Seria: INFORMATYKA z. 28 Nr kol. 1260

Andrzej KRÓL, Andrzej USZOK, Krzysztof ZIELIŃSKI

RÓWNOWAŻENIE OBCIĄŻENIA W OBIEKTOWYCH SYSTEMACH ROZPROSZONYCH, PODEJŚCIE ANSA [1]

Streszczenie. W niniejszym artykule przedstawiono realizację rozszerzenia systemu ANSA o mechanizmy równoważenia obciążenia. Własności tak zmodyfikowanego systemu przedstawiono na przykładzie realizacji obliczeń farmer-worker.

LOAD BALANCING IN OBJECT DISTRIBUTED SYSTEMS, ANSA APPROACH

Summary. In this paper, the extension of ANSA with load balancing mechanisms has been described. The features of modified version of the system bas been presented by the farmer-worker computational example.

EQUILIBRE DE CHARGE DANS LES SYSTEMS REPARTIS OBJECT ORIENTE, APPROACHE ANSA

Resumé. Dans l'article la réalisation d'une extention du systéme ANSA des outils pour l'équilibre de charge des systèmes. Les proprietés de système proposé est verifié sur l'example d'execution de farmer-worker algorithme de la repartition des tâches.

[1] Pracę wykonano w ramach grantu KBN nr 8 S503 015 06

1. Wstęp

Dynamicznie rozwijająca się obecnie dziedzina programowania obiektowego w rozproszonych systemach komputerowych uwidacznia konieczność wprowadzenia zarządzania takimi systemami. Jeden z aspektów zarządzania obiektowymi systemami rozproszonymi dotyczy problemu równoważenia obciążenia (ang. Load Balancing (**LB**)) i nierozłącznie z tym związanego problemu monitorowania systemów rozproszonych.

Równoważenie obciążenia może dotyczyć zagadnienia tworzenia nowych obiektów zgodnie z informacją o stanie systemu (LB statyczny), lub problemu migracji obiektów z bardziej obciążonych węzłów na te, które są w danej chwili mniej obciążone (LB dynamiczny). Prezentowane w artykule rozwiązanie dotyczy statycznego równoważenia obciążenia, tzn. alokacji tworzonych obiektów na podstawie informacji o stanie systemu, dostarczonej przez system monitorowania.

W obiektowo zorientowanych systemach rozproszonych monitorowanie może dotyczyć zarówno wewnętrznego zachowania się obiektu jak i zachowania zewnętrznego manifestującego się np. wykorzystaniem procesora, pamięci, czy też liczby wysłanych komunikatów.

Celem niniejszego artukułu jest przedstawienie implementacji systemu równoważenia obciążenia wraz z systemem monitorowania zewnętrznego, napisanego z uwzględnieniem specyfiki zarządzania obiektowo zorientowanymi systemami rozproszonymi. Jako platformę implementcji wybrano system ANSA [1], tak więc rozważany system równoważenia obciążenia **LB** stanowi sam obiektową aplikację rozproszoną. Punktem wyjścia dla prezentowanej implementcji systemu **LB** jest model funkcjonalny omówiony krótko w punkcie 2. Następnie w punkcie 3 podano sposób współpracy tego systemu z monitorem obciążenia **LM**. W punkcie 4 przedstawiono sposóby włączania mechanizmów **LB** do aplikacji ANSA. Wyniki eksperymentalnej weryfikacji przyjętej koncepcji zawiera punkt 5.

2. Model funkcjonalny systemu LB

Rozważając możliwości rozwiązania problemu **LB** dla systemu ANSA należy brać pod uwagę charakterystyczne cechy tego systemu:

- Jest to system zorientowany obiektowo, podstawowym elementem działającej aplikacji jest kapsuła (*capsule*), reprezentowana w systemie operacyjnym przez pojedyńczy proces.

- Kapsuła zawiera wiele obiektów, które udostępniają interfejsy do oferowanych przez siebie usług.

