PLANOWANIE ZADAŃ
DLA KLASTRA OBLICZENIOWEGO

Wojciech BOŻEJKO, Piotr NADYBSKI, Mieczysław WODECKI

Streszczenie: W pracy jest rozpatrywane zagadnienie związane z harmonogramowaniem zadań w klastrze obliczeniowym (HPC). Ich przydział i kolejność realizacji w węzłach obliczeniowych można sprowadzić do znanego w literaturze dwuwymiarowego problemu pakowania. Przedstawiamy dwuetapowy algorytm bazujący na strategii zachłannej oraz metaheuryste opartej na metodzie przeszukiwania z tabu. Przeprowadzone eksperymenty obliczeniowe wykazały dużą efektywność algorytmu i stanowią inspirację do dalszych badań nad problemami modelującymi znacznie lepiej współczesne środowisko obliczeń wieloprocesorowych.

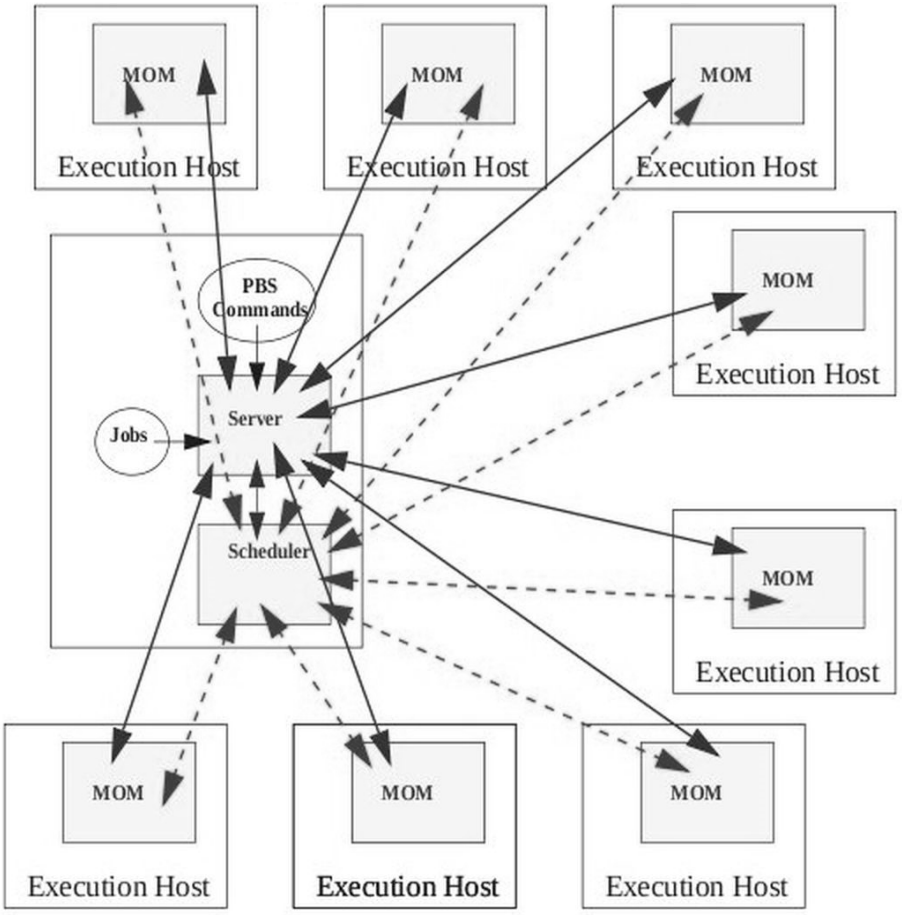
Słowa kluczowe: klaster obliczeniowy, szeregowanie zadań, minimalizacja kosztu, problem pakowania.

