POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Imię i nazwisko autora rozprawy: Artur Malinowski
Dyscyplina naukowa: Informatyka

ROZPRAWA DOKTORSKA

Tytuł rozprawy w języku polskim: Zastosowanie bajtowo adresowanej pamięci NVRAM do zwiększenia wydajności wybranych aplikacji równoległych wykorzystujących MPI I/O

Tytuł rozprawy w języku angielskim: Applying byte-addressable NVRAM memory to increase the performance of selected parallel applications based on MPI I/O

Promotor *podpis*
dr hab. inż. Paweł Czarnul, prof. nadzw. PG

Gdańsk, 2019 r.

POLITECHNIKA GDAŃSKA
WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

OPIS ROZPRAWY DOKTORSKIEJ

Streszczenie rozprawy w języku polskim: Obecnie wiele badań podejmuje temat rosnącego problemu wydajności operacji na plikach w środowiskach klastrowych. Jednocześnie, według ostatnich doniesień związanych z rozwojem technologii pamięci komputerowych, w najbliższej przyszłości na rynku powinny pojawić się układy trwałej pamięci o dostępie swobodnym, adresowanej bajtowo. Niniejsza rozprawa pokazuje, że przy użyciu takiej pamięci można zwiększyć wydajność wybranych aplikacji przetwarzających dane zgromadzone we współdzielonych plikach. Praca skupia się na autorskim rozwiązaniu – rozproszonej pamięci podręcznej, kompatybilnej z interfejsem popularnego standardu dostępu do plików w klastrach jakim jest MPI I/O. Do weryfikacji poprawy wydajności wykorzystano dwa syntetyczne benchmarki oraz cztery aplikacje użytkowe, a same testy przeprowadzono przy użyciu sprzętowego symulatora pamięci o nowych parametrach. Na rozprawę składa się wprowadzenie teoretyczne ze szczególnym uwzględnieniem najnowszych badań, szczegółowy opis architektury oraz implementacji zaproponowanego rozwiązania, charakterystyka zestawu aplikacji demonstrujących wraz z wynikami eksperymentów wydajnościowych i komentarzem rezultatów, oraz podsumowanie i nakreślenie dalszych kierunków prac.

Streszczenie rozprawy w języku angielskim: Nowadays, as reported in many scientific papers, the problem of file operations' performance in cluster environments becomes more and more important. At the same time, according to recent reports related to the development of computer memory technologies, in the nearest future we can expect new devices, equipped with byte addressable non-volatile random access memory. This thesis shows that using emerging memory technologies, it is possible to increase the performance of selected applications that process shared files. The work focuses on the proposed solution, based on a distributed cache, compatible with MPI I/O – the popular interface for file operations in cluster environment. Improvement of the performance was verified using two synthetic benchmarks and four utility applications. All tests were executed using a configurable hardware memory simulator. The dissertation covers following topics: a theoretical introduction including latest research, a detailed description of architecture and implementation of the proposed solution, properties of a set of demonstrating applications together with results of performance experiments and discussion, final conclusions, and an outline of planned future work.

MOST WIEDZY Downloaded from mostwiedzy.pl

Spis treści

1 Wstęp **15**

1.1 Teza i cele pracy 19

1.2 Struktura rozprawy 20

1.3 Publikacje 21

2 Adresowana bajtowo pamięć nieulotna o dostępie swobodnym **23**

2.1 Pamięć w komputerach obecnie 23

2.2 Pamięć uniwersalna 26

3 Obliczenia wysokiej wydajności i operacje na plikach **33**

3.1 Wprowadzenie do I/O w HPC 33

3.2 Optymalizacje niezależne od zastosowanej pamięci 41

3.3 Optymalizacje bazujące na pamięci opartej o SSD 45

3.4 Rozwiązania typu *burst buffer* 49

3.5 Podsumowanie 51

4 Zastosowanie adresowanej bajtowo pamięci NVRAM w obliczeniach wysokiej wydajności **55**

4.1 Optymalizacje obliczeń i zapobieganie utracie danych 55

4.2 Optymalizacje operacji I/O 62

5 Rozproszona pamięć podręczna dla aplikacji wykorzystujących MPI I/O **69**

5.1 Definicja problemu 69

5.2 Projekt rozwiązania 71

5.3 Praca w środowisku podatnym na awarie 76

MOST WIEDZY Downloaded from mostwiedzy.pl

5.4 Monitorowanie pamięci podręcznej 81
5.5 Potencjalne ograniczenia 83
5.6 Realizacja rozwiązania 85
5.6.1 Biblioteka libmpi-io-nvram 85
5.6.2 Główne problemy implementacji i rozwiązania 88