- System ANSA umożliwia uruchamianie wielu kapsuł na dowolnych węzłach systemu w postaci serwerów.
- Wszystkie interfejsy usług zarejestrowane są w specjalnej kapsule systemowej *Trader*, są stąd eksportowane przez klientów. Możliwy więc jest pewien sposób zarządzania wyborem potrzebnej usługi z pewnego zbioru identycznych usług, które udostępniane są przez kapsuły działające na różnie obciążonych węzłach.
- System ANSA nie udostępnia, jak na razie, przeźroczystego mechanizmu migracji kapsuł, co utrudnia skonstruowanie mechanizmów **LB** dla tego systemu.

Ponieważ aplikacja w systemie ANSA posiada budowę trój-poziomową składającą się z kapsuł, obiektów i interfejsów, zatem mechanizmy **LB** mogą dotyczyć każdego z nich. Czasy podstawowych operacji ANSA związanych z budowaniem i niszczeniem kapsuł zaprezentowano w Tabeli 1.

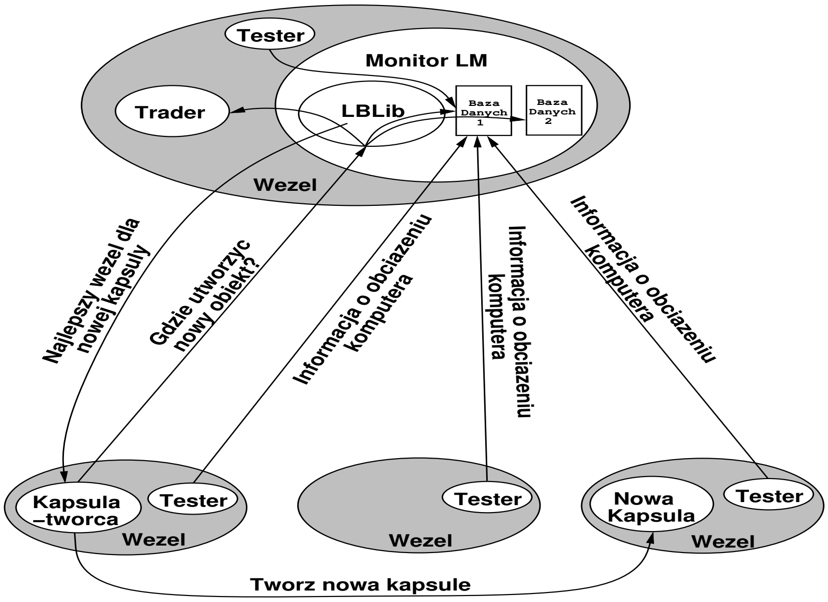
Tabela 1: Średnie czasy wykonania operacji ANSA na sieci komputerów SUN.

Nazwa Operacji	**średni czas wykonania [ms]**
Eksport interfejsu usługi Factory	24
Tworzenie nowej kapsuły	318
Niszczenie kapsuły (operacja Terminate)	49
Tworzenie nowego obiektu wewnątrz kapsuły	33
Tworzenie całej struktury (kapsuła, obiekt, interfejsy)	381
Niszczenia takiej struktury	54
Operacja Discard dla interfejsu	1
Operacja Create dla interfejsu	1
Operacja Destroy dla interfejsu	0.5
Operacja Importu referencji do interfejsu usług	12
Operacja Exportu referencji do interfejsu usług	22

Wyniki te uzyskano poprzez wielokrotne (400 pomiarów) wywołania wymienionych operacji w systemie ANSA pracującym w następującej konfiguracji: kapsuła usługi *Trader* zainstalowana była na komputerze SUN SPARCstation 2, system ANSA funkcjonwał dodatkowo na trzech komputerach SUN.

Najbardziej kosztowną operacją jest utworzenie nowej kapsuły i czas ten wynosi średnio 0.3[s]. Czas ten rośnie jednak nawet do ponad 6[s] na bardzo obciążonym komputerze. Tak więc decyzja o tworzeniu kapsuł musi być podejmowana szczególnie ostrożnie. Pozostałe czasy też nie są pomijalnie małe, jeśli operacje te wywoływane są bardzo często.

