Realizacja rozproszonej komunikacji międzyprocesowej (IPC) w środowisku OpenSSI

Projekt z przedmiotu RSO

Kamil Kołtyś
KJKoltys@elka.pw.edu.pl

16 czerwca 2005

Spis treści

1 Wprowadzenie

Komunikacja międzyprocesowa w środowisku OpenSSI realizowana jest poprzez te same obiekty IPC, jakie występują w tradycyjnym systemie Linux. Mamy zatem do dyspozycji:

- potoki
- kolejki FIFO
- sygnały
- kolejki wiadomości
- semafory
- pamięć dzieloną
- gniazda (Internet-domain i Unix-domain)

Warto wspomnieć, że OpenSSI zawiera pakiet oprogramowania PVM oraz implementuje interfejs MPI. Obu tych mechanizmów także można użyć do komunikacji międzyprocesowej.

Rozproszona komunikacja międzyprocesowa w środowisku OpenSSI jest przezroczysta. Oznacza to, że dla semaforów, kolejek wiadomości i pamięci dzielonej jest jedna przestrzeń nazw w całym klastrze, a potoki, kolejki FIFO i gniazda są wspólne w całym klastrze. Przestrzeń nazw dla IPC jest zarządzana przez IPC Nameserver, który jest automatycznie reaktywowany po awarii węzła, na którym się on znajdował. Każdy obiekt IPC jest tworzony na tym węźle, na którym wykonywany jest proces, który go ”powołuje do życia”. Ale jest on różnież dostępny z każdego innego węzła w klastrze, skąd każdy proces może z niego korzystać w dokładnie taki sam sposób, jakby znajdował się on na tym samym węźle, co dany obiekt IPC. Wyjątek stanowią te obiekty IPC, które zostały utworzone przez procesy wykonywane jako lokalne na danej maszynie. Ich zasięg, podobnie jak zasięg procesu jest lokalny i odwoływać się mogą do niego tylko procesy z tego węzła. Obiekty IPC nie mogą być przenoszone z jednego węzła na inny węzeł, jak np. procesy. Gdy zostaną utworzone na danym weźle, muszą już na nim pozostać aż do momentu ich usunięcia.

Dlatego właśnie szybkość działania mechanizmów IPC w środowisku OpenSSI w dużej mierze zależy od tego czy proces będzie migrował, czyli przemieszczał się na inny węzeł czy też nie. Jeżeli bowiem po migracji proces odwoła się do któregoś z obiektów IPC, które wcześniej (przed migracją) utworzył (na innym węźle), to wystąpi opóźnienie związane z koniecznością przesyłania komunikatów przez sieć.

Celem niniejszego opracowania jest przeanalizowanie wpływu migracji procesów na szybkość działania mechanizmów IPC. Tym samym będzie można ocenić wydajność środowiska OpenSSI w kontekście komunikacji międzyprocesowej. Jest to szczególnie ważne zważywszy na istotę systemów klastrowych, w których poprzez migrację procesów dokonuje się optymalnego przydziału zasobów (głównie procesora) poszczególnym procesom.

2 Test semaforów

2.1 Opis

Test semaforów będzie wykorzystywał program implementujący rozwiązanie klasycznego problemu synchronizacji 5 filozofów. Ogólnie rzecz ujmując, problem ten polega na tym, że każdy z filozofów w przerwach swoich rozmyślań musi spożyć posiłek. Na stole przy którym jedzą znajduje się 5 talerzy i 5 widelców między każdymi dwoma talerzami. Aby filozof mógł jeść potrzebuje on obu sąsiednich widelców. Należy tak zsynchronizować spożywanie posiłków przez filozofów, aby nie doszło do zakleszczenia. Program rozwiązujący ten problem korzysta z mechanizmu semaforów tworząc zbiór 5 semaforów (dla każdego widelca). Problem zakleszczenia jest tutaj rozwiązany dzięki zapewnieniu przez system operacyjny jednoczesnego przeprowadzania operacji na kilku semaforach. W związku z tym filozof albo podnosi dwa widelce naraz albo nie podnosi żadnego. Program kończy swoje działanie po ustalonym czasie 30 sekund zapamiętując liczbę spożytych w tym czasie posiłków przez każdego z filozofów. Test semaforów ma polegać na sprawdzeniu jak zmienia się liczba spożytych posiłków przez poszczególnych filozofów w zależności od tego jakie migracje nastąpiły w trakcie działania programu.

