Pro Dialog **20** (2005), 27–42
Wydawnictwo NAKOM – Poznań

Wybrane zastosowania spekulacji w systemach z rozproszoną pamięcią współdzieloną

Jerzy BRZEZIŃSKI, Arkadiusz DANILECKI

Politechnika Poznańska, Instytut Informatyki
ul. Piotrowo 3a, 60-965 Poznań
e-mail: arkadiusz.danilecki@cs.put.poznan.pl

Otrzymano 26 lipca 2005 roku

Streszczenie. Artykuł prezentuje wybrane zastosowania spekulacji w systemach z rozproszoną pamięcią współdzieloną. Przedstawiono w nim nową propozycję formalnych definicji pojęć predykcji, predyktorów oraz spekulacji. Wyjaśniono zasadę działania spekulacji, skupiając się na pojęciu wzorców współdzielenia i opisano charakterystyki jakości predyktorów. Następnie opisano najbardziej popularne i znane predyktory ogólnego zastosowania. Jako przykład zastosowania spekulacji pokazano metody sprowadzania stron z wyprzedzeniem.

Słowa kluczowe: spekulatywne przetwarzanie danych, predykcja, rozproszona pamięć współdzielona, pobieranie stron z wyprzedzeniem

1. Wstęp

W systemach rozproszonych pamięć współdzielona (DSM, skr. ang. *Distributed Shared Memory*) obejmuje pamięci lokalne wielu komputerów (jednostek przetwarzających, stacji roboczych) – *węzłów*, połączonych siecią komunikacyjną. Pamięć współdzielona jest zwykle podzielona na mniejsze części składowe, strony czy obiekty, które są następnie odwzorowywane na bloki pamięci lokalnej węzłów. Gdy węzeł odwołuje się do strony nie znajdującej się aktualnie w pamięci lokalnej, następuje tzw. *błąd strony*, powodujący przekazanie sterowania do mechanizmu zarządzania pamięcią współdzieloną, który powinien zapewnić sprowadzenie strony do pamięci lokalnej węzła.

Opóźnienia związane z błędami stron mogą mieć spory wpływ na wydajność, dlatego też duże znaczenie mają wszelkie metody ich unikania. Jedną z nich jest odpowiednio wczesne przewidywanie do jakich stron węzeł będzie się odwoływał, tak by móc je sprowadzić do pamięci lokalnej, zapobiegając wystąpieniu błędu strony. Stosownej do tego celu informacji może dostarczać analiza kodu źródłowego programu i próbnych jego wykonań, której wyniki albo wspierają programistów i projektantów [4, 15], albo są wykorzystywane przez kompilatory i specjalne narzędzia [20, 24]. Rozwiązania te są jednakże z swojej natury statyczne, podczas gdy działanie programów w rozproszonym środowisku cechuje zmienność przetwarzania. Dlatego też byłoby rzeczą pożądaną wyposażenie takich systemów w mechanizm, który potrafiłby dynamicznie dostosowywać się do zmieniających się warunków, próbując przewidywać przyszłe zachowanie procesów. Mechanizm taki powi-

nien być prosty i wydajny. Nie jest przy tym wymagana jego stuprocentowa dokładność, gdyż koszt pomyłek nie jest na ogół duży. Te wymagania od razu dyskwalifikują wiele dobrze znanych algorytmów uczenia maszynowego [18]. Również analogie do metod analizy szeregów czasowych [29] są zbyt odległe, aby można wykorzystać je w rozważanym zastosowaniu.

Interesującym i obiecującym podejściem do rozważanego problemu jest *spekulacja*, rozumiana jako wykonywanie akcji, które mogą potencjalnie zarówno przyśpieszyć, jak i spowolnić działanie systemu, w zależności od poprawności dokonanej *predykcji*, czyli przewidywania przyszłego zachowania systemu. W niniejszej pracy przedstawiono formalną definicję predykcji, budowę i zasadę działania najpopularniejszych *predyktorów* (modułów systemu zajmujących się wykonywaniem predykcji) oraz przykłady jednej z możliwych technik spekulacji, mianowicie pobierania stron z wyprzedzeniem w stosunku do faktycznego zażądania do nich dostępu.

2. Model systemu

Rozważany w artykule system DSM składa się ze skończonego zbioru procesów P_1, P_2, P_3, ..., P_n rezydujących w skończonym zbiorze węzłów N_1, N_2, N_3, ..., N_j połączonych siecią komunikacyjną. Węzły posiadają własne pamięci lokalne, z których każda jest podzielona na *bloki* o jednakowym rozmiarze. Pamięci lokalne węzłów tworzą wirtualną pamięć współdzieloną przez procesy i postrzeganą jako jedna spójna, liniowa przestrzeń adresowa podzielona na *strony*. Fakt rozproszenia tej pamięci na wielu węzłach jest przy tym ukrywany przez system. Niech $M_i = \{1, 2, ..., m\}$ oznacza zbiór numerów bloków pamięci operacyjnej dostępnych w *i*-tym węźle. Ponadto niech $X = \{1, 2, ..., n\}$ oznacza zbiór numerów stron wirtualnych. Numery stron pamięci należące do zbioru X są kojarzone z numerami bloków pamięci lokalnych M_i, przy czym jednej stronie może odpowiadać wiele bloków (kopii strony). Jeżeli proces P_i w węźle N_j odwołuje się do strony pamięci, której kopia znajduje się w bloku m_{ln} nie należącym do pamięci lokalnej M_l węzła N_l, to zachodzi *błąd strony*, zawieszenie pracy procesu i uruchomienie mechanizmu zarządzania pamięcią współdzieloną, który zrealizuje skopiowanie przez system żądanej strony do wybranego bloku $m \in M_j$ węzła N_j, w sposób przezroczysty dla procesu. Możliwość istnienia równocześnie wiele kopii (replik) jednej strony pamięci rodzi oczywiście problem zapewnienia spójności kopii (w uproszczeniu, identyczności). System DSM oferuje zwykle aplikacji zestaw gwarancji dotyczących spójności replik, określanych jako model spójności. Za utrzymanie przyjętego modelu spójności odpowiedzialny jest *protokół koherencji* (spójności), czyli (znowu w uproszczeniu) algorytm, gwarantujący zachowanie przyjętego modelu spójności.

Zazwyczaj protokół spójności wyróżnia pewien zbiór stanów dopuszczalnych stron. Najbardziej ogólny zbiór stanów określany jest jako skrótem MOESI od angielskich słów dotyczących strony: *modified* – zmodyfikowana, *owned* – prywatna (w prywatnym posiadaniu), *exclusive* – dla której istnieje węzeł posiadający wyłączny dostęp, *shared* – współdzielona, *invalid* – unieważniona (nie zezwala się na żadną operację na stronie). Z kolei typowy protokół koherencji posiada wyróżnione cztery stany: MESI – *modified, exclusive, shared, invalid*, a często używaną dla niego nazwą jest *protokół Illinois*.