6 Testy rozwiązania 93

6.1 Wprowadzenie do testów rozwiązania 93
6.2 Środowisko testowe 94
6.3 Benchmark ROMPIO 96
6.4 Benchmark – błądzenie losowe 98
6.5 Przetwarzanie dużych obrazów 101
6.6 Wyznaczanie potęgi grafu 107
6.7 Przeszukiwanie dwuwymiarowej mapy 109
6.8 Symulacja zachowania tłumu 110
6.9 Narzut mechanizmów ograniczających skutki awarii 115

7 Podsumowanie i dalsze kierunki prac 119

7.1 Podsumowanie i wnioski 119
7.2 Dalsze kierunki prac 122

MOST WIEDZY Downloaded from mostwiedzy.pl

If you need inspiring words, don't do it.

Jeśli potrzebujesz inspirujących słów, nie rób tego.

Elon Musk

MOST WIEDZY Downloaded from mostwiedzy.pl

Podziękowania

Dziękuję profesorowi Pawłowi Czarnulowi, zarówno za wszechstronną pomoc, której udzielał mi jako promotor tej pracy, jak i przede wszystkim za sumienne i wytrwałe wypełnianie roli opiekuna naukowego.

Dziękuję mojej żonie Ali za wszelkie wsparcie – nie takie, jakiego oczekiwałem, ale takie, jakiego potrzebowałem.

Dziękuję mojej rodzinie i przyjaciołom, głównie za to, że wytrzymali te kilka lat słuchania mojego powtarzanego jak mantrę „nie mam czasu”.

MOST WIEDZY Downloaded from mostwiedzy.pl

Wykaz oznaczeń

API	–	interfejs programistyczny aplikacji (ang. *application programming interface*)
DRAM	–	pamięć dynamiczna o dostępie swobodnym (ang. *dynamic random-access memory*)
FLOPS	–	operacje zmiennoprzecinkowe na sekundę (ang. *floating point operations per second*)
HDD	–	dysk twardy (ang. *hard disk drive*)
HPC	–	obliczenia wysokiej wydajności (ang. *high performance computing*)
I/O	–	wejścia-wyjście, w rozprawie: operacje na plikach (ang. *input/output*)
MPI	–	interfejs przesyłania komunikatów pomiędzy procesami (ang. *message passing interface*)
NVDIMM	–	standard modułów pamięci NVRAM (ang. *non-volatile dual in-line memory module*)
NVRAM	–	nieulotna pamięć o dostępie swobodnym (ang. *non-volatile random-access memory*)
PCM	–	pamięć zmiennofazowa (ang. *phase-change memory*)
PFS	–	równoległy system plików (ang. *parallel file system*)
PMDK	–	zestaw narzędzi dla programistów ułatwiający pracę z pamięcią NVRAM (ang. *persistent memory development kit*)
POSIX	–	przenośny interfejs dla systemu operacyjnego Unix (ang. *portable operating system interface for Unix*)

MOST WIEDZY Downloaded from mostwiedzy.pl

RAM	–	pamięć o dostępie swobodnym (ang. *random-access memory*)
RDMA	–	zdalny bezpośredni dostęp do pamięci (ang. *remote direct memory access*)
SRAM	–	statyczna pamięć o dostępie swobodnym (ang. *static random access memory*)
SSD	–	dysk półprzewodnikowy (ang. *solid-state drive*)
SSHD	–	dysk hybrydowy (ang. *solid-state hybrid drive*)

MOST WIEDZY Downloaded from mostwiedzy.pl

Wykaz symboli

APP	–	aplikacja (ang. *application*)
CMP	–	zbiór operacji obliczeniowych (ang. *computation operations*)
COM	–	zbiór operacji komunikacyjnych (ang. *communication operations*)
ET	–	czas działania aplikacji (ang. *execution time*)
HW	–	sprzęt (ang. *hardware*)
I	–	zbiór połączeń elementów sprzętowych węzła (ang. *interconnect*)
L	–	zbiór łącz komunikacyjnych (ang. *communication links*)
M	–	zbiór układów pamięci (ang. *memory*)
MOP	–	zbiór operacji wykonywanych na pamięci (ang. *memory operations*)
MW	–	oprogramowanie pośredniczące (ang. *middleware*)
N	–	zbiór węzłów klastra (ang. *nodes*)
O	–	zbiór pojedynczych operacji wykonywanych w ramach aplikacji (ang. *operation*)
OS	–	system operacyjny (ang. *operating system*)
P	–	zbiór procesów (ang. *process*)
PU	–	jednostki przetwarzające (ang. *processing units*)
SW	–	oprogramowanie (ang. *software*)
T	–	zbiór wątków (ang. *thread*)
n	–	węzeł klastra
o	–	pojedyncza operacja
p	–	proces
t	–	wątek