2.2 Wyniki

W każdym z poniższych przypadków wykonano pięciokrotnie program testowy, aby przedstawić rozpiętność zmian kolejnych wyników. W tabelach zawarta jest informacja o ilości zjedzonych posiłków przez poszczególnych filozofów.

2.2.1 Przypadek 1 - wszystkie procesy wykonywane na jednym węźle

W tym przypadku rozważane są dwie sytuacje: pierwsza, gdy wszystkie procesy wykonują się na tym samym węźle, gdzie zostały utworzone semafory i druga, gdy procesy wszystkie procesy na samym początku zostają przemieszczone na inny (ale wszystkie na ten sam) węzeł.

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	602	601	602	600	598
2	601	601	602	599	600
3	601	601	602	599	599
4	601	601	601	600	601
5	602	601	602	600	598
Średnia	601.4	601	601.8	599.6	599.2

Tablica 1: Test semaforów (przypadek 1a) - wszyskie procesy wykonują sie na węźle macierzystym (brak migracji)

2.2.2 Przypadek 2 - procesy wykonywane na kilku różnych węzłach (migracje na początku)

W tym przypadku rozważany jest wpływ na liczbę spożytych posiłków przez filozofów w zależności od rozmieszczenie procesów na kilku węzłach. Ewentualne migracje

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	598	597	597	598	597
2	600	599	600	599	599
3	599	599	599	599	599
4	600	599	600	599	599
5	600	599	600	599	599
Średnia	599.4	598.6	599.2	598.8	598.6

Tablica 2: Test semaforów (przypadek 1b) - wszystkie procesy wykonują się na innym węźle niż ten, na którym zostały utworzone (migracja na początku)

poszczególnych procesów występują tylko na początku działania programu. Rozpatrywane będą trzy sytuacje. W pierwszej procesy filozofów 1 i 4 będą wykonywać się pierwszym węźle, a pozostałe trzy na drugim. W drugiej podobnie procesy filozofów będą wykonywać się na pierwszym węźle, ale pozostałe trzy zostaną rozmieszczone na trzech różnych węzłach. Ostatnia sytuacja będzie rozpatrywać, podobnie jak pierwsza, rozmieszcznie wszystkich procesów na tylko dwóch węzłach, ale procesy te zostaną rozmieszczone w inny, następujący sposób: procesy filozofów 1 i 2 na pierwszym węźle, a procesy 3, 4 i 5 na drugim.

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	751	749	749	751	749
2	751	750	749	751	749
3	750	749	749	750	749
4	751	749	748	751	748
5	750	749	749	751	749
Średnia	750.6	749.2	748.8	750.8	748.8

Tablica 3: Test semaforów (przypadek 2a) - procesy filozofów 1 i 4 wykonują się na pierwszym węźle, a pozostałe trzy na drugim, tym samym węźle

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	750	750	749	750	750
2	750	750	750	751	750
3	751	749	750	751	750
4	750	750	750	750	750
5	751	749	750	750	750
Średnia	750.4	749.6	749.8	750.4	750

Tablica 4: Test semaforów (przypadek 2b) - procesy filozofów 1 i 4 wykonują się na pierwszym węźle, a pozostałe trzy na trzech różnych węzłach

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	668	667	668	667	666
2	668	667	667	667	666
3	668	667	667	666	667
4	668	667	668	667	666
5	668	667	667	666	666
Średnia	668	667	667.4	666.6	666.2

Tablica 5: Test semaforów (przypadek 2c) - procesy filozofów 1 i 2 wykonują się na pierwszym węźle, a pozostałe trzy na drugim, tym samym węźle

2.2.3 Przypadek 3 - jeden proces migruje wiele razy między dwoma węzłami

W tym przypadku rozważany jest wpływ wielokrotnej migracji jednego procesu między węzłami na liczbę spożywanych posiłków przez poszczególnych filozofów. Ten przypadek będzie uwzględniał dwa podprzypadki. W pierwszym proces będzie migrował pomiędzy węzłem macierzystym, którym wykonują się pozostałe procesy, a jakimś innym węzłem. Natomiast w drugim podprzypadku proces będzie migrował między dwoma węzłami różnymi od macierzystego, na którym wykonują się pozostałe procesy. Oba podprzypadki będą rozpatrywały dwie sytuacje w zależności od częstoliwości migracji: migracja procesu filozofa co 10 i co 100 posiłków.

