ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1999
Seria: INFORMATYKA z. 36 Nr kol. 1414

Mateusz NOWAK
Instytut Informatyki Teoretycznej i Stosowanej, PAN

WYRÓWNYWANIE OBCIĄŻENIA W WARUNKACH INTENSYWNEJ KOMUNIKACJI*

Streszczenie. Artykuł przedstawia prace autora nad wyrównywaniem obciążenia w heterogenicznych systemach rozproszonych, w szczególności w sieci stacji roboczych, dla pewnej klasy algorytmów. W pracy skoncentrowano się na algorytmach, dla których czas komunikacji między węzłami przetwarzającymi w stosunku do czasu obliczeń jest znaczący, natomiast ziarnistość zadania jest duża. Takie cechy charakteryzują często algorytmy do przetwarzania obrazów.

LOAD BALANCING IN INTENSIVE COMMUNICATION CONDITIONS

Summary. The article shows the author's work on load balancing in heterogeneous distributed systems, especially in networks of workstations, for an algorithm class. A special attention was paid to algorithms, in which time of communications is significant comparing to computations time, and granularity of the task is relatively big. Such features are typical for a lot of image processing algorithms.

1. Wstęp, geneza problemu

W sytuacji kiedy przetwarzanie równoległe coraz bardziej zyskuje na znaczeniu, coraz istotniejsze stają się problemy związane z optymalną organizacją tego przetwarzania. Problem ten widoczny jest zwłaszcza w środowiskach heterogenicznych, w których jednostki przetwarzające mają różną moc obliczeniową, a często są specjalizowane do pewnej klasy zadań. Różnice występują też w szybkości łącz komunikacyjnych, co ma znaczenie, jeżeli czas konieczny na komunikację jest znaczący w stosunku do czasu przeznaczonego na obliczenia.

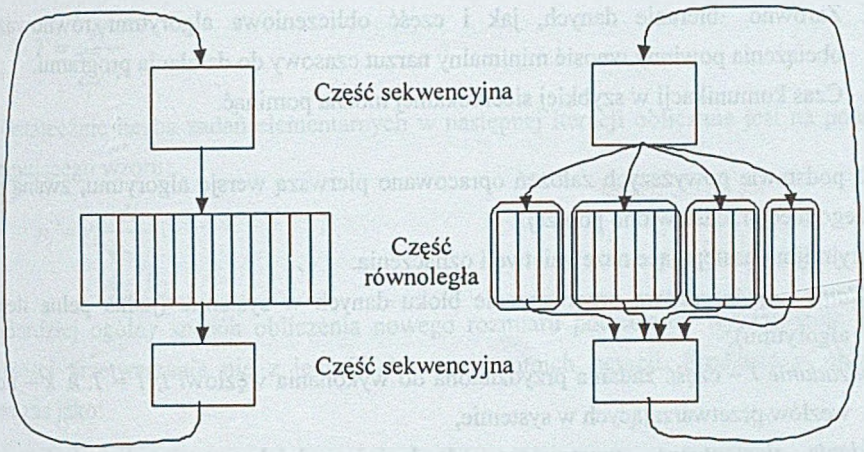
* Publikacja powstała w ramach grantu KBN 8T11C 006 14.

Prace autora koncentrują się na środowiskach mających następujące cechy: brak wspólnej pamięci dla procesów, różna i zmienna w czasie moc węzłów obliczeniowych dostępna dla użytkownika, różna i zmienna w czasie przepustowość łącz komunikacyjnych dostępna dla użytkownika. Cechy takie są charakterystyczne dla sieci stacji roboczych, występują także w środowiskach dużych, wieloprocesorowych serwerów obliczeniowych dostępnych dla wielu użytkowników.