Stany strony przechowywane są w specjalnej strukturze nazywanej *katalogiem* (*tablicą stron*). Zazwyczaj katalog ten znajduje się w wydzielonym węźle systemu, lecz może też być strukturą rozproszoną. Jeżeli proces zmodyfikuje kopię strony, uruchamiany jest jeden z dwóch ogólnych mechanizmów: uaktualniania lub unieważniania. W mechanizmie uaktualniania protokół przesyła uaktualnienie do wszystkich pozostałych węzłów posiadających kopię zmodyfikowanej strony (ang. *write-update*), z kolei w mechanizmie unieważniania przesyłane jest powiadomienie o nieaktualności kopii do odpowiednich węzłów (ang. *write-invalidate*). W niniejszym artykule przyjęto, że protokoły koherencji stosują mechanizm unieważnienia.

W literaturze przedmiotu pojęcia *spekulacji* oraz *predykcji* są traktowane w sposób intuicyjny. Dla uściślenia prezentacji warto jednak wprowadzić ich formalne definicje.

Oznaczmy przez o_i^j *j*-tą operację, wykonaną w węźle N_i, którą w szczególności może być operacja odczytu $r_i^j(x)$ bądź zapisu $w_i^j(x)$ strony *x*. Warto zauważyć, że indeks tej operacji wyznaczać może zarazem bieżący czas logiczny. Przyjmujemy przy tym, że operacje obejmują wymianę komunikatów protokołu koherencji (na przykład unieważnienie strony). Sekwencję następujących po sobie operacji $o_i^1, o_i^2, \ldots, o_i^{t-1}, o_i^t$ nazwiemy historią H_i^t operacji w węźle N_i.

Dla ułatwienia zapisu ciąg operacji $o_i^m, o_i^{m+1}, \ldots, o_i^{n-1}, o_i^n$ oznaczać będziemy przez $O_i(m, n)$.

Zbiór $BO_i(j)$ poprzedników operacji o_i^j zdefiniujemy następująco:

$$BO_i(j) \equiv \{o_i^k : k < j \wedge o_i^k \in H_i^t\}.$$

Zbiór $FO_i(j)$ następników operacji o_i^j określimy jako

$$FO_i(j) \equiv \{o_i^k : k > j \wedge o_i^k \in H_i^t\}.$$

Przez predykcję będziemy rozumieć dowolne przyporządkowanie ciągu operacji $O_i(k, j) \subset H_i^t$ nieuporządkowanego zbioru przewidzianych operacji $PO_i(j)$. Liczność zbioru $PO_i(j)$ nazywać będziemy *rozmiarem predykcji*, a liczność zbioru $O_i(k, j)$ – *głębią* bądź *wymiarem predykcji*. Predykcja jest poprawna, jeżeli $PO_i(j) \subset FO_i(j)$. Jeśli zachodzi predykat $\left(\exists o_i^j \in PO_i(j): o_i^j \in FO_i(j)\right) \wedge \left(\exists o_i^k \in PO_i(j): o_i^k \notin FO_i(j)\right)$, to predykcja jest *częściowo poprawna*. W pozostałych przypadkach jest ona *błędna* (*niepoprawna*). Poprawnie przewidzianymi operacjami nazwiemy takie operacje o_i^k, że $o_i^k \in FO_i(j)$ oraz $o_i^k \in PO_i(j)$.

Z uwagi na ograniczenia sprzętowe rzeczywistych komputerów tylko przewidywanie najbliższych w czasie operacji ma sens. Na przykład, przewidzenie zapotrzebowania na daną stronę w nieokreślonej, odległej przyszłości jest całkowicie bezużyteczne, gdyż sprowadzona strona może zostać unieważniona zanim faktycznie zostanie użyta. Z uwagi na to wprowadzamy dodatkowy parametr, którym jest o*dległość* $d(o_i^j, o_i^k)$ operacji o_i^j oraz o_i^k, rozumiana jako różnica $k - j$. Predykcją poprawną ze względu na odległość *n* nazywać będziemy takie przyporządkowanie operacji $O_i(k, j) \subset H_i^t$ elementom zbioru $PO_i(j)$, że $\forall o_i^j \in PO_i(j): d(o_i^j, o_i^l) < n \wedge o_i^l \in FO_i(j)$. Analogicznie definiujemy predykcję częściowo poprawną ze względu na odległość *n*.

Niech $EO_i(j)$ oznacza zbiór wszystkich operacji o_i^k wykonanych w węźle N_i powodujących błąd strony, takich że $o_i^k \in FO_i(j)$.

Spekulacją nazywać będziemy proces podejmowania decyzji i wykonywania akcji na podstawie dokonanych predykcji, a *predyktorem* Pr_i – wydzieloną (być może tylko koncepcyjnie) część systemu realizującą predykcję i ewentualnie doradzającą podjęcie odpowiednich akcji.

Jako kryterium jakości czy też efektywności predyktora możemy przyjąć jedną z kilku proponowanych w literaturze miar. Jeżeli $CP_i(j)$ oznacza zbiór operacji, takich że

$$o_i^k \in CP_i(j) \Leftrightarrow o_i^k \in PO_i(j) \wedge o_i^k \in EO_i(j),$$

to *pokrycie* predyktora Pr_i, oznaczone przez $coverage(Pr_i)$, zdefiniujemy następująco:

$$coverage(Pr_i) = \frac{\left| CP_i(j) \right|}{\left| EO_i(j) \right|}.$$

Tego rodzaju miara jest używana na przykład w pracy [31]. Z kolei w pracy [38] zaproponowano, aby oceniać efektywność predyktorów na podstawie trzech innych miar: *wrażliwości* (ang. *sensitivity*), *prewalencji* (ang. *prevalence*) oraz p*rzewidywalnej wartości pozytywnego testu* (ang. *predictive value of a positive test*, w skrócie: PVP), nazywane przez niektórych innych autorów *skutecznością przewidywań* albo *proporcją trafień* (ang. *hit ratio*). Wrażliwość określona została jako stosunek liczby poprawnie przewidzianych przypadków współdzielenia strony do liczby faktycznych przypadków współdzielenia, prewalencja – jako stosunek liczby faktycznych przypadków współdzielenia stron do liczby wszystkich użytych stron (a więc oznacza górną granicę możliwej poprawy efektywności), zaś PVP – jako stosunek liczby poprawnych przewidywań do liczby wszystkich przewidywań (poprawnych i niepoprawnych).

3. Wzorce współdzielenia

Wiele technik prezentowanych dalej w niniejszej pracy korzysta z następujących faktów:

- dostępy do bloków pamięci da się zaliczyć do jednego spośród stosunkowo małej liczby stałych, stabilnych i powtarzających się schematów,
- dostępy do bloków pamięci charakteryzują się tendencją do powtarzalności zachowań, które można zakwalifikować do stosunkowo małej liczby kategorii,
- zachowanie aplikacji przy dostępie do bloków pamięci wykazuje powtarzalność i daje się skategoryzować za pomocą stosunkowo małej liczby własności.