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	604	589	603	633	634
2	604	588	604	633	634
3	602	585	603	634	635
4	604	589	605	634	633
5	604	588	604	633	634
Średnia	603.6	587.8	603.8	633.4	634

Tablica 6: Test semaforów (przypadek 3a) - proces filozofa drugiego migruje pomiędzy węzłem macierzystym a innym węzłem co 10 posiłków

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	630	628	630	633	634
2	633	627	633	633	631
3	630	628	630	634	633
4	631	628	631	634	633
5	631	628	631	633	634
Średnia	631	627.8	631	633.4	633

Tablica 7: Test semaforów (przypadek 3b) - proces filozofa drugiego migruje pomiędzy węzłem macierzystym a innym węzłem co 100 posiłków

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	623	638	639	671	670
2	625	636	638	670	671
3	639	637	624	671	670
4	633	631	633	671	670
5	640	639	624	671	670
Średnia	632	636.2	631.6	670.8	670.2

Tablica 8: Test semaforów (przypadek 3c) - proces filozofa drugiego migruje pomiędzy dwoma węzłami różnymi od węzła macierzystego co 10 posiłków

Nr	Filozof 1	Filozof 2	Filozof 3	Filozof 4	Filozof 5
1	664	662	662	667	667
2	662	664	665	668	667
3	665	664	662	667	667
4	661	659	661	668	668
5	662	661	661	668	668
Średnia	662.8	662	662.2	667.6	667.4

Tablica 9: Test semaforów (przypadek 3d) - proces filozofa drugiego migruje pomiędzy dwoma węzłami różnymi od węzła macierzystego co 100 posiłków

2.3 Wnioski

Pierwszy przypadek pokazuje sytuację, w której wszystkie procesy wykonują się na jednym węźle. W pierwszym z nich procesy znajdują się na tym samym węźle, co obiekty synchronizujące (semafory), w drugim zaś przypadku na innym węźle. Jak można zauważyć różnica pomiędzy średnio zjedzonymi posiłkami przez poszczególnych filozofów jest nieznaczna i wskazuje na znikomy narzut czasowy w przypadku korzystania z semofora znajdującego się na innym węźle.

W drugim przypadku w pierwszej z sytuacji testowych można zaobserwować znaczny wzrost liczby spożywanych przez filozofów posiłków w porównaniu z dwoma sytuacjami z pierwszego przypadku. Już odpowiednie wykorzystanie dwóch węzłów podnosi wydajność o 25%. Co ciekawe rozmieszczenie procesów na czterech węzłach, jak w drugiej sytuacji tego przypadku, nie poprawiło specjalnie tych rezultatów. Można to wytłumaczyć tym, że jednocześnie i tak może jeść conajwyżej dwóch filozofów. Tak więc dysponując nawet większą liczbą węzłów, nie daje to możliwości wykonywania większej liczby procesów równolegle. Uwzględniając ten fakt i to, że wyniki dla czterech węzłów nie różniły się (a w każdym razie na pewno nie były gorsze) od wyników otrzymanych dla dwóch węzłów, potwierdza się dobra wydajność mechanizmu semaforów. Komunikacja między czterema węzłami była równie efektywna jak komunikacja między dwoma. Trzecia sytuacja tego przypadku pokazuje, że rozmieszczenie procesów na węzłach ma znaczenie i gdy trzech sąsiednich filozofów jest wykonywanych na tej samej maszynie, to otrzymywane wyniki są gorsze.

