej. Bufory impulsowe pozwalają także m.in. na załadowanie danych wejściowych przed rozpoczęciem zadania czy rozszerzenie pamięci RAM.



Ćwiczenie dla Czytelnika Znającego Programowanie Liniowe: Dany jest superkomputer posiadający N węzłów obliczeniowych i M buforów impulsowych, pojemność każdego bufora wynosi B. W kolejce znajduje się K zadań. Każde zadanie to trójka (n_i, b_i, t_i) oznaczająca kolejno: wymaganą liczbę węzłów, pojemność bufora per węzeł oraz czas rezerwacji. Zadanie może otrzymać wiele buforów impulsowych. Dla danego węzła wymagana pojemność bufora nie może zostać przydzielona do więcej niż jednego bufora impulsowego. Napisać program całkowitoliczbowy, który będzie maksymalizował wykorzystanie zasobów.

Algorytm 1. Agresywne szeregowanie z dopełnianiem

```
 1: procedure EASY-Backfilling(Q)
 2:     for J ∈ Q do                         ▷ Q – kolejka zadań oczekujących na wykonanie
 3:         if J może otrzymać wymagane procesory i bufory impulsowe then
 4:             Uruchom J i usuń je z Q         ▷ Alokacje nie mogą na siebie nachodzić
 5:         else
 6:             Przerwij pętlę
 7:     J* ← zdejmij pierwsze zadanie z Q
 8:     Zarezerwuj procesory [i bufory impulsowe] dla J* w najwcześniejszym
          możliwym momencie w przyszłości
 9:     for J ∈ Q do                  ▷ Alokacje i rezerwacje nie mogą na siebie nachodzić
10:         if J może otrzymać wymagane procesory i bufory impulsowe then
11:             Uruchom J i usuń je z Q
12:     Usuń rezerwacje zasobów dla J*
13:     Wstaw J* na początek Q
```

Problem szeregowania zadań

Z superkomputera korzystają zazwyczaj dziesiątki osób jednocześnie. Każdy użytkownik wysyła do systemu swoje zadania, które następnie są kolejkowane i oczekują na wykonanie. Zadanie składa się z programu do wykonania oraz deklaracji wymaganych zasobów: ile węzłów/rdzeni obliczeniowych, pamięci RAM i buforów impulsowych jest potrzebnych i na jaki czas. Kluczowy jest ostatni parametr: jeśli program nie zakończy się samoistnie, to po upływie przydzielonego czasu zostanie on wywłaszczony.

Superkomputery sterowane są przez zaawansowane oprogramowanie nazywane systemami zarządzania zasobami i szeregowania zadań. Najpopularniejszym obecnie systemem tego typu jest SLURM (*Simple Linux Utility for Resource Management*) . To te systemy odpowiadają za przydział zasobów do zadań i kolejność ich wykonania. Celami tych systemów są:

1. Maksymalizacja ciągłego wykorzystania zasobów superkomputera.
2. Minimalizacja czasu spędzonego w systemie przez każde zadanie (czas oczekiwania na rozpoczęcie + czas wykonania).
3. Sprawiedliwy przydział zasobów i unikanie „zagłodzenia”, czyli sytuacji, w której zadanie oczekujące w kolejce przez długi czas nie otrzymuje potrzebnych zasobów.

W rozważaniach teoretycznych zakłada się często, że zadania nie mogą zażądać pojedynczych rdzeni, tak jak ma to miejsce w praktyce, a całe węzły obliczeniowe. Pamięć RAM, jako zasób lokalny względem węzłów, może zostać wtedy pominięta w rozważaniach. Problem szeregowania zadań sprowadza się wówczas do optymalizacji przydziału procesorów – węzłów obliczeniowych. Wprowadzenie buforów impulsowych znacząco komplikuje ten problem, ponieważ podobnie jak procesory, bufory impulsowe są zasobem globalnym.

Algorytmy szeregowania zadań

Najpopularniejszym algorytmem szeregowania zadań jest szeregowanie z dopełnianiem (*backfilling*). Istnieje kilka wariantów tego algorytmu. SLURM implementuje konserwatywny algorytm szeregowania z dopełnianiem. My natomiast skupimy się na wariancie „agresywnym”, który przedstawia **algorytm 1**.