Własności te nazywane są *wzorcami współdzielenia* [9, 10, 25, 26, 44]. Przykładem może tutaj być typowy problem producent-konsument, w którym na pewnej stronie x są przechowywane ważne dane modyfikowane tylko przez jeden proces („producenta") i odczytywane przez wiele innych procesów („konsumentów"). Jeżeli proces „producent" w węźle N_j dokona zapisu $w_j(x)$ na tej stronie (co zazwyczaj w typowych protokołach koherencji typu *write-invalidate* powoduje unieważnienie wszystkich pozostałych kopii), wtedy można się

spodziewać, że wszyscy „konsumenci" w węzłach N_k, N_l, N_m wkrótce wykonają operację odczytu $r_k(x)$, $r_l(x)$, $r_m(x)$. Wzorzec współdzielenia dla tej strony wyglądałby następująco: $w_j(x) \rightarrow r_k(x), r_l(x), r_m(x)$.

W tabeli 1 przedstawiono pojawiające się w literaturze kategorie wzorców współdzielenia. Wadą cytowanych prac jest skupienie uwagi na aplikacjach naukowych, obliczeniowych, dość nietypowych z punktu widzenia komercyjnych zastosowań. Z kolei analizy komercyjnych aplikacji zazwyczaj ignorują całkowicie zagadnienie istnienia wzorców współdzielenia (pewnym wyjątkiem jest tutaj system opisany w pracy [8]), koncentrując się raczej na kwestii poprawnego modelowania w warunkach laboratoryjnych obciążeń komercyjnych (ang. *commercial workloads*).

Właściwości większości z kategorii przedstawionych w tabeli 1 są w intuicyjny sposób określone przez ich nazwy. Wyjątkiem są tutaj:

- strony migracyjne, na których po kolei wiele procesów domaga się dostępu w trybie do zapisów, co zazwyczaj powoduje unieważnienie wszystkich pozostałych kopii strony (w efekcie takie strony często są przenoszone między węzłami),
- strony zapisywane raz, które są inicjowane podczas początku przetwarzania i potem tylko czytane,
- strony zawierające rezultaty obliczeń – zapisywane przez wiele węzłów, a potem odczytywane przez jeden węzeł.

W pracach [25] i [26] przeprowadzono głównie analizę wpływu przynależności do określonej kategorii wzorców współdzielenia na liczbę unieważnień strony, ignorując kwestię częstości ich występowania.

Prace [9] i [10] skupiają się natomiast na zbiorze aplikacji obliczeniowych działających w systemie Munin [9]. Ważnym efektem tych badań było wykazanie, że w najlepszym razie liczba stron niepasujących do żadnego z ogólnych wzorców (czyli strony typu *general read-write*) jest nie większa niż 4,1%, zazwyczaj zaś w ogóle tego rodzaju obiekty nie pojawiały się.

W studium [44] przeanalizowano dziesięć aplikacji z zestawu SPLASH i dodatkowo dwie udostępnione przez U. C. Berkeley. Studium to wykazało występowanie w wielu z przebadanych aplikacji dużej, kilkakrotnej przewagi danych prywatnych nad współdzielonymi (fenomen tzw. martwego współdzielenia – ang. *dead sharing*). Kolejną zbadaną cechą była olbrzymia przewaga odczytów nad zapisami współdzielonych danych. Bardzo rzadko pojawiały się też operacje na zmiennych synchronizujących (dostępy do zamków). Zauważono również, że blisko połowa unieważnień była spowodowana przez przejście z trybu do odczytu do trybu do zapisu w jednym węźle, po którym żaden inny procesor nie żądał odczytu przed dokonaniem kolejnego unieważnienia. W połowie bloków zapisane dane nie były odczytywane przez inne węzły przed kolejnym unieważnieniem, a jedna piąta bloków posiadała współdzielenie na poziomie pojedynczych słów maszynowych (tzn. zaledwie pojedyncze słowa w danym bloku były faktycznie współdzielone przez węzły). Równocześnie mniej więcej połowa bloków była unieważniana zanim węzeł zdążył zapisać bądź odczytać więcej niż jedno słowo.

Tabela 1. Kategorie wzorców współdzielenia według prac [9], [10], [25], [26] i [44]

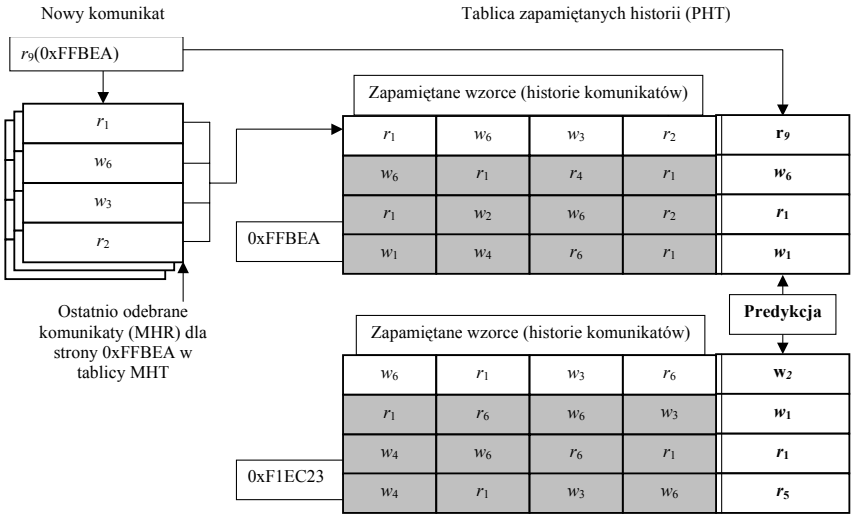
Wzorce współdzielenia	*Artykuł*		
	[25] i [26]	[9] i [10]	[44]
Migracyjne	obiekty migracyjne (ang. *migratory objects*)	strony migracyjne	migracyjne
Prywatne		prywatne – wszystkie dostępy pochodzą od jednego procesu	prywatne – tzw. martwe współdzielenie (ang. *dead sharing*)
Inicjowane raz		inicjowane raz (ang. *write-once*)	
Tylko do odczytu	tylko do odczytu (ang. *code and read-only*)		tylko do odczytu
Częsty odczyt	często czytane (ang. *frequently read*)	często czytane (ang. *read-mostly*)	przeważnie czytane (ponad 75% odczytów)
Odczyt/zapis	często czytane i zapisywane (ang. *frequently read/written*)	odczyt/zapis (ang. *general read-write*)	odczyt-zapis (ang. *read-write*)
Producent-konsument		producent-konsument (ang. *producent-consument*)	strony rozgłaszane (ang. *broadcast*)
Synchronizacyjne	obiekty synchronizujące (ang. *synchronization objects*)	strony synchronizujące (zamki, semafory)	zamki o niskim (wysokim) stopniu współdzielenia (ang. *low (high) contention locks*)
Częsty zapis		często zapisywane (ang. *write-many*)	
Rezultaty		rezultaty (ang. *results*)	

4. Przykład predykcji – predyktory uniwersalne

Jak pokazano w poprzednim punkcie, dostępy do stron wykazują dużą regularność. Zakładając, że poprzednie zachowania systemu będą powtarzalne, można automatycznie wykrywać na podstawie dotychczasowej historii *wzorce współdzielenia* nieznane *a priori*. Jeżeli w przeszłości dostęp do strony przez węzeł N_i poprzedzał zawsze (bądź wystarczająco często) dostęp do tej strony przez węzeł N_j, to istnieje duże prawdopodobieństwo, że w przyszłości taka sytuacja będzie się powtarzać.