W wielu dotychczasowych pracach nad wyrównywaniem obciążenia w systemach rozproszonych dla różnego typu algorytmów i środowisk autorzy pomijali wpływ komunikacji na jakość równoważenia obciążenia (por. [4]). Ponieważ największa potrzeba zrównoleglenia obliczeń, a co za tym idzie - równoważenia obciążenia, występuje dla algorytmów intensywnych obliczeniowo, w których w przypadku równoległej implementacji potrzeby komunikacyjne są stosunkowo niewielkie, podejście takie jest wystarczające. Istnieją jednak algorytmy, których zrównoleglanie może przynosić opłacalne zyski czasowe, a dla których czas komunikacji między węzłami jest istotną częścią całego czasu przetwarzania. Takimi cechami charakteryzuje się wiele algorytmów przetwarzania obrazów. Z jednej strony ilość obliczeń w stosunku do ilości danych może być niewielka, z drugiej strony użytkownicy często wymagają, aby algorytm był realizowany szybko, np. w czasie rzeczywistym, kiedy czas przetwarzania musi być mniejszy niż odstęp między kolejnymi klatkami obrazu ruchomego dostarczanymi z urządzenia akwizycji.

2. Charakterystyka zadania

Charakterystyczną cechą wielu algorytmów przetwarzania obrazów jest ich średnia bądź duża drobnoziarnistość oraz praca zgodnie ze schematem master-slave. Obraz może być podzielony na wiele części (np. piksel, linia, klaster), dla których wykonuje się te same operacje obliczeniowe. Liczba identycznych operacji wykonywanych niezależnie od siebie dla różnych danych jest znacznie większa od liczby węzłów przetwarzających. Pozwala to w stosunkowo łatwy sposób podzielić całe zadanie na podzadania różnej wielkości i rozdzielić przetwarzanie między węzły. Równocześnie algorytmy te pracują iteracyjnie zgodnie ze schematem master-slave. Węzeł nadzorujący (master) dzieli zadanie na podzadania i zleca ich wykonanie węzłom przetwarzającym (slave), a po ich wykonaniu odbiera wyniki ich pracy i scala wyniki. Operacje te są powtarzane wielokrotnie tak długo, aż błąd rozwiązania będzie mniejszy od założonego. Typowy schemat blokowy takiego algorytmu i sposób jego realizacji przedstawiono na rysunku 1.



Rys. 1. Algorytm drobnoziarnisty i jego równoległe wykonanie
Fig. 1. Fine-grain algorithm and its distributed execution

W ramach prac prowadzonych przez autora opracowano metodę wyrównywania obciążenia dla takich algorytmów. Celem było zminimalizowanie strat wydajności systemu wynikających z różnic między mocą obliczeniową procesorów udostępnianą określonemu użytkownikowi. Ze względu bowiem na to, że węzły sieci przetwarzającej pracują pod kontrolą systemów operacyjnych wieloprogramowych (i wieloużytkownikowych) dzielących czas przetwarzania między procesy zgodnie z zasadą podziału czasu (ang. *time-sharing*), efektywna moc obliczeniowa przypadająca na jeden proces jest na każdym procesorze inna i dodatkowo może być zmienna w czasie.

3. Algorytmy LB

Problem wyrównywania obciążenia dla realizowanych równolegle algorytmów przetwarzania obrazów wyniknął z potrzeb Zespołu Komputerowych Systemów Wizyjnych IITiS PAN. Implementacja i badania wykonywane były w środowisku sieci stacji roboczych Sun połączonych siecią ATM i Ethernet.

Dla rozwiązania problemu w pierwszym etapie przyjęto następujące założenia:

1. Problem wyrównywania obciążenia jest w ogólnym przypadku NP-zupełny ([5]), więc należy zastosować algorytm heurystyczny.
2. Algorytm wyrównywania obciążenia powinien reagować na bieżące zmiany obciążenia węzłów w sieci, więc powinien być oparty na danych zbieranych na bieżąco w trakcie działania programu.

3. Zarówno bieranie danych, jak i część obliczeniowa algorytmu równoważenia obciążenia powinny wnosić minimalny narzut czasowy do działania programu.
4. Czas komunikacji w szybkiej sieci lokalnej można pominąć.

Na podstawie powyższych założeń opracowano pierwszą wersję algorytmu, zwaną dalej LB1. Jego ideę przedstawiono poniżej.