Na wejściu dana jest kolejka zadań oczekujących na wykonanie. Na początku uruchamiamy kolejne zadania z kolejki tak długo, aż natrafimy na pierwsze zadanie, dla którego nie ma wystarczająco dużo wolnych zasobów w systemie (linie 2–7). Wówczas tworzymy rezerwacje zasobów dla tego zadania w najwcześniejszym możliwym momencie w przyszłości, w oparciu o przewidywane czasy zakończania uruchomionych już zadań (linia 8). Następnie staramy się uruchomić pozostałe zadania w kolejce, tak aby nie opóźnić zadania z rezerwacjami (linie 9–11), i w końcu wstawiamy to zadanie na początek kolejki (linie 12–13).

Istotą problemu w obecnych implementacjach szeregowania z dopełnianiem jest naiwne rozszerzenie ich o obsługę buforów impulsowych. Alokacja zasobów w systemie SLURM została dodana poza procedurą szeregowania z dopełnianiem (by to odwzorować w linii 8, bufory impulsowe ujęliśmy w nawiasy kwadratowe). To rozwiązanie może prowadzić jednak do głodzenia zadań wymagających buforów impulsowych.

Symulowane wyżarzanie

Jak widać, algorytm szeregowania z dopełnianiem jest stosunkowo prosty. Szeregowanie zadań można ulepszać na dwa główne sposoby:

1. Użytkownicy zazwyczaj deklarują wymagany czas dla zadania znacznie dłuższy od rzeczywistego czasu jego działania. Zamiast korzystać z czasów podanych przez użytkowników przy tworzeniu rezerwacji, można oszacować rzeczywisty czas działania każdego zadania. Dobrym narzędziem do tego jest uczenie maszynowe, np. drzewa decyzyjne czy wzmacnianie gradientowe (*gradient boosting*).
2. Stworzyć bardziej wyrafinowane algorytmy szeregowania zadań. Do tego można wykorzystać znane metody optymalizacyjne, takie jak algorytmy

genetyczne lub symulowane wyżarzanie, bądź ponownie zdać się na uczenie maszynowe za pomocą uczenia ze wzmocnieniem.

My skupimy się na tym drugim podejściu, a konkretniej stworzymy algorytm szeregowania zadań przy użyciu symulowanego wyżarzania. Symulowane wyżarzanie to algorytm heurystyczny przeszukujący pewną przestrzeń rozwiązań w poszukiwaniu minimum globalnego (najlepszego rozwiązania) dla zadanej funkcji celu. W każdej kolejnej iteracji symulowanego wyżarzania znajdujemy alternatywne rozwiązanie i wyznaczamy wartość funkcji celu. Jeśli jest ono lepsze od poprzedniego rozwiązania, to akceptujemy je, a jeśli jest gorsze, to z pewnym prawdopodobieństwem akceptujemy lub odrzucamy. Postępujemy w ten sposób, aby uniknąć utknięcia w minimum lokalnym przeszukiwanej przestrzeni. Ważne jest także to, aby prawdopodobieństwo malało wraz z kolejnymi iteracjami i zbliżaniem się do minimum globalnego – za to odpowiada parametr nazywany *temperaturą*. Symulowane wyżarzanie zostało opisane także w artykule *O układaniu optymalnych planów zajęć w systemie USOS* (Δ^{12}_{13})

Zauważmy, że efektywność szeregowania w algorytmie szeregowania z dopełnianiem zależy od kolejności zadań w kolejce. Naszą przestrzenią poszukiwań będą zatem permutacje zadań! Zdefiniujmy teraz funkcję celu. Niech Q ponownie oznacza kolejkę zadań czekających na rozpoczęcie, a P jedną z możliwych permutacji zadań w kolejce. Przez W_j^P oznaczymy oczekiwany czas na rozpoczęcie j-tego zadania w kolejce, gdy przydział zasobów do zadań następuje zgodnie z permutacją P.

Naszą funkcją celu, którą będziemy minimalizować, jest:

$$f(P) = \sum_{j \in Q} (W_j^P)^\alpha,$$

gdzie $\alpha \in \mathbb{R}^+$. W praktyce będziemy stosować $\alpha = 2$, czyli minimalizować sumę kwadratów czasów oczekiwania, gdyż daje to dobre rezultaty, a jednocześnie zapobiega głodzeniu zadań.