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 1

Wstęp

Obliczenia wysokiej wydajności (*High Performance Computing*, HPC) od zawsze wiązały się z kosztownym sprzętem. Duże zapotrzebowanie na moc obliczeniową spowodowało szybki rozwój superkomputerów, związany z jednej strony ze wzrostem wydajności pojedynczych procesorów, a z drugiej z ich liczbą. Obecnie najszybsze superkomputery to klastry składające się z dziesiątek tysięcy węzłów, z których każdy jest wyposażony w kilka procesorów, przeważnie wielordzeniowych. W praktyce wykorzystuje się od kilku rdzeni w zwykłych procesorach, do kilkudziesięciu w rozwiązaniach takich jak Intel Xeon Phi drugiej generacji[1]. Moc obliczeniową dodatkowo zwiększają akceleratory sprzętowe, między innymi karty graficzne, na przykład zgodne z technologią Nvidia CUDA[2] lub OpenCL[3], czy specjalne wielordzeniowe koprocesory, takie jak Intel Xeon Phi pierwszej generacji[4]. Według listy top500, opublikowanej w listopadzie 2018 roku[5], najszybszy sklasyfikowany superkomputer, Summit, był wyposażony w łącznie ponad 2 miliony rdzeni i osiągnął wydajność ponad 140 PFLOPS ($140 \cdot 10^{15}$ operacji zmiennoprzecinkowych na sekundę). Znaczną część wysokich miejsc na liście zajęły również klastry, których węzły wyposażono w akceleratory takie jak Intel Xeon Phi czy NVIDIA Tesla.

Przy tak dużej mocy obliczeniowej wiele aplikacji wymaga także szybkiego dostępu do danych. O ile w ramach pojedynczego węzła jest to stosunkowo łatwe do zrealizowania, głównie dzięki dużej pamięci operacyjnej i szybkiej pamięci masowej, o tyle możliwość wydajnego odczytu i zapisu danych z poziomu każdego węzła klastra jest dużym wyzwaniem. Oprócz dodatkowego narzutu czasowego na transport danych przy wykorzystaniu infrastruktury sieciowej i konieczności zaprojektowania właściwego przepływu danych po-

[1] https://ark.intel.com/products/series/92650/Intel-Xeon-Phi-x200-Product-Family
[2] https://developer.nvidia.com/cuda-zone
[3] https://www.khronos.org/opencl/
[4] https://ark.intel.com/products/series/92649/Intel-Xeon-Phi-x100-Product-Family
[5] https://www.top500.org/list/2018/11/

MOST WIEDZY Downloaded from mostwiedzy.pl

między maszynami, pojawia się problem organizacji dużego wolumenu żądań, nierzadko kierowanych do pojedynczego zasobu.