1. Klastry obliczeniowe

Minione dziesięciolecia rozwoju technologii z obszaru informatyki charakteryzują się stałym wzrostem mocy obliczeniowej komputerów oraz ich procesorów. Zadania wymagające znaczącej mocy obliczeniowej, które jeszcze kilkanaście lat temu pozostawały w sferze ściśle zarezerwowanej dla niewielkiej liczby superkomputerów na świecie, dzisiaj są z powodzeniem wykonanie na układach instalowanych w komputerach osobistych. Niemniej dalej pozostają pewne obszary, w których odpowiednia dokładność obliczeń wymaga na tyle dużej mocy obliczeniowej, że nawet najwydajniejsze, dostępne na rynku procesory, potrzebują zbyt długiego czasu na wykonanie tych obliczeń, by mówić o ich użyteczności. Wśród nich wymienić można matematykę kwantową, prognozę pogody oraz badania klimatyczne, zarządzanie systemami transportowymi, prognozowanie i eksploracja złóż gazu czy ropy, symulacje z modelami przebiegu reakcji chemicznych, biologicznych, medycznych oraz fizycznych (np. motoryzacja, lotnictwo, wytrzymałość materiałowa itp.) czy ekonomicznych (prognozowanie rynku, giełdy itp.), a także logistyki oraz zarządzania procesami produkcyjnymi.

Obecny poziom wiedzy z pozwala na konstruowanie bardzo złożonych modeli uwzględniających dużą liczbę parametrów. Dzięki temu wyniki prognoz i symulacji z wykorzystaniem skomplikowanych obliczeń są coraz bardziej trafne. Konsekwencją stosowania modeli o znacznej liczbie parametrów jest duża złożoność obliczeniowa, a co za tym idzie wysokie wymagania dotyczące mocy obliczeniowej systemów, w których uruchamia się tego typu procesy. Dlatego takie problemy rozwiązuje się korzystając ze środowisk o dużej mocy obliczeniowej (wielokrotnie przewyższającej możliwości typowych komputerów, powszechnie obecnie stosowanych). Koszty prowadzenia badań oraz proces projektowania i produkcji dedykowanych procesorów do tego typu zastosowań byłby ogromny (wystarczy spojrzeć na budżety działów R&D takich firm jak np. Intel, które sięgają wartości liczonych w miliardach dolarów [1]). Dlatego też do budowy komputerów dużej mocy wykorzystuje się produkowane seryjnie komponenty, łącząc je

przy wykorzystaniu odpowiednich technologii oraz protokołów. Jednym z przykładów takiego podejścia są klastry obliczeniowe. Są to zestawy odrębnych komputerów, nierzadko indywidualnie posiadających relatywnie przeciętną moc obliczeniową, które z perspektywy użytkownika stanowią jeden system o zasobach wielokrotnie większych niż pojedynczy komputer (nazywany węzłem) wchodzący w skład klastra. Iluzja pojedynczego systemu możliwa jest dzięki odpowiednim technologiom sprzętowym (komponenty sieci komputerowej - interfejsy sieciowe, okablowanie, urządzenia sieciowe) oraz softwareowym (oprogramowanie sieciowe, oprogramowanie do centralnego zarządzania zasobami). Ogólną ideę budowy klastra przedstawiono na rysunku 1.



Rys. 1. Ogólna idea budowy klastra obliczeniowego [2]

Koncepcja klastra obliczeniowego jest dość powszechnie stosowana z powodu wielu swoich zalet. Jego budowa jest relatywnie dość tania z racji wykorzystania powszechnie dostępnych komponentów. Jednocześnie dzięki wykorzystaniu dziesiątków czy nawet setek węzłów, środowisko to może zapewniać bardzo dużą moc obliczeniową. Dodatkowo, klaster taki dość dobrze skaluje się poprzez zwiększenie liczby węzłów, tym samym

jednocześnie wzrasta również poziom zapewnienia dostępności (wyłączenie lub awaria pojedynczego węzła nie powoduje przestoju w pracy klastra).