Pojawia się tutaj kilka zagadnień: co właściwie ma obejmować zapamiętywana historia, jak wykorzystywać zapamiętywaną historię w celu wykonywania predykcji, a w szczególności w jaki sposób wyodrębnić zachowania powtarzalne od takich, które występują rzadko

lub w ogóle, jak wychwycić powtarzalność zachowań mimo pewnych odchyleń od spodziewanego wzorca (tzw. „filtrowanie szumów"), kiedy rozpoczynać predykcję. Wreszcie, ponieważ rzeczywiste systemy posiadają ograniczenia pamięciowe, pojawia się kwestia rozmiaru zapamiętywanej historii.



Rys. 1. Budowa predyktora *Cosmos*

Pierwsza próba wykorzystania wyżej wspomnianych pomysłów w celu utworzenia predyktora o ogólnym przeznaczeniu pojawiła się w pracy [41]. Predyktor ten, nazwany *Cosmos*, oparty jest na dwupoziomowym algorytmie przewidywania skoków Yeha i Patta [42]. Założono, że system DSM utrzymuje katalog stron zawierający dla każdej z nich informację o bieżącym właścicielu. Węzły (lub procesy w tych węzłach) uzyskują dostęp do tych informacji z katalogu za pomocą protokołu koherencji. Historia wymiany komunikatów jest zapamiętywana w celu przewidzenia następnych komunikatów. Autorzy artykułu starają się abstrahować, jak to tylko jest możliwe, od protokołów koherencji i możliwych akcji wynikających z dokonania predykcji. Wynikiem predykcji może być zmiana stanu strony, przesłanie jej kopii do danego węzła, zmiana jej właściciela itd.

Opisywany predyktor utrzymuje dla każdego z węzłów i katalogu dwie proste struktury: Tablicę Historii Komunikatów (ang. *Message History Table*, w skrócie: MHT) oraz Tablicę Historii Wzorców (ang. *Pattern History Table*, w skrócie: PHT) – zob. rys. 1.

Tablica MHT posiada wpis dla każdej strony współdzielonej pamięci. Każdy taki wpis, nazywany Rejestrem Historii Komunikatów (ang. *Message History Register*, w skrócie: MHR), zawiera informację o ostatnio odebranych komunikatach w postaci sekwencji <nadawca, typ komunikatu> (rys. 1 zawiera taki wpis dla strony 0xFFBEA: r_1, w_6, w_3, r_2 – odczyt strony wykonany przez węzeł pierwszy, zapis dokonany przez węzeł szósty itd).

Tablica PHT posiada wpis dla każdej strony współdzielonej pamięci. Każdy wpis jest indeksowany listą sekwencji o strukturze i rozmiarze identycznych jak w tablicy MHT, a jej każdy wpis zawiera przewidywany kolejny komunikat (na rys. 1 pokazane są wpisy dla strony 0xFFBEA i strony 0xF1EC23 tablicy PHT).

Przy nadejściu nowego komunikatu dotyczącego strony x (na przykładzie – r_9), predyktor odnajduje wpis dla tej strony w tablicy MHT (czyli historię komunikatów) i używa go jako indeksu do tablicy PHT, w uzyskanej pozycji dopisując nowo otrzymany komunikat (w przykładzie, pokazanym na rys. 1, na pozycji indeksowanej sekencją r_1, w_6, w_3, r_2 wpisuje r_9). Wreszcie, komunikat ten jest dopisywany na koniec wpisu w tablicy MHT, usuwając z niego, jeśli trzeba, najstarszy komunikat w celu zachowania odpowiedniego rozmiaru elementu tablicy (w przykładzie z rys. 1 po usunięciu r_1 i dopisaniu r_9 tablica MHT zawierałaby wpis r_9, r_1, w_6, w_3). Predykcja polega na odnalezieniu pozycji w tablicy PHT na podstawie aktualnej historii, po czym przewiduje się, że ponownie pojawi się komunikat znajdujący się w odnalezionej pozycji.

Analiza wykazała, że skuteczność przewidywań (czyli proporcja trafień, w skrócie: PVP) sięga od 62% do 93% zależnie od używanego protokołu koherencji. W celu zwiększenia skuteczności predykcji można wprowadzić albo filtry szumu (które powodują, że dopiero kolejne nieudane predykcje powodują modyfikacje w tablicy PHT) albo zwiększyć wymiar predykcji (elementów składających się na pojedynczy wpis w tablicy MHT).

Potencjalny zysk wynikający ze stosowania predykcji autorzy oceniają na podstawie następującego wzoru:

$$\frac{czas\,bez\,dokonania\,predykcji}{czas\,z\,predykcją} = \frac{1}{p \cdot f + (1-p)(1+r)},$$

gdzie p oznacza *dokładność predykcji* (prawdopodobieństwo, że predyktor poprawnie odgadnie przyszłe zdarzenia), f oznacza *narzut* (dodatkowe opóźnienie powstałe na skutek działania mechanizmów predykcji przy poprawnej predykcji), a r – koszt popełnienia błędu, czyli opóźnienie powstałe w wyniku dokonania błędnej predykcji.

Na podstawie predyktora *Cosmos* w pracy [22] zaproponowano nowy predyktor o skuteczności przewidywań większej od poprzednika. Główna różnica w tym rozwiązaniu polega na ograniczaniu zbioru przewidywanych komunikatów tylko do tych, które faktycznie mają wpływ na stan stron. Dzięki temu dwa nowe predyktory: *MSP* (skr. ang. *Memory Sharing Predictor* – Predyktor Współdzielenia Pamięci) oraz *VMSP* (skr. ang. *Vector Message Sharing Predictor* – Wektorowy Predyktor Współdzielenia Pamięci) zwiększają dokładność predykcji przy mniejszym koszcie (w sensie rozmiaru potrzebnych struktur).

Predyktor *MSP* różni się od wcześniej omawianego rozwiązania tylko rodzajem przechowywanych komunikatów. Tylko prośby o dostęp w trybie do odczytu bądź zapisu (autorzy rozróżniają trzy takie komunikaty: *read, write* oraz *upgrade*) są zapisywane w odpowiednich strukturach. Zmniejsza to wymagania wobec pamięci oraz wrażliwość na szumy.