Zaproponowane w dalszej części artykułu mechanizmy **LB** umożliwiają podejmowanie decyzji odnośnie tworzenia i lokalizacji elementów systemu z każdego z wymiennionych wcześniej poziomów. Koncepcja działania proponowanego systemu **LB** zakłada, że przed utworzeniem nowego elementu systemu istnieje możliwość uzyskania informacji o stanie obciążenia systemu. W odniesieniu do poziomu kapsuł rozważany schemat działania został przedstawiony na rys. 1.



Rysunek 1: System równoważenia obciążenia.

Jego realizacja wymaga rowiązania dwu problemów: (i) konstrukcji systemu monitorowania stanu systemu **LM**, oraz (ii) sposobu wprowadzenia do kodu aplikacji odwołań do tego systemu. Zagadnienia te zostaną przeanalizowane w kolejnych punktach.

3. System monitorowania LM

Strukturę systemu monitorowania zewnętrznego jaką przyjęto w rozważanym systemie zaznaczono na rys. 1. Składa się on z dwu rodzajów kapsuł: kapsuły `Monitor` oraz zbierających dane kapsuł typu `Tester`. Kapsuły te są wielowątkowe. Umożliwia to współbieżną obsługę zdarzeń występujących w systemie. Szczegóły implementacyjne systemu LM zostały opisane w [2]. Z punktu widzenia systemu **LB** najważniejsze jest działanie kapsuły `Monitor`. Jest ona uruchamiania na samym początku pracy systemu **LM**. Wynikiem jej działania jest: (i) Uruchomienie na każdym węźle tego systemu kapsuły `Tester`, zbierającej dane na temat obciążenia komputera, na którym działa. (ii) Inicjacja bazy danych zawierającej informacje na temat stanu komputerów systemu ANSA. Polega to na inicjacji elementów tablicy *Nodes* na podstawie pliku zawierającego dane na temat zasobów poszczególnych węzłów systemu. (iii) Wyeksportowanie referencji do interfejsów, tak aby mogły one być dostępne dla innych aplikacji ANSA. (iv) Instalacja funkcji pozwalającej na zakończenie pracy systemu. Po zakończeniu wykonywania procedury inicjacji, system monitowania pracuje, a zbierane przez niego informacje są dostępne dla wszystkich aplikacji ANSA, które dokonają importu referencji jego interfejsów `SlmAcd` lub `LBLib` kapsuły `Monitor`.

Interfejs `LBLib` oferuje operacje umożliwiające programom dostęp do zebranych przez kapsułę `Monitor` danych w formie specjalnie opracowanej dla systemu **LB** i ma postać:

```
LBLib : INTERFACE =
NEEDS SlmDc;
BEGIN
-- Operations
BestNode      : OPERATION [ Dist: INTEGER ]
                  RETURNS [ HName: STRING ];
GetHostList   : OPERATION [ ]
                  RETURNS [ Hosts: HostsList ];
GoodNodes     : OPERATION [ Dist: INTEGER; Delim: INTEGER ]
                  RETURNS [ Hosts: HostsList ];
NewGoodNodes  : OPERATION [ Hosts: HostsList; Dist: INTEGER; Delim: INTEGER ]
                  RETURNS [ NewHosts: HostsList  ];
BadNodes      : OPERATION [ Hosts: HostsList; Dist: INTEGER; Delim: INTEGER ]
                  RETURNS [ BadHosts: HostsList ];
StillGoodNode : OPERATION [ Host: STRING; Dist: INTEGER; Delim: INTEGER ]
END.
```

Interfejs ten może być łatwo rozszerzony, a zawarte w nim przykładowe funkcje mają następującą interpretacje:

- *BestNode* – funkcja ta zwraca nazwę najlepszego, według aktualnie posiadanych informacji, komputera wyszukanego w zbiorze węzłów systemu ANSA przy pomocy algorytmu o nazwie będącej argumentem tej funkcji.
- *GetHostList* – funkcja ta zwraca listę nazw dostępnych komputerów
- *GoodNode* – funkcja ta zwraca listę nazw komputerów, których obciążenie jest mniejsze od określonego przez argument jej wywołania.
- *NewGoodNodes* – funkcja ta wyznacza, które komputery, oprócz podanych w jej wywołaniu, są mniej obciążone niż określa to jeden z argumentów jej wywołania.
- *BadNodes* – funkcja ta bada, które z podanych komputerów są bardziej obciążone niż określa to argument jej wywołania. Można ją wykorzystać do badania, które komputery zwrócone poprzednio przez wywołanie funkcji *GoodNodes* utraciły swoje własności.
- *StillGoodNodes* – funkcja ta sprawdza czy obciążenie komputera o podanej nazwie jest mniejsze od podanej wartości.