Typowe środowisko klastra obliczeniowego pozwala na uruchamianie wielu zadań wsadowych, optymalnie wykorzystując dostępne zasoby. System może się składać z identycznych lub różnych węzłów obliczeniowych (uwarunkowane jest to przede wszystkim oprogramowaniem zarządzającym klastrem). Każdy węzeł cechuje się określoną architekturą, dysponuje ustaloną liczbą procesorów, pamięcią RAM. Ilość wymaganych zasobów dla wybranego zadania określana jest przez użytkowników, zgodnie z ich szacunkami odnośnie zapotrzebowania na te zasoby. Wszystkie zadania zlecane przez użytkowników w trakcie pracy systemu gromadzone są w tzw. kolejce zadań oczekujących na realizację, następnie są z niej pobierane i sukcesywnie uruchamiane. W ramach danego systemu dostępne mogą być więcej niż jedna kolejka. Zazwyczaj różnią się one pomiędzy sobą priorytetami zadań w nich umieszczanych. Jednym z ważniejszych elementów takiego systemu jest tzw. menadżer kolejki, (ang. *scheduler*). Rolą *schedulera* zarządzającego kolejkami jest takie uruchamianie zadań oczekujących w kolejkach, by optymalnie wykorzystać zasoby systemu względem przyjętych kryteriów. W użytkowanych obecnie klastrach HPC wykorzystywanych jest kilka narzędzi realizujących proces zarządzania kolejką. Wśród nich wymienić można m.in. *Moab, Univa Grid Engine, Portable Batch System, LoadLeveler, Simple Linux Utility for Resource Management (SLURM), OpenLava IBM's Platform LSF*. Jednym z popularniejszych jest *PortableBatch System (PBS)* przez lata rozwijany w kilku wersjach (*OpenPBS, TORQUE, PBS Professional).* Niezależnie od wersji, ich zadaniem jest takie szeregowanie zadań zlecanych przez użytkowników, by w sposób optymalny wykorzystać dostępne zasoby i spełnić kryteria zadane przez administratora. W samym procesie szeregowania zadań najczęściej wykorzystywane są dość proste algorytmy zachłanne. Jednak budowa modułowa systemów zarządzania kolejkami umożliwia wykorzystanie zewnętrznych, zdecydowanie efektywniejszych, algorytmów [2].

Celem pracy jest opis modelu zarządzania zasobami klastra obliczeniowego oraz konstrukcja bardziej efektywnego algorytmu szeregowania, który mógłby zostać wykorzystany do kolejkowania zadań w klastrze pracującym z PortableBatch System.

2. Sformułowanie problemu

Współczesne systemy zarządzania kolejkami zadań do wykonywa na klastrze obliczeniowym są mocno zróżnicowane i posiadają dużą liczbę różnych parametrów charakteryzujących poszczególne zasoby. Z tego powodu są trudne w konkretnych zastosowaniach i mało efektywne. W tym rozdziale przedstawiamy pewien uproszczony model, który w dalszych badaniach będzie sukcesywnie rozbudowany.

2.1. Model systemu

W rozpatrywanym problemie, dany jest zbiór *n* zadań $Z = \{1, 2, .., n\}$ do wykonania oraz zbiór *m* identycznych węzłów obliczeniowych $U = \{1, 2, .., m\}$ klastra. Zakładamy, że wszystkie węzły posiadają identyczną konfigurację sprzętową. Nie ma zatem znaczenia, do którego węzła, spośród dostępnych w danym momencie, zadanie zostanie przypisane. Zakładamy, że każde zadanie jest opisane przez następujące parametry:

- p_i - założony czas wykonania zadania *i* (*ang. wall time*). Jest to jednocześnie czas,

w którym węzeł jest rezerwowany na wyłączność dla użytkownika,

- S_i - moment, w którym rozpoczyna się wykonanie zadania,
- C_i - moment, w którym według prognozy użytkownika powinno zakończyć się wykonanie zadania, $C_i = S_i + p_i$,
- u_i - liczba węzłów, które są rezerwowane dla realizacji zadania,

W rozpatrywanym problemie przydziału zadaniom węzłów klastra obliczeniowego (w skrócie problem **PW**), dla każdego zadania należy przydzielić odpowiednią liczbę węzłów w odpowiednim przedziale czasowym w taki sposób, aby były spełnione następujące ograniczenia:

(1) każde zadanie należy wykonać na odpowiedniej, zadeklarowanej przez użytkownika, liczbie węzłów,
(2) wykonywanie zadania na wszystkich zaplanowanych węzłach musi rozpocząć się w tym samym momencie,
(3) każdy węzeł może w danym momencie wykonywać co najwyżej jedno zadanie,
(4) wykonywanie zadania, na żadnym z przydzielonych węzłów, nie może być przerwane,
(5) wcześniejsze zakończenie zadania nie powoduje zwolnienia zasobów.