Predyktor *VMSP* odróżnia się od poprzednika tym, że wiele następujących po sobie sekwencji odczytu postaci <*nadawca*, *typ*> zamieniane jest na <<*wektor nadawców*>, *typ*>. Dzięki temu nie ma potrzeby utrzymywania osobnych wpisów w tablicy PHT dla kolejnych operacji odczytu, co w efekcie zwiększa efektywność dostosowywania się do wzorców

współdzielenia, zmniejsza wrażliwość na szumy i różnice w kolejności nadejścia komunikatów (żądań odczytu).

Aby przewidzieć, kiedy należy wykorzystać predykcję i rozpocząć odpowiednie działanie (tak, by uniknąć przedwczesnej spekulacji, która mimo dokonania poprawnej predykcji może zwiększyć opóźnienie) zaproponowano dwie heurystyki. Pierwsza, nazwana FR (skr. ang. *First Read*) rozpoczyna spekulację, przy nadejściu pierwszego żądania odczytu danej strony, zakładając, że wkrótce nadejdzie cała seria takich żądań. Druga o nazwie SWI (skr. ang. *Speculative Write Invalidation*) zakłada, że jeżeli proces odwoła się do innej strony niż dotychczas, oznacza to, że można rozpocząć spekulację i dokonać unieważnienia bieżącej strony. W obu przypadkach błędy w predykcji nie powodują potrzeby konstrukcji wyrafinowanych procedur obsługi. Na przykład przy użyciu drugiej heurystyki, przedwczesne unieważnienie spowoduje po prostu wysłanie kolejnego żądania o dostęp do strony.

Przedstawione techniki posiadały wadę polegającą na abstrahowaniu od rzeczywistych rozmiarów tablic predyktorów, zakładając, że są one wystarczająco duże lub o nieograniczonym rozmiarze. Nałożenie ograniczeń na rozmiary struktur danych predyktorów powoduje, że przy dodaniu nowego wpisu do tablicy może pojawić się konieczność usunięcia jednego z już istniejących. Problemem tym zajęto się w pracach [34], [36] i [37], analizując zarówno wpływ rozmiaru tablic, jak i algorytmów dodawania nowych wpisów wykorzystywanych przez predyktory na skuteczność przewidywań.

5. Zastosowanie predykcji i spekulacji – sprowadzanie z wyprzedzeniem

Jedną z używanych technik spekulacji jest spekulatywne sprowadzanie stron przed ich użyciem (ang. *prefetching*). Polega on na przewidywaniu, jakich stron węzeł będzie potrzebować w przyszłości i sprowadzaniu ich do pamięci lokalnej węzła przed ich użyciem (za pomocą zwykłych akcji przewidzianych protokołem koherencji) w celu uniknięcia opóźnień spowodowanych przez błąd strony. Ponieważ strony są sprowadzane jawnie z użyciem protokołu koherencji, błędna predykcja nie wpłynie w żaden sposób na spójność systemu, co najwyżej wprowadzając dodatkowe opóźnienie. Przyczyny takich opóźnień mogą być różne, a przykładem może być przypadek, gdy strona, która powinna być w stanie do wyłącznego użytku jednego procesu P_1, w wyniku spekulatywnej akcji procesu P_2 (będącej efektem błędnej predykcji) znajdzie się w stanie współdzielenia między procesem P_1 i P_2, co – przy próbie wykonania zapisu przez proces P_1 – spowoduje niepotrzebny błąd strony i konieczność unieważnienia innych kopii.

Wiele rozważanych dalej przykładów dotyczy systemu *TreadMarks* [19], w którym stosowany jest protokół typu LRC (skr. ang. *lazy release consistency*) opisany pokrótce dalej. Przetwarzanie w węźle podzielone jest na fazy (interwały) ograniczone dwiema następującymi po sobie operacjami synchronizującymi (barierami). Modyfikacje dokonywane na stronach są przesyłane w postaci tzw. różnic (ang. *diffs*), czyli listy zmian dokonanych na stronie przez dany węzeł. Przy błędzie dostępu do strony węzeł gromadzi potrzebne różnice i następnie aplikuje je (dokonuje zmian wyszczególnionych w nadesłanej różnicy) do posia-

danej strony w kolejności określonej przez znaczniki czasowe zawarte w różnicach (od najmniejszych do największych).

W pracy [6] opracowano technikę nazwaną Adaptive++ służącą do określania kiedy i czy dokonywać spekulacyjnego pobrania stron. Predyktor Adaptive++ działa w dwóch trybach: powtarzalnej sekwencji (ang. *repeated-phase*) oraz powtarzalnego odstępu (ang. *repeated-stride*). W pierwszym trybie zakłada się, że węzeł będzie potrzebował takiego samego zestawu stron jak ostatnio. W drugim trybie przyjmuje się natomiast, że węzeł będzie potrzebował stron o numerach w określonych odstępach (ang. *stride*).

W trybie powtarzalnej sekwencji predyktor korzysta z dwóch list: *previous-list* oraz *before-previous-list*, zawierających historię błędów dostępu do strony (wyłączając występujące w obrębie jakiejkolwiek sekcji krytycznej, tj. pomiędzy nabyciem a zwolnieniem zamka), które pojawiły się odpowiednio w ostatniej i przedostatniej fazie przetwarzania. Jeżeli zawartość tych list jest podobna (co najmniej 50% stron zawiera się w obu listach) zakłada się, że kandydatami do spekulacyjnego pobrania (do zbioru *expected*) są strony z listy *previous-list*, a w przeciwnym przypadku – z listy *before-previous-list*.

W trybie rozstępu zbiór kandydatów do spekulacyjnego pobrania tworzy się na podstawie najczęstszego odstępu między stronami w jednej z poprzednich list. Listę *expected* do analizy (w celu wyszukania rozstępu) wybiera się w identyczny sposób, jak w poprzednim trybie (tj. listę *previous-list*, gdy podobieństwo przekracza 50% itd). Kandydatami do spekulacyjnego pobrania są natomiast strony oddalone o wielokrotności odstępu od strony, dla której wykryto dopasowanie do oczekiwanego rozstępu i dla której nastąpił błąd dostępu.

Wybór trybu następuje na początku każdej fazy na podstawie miary nazwanej *użytecznością* każdego trybu. Użyteczność pierwszego trybu jest określona jako procent poprawnych spekulacyjnych pobrań w poprzedniej fazie (lub, gdy wybrano drugi tryb, procent pobrań, które byłyby poprawne, jeżeli wybrano by tryb powtarzalnej sekwencji). Dla drugiego trybu jest to częstotliwość występowania najczęstszego odstępu w wybranej liście (*previous-list* albo *before-previous-list*). W przypadku remisu, wybierany jest tryb powtarzalnej sekwencji.