Przyjmijmy następujące nazewnictwo i oznaczenia:

zadanie – jednorazowe przetworzenie bloku danych w systemie (jedna pełna iteracja algorytmu),

podzadanie i – część zadania przydzielona do wykonania węzłowi *i*; $i = 1..k$, k – liczba węzłów przetwarzających w systemie,

zadanie elementarne – najmniejsze podzadanie przydzielane węzłowi; zadanie można podzielić na N niezależnych zadań elementarnych, które rozdzielane są między węzły obliczeniowe,

N – liczba zadań elementarnych w zadaniu,

n_i – rozmiar podzadania przydzielonego węzłowi (liczba zadań elementarnych),

n_i' – rozmiar następnego podzadania, które powinno być przydzielone węzłowi,

i – numer węzła przetwarzającego, $i = 1..k$,

t_i – całkowity czas przetwarzania podzadania na węźle i,

v_i – liczba zadań elementarnych przetwarzanych w jednostce czasu przez węzeł i.

Z powyższych definicji wynika zależność:

$$v_i = \frac{n_i}{t_i} \tag{1}$$

Obciążenie jest wyrównane, gdy:

$$t_1 = t_2 = \ldots = t_k = T \tag{2}$$

Na podstawie tego warunku możemy przyjąć, że:

$$n_i' = T\frac{n_i}{t_i} = Tv_i \tag{3}$$

W ten sposób podstawowym parametrem do obliczenia rozmiaru podzadania przypadającego na węzeł w następnej iteracji jest prędkość przetwarzania w poprzedniej iteracji. Dane do obliczenia obciążenia węzła uzyskujemy mierząc czasy wykonania podzadań na poszczególnych węzłach. Równocześnie łatwo zauważyć, iż idealny czas wykonania podzadania T wynosi:

$$T = \frac{N}{\sum_i v_i} \tag{4}$$

Ostatecznie liczba zadań elementarnych w następnej iteracji obliczana jest na podstawie następującego wzoru:

$$n_i' = v_i \frac{N}{\sum_{i=1}^{k} v_i} \tag{5}$$

Bardziej ogólny sposób obliczenia nowego rozmiaru podzadania wykorzystuje dane o prędkości przetwarzania nie z jednej, ale z m ostatnich iteracji. Prędkość v_i obliczamy wówczas jako:

$$v_i = \frac{\sum_{j=0}^{m-1} a_j \frac{n_{ij}}{t_{ij}}}{\sum_{j=0}^{m-1} a_j} \tag{6}$$

gdzie n_{ij} – rozmiar podzadania na węźle i w j-tej iteracji wstecz, t_{ij} – czas wykonania tego podzadania, natomiast a_j to współczynniki wagowe danych poszczególnych iteracji spełniające warunek:

$$p < q \Rightarrow a_p > a_q \tag{7}$$

Użycie takiego algorytmu dało obiecujące rezultaty (przykładowe wyniki zamieszczono poniżej). Jednak dokładniejsza analiza przebiegu wykonywania programów wykazała, że ostatnie z przyjętych przed opracowaniem algorytmu założeń (założenie 4 o pomijalnym czasie komunikacji) nie oddaje rzeczywistych warunków panujących w sieci – czas komunikacji stanowi od kilku do nawet 30% czasu przetwarzania i może mieć istotny wpływ na jakość wyrównywania obciążenia w tych zadaniach, w których ilość przesyłanych danych nie jest proporcjonalna do rozmiaru zadania. Na podstawie analizy działania przykładowych algorytmów do przetwarzania obrazów przyjęto, że dane przesyłane do węzła przetwarzającego można podzielić na dwie części – część stałą i część proporcjonalną. Część stała to dane, które muszą zostać przesłane w całości do każdego węzła niezależnie od rozmiaru podzadania przetwarzanego przez ten węzeł. Wielkość części stałej danych nie zależy więc od rozmiaru podzadania. Rozmiar części proporcjonalnej jest natomiast zależny od rozmiaru podzadania. Typowym przykładem takiego zadania, dla którego występuje podział danych na część stałą i proporcjonalną, jest opisany dalej algorytm wykrywania ruchu z wykorzystaniem pasowania klasterów w wersji rozproszonej. Każdy z węzłów musi otrzymać cały obraz niezależnie od liczby przetwarzanych klasterów – jest to część stała danych. Każdy węzeł otrzymuje również pewne dane liczbowe opisujące każdy z przetwarzanych klasterów – rozmiar danych jest zależny od liczby przetwarzanych

klasterów, a więc od rozmiaru podzadania. W związku z powyższym założenie 4 otrzymało treść:

4. Czas komunikacji między węzłami jest istotnym czynnikiem branym pod uwagę przez algorytm wyrównywania obciążenia. Na czas komunikacji składają się następujące elementy: czas przesłania części stałej danych (rozmiar niezależny od rozmiaru zadania), czas przesłania części proporcjonalnej danych (rozmiar proporcjonalny do wielkości podzadania), opóźnienie wnoszone przez sieć. Założenia upraszczające: prędkość przesyłu jest dla danego węzła taka sama dla obu części danych i taka sama w obu kierunkach.