Rozwiązanie problemu **PW** może być reprezentowane przez macierz momentów rozpoczęcia zadań

$$S=[S_{i,j}]_{nxm} \tag{1}$$

gdzie $S_{i,j}$ jest momentem rozpoczęcia wykonywania zadania *i* w węźle *j*. Zakładamy przy tym, że jeżeli zadanie *i* nie jest wykonywane przez węzeł *j*, to $S_{i,j}$=0. Dowolne zadanie, zgodnie z założeniami, może być realizowane na wielu węzłach. Jego wykonywanie musi się jednak rozpoczynać, na każdym z węzłów, w tym samym momencie, tj.

$$\forall_{i\in\{1,2,..,n\}} S_{i,j} = S_{i,1} = S_{i,2} =,...,= S_{i,m} \text{ albo } S_{i,j} = 0 .$$

Przez Ω oznaczamy zbiór wszystkich rozwiązań dopuszczalnych rozpatrywanego problemu, tj. macierzy postaci (1) spełniających ograniczenia (1)-(5). Jeżeli U_i jest zbiorem wszystkich węzłów zarezerwowanych do realizacji zadania *i*, to

$$|U_i|=u_i.$$

Z kolei, jeżeli poprzez Z_j oznaczymy zbiór wszystkich zadań realizowanych w węźle *j*, a czas przez *t*, wówczas

$$\forall_{j\in\{1,2,...,m\}} |Z_j,t|=1 \text{ albo } |Z_j,t|,=0 \tag{2}$$

gdzie $|Z_j,t|$ oznacza liczbę elementów zbioru Z_j w chwili *t*.

Jeżeli S jest dopuszczalnym rozwiązaniem, to termin zakończenia wszystkich zadań

$$F_{max}(S) = max(S) = max([S_{i,j} + p_i]) , \tag{3}$$

Jednym z najważniejszych kryteriów, istotnym z punkt widzenia eksploatacji systemu, jest minimalizacja momentu zakończenia wykonywania wszystkich zadań, tj. wyznaczenie przydziału dopuszczalnego S^* takiego, że

$$F_{min}(S^*) = min(F_{max}(S)) = min(max([S_{i,j} + p_i])) . \tag{3}$$

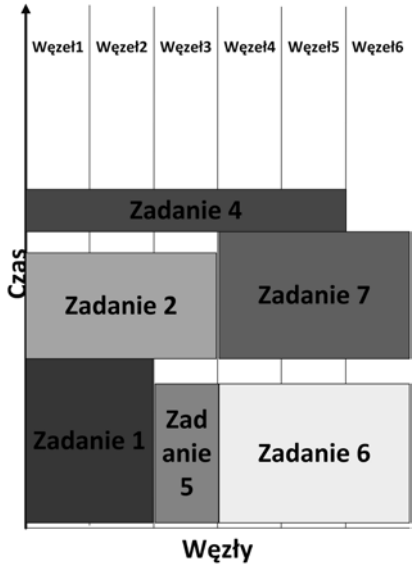
W dalszej części pracy będziemy rozpatrywali problem **PW** wraz z powyższym kryterium.

Wymienione wcześniej parametry charakteryzujące zadanie nie są jedynymi, które mogą wystąpić w realnych systemach obliczeniowych. W szczególności może to być: minimalna wielkość pamięci, licencjonowane oprogramowanie, wejście/wyjście, itp.

3. Harmonogramowanie zadań jako dwuwymiarowy problem pakowania

Można łatwo zauważyć, że przedstawiony w poprzednim rozdziale problem harmonogramowania jest równoważny problemowi dwuwymiarowego pakowania (ang. *two-dimensional bin packing problem*). Został on opisany w latach siedemdziesiątych ubiegłego wieku (zobacz np. prace [3] i [4]). Pierwsze badania były ściśle związane z praktycznymi zastosowaniami, tj. optymalnym cięciem materiału (np. drewno, metal, itp.). Podobnej natury problemy optymalizacyjne występują również, między innymi, w informatyce oraz telekomunikacji.