Wymienione operacje mogą być wywołane przez aplikacje ANSA w celu wyznaczenia najbardziej odpowiedniej lokalizacji nowo tworzonego elementu systemu.

4. Sposoby korzystania z mechanizmów LB

Rozważając sposób włączania strategii **LB** do tekstów programów należy rozważyć dwie możliwości:

- Przeźroczyste (*transparent*) włączanie mechanizmów **LB**, polegające na tym, że twórca aplikacji nie musi być świadomy ich budowy, sposobu działania a nawet istnienia. Natomiast dowolny stworzony przez niego program powinien móc automatycznie korzystać z tych mechanizmów.
- Bezpośrednie dołączanie mechanizmów **LB**. W tym przypadku cała inicjatywa leży po stronie programisty, od którego decyzji zależy sposób włączania mechanizmów **LB** do tekstu aplikacji.

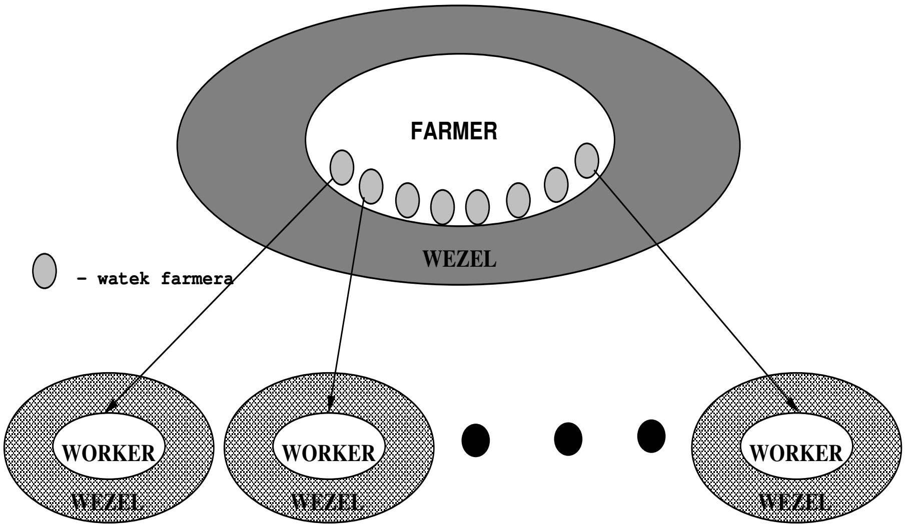
W przypadku systemu ANSA brak przeźroczystego mechanizmu włączania strategii **LB** do programów bardzo utrudnia pracę ich twórcom. Programy ANSA są to najczęściej dość rozbudowane aplikacje, składające się z wielu plików definiujących kapsuły i ich interfejsy. Cały program opisywany jest natomiast przez odpowiednio skonstruowany plik *imakefile*. Aby przedstawiony mechanizm **LB** uczynić przeźroczystym należy napisać taki preprocesor pliku *imakefile* aby włączanie strategii **LB** do aplikacji polegało na uruchomieniu tylko tego jednego przekształcającego go programu.

Mechanizmy bezpośrednie opierają się na wykorzystaniu operacji dostarczanych przez monitor **ML** wywoływanych wprost przez programistę, podobnie jak funkcje biblioteczne. W rozważanym systemie zapewniono obydwa mechanizmy korzystania z systemu **LB**.

5. Eksperymentalne badania mechanizmów LB

Doświadczalna weryfikacją zaprezentowanych mechanizmów **LB** jest ostatecznym kryterium ich działania i efektywności. W tym punkcie punkcie zaprezentowane zostaną wyniki doświadczeń jakie przeprowadzono z opracowanym modułem **LB**. Jako aplikację, do której wprowadzono mechanizmy **LB** wybrano model obliczeń rozproszonych *farmer – worker*. Jest to typowy i dość powszechnie stosowany model organizacji przetwarzania w środowisku rozproszonym [5].