Poniżej omówiono zmodyfikowany algorytm wyrównywania obciążenia uwzględniający prędkości komunikacji między węzłami. Dodatkowe użyte oznaczenia:

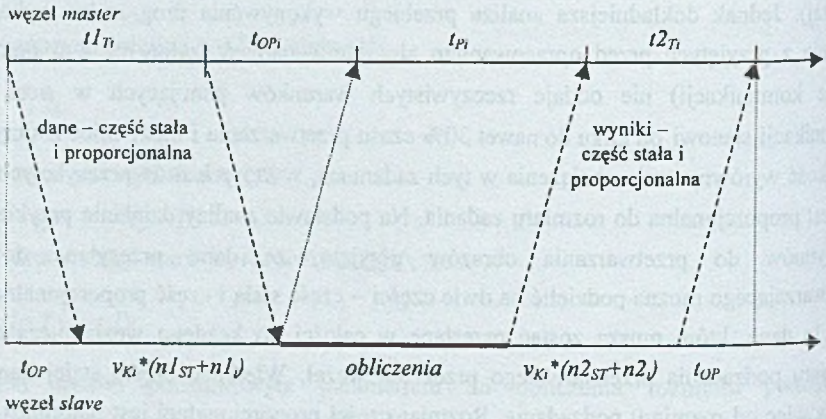
t_{Ki} – czas trwania komunikacji między węzłem nadzorującym a węzłem i,

t_{Pi} – czas trwania obliczeń na węźle i,

t_{OPi} – opóźnienie wnoszone przez sieć,

t_{Ti} – czas właściwego przesyłu danych,

n_{ST} – rozmiar części stałej danych (znormalizowany, gdy przyjmie się, że rozmiar danych dla jednego podzadania w części proporcjonalnej wynosi 1).



Rys. 2. Typowy przebieg komunikacji między węzłami master i slave
Fig. 2. Typical master-slave communication

Zgodnie z powyższymi oznaczeniami mamy:

$$t_i = t_{Ki} + t_{Pi} \tag{8}$$

$$t_i = t_{OPi} + t_{Ti} + t_{Pi} \tag{9}$$

Na tej podstawie uzyskujemy sposób na obliczenie obciążenia węzła w następnej iteracji, korzystając z danych dostępnych do pomiarów:

$$n_i' = \frac{\dfrac{N + \sum_i \left(B_i / A_i \right)}{\sum_i \left(1 / A_i \right)} - B_i}{A_i}, \text{ gdzie } A_i = \frac{t_{Ti}}{n_C + n_i} + \frac{t_{Pi}}{n_i}, \quad B_i = n_C \frac{t_{Ti}}{n_C + n_i} \tag{10}$$

Dane pomiarowe wykorzystywane w powyższym wzorze wymagają – w porównaniu z programem działającym bez wyrównywania obciążenia - wykonania dla każdej iteracji i dla każdego węzła jednej dodatkowej transmisji (potwierdzenie odebrania danych) nie powodującej narzutu czasowego, gdyż – jak widać na rys. 1 – transmisja potwierdzenia odbywa się równolegle do obliczeń. Tym samym można uznać, że założenie 3 (o minimalnym narzucie algorytmu wyrównywania obciążenia) dla części komunikacyjnej jest spełnione w stopniu zadowalającym.

4. Implementacja

Algorytm wyrównywania obciążenia zaimplementowano w formie biblioteki funkcji przeznaczonej do wykorzystania w programach tworzonych w języku C. Biblioteka ma postać zestawu procedur, które można podzielić na dwie grupy - procedury komunikacyjne i pozostałe. Procedury komunikacyjne korzystają z bibliotek MPI [1] i udostępniają podstawowy podzbiór funkcji standardu. Ich rolą w ramach działania algorytmu wyrównywania obciążenia jest zbieranie danych o czasach trwania tych fragmentów programu, które są istotne dla algorytmu. W drugiej grupie, poza procedurami o charakterze pomocniczym (np. inicjalizacja i zamknięcie biblioteki), znajdziemy procedurę obliczającą obciążenie węzłów (rozmiary podzadań) w następnej iteracji.