Jako przykład rozważmy następującą sytuację: niech lista *previous-list* zawiera strony o numerach 12, 14, 15, 17, 19 i 29, a lista b*efore-previous-list* – o numerach 12, 17, 19, 29, 33 i 34. Cztery numery stron powtarzają się na obu listach, a więc podobieństwo wynosi około 67% (2/3). W obu trybach wybralibyśmy listę *previous-list* do dalszej analizy (jako *expected*). W pierwszym trybie (powtarzalnej sekwencji) predyktor zalecałby pobieranie stron o numerach zawartych na wybranej liście. Użyteczność tego trybu zależałaby oczywiście od liczby poprawnych predykcji w poprzedniej fazie.

W drugim trybie (powtarzalnego odstępu) zostałby określony najczęściej występujący odstęp, w tym przypadku 2 (np. 12-14, 15-17-19). Użyteczność tego trybu wynosiłaby 60% (trzy odstępy o rozmiarze dwa, w stosunku do łącznej liczby pięciu odstępów). Gdyby tryb ten został wybrany i pojawiłby się błąd strony dla strony zawartej w wybranej liście, na przykład o numerze 17, to predyktor radziłby pobrać strony o numerach 19, 21, 23 itd.

Narzut czasowy związany z wyborem trybu jest na tyle niewielki, że w przeprowadzonych przez autorów symulacjach pokrywał się z kosztem czasowym związanym z synchronizacją, czyli algorytm wyboru stron kończył się zanim procesy rozpoczynały następny interwał.

Po dokonaniu wyboru trybu, rozpoczynane są spekulacyjne pobrania. W przypadku wyboru trybu powtarzalnej sekwencji pobiera się *n* (liczba ustalona przez użytkownika – 24 w przedstawionym artykule) stron ze zbioru *expected*, oraz kolejne *k* (w artykule 4) za każdym razem, gdy pojawi się błąd dostępu do którejś z stron zawierających się w tym zbiorze. Pobrania są oczywiście wykonywane dla stron, które nie znajdują się już w pamięci lokalnej węzła w odpowiednim stanie. Dla trybu rozstępu pobrania wykonywane są tylko, gdy pojawi się błąd dostępu do którejś ze stron zawartej w zbiorze *expected*. Żadna akcja nie jest wykonywana przy błędach dostępu do stron pojawiających się w obrębie sekcji krytycznej (czyli pomiędzy operacjami nabycia i zwolnienia zamka).

Otrzymane odpowiedzi (różnice) są kolejkowane do czasu, gdy pojawi się przewidziany błąd dostępu do strony i dopiero są one uwzględniane (aplikowane), dodatkowo ewentualnie zbierane są i aplikowane te różnice, dla których nie dokonano spekulacyjnego pobrania, po czym strona zmienia odpowiednio stan.

Propozycje spekulatywnego pobierania stron można również znaleźć w pracach [1], [7] i [38].

Kolejną pracą opisującą wpływ spekulacyjnego pobrania na system DSM jest publikacja [14] przedstawiająca budowę systemu *Delphi* – programowego systemu pamięci rozproszonej, opartego o protokół z węzłami domowymi (ang. *home nodes*). Każdy węzeł w tym systemie posiada programowy predyktor określający, jakie strony prawdopodobnie będą potrzebne w przyszłości. Początkowo każdy węzeł otrzymuje przypisane (według algorytmu *round robin*) grupy zawierające po 10 stron. Jeżeli jakiś węzeł jest jedynym dokonującym zapisów na jakiejś stronie, strona ta jest migrowana do tego węzła, czyli staje się on nowym węzłem domowym tej strony. System *Delphi* stosuje protokół typu *multiple-writer*, to znaczy, że (podobnie jak w opisanym bliżej systemie *TreadMarks*) wiele węzłów naraz może dokonywać zapisów na stronie, a wynikające z tych zapisów różnice są przesyłane do węzła domowego. Wszystkie strony są unieważniane podczas wystąpienia dowolnego zdarzenia synchronizującego, takiego jak bariera, powodując równocześnie utworzenie różnic i wysłanie ich do węzła domowego.

Predyktor używany w tym systemie używa metody różnicowania na podstawie kontekstu o skończonej długości (ang. *differential finite context method*, w skrócie: DFCM). Monitoruje on dostępy do pamięci współdzielonej i dla każdych trzech następujących po sobie błędów dostępu, zapamiętuje on adres strony, dla której nastąpił później błąd dostępu. Predyktor jest dwupoziomowy. Pierwszy poziom przechowuje numery stron z trzech ostatnich dostępów, przy czym przechowywany jest adres ostatniej strony oraz odstęp w stosunku do tego adresu dla pozostałych dwóch stron (liczba ta oznacza rząd predyktora DFCM, a liczba trzy została wybrana eksperymentalnie jako dająca najlepsze wyniki). Odstępy są argumentem dla funkcji haszowej, która na ich podstawie tworzy indeks do tablicy haszowej, będącej drugim poziomem predyktora. Numery stron przechowywane w tej tablicy są zapisywane w postaci różnic (odległości) od bieżącej strony i są aktualizowane (bądź dopisywane) przy każdym błędzie dostępu. Tak więc, jeżeli ostatnio węzeł odwoływał się do stron 0xFFBE9, 0xFFBEA i 0xFFBEC, a predyktor przewiduje, że kolejny dostęp będzie do strony 0xFFBEE, to pierwszy poziom będzie zawierał numery 0xFFBE9, 1, 3, a drugi poziom numer 5.

Podczas aktualizacji tablic stron istniejące wpisy są nadpisywane. Gdy nastąpi błąd dostępu do strony, predyktor sugeruje pobranie *n* kolejnych stron (rozdzielonych w stosunku

do strony powodującej błąd odstępem, przewidzianym na podstawie zapamiętanej historii), jednakże tylko tych, które są w tym samym węźle domowym, co strona powodująca błąd dostępu. Przetwarzanie jest wznawiane po otrzymaniu pierwszej z tych stron.

Przedstawienie samych algorytmów bez pokazania wynikających z nich korzyści mogłoby nie przekonać do słuszności stosowania prezentowanych pomysłów; jednakże uważamy, że zestawienie wyników publikowanych przez autorów przedstawionych prac byłoby pewnym nadużyciem, gdyż każdy z tych algorytmów został zbadany w innym środowisku. Dlatego też przedstawiamy najlepsze oraz średnie wyniki (przyśpieszenie pracy programu dzięki zastosowaniu spekulacji) osiągane za pomocą prezentowanych technik, zaznaczając jednak, że należy je traktować poglądowo, po szczegółową zaś analizę odsyłamy po odpowiednich pozycji literatury.