Typową budowę modelu *farmer - worker* przedstawiono na Rys. 2. Na jednym z węzłów pracuje proces zarządzający - *farmer*, który kieruje obliczeniami, tzn. przekazuje *worker'om* kolejne partie danych i odbiera od nich wyniki ich pracy. Kapsuła *FARMER* zawiera współbieżne wątki, kontrolujące pracę każdej kapsuły *WORKER* z osobna.



Rysunek 2: Model obliczeń *farmer - worker*

W czasie opisanych doświadczeń obliczaniami, które wykonywała farma procesów, były obliczenia fraktala Mandelbrota. Dla potrzeb rozważanego eksperymentu nie jest istotne oczywiście co jest obliczane, lecz sam proces obliczeniowy, a zaprezentowana implementacja farmy jest ogólna. Ważne, jest natomiast, że można sterować wielkością pojedyńczego zadania kierowanego do kapsuły *WORKER* zmieniając ilość, punktów obrazu fraktala, które ma ona obliczyć. Szczególną uwagę zwrócono na ilustrację przeźroczystej oraz bezpośredniej metody włączania mechanizmów **LB**.

5.1. Przeźroczyste włączanie mechanizmów LB

Wprowadzenie mechanizmów **LB** w dskutowanym przykładzie jest możliwe na etapie tworzenia kapsuł typu *WORKER*. Obecnie zostanie pokazane w jaki sposób dokonuje tego skonstruowany w tym celu preprocesor. Rozważana aplikacja składa się z kilku odddzielnych plików powiązanych w jedną całość odpowiednim plikiem *imakefile*. Efekt działania preprocesora zostanie pokazany na przykładzie funkcji *TworzKapsuleWorkera*, tworzącej nowe kapsuły *WORKER* W pliku wejściowym funkcja ta miała następującą postać:

```
void TworzKapsuleWorkera(wstat)
StrukturaWorker *wstat;
{
```

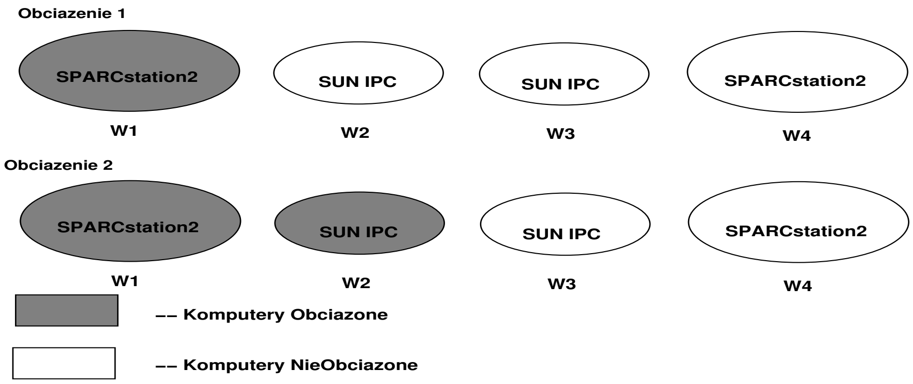
```
 ansa_Boolean r;
/* Importuj referencje do uslugi Factory */
! {FacRef} <- traderRef$Import("Factory", "/", "")
/* Tworz kapsule Worker na odpowiednim wezle */
! {CapRef, Cid} <- FacRef$Instantiate \
      (WorkerPATH, "server", "", "", nullRef, 0)
/* Tworz obiekt w tej kapsule inportujacy interfejs WORKER */
! {ObjRef, Res} <- CapRef$Instantiate(nullRef, nullRef,\
          "Worker", "", "")
}
```