Trzeci, a tym samym ostatni przypadek prezentuje wpływ liczby migracji na ilość zjedzonych posiłków przez poszczególnych filozofów. W pierwszym podprzypadku widać, że gdy filozof drugi często migrował (co 10 posiłków) to on i jego sąsiedzi jedli

średnio mniej niż filozofowie 4 i 5. Stąd wniosek, że sam proces migracji może być obciążający i ten proces, który jej często podlega rzadziej będzie uzyskiwał do dostęp do zasobów współdzielonych. Także inne procesy, które z nim będą konkurować, będą miały mniej okazji do korzystania z tych zasobów. Warto jednak zwrócić uwagę, że w ostatecznym rozrachunku i tak wyniki otrzymane w tej sytuacji były lepsze niż wtedy, gdy wszystkie procesy wykonywały się na tym samym węźle. Zatem koszty migracji są mniejsze niż zysk płynący z równoległego wykonywania procesów. Ten koszt jest tym mniejszy, im rzadziej następuje migracja. Gdy proces filozofa drugiego migrował co 100 posiłków, to różnica w jedzonych posiłkach między tym filozofem i jego sąsiadami a pozostałymi dwoma nieznacznie przemawiała na korzyść tych dwóch ostatnich.

Na podstawie trzeciego przypadku testowego można porównać jak migracja jednego procesu wpływa na działanie programu, gdy migruje on pomiędzy dwoma węzłami różnymi od tego, na którym wykonują się pozostałe procesy. Liczba zjedzonych posiłków przez poszczególnych filozofów jest odpowienio większa zarówno, gdy migracja następuje po dziesięciu jak i po stu posiłkach. Nawet w przypadku częstej migracji otrzymano dużo lepsze wyniki niż przy braku jakiejkolwiek migracji. Migrujący proces i jego sąsiedzi jedli średnio o 30 posiłków więcej, a pozostali nawet o 70 posiłków więcej. Gdy migracja była rzadsza, otrzymano podobny wynik jak w drugim przypadku dla ostatniej sytuacji, gdy dwa procesy wykonywały się na jednym węźle a trzy na drugim.

Reasumując, wykorzystanie kilku węzłu w przypadku synchronizacji za pomocą semaforów, jest opłacalne i koszt związany z koniecznością komuniakcji przez sieć jest rekompensowany z dość dużą nawiązką przez równoległość wykonywania procesów.

3 Test potoków

3.1 Opis

Test potoków będzie wykorzystywał prosty program implementujący komunikację przez dwa potoki. Komunikacja będzie zachodzić pomiędzy procesem generującym (proces ”generator”) wektory liczb naturalnych i procesem ”kalkulatorem”, który będzie wykonywał na nich odpowiednie działania i zwracał wynik pierwszemu procesowi. Po przeprowadzeniu wszystkich obliczeń program kończy swoje działanie. Test potoków będzie polegał na wyznaczeniu czasu wykonywania się programu w zależności od tego, które procesy zostały przeniesione na inne węzły. Czas będzie mierzony za pomocą polecenia */usr/bin/time*. Testy będą przeprowadzone dla różnych rozmiarów wektora.

3.2 Wyniki

Współczynnik w trzeciej kolumnie w poniższych tabelach wyraża stosunek otrzymanego czasu dla danej sytuacji do najlepszego (najmniejszego) z uzyskanych czasów. Wartości czasów w nawiasach dotyczą sytuacji migracji obu procesów na różne węzły. W przypadku migracji jednego procesu nie miało większego znaczenia, który z nich migrował (generator czy kalkulator).

3.2.1 Przypadek 1 - mały rozmiar wektora i duża liczba działań

W pierwszym przypadku rozważany jest rozmiar wektora równy 1 element. Ilość działań wykonywanych w trakcie trwania programu jest równe 100000.