Problem dwuwymiarowego pakowania (w skrócie problem **DW**) można sformułować następująco (za [6]). Dany jest zbiór obiektów w kształcie prostokąta $J = \{1,..,n\}$. Każdy prostokąt definiowany jest przez dwa parametry: szerokość w_j oraz wysokość h_j. W przypadku problemów produkcyjnych wartości te mogą odpowiadać np. fizycznym wymiarom detali wycinanych z materiału (problem optymalnego wykorzystania materiałów). W modelowaniu problemów z optymalizacyjnych z dziedziny informatyki wymiary rozpatrywanych prostokątów mogą odzwierciedlać inne zasoby fizyczne, na przykład liczbę użytych jednostek wykonawczych w procesie produkcyjnym (szerokość prostokąta) czy czas (najczęściej wysokość prostokąta). Problem polega rozmieszczeniu prostokątów na płaszczyźnie w taki sposób, by zajmowały one jak najmniejszą powierzchnię, jednocześnie bez wzajemnego nakładania się na siebie (podobnie, jak w pracy [5]). Na rysunku 2 przedstawiono pewien przykład problemu przydzielenia węzłów klastra obliczeniowego do wykonywania zadań, który został sprowadzony do dwuwymiarowego problemu pakowania.



Rys. 2. Dwuwymiarowe pakowanie zadań

W praktyce najczęściej można spotkać dwa warianty ogólnego problemu pakowania (szerzej opisane w pracy [7]).

A. Dwuwymiarowy problem pakowania pojemników.

Dany jest nieskończony zbiór identycznych pojemników o wymiarach *W* (ang. *width* - szerokość) oraz *H* (ang. *height* - wysokość). Problem polega na minimalizacji liczby pojemników wykorzystanych do zapakowania wszystkich elementów. Przykładem może być tu zadanie, w którym z płyty metalowej o określonych wymiarach należy wyciąć jak największą liczbę detali, w taki sposób by zminimalizować ilość odpadu. W przypadku problemów optymalizacji procesów, wysokość pojemnika może reprezentować tak zegara, cykl zmiany załogi, itp.

B. Dwuwymiarowy problem pakowania pojemnika o nieograniczonej wysokości.

W tym przypadku prostokąty zostają umieszczone w jednym pojemniku, a celem jest zminimalizowanie wysokości, do której pojemnik zostanie wypełniony. W praktyce jest to przypadek, gdy w procesie produkcyjnym materiał podawany jest np. w postaci taśmy z której wycinane są detale (dla uproszczenia zakłada się że taśma jest dostatecznie długa, a więc, że nie zabraknie materiału) przy jednocześnie jak najmniejszej długości wykorzystanej taśmy.

Problem pakowania był jednym z pierwszych problemów optymalizacji dyskretnej, dla którego udowodniono NP-trudność [8]. Dlatego też, do rozwiązywania różnych wariantów tego problemu zazwyczaj są stosowane algorytmy przybliżone. Zwykle są to hybrydy bazujące na jednej z wielu metaheurystyk oraz strategii zachłannej.

3.1. Metody rozwiązanie problemu pakowania

Rozwiązanie dwuwymiarowego problemu pakowania prostokąta o wymiarach *W*x*H* może być reprezentowany przez macierz $M_{W\times H} = [x_{w,h}]$. Poszczególne elementy macierzy reprezentują pozycje umieszczenia prostokątów, począwszy od lewego dolnego rogu w prawą stronę oraz w górę. Jeżeli więc $x_{w,h}=j$ to oznacza, że na tę pozycję jest wstawiony *j*-ty prostokąt. W przeciwnym przypadku $x_{w,h}=0$.

Strategie pakowania prostokątów.

W literaturze opisano wiele strategii umieszczania prostokątów, które są stosowane w algorytmach pakowania. Wiele z nich przedstawiono w pracy [9]. Najważniejsze założenia, wspólne dla wszystkich strategii:

- elementy są wstępnie ułożone są w postaci ciągu (zwykle nierosnący ze względu na pewne kryterium),
- kolejne elementy są układane od lewego dolnego rogu w prawą stronę, a następnie w górę,
- w większość strategii przyjmuje się tzw. podejście "półkowe". Kolejne elementy są układane na prawo, aż do momentu, gdy nie da się już ułożyć kolejnego elementu. Wtedy wstawiania jest niejawna linia zaczynająca nowy poziom ("półkę") i na niej układane są kolejne elementy. Sytuacja powtarza się aż do napotkania warunku stopu - wypełnienia pojemnika lub ułożenia wszystkich elementów.