Po wykonaniu preprocesingu jej postać uległa zmianie. Włączone zostały do niej nowe instrukcje i wywołanie funkcji *BestNode* interfejsu `LBLib`. Otrzymana postać jest następująca:

```
/* Funkcja tworzaca nowa kapsule workera */
void TworzKapsuleWorkera(wstat)
StrukturaWorker *wstat;
 {
   ansa_Boolean r;
   /* Importuj referencje do Factory */
   if(strstr(propbuf, "Node"))
!  {FacRef} <- traderRef$Import("Factory", "/", "")
    else{
         char pbuf[1024];
         ansa_String wezel;
         if(!_LBLib.length)
!        {_LBLib} <- traderRef$Import("LBLib", "", "")
!        {wezel} <- _LBLib$BestNode(NrAlg)
         (void)strcpy(pbuf, propbuf);
         (void)strcat(pbuf, " Node==’");
         (void)strcat(pbuf, wezel);
         (void)strcat(pbuf, "’");
!        { Facref } <- traderRef$Import("Factory", dir, pbuf)
    }
   /* Tworz kapsule Worker na odpowiednim wezle */
!  {CapRef, Cid} <- FacRef$Instantiate \
          (WorkerPATH, "server", "", "", nullRef, 0)
  /* Tworz obiekt w tej kapsule eksportujacy interfejs WORKER */
!  {ObjRef, Res} <- CapRef$Instantiate(nullRef, nullRef,\
          "Worker", "", "")
}
```

Tak więc przed każdym utworzeniem nowej kapsuły *WORKER* uzyskiwana jest najpierw informacja na jakim węźle powinien być on utworzony. Należy podkreślić, że opisany preprocesor jest ogólny w tym sensie, że może przetwarzać dowolne aplikacje. Wykrywa on instrukcje tworzenia kapsuł i podmienia na odpowiednio zmodyfikowany kod zawierający odwołania do operacji **LB**.

Aby pokazać wpływ wprowadzonych mechanizmów **LB** zdecydowano się przeprowadzić uproszczone eksperymenty na systemie, w którym sztucznie wywoływano stałe obciążenie komputerów na dwa sposoby, pokazana na rys. 3. Starano się ocenić maksymalny zysk ze stosowania mechanizmów **LB** – aplikacja bez wprowadzonych do niej mechanizmów **LB** wybierała najgorsze rozłożenie kapsuł *WORKER* w systemie, natomiast z mechanizmem **LB** najlepsze. Eksperymenty przeprowadzano w systemie ANSA o konfiguracji identycznej jak opisana w punkcie 2.



Rysunek 3: Rozkłady obciążenia systemu

Tabela 2: Czasy wykonania obliczeń z i bez mechanizmów **LB**

Rozkład obciążgnie	**Ilosc** *workerów*	**Obliczenia z LB** W1	W2	W3	W4	**Czas[s]**	**Obliczenia bez LB** W1	W2	W3	W4	**Czas[s]**
1	1				X	53.08	X				103.45
	2			X	X	33.99	X	X			56.83
	3	X		X	X	25.08	X	X	X		36.09
	4	X	X	X	X	22.93	X	X	X	X	22.93
2	1				X	53.08		X			143.46
	2			X	X	33.99	X	X			71.25
	3	X		X	X	27.66	X	X	X		45.25
	4	X	X	X	X	25.91	X	X	X	X	25.91

W tabeli 2 przedstawiono uzyskane rezultaty eksperymentu. Obecność symbolu X w danej kolumnie oznacza, że na tym węźle pracowała kapsuła *WORKER*. Analiza tych wyków pozwala stwierdzić poprawność przyjętej koncepcji działania systemu **LB** w tym sensie, iż jego zastosowanie może prowadzić do zanaczącego skrócenia czasów obliczeń.