Opis migracji	Czas [sec]	Współczynnik
brak migracji	1.02	1
migracja generatora/kalulatora przed pierwszym działaniem	37.6	37
migracja generatora/kalkulatora przed 25000 działaniem	28.6	27.7
migracja generatora/kalkulatora przed 50000 działaniem	19.4	20
migracja generatora/kalkulatora przed 75000 działaniem	10.2	10
migracja generatora/kalkulatora przed 99000 działaniem	1.4	1.37
migracja generatora i kalulatora przed pierwszym działaniem na ten sam węzeł (na różne węzły)	50.5 (46.7)	49.5 (45.7)
migracja generatora i kalkulatora przed 25000 działaniem na ten sam węzeł (na różne węzły)	38.6 (35.8)	37.84 (35.09)
migracja generatora i kalkulatora przed 50000 działaniem na ten sam węzeł (na różne węzły)	26.1 (24.2)	25.5 (23.72)
migracja generatora i kalkulatora przed 75000 działaniem na ten sam węzeł (na różne węzły)	13.6 (12.6)	13.3 (12.35)
migracja generatora i kalkulatora przed 99000 działaniem na ten sam węzeł (na różne węzły)	1.64 (1.57)	1.6 (1.53)

Tablica 10: Test potoków (przypadek 1) - mały rozmiar wektora i duża liczba działań

3.2.2 Przypadek 2 - duży rozmiar wektora i mała liczba działań

W drugim przypadku rozważany jest rozmiar wektora równy 100000 element. Ilość działań wykonywanych w trakcie trwania programu jest równe 10.

3.3 Wnioski

Niezależnie od rozmiaru i ilości wykonywanych działań migracja jednego procesu powoduje znaczne wydłużenie czasu wykonywania wszystkich obliczeń. Im wcześniej następowała migracja tym ten czas był większy. W przypadku pierwszym nawet bardzo późna migracja procesu generatora (lub kalkulatora) - po wykonaniu 99% obliczeń - powoduje ok. 1.37 razy dłuższe wykonywanie się programu. W drugim przypadku wykonanie tylko ostatniego działania po migracji skutkuje już wzrostem czasu 4.52 razy. Migracja obu procesów jeszcze bardziej pogarsza wydajność. Dla obu przypadów migracja przed rozpoczęciem wszystkich obliczeń powoduje wydłużenie czasu wykonania programu 50-ciokrotnie. Wynika to z konieczności komunikowania się obu procesów z potokiem przez sieć. Stąd wniosek, że im więcej procesów będzie się znajdywało na innym węźle niż potok, przez który się komunikują, tym ta komunikacja będzie wolniejsza.

Opis migracji	Czas [sec]	Współczynnik
brak migracji	0.73	1
migracja generatora/kalulatora przed pierwszym działaniem	19.2	26.3
migracja generatora/kalkulatora przed 5 działaniem	10.6	14.5
migracja generatora/kalkulatora przed 9 działaniem	3.3	4.52
migracja generatora i kalulatora przed pierwszym działaniem na ten sam węzeł (na różne węzły)	35.2 (37.4)	48.2 (51.2)
migracja generatora i kalkulatora przed 5 działaniem na ten sam węzeł (na różne węzły)	19.2 (20.4)	26.3 (27.9)
migracja generatora i kalkulatora przed 9 działaniem na ten sam węzeł (na różne węzły)	5.6 (5.9)	7.6 (8.08)

Tablica 11: Test potoków (przypadek 2) - duży rozmiar wektora i mała liczba działań

Warto zwrócić uwagę na istotną różnicę pomiedzy przedstawionymi przypadkami w sytuacji, gdy oba procesy migrują albo na ten sam węzeł albo na inne węzły. Otóż w przypadku małego wektora i dużej liczby obliczeń korzystniej jest jak oba procesy wykonują się na różnych węzłach. Natomiast gdy wykonujemy małą liczbę dużych transmisji (jak w drugim przypadku drugim), to wykonywanie obu procesów na tym samym węźle daje lepsze rezultaty. Uzasadnieniem tego faktu może być to, że gdy wykonujemy dużą liczbę małych transmisji, to w przypadku wykonywania się procesów na jednym węźle konieczna jest częsta zmiana kontekstu procesora, która może być kosztowna. Gdy procesy wykonują się na różnych węzłach zmiana kontekstu oczywiście nie jest potrzebna. Z kolei, gdy wykonujemy mało dużych transmisji, to koszt zmiany kontekstu nie jest ważny w porównaniu z czasem potrzebnym na przesłanie danych. Wówczas komunikacja pomiędzy dwoma węzłami jest wydajniejsza niż pomiedzy trzema. Stąd można wyciągnać wniosek, że jeżeli nie będziemy chcieli przesyłać dużej ilości danych przez potok, a większą część działania procesów będą stanowiły obliczenia, to równomierne rozłożenie procesów na kilku węzłach może zrekompensować koszt komunikacji przez sieć.