Najczęściej, w algorytmach pakowania, stosowane są następujące strategie (w opisie, *j* jest kolejnym elementem z listy, a *t* numerem ostatnio utworzonej warstwy):

- Następna Pasująca Pozycja (ang. *Next-Fit DecreasingHeight* - NFDH). Element *j* jest umieszczany w lewym dolnym rogu najbliższego wolnego miejsca na

poziomie *t*, w którym może zostać umieszczony. W przeciwnym wypadku jest tworzony nowy poziom (*t:=t+1*), a element *j* jest umieszczany jako pierwszy z wyrównaniem do lewej strony,

- Pierwsza Pasująca Pozycja (ang. *First-Fit DecreasingHeight* - *FFDH*). Element *j* jest pakowany z wyrównaniem do lewej na pierwszym poziomie, na którym jest wolne miejsce. Jeżeli na żadnym poziomie nie ma dostatecznego miejsca, wówczas utworzony zostaje nowy poziom (*t:=t+1*) i tam umieszcza się *j* (z wyrównaniem do lewej krawędzi),
- Najlepiej Pasująca Pozycja (ang. *Best-Fit DecreasingHeight* - BFDH). Element *j* jest pakowany, z wyrównaniem do lewej strony na tym poziomie spośród wszystkich pasujących, na którym ilość niewykorzystanego miejsca na tym poziomie po umieszczeniu *j* pozostaje najmniejsza. Jeżeli na żadnym poziomie nie ma dostatecznego miejsca, to tworzony zostaje nowy poziom (*t:=t+1*) i na nim umieszcza się element *j*, z wyrównaniem do lewej krawędzi obszaru.

W przypadku rozpatrywanego w pracy problemu harmonogramowania, każde zadanie można przedstawić w postaci prostokąta o wymiarach $w \times p$, gdzie w oznacza liczbę węzłów klastra zarezerwowanych do realizacji zadania, natomiast p oznacza czas zarezerwowany przez użytkownika na wykonanie zadania. Wówczas, optymalny przydział zadań do realizacji w węzłach klastra sprowadza się do dwuwymiarowego problemu pakowania odpowiednich prostokątów.

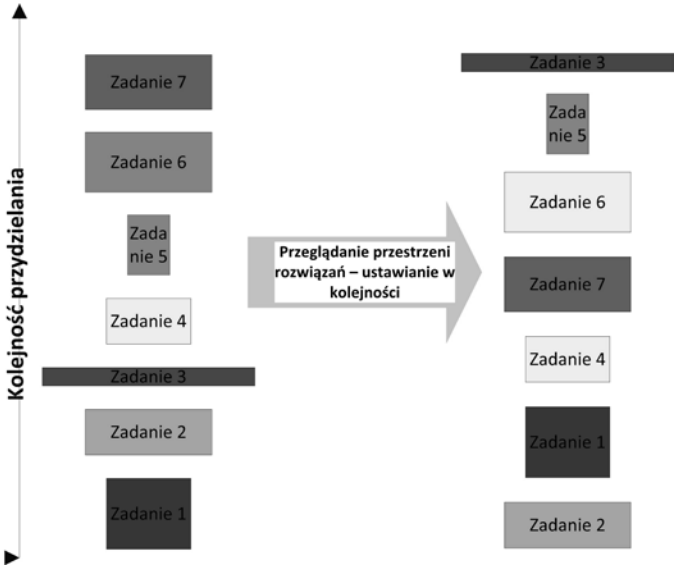
3.2. Algorytm pakowania

Przedstawione w poprzednim podrozdziale strategie pakowania są szybkie (tj. mają niewielką złożoność obliczeniową) i relatywnie łatwe w implementacji. Niestety, wykazują one przypadłość typową dla wielu algorytmów bazujących na metodzie zachłannej polegającą na relatywnie niskiej skuteczności. Dokładność uzyskanych wyników w dużej mierze zależną od doboru danych testowych. Aby chociaż częściowo zminimalizować tę wadę, do rozwiązania rozpatrywanego w pracy problemu, przedstawiamy metodę składającą się z dwóch kroków (jej ogólny schemat przedstawiono na rysunku 3).