5.2. Bezpośrednia metoda włączania mechanizmów LB

Metoda ta polega na włączanie wywołań funkcji biblioteki *LBLib* do tekstu programu przez jego twórcę i jest przydatna w sytuacji gdy równoważenie obciążenia wymaga wykorzystania informacji o semantyce programu. W rozważamym przykładzie zdecydowano się na sterowaniem ilością elementarnych zadań przekazywanych *worker'om*, na podstawie obciążenia węzła, na którym działają. W aplikacji wyjściowej odpowiedzialna jest za to funkcja *PrzeslijWorkerowiNoweDane* o postać:

```
void  PrzeslijWorkerowiNoweDane(NrWorkera, Dane, NrAdresuBuforaNaWyniki)
int NrWorkera;
Data Dane;
Result *NrAdresuBuforaNaWyniki;
 {
 /* Dopoki sa dane do obliczen */
 while(DaneDoObliczen())
 /* Przeslij nastepna paczke punktow do obliczen */
!  {} <- worker[NrWorkera]$Obliczenia(Dane, NrAdresuBuforaNaWyniki)
 }
```

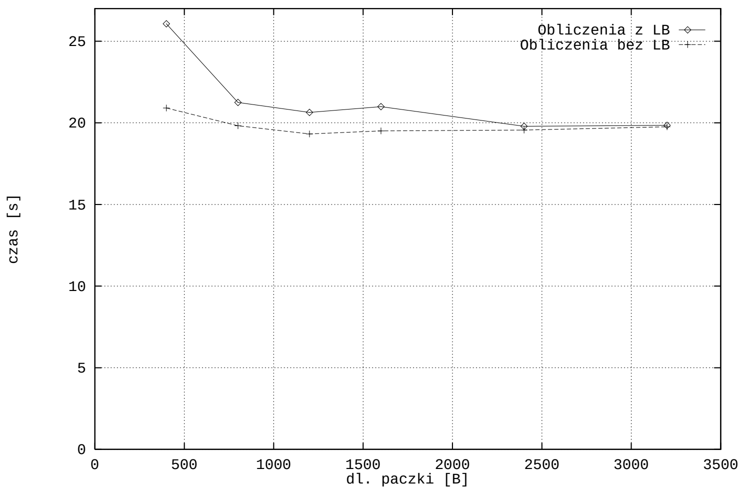
Wprowadzono do niej pętlę czekającą, aż obciążenie danego węzła spadnie do poziomu kiedy opłacalne będzie wysłanie do *worker'a* rezydującego na nim nowego zadania. Badana jest najpierw średnia długość kolejki procesów w ciągu ostatniej minuty, jeśli jest ona mniejsza niż 1.2 to dane przesyłane są do węzła. Jeśli jednak jest ona większa, to sprawdzane jest aktualne wykorzystanie procesora, jeśli jest ono mniejsze niż 65% to również dane są przesyłane. Jeśli jednak jest ono większe to wątek kontrolujący danego *worker'a* zasypia na pewien czas (3[s]), czekając na zmianę stanu komputera, a następnie powtarza opisane czynności. Podane parametry granicznego obciążenia wybrano na podstawie obserwacji zachowania się systemu podczas wielu tego typu eksperymentów.

```
void  PrzeslijWorkerowiNoweDane(NrWorkera, Dane, NrAdresuBuforaNaWyniki)
int NrWorkera;
Data Dane;
Result *NrAdresuBuforaNaWyniki;
 {
  ansa_Boolean rr;
   /* Eksportuj referencje LBLib */
! {LBLRef} <- traderRef$Import("LBLib", "/", "")
   /* Dopoki sa dane do obliczen */
   while(DaneDoObliczen())
   /* Czekaj dopuki wezel nie bedzie spelnial warunki */
```

```
       do{
!       {rr} <- LBLRef$StillGoodNode(worker[NrWorkera].name, 5, 120)
        if(!rr){
!        {rr} <- LBLRef$StillGoodNode(worker[NrWorkera]name, 2, 65)
        if(!rr)
          sleep(3);
       }
     }while(!rr);
        /* Przeslij nastepna paczke punktow do obliczen */
!     {} <- worker[NrWorkera]$Obliczenia(Dane, NrAdresuBuforaNaWyniki)
    }
```

W programie tym w dość intensywny sposób wykorzystuje się operacje interfejsu *LBLib*. Należy zatem oszacować jakie koszty czasowe się z tym wiążą. Badania przeprowadzano w czasie gdy komputery nie były wykorzystywane tzn. pracowały na nich jedynie kapsuły systemowe ANSA i aplikacji *farmer – worker*.