W jaki sposób uzyskać oczekiwane czasy na rozpoczęcie zadań? Mając daną permutację P, wystarczy dla każdego kolejnego zadania znaleźć najwcześniejszy możliwy przydział zasobów, który nie nachodzi na rezerwacje dla pozostałych zadań. Innymi słowy, stworzyć plan wykonania dla wszystkich zadań.

Algorytm 2 przedstawia nasz schemat symulowanego wyżarzania. Na początku w oparciu o wiedzę z dziedziny wyznaczamy niewielki zbiór permutacji będących kandydatami na minimum globalne. Spośród nich wyłaniamy najlepszą i najgorszą permutację na podstawie funkcji celu (linie 2–3). Posłużą nam one do wyznaczenia początkowej permutacji oraz temperatury (linie 4–6). W każdej iteracji wyżarzania wyznaczamy nową permutację, zamieniając ze sobą kolejnością dwa zadania w obecnej permutacji (linia 9). Następnie symulujemy przydział zasobów, tworząc plan wykonania i obliczamy nową wartość funkcji celu (linie 10–11). Jeżeli jest lepsza od obecnej, to akceptujemy zmianę, a jeżeli jest gorsza, to akceptujemy ją z pewnym prawdopodobieństwem (linie 12–13). Prawdopodobieństwo to zależy od temperatury, którą zmniejszamy co N iteracji (linia 16). Jeżeli nowa permutacja jest lepsza od wszystkich poprzednich, to zapisujemy ten fakt (linie 14–15). Algorytm kończy wykonanie najlepszej znalezionej permutacji (linia 17).

Zauważmy, że powyższy algorytm nie wspomina nic o buforach impulsowych. Istotne jest, aby były one rezerwowane jednocześnie z procesorami podczas tworzenia planu wykonania. Schemat ten jest na tyle ogólny, że można go w prosty sposób rozszerzyć o inne typy zasobów obecnych w superkomputerach: pamięć RAM, pamięć HBM, karty graficzne, karty FPGA czy licencje na oprogramowanie.

Algorytm 2. Szeregowanie oparte na planie wykonania

```
 1: procedure PLAN-BASED(Q, r, N, M)
 2:   P_best, S_best ← znajdź permutację o najniższym wyniku spośród kandydatów
 3:   P_worst, S_worst ← znajdź permutację o najwyższym wyniku spośród kandydatów
 4:   if S_best ≠ S_worst then                 ▷ S – Wynik funkcji celu (energia)
 5:     T ← S_worst − −S_best                   ▷ T – Temperatura początkowa
 6:     P ← P_best; S ← S_best                  ▷ P – Permutacja początkowa (stan)
 7:     for i = 1..N do                         ▷ Symulowane wyżarzanie
 8:       for j = 1..M do
 9:         P′ ← zamień dwa zadania na losowych pozycjach w P
10:         Stwórz plan wykonania dla P′
11:         S′ ← oblicz funkcję celu dla P′ na podstawie planu wykonania
12:         if S′ < S ∨ random(0, 1) < e^((S−S′)/T) then
13:           P ← P′; S ← S′
14:         if S′ < S_best then
15:           P_best ← P′; S_best ← S′
16:       T ← r · T                             ▷ Zmniejszanie temperatury, r ∈ (0, 1)
17:   Zarezerwuj zasoby i uruchom zadania zgodnie z planem wykonania dla P_best
```

Aby wykazać, że nasz nowy algorytm szeregowania zadań jest lepszy niż istniejące metody, musimy przeprowadzić eksperymenty. By nasze eksperymenty były wiarygodne, powinniśmy wykorzystać rzeczywiste dane. A co, jeśli dane nie istnieją dla nowego rodzaju sprzętu – buforów impulsowych? Wówczas możemy wykorzystać tradycyjny zbiór zadań i wzbogacić go o wiarygodnie wygenerowane żądania buforów impulsowych. Ponadto mało prawdopodobne jest, aby ktoś udostępnił nam superkomputer na wyłączność do testowania naszych algorytmów. Potrzebny jest nam zatem symulator superkomputera! Na potrzeby naszego problemu możemy stworzyć taki symulator od podstaw lub dostosować do niego jeden z istniejących symulatorów. Zasadniczą zaletą drugiej opcji jest wykorzystanie sprawdzonego już w innych pracach rozwiązania, co może uczynić wyniki naszych eksperymentów bardziej wiarygodnymi.