Bibliotekę można wykorzystać do optymalizacji wykonania programów rozproszonych działających w sposób przedstawiony na rysunku 1 i opisanych w rozdziale 2. Osoba chcąca wykorzystać bibliotekę do optymalizacji wykonania rozproszonej wersji swojego programu jest odpowiedzialna za przygotowanie takiej wersji. Biblioteka umożliwia tylko – przez

optymalizację wykorzystania węzłów przetwarzających – przyspieszenie wykonania programu dzięki lepszemu wykorzystaniu czasu procesorów w węzłach obliczeniowych.

5. Zastosowania

Algorytmy równoważenia obciążenia wykorzystano dla kilku algorytmów przetwarzania obrazów. Poniżej szczegółowo zaprezentowano dwa z nich.

5.1. Wykrywanie ruchu z wykorzystaniem pasowania klasterów

Algorytm wykrywania ruchu z wykorzystaniem pasowania klasterów [6] można podzielić na dwie autonomiczne części. W pierwszej części dzielimy obraz na klastery – grupy pikseli o zbliżonych cechach. Przestrzeń cech jest trójwymiarowa – dwa wymiary położenia oraz jasność. Oznacza to, że klaster grupuje punkty położone blisko siebie i mające zbliżoną jasność. W drugiej części algorytmu wykonujemy dopasowanie klasterów z dwóch obrazów.

Pierwsza część algorytmu to typowy przypadek należący do klasy algorytmów opisanych w rozdziale 2. Na początku obraz dzielony jest na K prostokątnych regionów – klasterów. W kolejnych iteracjach obliczamy parametry klasterów – środek ciężkości i średnią jasność, a następnie przyporządkowujemy każdy piksel obrazu do jednego z klasterów – do tego, do którego odległość euklidesowa jest najmniejsza. Działanie algorytmu kończy się w chwili, kiedy w kolejnych iteracjach odpowiednio niewielka liczba punktów zmienia przynależność do klasterów.

W algorytmie podziału obrazu na klastery mamy dwie operacje, które mogą zostać rozproszone. Pierwsza to obliczanie parametrów klasterów – dla każdego klastera oblicza się je niezależnie i łatwo jest rozdzielić to zadanie między węzły obliczeniowe. Operacja elementarna to obliczenie parametrów jednego klastera. Drugą rozpraszaną operacją jest przyporządkowywanie punktów klasterom. Znając położenie klasterów w trzech wymiarach (współrzędne środka ciężkości oraz średnia jasność), możemy dla każdego punktu obrazu obliczyć jego odległość euklidesową do każdego klastera i przyporządkować punkt temu klasterowi, dla którego odległość ta jest najmniejsza. Dla tej części algorytmu jako operację elementarną wybrano obliczenie odległości dla pikseli w jednej linii obrazu (zazwyczaj 256, 512 lub 1024 piksele).

5.2. Odnajdywanie konturów na obrazie z wykorzystaniem konturu aktywnego

Technika odnajdywania konturów na obrazie z wykorzystaniem konturu aktywnego wykorzystuje schemat minimalizacji energii [7,8]. Kontur ma dwie funkcje energetyczne o wartościach zależnych od jego chwilowego stanu: energię wewnętrzną i energię zewnętrzną. Energia wewnętrzna mierzy pożądane własności kształtu konturu, takie jak gładkość i ciągłość. W typowym przypadku energia jest zmniejszana przez zbliżanie punktów konturu i wygładzanie jego krawędzi. Energia zewnętrzna jest określana na podstawie pewnych cech obrazu, takich jak krawędzie, regiony, linie, tekstury. Kiedy kontur znajduje się bliżej takich elementów obrazu, energia jest mniejsza. Algorytm dąży do osiągnięcia minimalnej wartości energii całkowitej dla konturu. Jej minimalizacja może być widziana jako wieloetapowy proces decyzyjny. Dla każdego punktu należy sprawdzić wszystkie możliwe pozycje w określonym sąsiedztwie i wybrać najlepszą nową pozycję. Problem składa się więc z pewnej liczby problemów NP-zupełnych i jest bardzo czasochłonny. Jednocześnie obliczenia dla każdego punktu konturu są wykonywane niezależnie i całe zadanie może zostać zrównoległone. Ponieważ operacje minimalizacji energii przeprowadza się w iteracjach tak długo, aż zmiany kształtu konturu nie będą mniejsze od założonych, widać, że algorytm należy do klasy, dla której możemy zastosować opisaną wyżej technikę wyrównywania obciążenia.