Metoda MTS
Repeat
Krok1: Wyznacz jeszcze nie rozpatrywaną permutację elementów zbioru zadań;
Krok2. Stosując jedną ze strategii pakowania oraz korzystając z kolejności zadań wyznaczonej w Kroku 1, wyznacz minimalny obszar, tj. minimalny moment zakończenia wykonywania wszystkich zadań przez węzłach klastra.
Until {*warunek zakończenia*}

W realizacji metody **MTS** będą stosowane są dwie strategie:

1. przeszukiwania z tabu, dla wyznaczania kolejności zadań,
2. algorytmu zachłannego, do wygenerowania rozwiązania (zapakowania elementów), tj. czasu wykonywania zadań oraz wyznaczenia wartości funkcji celu.



Rys. 3. Idea dwukrokowego algorytmu pakowania

Niech Φ będzie zbiorem wszystkich permutacji stanowiących kolejność pakowania elementów - przydziału zadań zbioru *Z*. Przedstawioną powyżej ogólną ideę metody pakowania można, bardziej formalnie, zapisać następująco:

Algorytm ATS

1. Wylosować permutację startową $\pi \in \Phi$;
2. $\pi_{min} \leftarrow \pi$;
3. Wyznacz otoczenie $\Phi_{\pi} \in \Phi$ permutcji π generowane przez ruchy typu zamień (ang. *swap*) (zobacz np. praca [10]);
4. Dla każdej permutacji $\pi' \in \Phi_{\pi}$ wykonaj pakowanie korzystając z metody zachłannej oraz oblicz wartość funkcji kosztu $F_{max}(\pi)$;
5. Jeżeli $F_{max}(\pi') < F_{max}(\pi_{min})$, to podstaw $\pi_{min} \leftarrow \pi'$;
6. Jeżeli nie wystąpiła warunek zakończenia, to podstaw $\pi \leftarrow \pi'$ oraz przejdź do punktu 3;
7. Zwróć wartość $F_{\max}(\pi_{min})$, przybliżoną wartość rozwiązania problemu przydziału węzłów do wykonywania zadań.

Algorytm **ATS** wyznacza rozwiązanie π_{min} - kolejność pakowania elementów**.** Korzystając z jednej ze strategii pakowania otrzymujemy przybliżone rozwiązanie - termin zakończenia wykonywania zadań przez węzły klastra obliczeniowego. Złożoność obliczeniowa algorytmy zależy od liczby iteracji, tj. warunku zakończenia sprawdzanego w wierszu 6.

4. Eksperymenty obliczeniowe

Przedstawiony w poprzednim rozdziale algorytm został zaimplementowany w języku C# oraz uruchamiany na komputerze z procesorem Intel i7 3537U (3.1 GHz). Obliczenia wykonano dla różnych rozmiarów danych. Liczba zadań n=20,30,40,50,100, a liczba węzłów m=5,10. Czasy wykonywania zadań p_i oraz liczby węzłów deklarowanych do wykonania zadania p_i generowano losowo, zgodnie z rozkładem jednostajnym na przedziale [1,m]. Dla każdego pary (n,m) wygenerowano 10 przykładów. W sumie więc przygotowano 50 przykładów danych testowych. Obliczenia wykonano w celu porównania dwóch algorytmów:

1. zachłannego bazującego na strategii FFDH,
2. **ATS** z przeszukiwaniem z tabu (linia 3).

Po wstępnych obliczeniach przyjęto maksymalną liczbę iteracji równą 50, a długość listy tabu równą 7. Dodatkowo, dla każdego przykładu wyznaczono dolne ograniczenie wartości funkcji celu $LB = \sum_{i=1}^{n} p_i * u_i m$. Otrzymane wyniki przedstawiono w Tabeli 1. W każdym wierszu znajduje się średnia obliczeń z 10 przykładów danego rozmiaru, tj. pary (n,m). Z kolei w ostatniej kolumnie zamieszczono procentową poprawę przez algorytm TS rozwiązania wyznaczonego przez algorytm zachłanny.