Mierzono czas obliczeń z mechanizmem **LB** i bez niego, przyczym zmieniano wielkość zadania przesyłanego do *worker'a*. modyfikując ilość przekazywanych mu do obliczeń punktów. Zmieniano w ten sposób jednocześnie ilość odwołań do interfejsu *LBLib*. Średnie z otrzymanych wyników przedstawiono na rys. 4.



Rysunek 4: Koszty mechanizmu **LB**

Jak można zaobserwować dla małej ilości punktów, czyli w przypadku gdy zadanie obliczeniowe było małe, a jednocześnie bardzo często wywoływano funkcje *LBLib* to koszty mechanizmu **LB** były wysokie, jednakże wraz z wzrostem rozmiarów zadania koszty te znacznie malały.

Wyniki zastosowania rozważanego w tym punkcie mechamizmu **LB** dla obciążenia typu 2 z rys. 3 przedstawiono w tabeli 3. Eksperyment ten przeprowadzono w sposób podobny jak w poprzednim punkcie dla zadań obliczających 800 punktów.

Tabela 3: Porównanie ilości zadań przydzielonych poszczególnym węzłom

Nazwa	**Ilość zadań przydzielonych węzłom**	
węzła	**Obliczenia z LB**	**Obliczenia bez LB**
W1	103	150
W2	89	115
W3	136	128
W4	272	207

Średni czas wykonania się obliczeń w wariancie bez **LB** wyniósł: 26.64[s], natomiast wariant z **LB** miał średni czas zakończenia: 24.01[s]. Czyli poniesione koszty na wprowadzenie mechanizmy **LB** były celowe. Wynika to z faktu, iż poszczególne kapsuły *WORKER* otrzymywały ilość zadań dostosowaną do obciążenia węzła, na których pracowały.

W przypadku zmiennego obciążenia węzłów, algorytm **LB** wybiera najmniej obciążone w danym momencie węzły, do realizacji kolejnych kroków obliczeń (tabela 4).

Tabela 4: Czasy obliczeń dla zmiennego obciążenia wezłów

wielkość paczki ilość punktów	**Obliczenia z LB [s]**	**Obliczenia bez LB [s]**
800	41.24	41.98
2000	34.87	36.52
4000	33.43	40.82
8000	32.44	38.85

6. Podsumowanie

Przeprowadzone badania wykazały możliwość i celowość uzupełnienia systemu ANSA o mechanizmy statycznego równoważenia obciążenia. Stwierdzono przy tym, że ważne jest uwzględnienie trój-poziomowej hierachii elementów tego systemu. Zaprezentowano architekturę systemu **LB** oraz podano opis jego najważniejszych elementów składowych. Zaproponowano i praktycznie zweryfikowano dwie metody włączania mechanizmów **LB** do aplikacji ANSA wskazując, iż bardziej ogólna i celowa jest technika preprocesingu.

Uzyskane wyniki eksperymentalne posiadają raczej charakter jakościowy a nie ilościowy, gdyż te ostatnie zależą od konkretnej aplikacji. Znacznie większe znaczenie posiadają ogólne cechy powstałej rozszerzonej architectury ANSA i przyjęta metodologia, którą można przenieść na inne środowiska rozproszone.

LITERATURA

[1] *ANSAware 4.0 — System Programmer's Manual.* APM Ltd. Cambridge (1992).

[2] A. Król, A. Uszok, K. Zieliński *Monitorowanie obiektowo zorientowanych aplikacji rozproszonych, podejście ANSA.* Konferencja Informatyka na wyższych uczelniach dla gospodarki narodowej, Gdańsk, 17-19 listopada (1994).

[3] M. Mansouri-Samani, M. Sloman *Monitoring Distributed Systems (A Survey)* Imperial College Research Report No. DOC92/23 Apr. (1993).

[4] A. Uszok, A. Król, K. Zieliński *Cluster Management Software for Open Object Oriented Systems* HPCN94, Munich, Lecture Notes in Computer Science 797, Springer-Verlag, (1994).

[5] A. Uszok, *Równoważenie obciążenia w obiektowo zorientowanych systemach rozproszonych.* Praca magisterska, AGH Kraków, Instytut Informatyki (1993).