Tabela 2. Przykładowe przyśpieszenie możliwe do osiągnięcia dzięki zastosowaniu spekulacji

Predyktor	*Największe przyśpieszenie*	*Średnie przyśpieszenie*
Adaptive++	34%	13%
Cosmos	25%	12%
DFCM [14]	14%	4,9%

6. Inne zastosowania spekulacji w systemach rozproszonych

Opisane w poprzednim punkcie przykłady nie są oczywiście jedynymi możliwymi zastosowaniami spekulacji. Metodą symetryczną wobec spekulacyjnego sprowadzania stron jest spekulacyjne wysyłanie (ang. *speculative push*) stron do wybranego węzła przewidując, że wkrótce będzie tych stron potrzebować. Tego rodzaju rozwiązania są opisane w pracach [5] i [43]. Opierają się one na obserwowaniu związków między stronami a obiektami synchronizacyjnymi (tj. stronami, do których często następują odwołania zaraz po nabyciu obiektów synchronizacyjnych).

Inną techniką jest spekulacyjne samounieważnianie stron, jeżeli istnieje podejrzenie, że węzeł nie będzie już ich potrzebował [17, 23, 35]. Pozwala to zredukować koszt aktualizacji replik stron w protokołach klasy *write-update*.

Ponieważ dla poszczególnych wzorców współdzielenia stron istnieją dobrze znane techniki przyśpieszające działanie aplikacji, zaproponowano cały szereg rozwiązań wybierających dynamicznie pewien sposób zachowania na podstawie próby odgadnięcia, jaki typ wzorca współdzielenia reprezentuje strona [21, 30, 32].

Kolejną metodą może być przewidywanie właścicieli stron [2] tak, by móc ominąć kosztowne odwołania do katalogu, czy przewidywanie adresatów komunikatów [11, 39] dla systemów opartych na protokołach koherencji typu „nasłuchiwania" (ang. *snooping*), a nawet przewidywanie wartości [13] (na podstawie posiadanych nieaktualnych wartości pochodzących z unieważnionej strony, przewidując w oparciu o fenomen „fałszywego współdzielenia", że nie uległy one zmianom). Innym podejściem może być propozycja przetwarzania

quasitransakcyjnego [12, 28], w którym wymaga się, by aplikacja jawnie rozpoczynała spekulację określając, jakie warunki musi spełniać stan przetwarzania, by spekulacja została zatwierdzona i akcje przywracające stan normalny w wypadku niemożliwości zatwierdzenia spekulacji. Istnieją również propozycje tzw. maszyn spekulatywnych zawierających sprzętowe wsparcie dla spekulacji [16], spekulatywnej synchronizacji [40] oraz wykorzystywania współbieżności na poziomie wątków (ang. *thread level parallelism*) w celu równoczesnego uruchomienia kilku alternatywnych ścieżek przetwarzania i następnie zatwierdzania tylko tych, które okazały się prawdziwe [33, 40].

7. Wnioski i uwagi końcowe

Zastosowanie spekulacji w systemach rozproszonych jest dziedziną ciekawą zarówno z punktu widzenia poznawczego, jak i praktyki zastosowań, ważną, aktualną i wartą bliższego poznania. Eksperymenty przeprowadzane przez badaczy pozwalają mieć nadzieję, że spekulacja może w znaczący sposób przyśpieszyć działanie systemów z rozproszoną pamięcią współdzieloną. Niniejsza praca jest próbą przybliżenia pojęć związanych z spekulacją.

Podjęta w pracy próba sformalizowania pojęć predykcji i spekulacji powinna pomóc w próbach ścisłego ujęcia przedstawionych problemów i może stanowić podstawę do badań nad poprawnością spekulacji w uogólnionych modelach oraz wprowadzenia jej do systemów pamięci rozproszonej opartej o obiekty.

Przedstawiono budowę najpopularniejszych predyktorów, problemy związane z ich implementacją i niektóre propozycje rozwiązania tych problemów. Opisany przykład działania spekulacji powinien pomóc Czytelnikowi w zrozumieniu oraz w dalszym samodzielnym poznawaniu tej interesującej dziedziny wiedzy.

8. Kierunki dalszych badań

Kwestie, które wydają się nam szczególnie interesujące i warte dodatkowej pracy, obejmują zagadnienia wpływu mechanizmów spekulacji na zapisywanie i odtwarzanie stanu aplikacji. Przygotowywany jest artykuł analizujący możliwe problemy pojawiające się przy wprowadzeniu spekulacji. Drugim zagadnieniem wartym analizy jest kwestia wprowadzenia mechanizmów spekulacji do systemów z współdzieloną pamięcią rozproszoną opartą o obiekty oraz idea spekulatywnego wywoływania metod obiektów.

Literatura

[1] Abud, M., Amorim, C. L., , De Maria, M.., Bianchini, R., Kontothanassis, L. I., Pinto, R., Hiding Communication Latency and Coherence Overhead in Software DSMs, w: *Proceedings of the 7th ASPLOS*, 1996.

[2] Acacio, M.. E., Duato, J., Gonzales, J., Garcia, J. M., Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in a cc-NUMA Architecture, w: *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, 2002.