Tabla 1. Wyniki eksperymentów obliczeniowych.

liczba zadań	liczba węzłów	dolne ograniczenie	algorytm zachłanny	algorytm z tabu	poprawa rozwiązania
20	5	64,4	84	67	20,24%
30	5	82,8	98	88	10,20%
40	5	134,6	162	139	14,20%
50	5	142,4	170	149	12,35%
50	10	154,4	194	168	13,40%
100	10	308,4	408	341	16,42%

Na podstawie zamieszczonych wyników można stwierdzić, że wyznaczane przez algorytm przeszukiwania z tabu kolejności pakowania elementów są znacznie lepsze od tych, wyznaczanych przez algorytm zachłanny. Średnia poprawa rozwiązania wynosi kilkanaście procent. Należy podkreślić, że średnie wartości rozwiązań wyznaczonych przez algorytm z TS są niewiele większe dolnych ograniczeń (średni błąd względny nie przekracza 10%).

5. Podsumowanie

W pracy rozpatrywano problem planowania zadań wymagających wielu węzłów, dla klastra obliczeniowego, z minimalizacją terminu ukończenia wykonywania wszystkich zadań. Wyznaczenie rozwiązania można sprowadzić do pewnego dwuwymiarowego problemu pakowania prostokątów Przedstawiono algorytm przeszukiwania z tabu z zastosowaniem zachłannej strategii pakowania. Zadawalające wyniki przeprowadzonych eksperymentów obliczeniowych są inspiracją do dalszych badań z wykorzystaniem bardziej złożonych modeli, uwzględniających większą liczbę rzeczywistych parametrów systemów oraz dodatkowych zasobów.

Literatura

1. Baker B.S., Brown D.J., Katseff H.P., A 5/4 algorithm for two-dimensional packing, Journal of Algorithms, 2, 1981, 348-368.
2. Borowicka K., Bożejko W., Kacprzak Ł., Wodecki M., Problem harmonogramowania zadań wielomaszynowych, Innowacje w Zarządzaniu i Inżynierii Produkcji (red. R. Knosala), Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją, Tom I, Opole 2016, 547-558.
3. Coffman E. G., Jr., Garey, M. R., Johnson D. S., Tarjan R. E., Performance bounds for level-oriented two-dimensional packing algorithms. SIAM Journal on Computing, 9, 1980, 801-826.
4. Garey M. R., Johnson D.S., Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co, 1979.
5. Harren R., Two - dimensional packing problem, Saarbruken, 2010, http://scidok.sulb.unisaarland.de/volltexte/2010/3470/pdf/Dissertation_1302_Harr_Rolf_2010.pdf
6. Intel Research and Development Expense https://ycharts.com/companies/INTC/r_and_d_expense
7. Lodi A., Martello S., Monaci M., Two-dimensional packing problems: A survey, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy, 2001.
8. Lodi A., Martello S, Vigo D., Recent advances on two-dimensional bin packing problems, Discrete Applied Mathematics Volume 123, Issues 1-3, 15, 2002, 379-396.
9. PBS ProfessionalÂ® 13.0 Reference Guide.
10. Wodecki M., A block approach to earliness-tardiness scheduling problems International Journal on Advanced Manufacturing Technology, 40, 2009, 797-807.

Prof. nadzw. dr hab. Wojciech BOŻEJKO
Katedra Automatyki, Mechatroniki i Systemów Sterowania
Politechnika Wrocławska
ul. Janiszewskiego 11/17, 50-372 Wrocław
e-mail: wojciech.bozejko@edu.pwr.wroc.pl

Mgr inż. Piotr NADYBSKI
Państwowa Wyższa Szkoła Zawodowa
im. Witelona w Legnicy
ul. Sejmowa 5a,
e-mail: nadybskip@pwsz.legnica.edu.pl

Prof. nadzw. dr hab. Mieczysław WODECKI
Instytut Informatyki Uniwersytetu Wrocławskiego
ul. Joliot-Curie 15, 50-383 Wrocław
e-mail: mwd@ii.uni.wroc.pl