4 Test pamięci dzielonej

4.1 Opis

Test pamięci dzielonej wykorzystuje program, w którym dwa procesy zwiększają o 1 wartość każdego elementu tablicy współdzielonej (zainicjowanej wartością 0). Następnie liczą sumę wszystkich elementów i sprawdzają jej wartość. Jeżeli jest ona większa od ustalonej wartości, to następuje koniec działania programu. Pamięci dzielona jest synchronizowana za pomocą semafora. Test pamięci dzielonej będzie polegał na porównaniu czasów wykonywania się obu programów w zależności od wystąpienia określonych migracji procesów. Podobnie jak w poprzednim teście czas będzie mierzony za pomocą polecenia */usr/bin/time* i będzie rozpatrywał różne wielkości dzielonej tablicy.

4.2 Wyniki

W poniższych testach rozważamy tylko migracje na samym początku, tzn. przed rozpoczęciem wykonywania działań na pamięci dzielonej przez procesy. Podobnie jak w poprzednim teście współczynnik w trzeciej kolumnie w poniższych tabelach wyraża stosunek otrzymanego czasu dla danej sytuacji do najlepszego (najmniejszego) z uzyskanych czasów.

4.2.1 Przypadek 1 - rozmiar pamięci dzielonej 4 bajty

W pierwszym przypadku rozważany jest rozmiar tablicy równy 1 element. Wartość sumy, po której następuje zakończenie programu jest równa 1000, co oznacza że oba procesy będą musiały wykonać w sumie 1000 inkrementacji każdego elementu tablicy.

Opis migracji	**Czas [sec]**	**Współczynnik**
brak migracji	10.02	1
migracja jednego procesu	10.23	1.02
migracja obu procesów na ten sam węzeł	10.23	1.02
migracja obu procesów na różne węzły	10.23	1.02

Tablica 12: Test pamięci dzielonej (przypadek 1) - rozmiar pamięci dzielonej 4 bajty

4.2.2 Przypadek 2 - rozmiar pamięci dzielonej 40 kilobajtów

W drugim przypadku rozważany jest rozmiar tablicy równy 10000 element. Wartość sumy, po której następuje zakończenie programu jest równa 1000000, co oznacza że oba procesy będą musiały wykonać w sumie 100 inkrementacji każdego elementu tablicy.

Opis migracji	**Czas [sec]**	**Współczynnik**
brak migracji	1.018	1
migracja jednego procesu	1.199	1.17
migracja obu procesów na ten sam węzeł	1.134	1.11
migracja obu procesów na różne węzły	1.715	1.68

Tablica 13: Test pamięci dzielonej (przypadek 2) - rozmiar pamięci dzielonej 40 kilobajtów

4.2.3 Przypadek 3 - rozmiar pamięci dzielonej 4 megabajty

W trzecim przypadku rozważany jest rozmiar tablicy równy 1000000 element. Wartość sumy, po której następuje zakończenie programu jest równa 100000000, co oznacza że oba procesy będą musiały wykonać w sumie 100 inkrementacji każdego elementu tablicy.

Opis migracji	Czas [sec]	Współczynnik
brak migracji	1.178	1
migracja jednego procesu	50.666	43
migracja obu procesów na ten sam węzeł	2.269	1.92
migracja obu procesów na różne węzły	99.129	84.15

Tablica 14: Test pamięci dzielonej (przypadek 3) - rozmiar pamięci dzielonej 4 megabajty