[3] Acquaviva, J. T., Jalby, W., Hardware Prediction for Data Coherency of Scientific Codes on DSM, w: *Proceedings Supercomputing*, 2000.
[4] Ahamad, M., Schwan, K., Tacic, I., West, R., Exploiting Temporal & Spatial Constraints on Distributed Shared Objects, w: *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS'97)*.
[5] Alvisi, L., Kistler, M., *Improving the Performance of Software Distributed Shared Memory with Speculation. Department of Computer Sciences*, Technical Report TR-02-57, The University ofTexas at Austin, Austin 2002.
[6] Amorim, C. L., Bianchini, R., Pinto, R., Data Prefeteching for Software DSMs, w: *Proceedings of the International Conference on Supercomputing*, 1998.
[7] Amza, C., Cox, A., Rajamani, K., Zwaenepoel, W., Tradeoffs Between False Sharing and Aggregation in Software Distributed Shared Memory, w: *Proceedings of the 6th PpoPP*, 1997.
[8] Barroso, L. A., Bugnion, E., Gharachorloo, K., Memory System Characterisation of Commercial Workloads, w: *Proceedings of the 25th International Symposium on Computer Architecture*, 1998.
[9] Bennett, J. K., Carter, J. B., Zwaenepoel, W., Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence, w: *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 168–176, 1990.
[10] Bennett, J. K., Carter, J. B., Zwaenepoel, W., Adaptive Software Cache Management for Distributed Shared Memory Architectures, w: *Proceedings of the 17th International Symposium on Computer Architecture*, 125–134, 1990.
[11] Bilir, E. E., Dickson, R. M., Hu, Y., Plakal, M., Sorin. D. J., Hill, M.. D., Wood, D.A., Multicast Snooping: A New Coherence Method Using a Multicast Address Network, w: *Proceedings of the 26th Annual Int'l Symp. on Computer Architecture (ISCA'99)*, 1999.
[12] Borkowski, J., Interrupt and Cancellation as Synchronization Methods, *PPAM* 2001, 3–9.
[13] Burger, D., Chang, J., Desikan, R., Huh, J., Sohi, G., Using Coherent Value Speculation to Improve Multiprocessor Performance, *First Value Prediction Workshop*, 2003.
[14] Burtscher, M., Speight, E., Delphi: Prediction-Based Page Prefetching to Improve the Performance of Shared Virtual Memory Systems, w: *2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 49–55, 2002.
[15] Chapman, B., Hernandez, D., Patil, A., Prabhakar, A., Program Development for OpenMP Programs on ccNUMA Architectures, w: *Proceedings of the Large Scale Computations,* Sozopol 2001.
[16] Chen, M., Hammond, L., Hubbert, B. A., Olokotun, K., Prabhu, M.. K., Siu, M., The Stanford Hydra CMP, w: *IEEE Micro*, 71–84, 2000.
[17] Cho, J. W., Lee, J., Sihn, K-H., A Speculative Coherence Scheme using Decoupling Synchronization for Multiprocessor Systems, *Computer Architecture Letters* 2 (2003).
[18] Cichosz, P., *Systemy uczące się*, Wydawnictwa Naukowo-Techniczne, Warszawa 2000.
[19] Cox, A. L., Dwarkadas, S., Keleher, P., Zwaenepoel, W., TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems, w: *Proceedings of the 1994 Winter USENIX Conference*, 1994.
[20] Cox, A. L., Dwarkadas, S., Lu, H., Rajamon, R., Zwaenepoel, W., Compiler and Software Distributed Shared Memory Support for Irregular Applications, w: *Proceedings of the Sixth ACM Sigplan Symposium on Principles and Practice of Parallel Programming (PpoPP'97)*, 1997.
[21] Cox, A. L., Fowler, R. J., Adaptive Cache Coherency for Detecting Migratory Shared Data, w: *Proceedings of the International Symposium on Computer Architecture (ISCA'93)*, 1993.
[22] Falsafi, B., Lai, A-C., Memory Sharing Predictor: The Key to a Speculative Coherent DSM, w: *Proceedings of the 26th International Symposium on Computer Architecture (ISCA 26)*, 1999.
[23] Falsafi, B., Lai, A-C., Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction, w: *Proceedings of the 27th International Symposium on Computer Architecture (ISCA 27)*, 2000.
[24] Govindarajan, R., Manuj, N. P., *CAS-DSM. A Compiler-Assisted Software DSM*, 1999.
[25] Gupta, A., Weber, W-D., Analysis of Cache Invalidation Patterns in Multiprocessors, w: *Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Systems (ASPLOS III)*, 243–256, 1989.
[26] Gupta, A., Weber, W-D., Cache Invalidation Patterns in Shared-Memory Multiprocessors, *IEEE Transactions on Computers* 41 (7) (1992).

[27] Hammond, L., Olukotun, K., Willey, M., Improving the Performance of Speculatively Parallel Applications on the Hydra CMP, w: *Proceedings of International Conference on Supercomputing (ICS'99)*, 1999.
[28] Hickey, J., Smith, J. D., Tapus, C,. Kernel Level Speculative DSM, w: *Workshop on Distributed Shared Memory on Clusters (DSM 2003)*, 2003 (IEEE International Symposium on Cluster Computing and the Grid – CCGRID 2003).
[29] Jóźwiak, J., Podgórny, J., *Statystyka od podstaw*, PWN, Warszawa 1994.
[30] Kaxiras, S., *The Use of Instruction Based Prediction in Hardware Shared Memory*, University of Wisconsin, Computer Sciences Dept., TR-1368, 1998.
[31] Kaxiras, S., Young, C., Coherence Communication Prediction in Shared-Memory Multiprocessors, w: *Proc. HPCA-6*, 2000.
[32] Kim, J-H., Vaidya, N. H., Adaptive Migratory Scheme for Distributed Shared Memory, w: *International Conference on Supercomputing ICS'97*, 1997.
[33] Lam, M.. S., Oplinger, J., Enhancing Software Reliability with Speculative Threads, w: *Proceeding of the Conference on Architectural Support for Programming Languages and Operating Systems*, 184–196, 2002.
[34] Landin, A., Nilsson, J., Stenstrom, P., The Coherence Predictor Cache: A Resource-Efficient and Accurate Coherence Prediction Infrastructure, w: *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.
[35] Lebeck, A. R., Wood, D. A., Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors, w: *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, 48–59, 1995.
[36] Lee, J., Maeng, S. R., Park, S., *Pattern Cache: Design of a Coherence Message Predictors with Reduced Storage Overhead*, 2001.
[37] Lee, J., Maeng, S. R., Park, S., A Dynamic Table Allocation Scheme for Reducing Storage Overhead of Coherence Message Predictors, w: *International Conference on Computer and Information Science (ICIS'01)*, 2001.
[38] Lee, S-K., Lee, J., Maeng, S., Yun, H-Ch., Adaptive Prefetching Technique for Shared Virtual Memory, w: *The 3rd International Workshop on Software Distributed Shared Memory System*, 2001.
[39] Martin, M.. M.. K., Harper, P.. J., Sorin, D. J., Hill, M.. D., Wood, D. A., Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors, w: *The 30th Annual International Symposium on Computer Architecture*, 206–217, 2003.
[40] Martinez, J. F., Torrellas, J., Speculative Synchronisation: Applying Thread-Level Speculation to Explicitly Parallel Applications, w: *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.
[41] Mukharjee, S. S., Hill, M.. D., Using Prediction to Accelerate Coherence Protocols, w: *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA)*, 1998.
[42] Patt, Y., Yeh, T-Y., Alternative implementation of Two-Level Adaptive Branch Prediction, w: *Proceedings of the 19th Annual International Symposium on Computer Achitecture*, 124–134, 1992.
[43] Rajwar, R., Kagi, A., Goodman, J. R., Inferential Queueing and Speculative Push for Reducing Critical Communication Latencies, w: *International Conference on Supercomputing ISC'03*, 2003.
[44] Rothman, J. B., Smith, A. J., *Analysis of Shared Misses and Reference Patterns*, Report No. UCB/CSD-99-1064, 1999.

Selected Speculation Applications in Distributed Shared Memory Systems

Jerzy BRZEZIŃSKI, Arkadiusz DANILECKI

Poznań University of Technology, Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznań, Poland
e-mail: arkadiusz.danilecki@cs.put.poznan.pl

Received July 26, 2005

Abstract. This paper presents and discusses the use of speculation in distributed shared memory systems. It explains the basis of speculation, especially in the context of sharing patterns. Additionally, the most popular and well-known general predictors are outlined. In this paper new formal definitions of speculation and prediction are proposed, and the characteristics of predictors' quality are discussed. Finally, prefetching methods are shown as an example of the application of speculation.

Key words: data speculation, prediction, distributed shared memory, prefetching