6. Wyniki

Dla algorytmu odnajdywania konturów – jak można było przewidzieć – zysk czasowy ze zmiany algorytmu równoważenia z LB1 na LB2 był bardzo niewielki. Wynika to z faktu, że liczba danych do przesłania jest poza jednorazową inicjalizacją stosunkowo niewielka (po jednorazowym przesłaniu obrazu do wszystkich węzłów wymieniane są tylko informacje o współrzędnych punktów konturu). Inna sytuacja zachodzi dla algorytmu klasteryzacji obrazu (pierwsza część opisanego w rozdziale 5.1 algorytmu wykrywania ruchu).

Poniżej przedstawiono przykładowe wyniki czasowe dla algorytmu klasteryzacji. Badania wykonano w sieci stacji roboczych, wykorzystując komputer PC z procesorem Pentium 166 MHz (komputer 0), 3 komputery Sun Ultra II z procesorem UltraSPARC 166 MHz (komputery 1, 2 i 3) oraz komputer Sun Ultra 1 z procesorem UltraSPARC 133 MHz (komputer 4). Komputery 1, 2 i 3 połączone były ze sobą siecią ATM 155 MB, natomiast z komputerami 0 i 4 siecią Ethernet 10 MB.

	bez równoważenia	LB1	LB2
Master – 0, slaves – 1, 3, 4			
	6,5660	6,1388	5,8112
Master – 0, slaves – 1, 2, 3			
	6,1302	5,9135	5,8264
Master – 3, slaves – 1, 2, 3			
	12,8629	7,0840	6,4212
Master – 4, slaves – 1, 2, 3			
	5,7464	5,7183	5,7203
Master – 4, slaves – 1, 2, 3, węzeł 3 obciążony przez inne aplikacje			
	9,5174	6,9470	6,3576
Master – 2, slaves – 2, 2, 2			
	22,2591	23,0261	23,0815
Master – 0, slaves – 0, 1, 3			
	7,0525	6,8193	6,3308

Powyższe rezultaty pokazują, że quasi-dynamiczne równoważenie obciążenia daje dobre rezultaty w sieci stacji roboczych.

LITERATURA

1. „MPI: A Message Passing Interface Standard", Message Passing Interface Forum, University of Tenesee, Knoxville 1994.
2. Geist A. et al.: PVM: Parallel Virtual Machine, MIT Press, Cambridge 1994
3. Brudło P. E.: Fully distributed task processing in local area networks, PhD Thesis, Technical University of Gdansk, 1995.
4. Drozdowski M.: Selected problems of Scheduling Tasks in Multiprocessor Computer Systems, Instytut Informatyki, Politechnika Poznańska 1997.
5. Keyser J., Lust K., Roose D.: Run time load balancing support for a parallel multiblock Euler/Navier-Stokes code with adaptive refinement on distributed memory computers, Parallel Computing, 1994, 20.
6. Kottke D. P., Sun Y.: Motion Estimation via Cluster Matching, IEEE Trans. PAMI vol.16, no.11, 1994.
7. Kass M., Witkin A., Terzopulos D.: Snakes: Active Contour Models, International Journal of Computer Vision, no.1, 1988.
8. Amini A., Weymouth T. E., Jain R. C.: Using Dynamic Programming for Solving Variational Problems in Vision, IEEE Trans. PAMI, vol.12, no.9, 1990.

Recenzent: Doc. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 31 marca 1999 r.

Abstract

The paper presents author's approach to a problem of load balancing in generally defined environment: multi-program and multi-user, with changing load of processing units and communication links. A heuristic solution for iterative fine-granularity algorithms was found. The quasi-dynamic load-balancing algorithm computes load of processing node in next iteration on the basis of performance measurements in previous iteration. Performance of processing nodes is considered as well as performance of communication links.