4.3 Wnioski

Dla małego obszaru pamięci dzielonej (1 element typu integer - 4 bajty) wpływ migracji na czas wykonywania się programów był nieznaczny, jakkolwiek jednak migracja spowodowała jego niewielki wzrost. Wraz ze zwiększeniem rozmiaru tablicy dzielonej rożnice pomiędzy poszczególnymi przypadkami stawały się coraz większe. Dla tablicy o rozmiarze 10000 elementów (40kB) migracja obu procesów na różne węzły sprawiła wzrost czasu wykonania programu 1.6 razy. Co ciekawe w przypadku migracji obu procesów na jeden, ten sam węzeł uzyskano lepszy czas niż dla migracji jednego procesu. Dla rozmiaru tablicy równego 1000000 elementów (4Mb) wpływ migracji na wydajność programu był ogromny. Migracja obu procesów na ten sam węzeł spowodowała wprawdzie tylko dwukrotne wydłużenie czasu, ale migracja jednego procesu aż 43-krotne, zaś migracja obu na różne węzły obniżyła wydajność aż 84 razy.

Można zatem wyciągnąć wniosek, że gdy oba procesy są wykonywane na dwóch różnych węzłach, to komunikacja między nimi poprzez pamięć dzieloną nie jest wydajna, zwłaszcza jeżeli mamy do czynienia z dużym obszarem pamięci dzielonej. Jest to o tyle dotkliwe, że praktycznie czyni bezsensownym przydzielenie procesom komunikującym się przez pamięć dzieloną różnych węzłów. Zysk bowiem z równoległego wykonywania się procesów może skutecznie zostać zaprzepaszczony przez komunikację za pomocą pamięci dzielonej.

5 Podsumowanie

W powyższych testach wydajnościowych poddano analizie następujące obiekty IPC służące do rozproszonej komunikacji międzyprocesowej w środowisku OpenSSI: semafory, potoki i pamięć dzieloną. Mechanizm semaforów okazał bardzo wydajny. Konieczności komunikowania się przez sieć z obiektami IPC znajdującymi się na innych węzłach nie przynosiła większych nakładów czasowych, a zysk płynący z równoległego wykonywania procesów był znaczący. Nawet częste migracje w trakcie działania programu nie powodują znacznego obniżenia wydajności, zwłaszcza gdy dotyczą migracji na jakiś dodatkowy węzeł - a taki charakter ma większość migracji, które mają na celu równomierne obciążenie klastra. Dzięki temu programy wykonywane w środowisku OpenSSI, które wykorzystują ten rodzaj obiektu IPC, nie powinny zachowywać się gorzej pod tym względem. Wręcz przeciwnie, korzyści płynące z rozmieszczenia procesów na wielu węzłach klastra mogą znacznie podnieść efektywność wykonywania programu.

Gorzej rzecz wygląda w przypadku potoków. Testy dla tego obiektu IPC wykazały, że migracja ma znaczący wpływ na szybkość działania programu i im więcej

procesów komunikujących z potokiem wykonuje się na innym węźle niż ten, na którym znajduje się potok, tym większe opóźnienia są wprowadzane przez komunikację sieciową. Niemniej jeśli procesy wykonują dużo obliczeń, które można przeprowadzać równolegle, a dane przesyłane przez potok mają niewielki rozmiar, to ten narzut czasowy może okazać się mniejszy niż zysk płynacy z optymalnego rozdysponowania zasobów klastra pomiędzy procesy. Takie programy mogą wydajnie działać w środowisku OpenSSI.

Test pamięci dzielonej pokazał, że efektywność jej działania w dużym stopniu zależy od jej rozmiaru. Gdy pamięć jest mała, to straty czasu wynikajace z komunikacji sieciowej też są małe. Jednakże pamięć dzielona dużo lepiej sprawdza się w przypadku, gdy procesy komunikujące się za jej pomocą znajdują się na tym samym węźle (niekoniecznie na tym samym co pamięć). Dla dużych obszarów pamięci dzielonej różnice czasów wykonania są znaczne i o wiele gorzej wypadają przy migracji procesów na różne węzły. O ile pamięć dzielona jest najszybszym sposobem komunikacji międzyprocesowej w tradycyjnym systemie Linux, o tyle w rozproszonym środowisku OpenSSI traci ona bardzo wiele ze swojej wydajności. Dlatego należy być ostrożnym podczas uruchamiania w tym systemie klastrowym programów korzystających z tego mechanizmu komunikacji. Istnieje bowiem duże prawdopodobieństwo, że będą one działały wolniej niż w środowisku jednoprocesorowym.