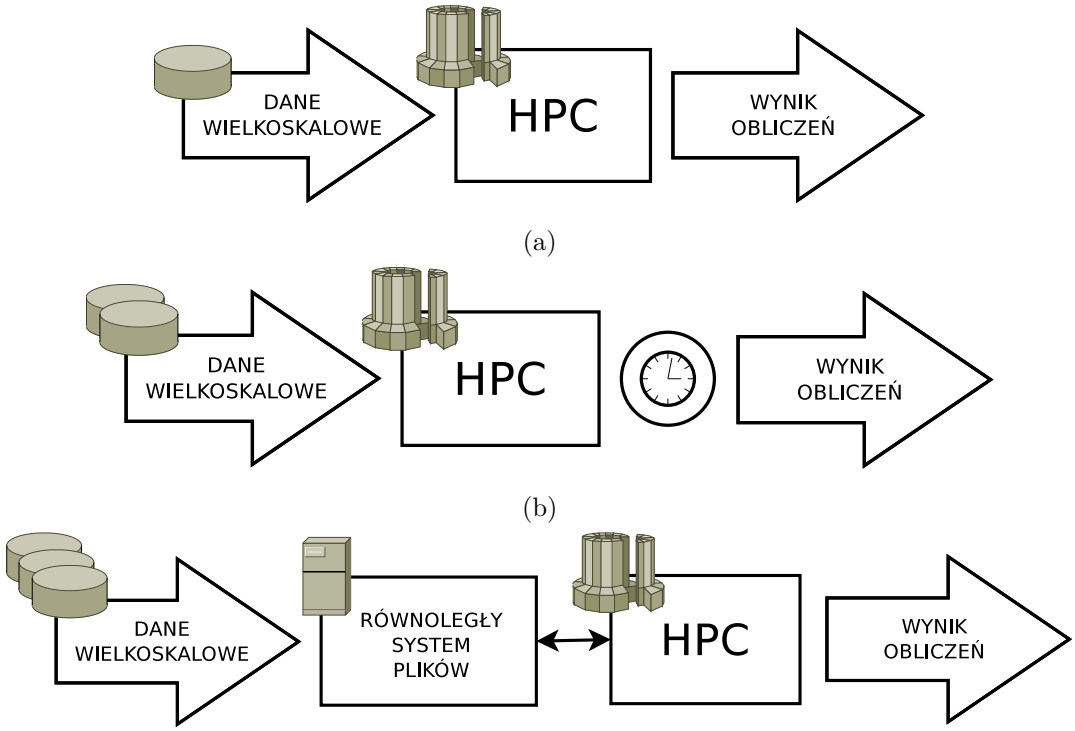
Podstawowym sposobem przechowywania danych w aplikacjach równoległych wysokiej wydajności są współdzielone pliki. Takie pliki zarządzane są przez klastrowe systemy plików, nazywane również równoległymi lub PFS (*Parallel File System*) ze względu na możliwość pracy w konfiguracji złożonej z wielu instancji serwerów. Chociaż obecnie wykorzystuje się różne systemy plików, takie jak Lustre[6], OrangeFS[7], czy HDFS[8], dostęp do danych realizowany jest zazwyczaj albo przez podstawowy interfejs oferowany przez system operacyjny, albo przez dedykowane biblioteki (na przykład MPI I/O) [58]. Jeśli chodzi o zapewnienie wysokiej wydajności pracy, to zarówno w systemach plików, jak i w bibliotekach pośredniczących, korzysta się z szeregu optymalizacji, mających na celu zwiększenie przepustowości i zmniejszenie opóźnień odczytu oraz zapisu. Dodatkowo, projektant aplikacji powinien stosować się do zalecanych dobrych praktyk, takich jak unikanie dostępu do małych fragmentów plików czy organizacja kolejnych żądań w takich sposób, żeby operowały na fragmentach niedaleko od siebie oddalonych [18, 30, 84, 113]. Niestety, mimo wszystko dla wielu aplikacji intensywnie przetwarzających dane, generowane obciążenie jest na tyle wysokie, że operacje plikowe stają się wąskim gardłem całego systemu [6, 58].

Grafiki zgromadzone na rysunku 1.1, oparte o materiały szkoleniowe dostępne w Internecie, w prosty sposób ilustrują powyższe rozważania. W celu przyspieszenia obliczeń zorientowanych na dane wielkoskalowe zgromadzone w plikach, powszechnie stosowane są duże klastry obliczeniowe, wspierane różnorodnymi technikami tworzenia aplikacji HPC (a). Niestety, wraz ze wzrostem rozmiaru danych, szybkim rozwojem jednostek obliczeniowych i stosunkowo powolnym wzrostem wydajności pamięci masowych, operacje na plikach okazują się często wąskim gardłem całego systemu (b). Rozwiązaniem są wyspecjalizowane, równoległe systemy plików (c), oczywiście odpowiednio skonfigurowane i zoptymalizowane.

Problemy niedostatecznej mocy obliczeniowej są rozwiązywane jednocześnie przez optymalizację algorytmów oraz wykorzystanie nowego sprzętu komputerowego, takiego jak akceleratory sprzętowe. W przypadku równoległego dostępu do plików jest podobnie. Do oprogramowania systemów plików i bibliotek klienckich wdrażane są kolejne optymalizacje i usprawnienia, na przykład bardziej wydajna obsługa współbieżności, poprawione mecha-

[6] `http://lustre.org/`
[7] `http://www.orangefs.org/`
[8] `https://wiki.apache.org/hadoop/HDFS`

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 1.1: Grafika poglądowa ilustrująca wyzwania stawiane dzisiaj przed HPC

nizmy zarządzania pamięcią podręczną czy lepiej dobrane parametry domyślne. Duża część z nich, oprócz zmian w oprogramowaniu, wykorzystuje zalety sprzętu, między innymi pamięci masowej opartej o napędy półprzewodnikowe SSD (*Solid-State Drive*) czy szybkie sieci InfiniBand.

Dotychczas, twórcy rozwiązań usprawniających operacje plikowe w klastrach obliczeniowych wspierali się dwoma rodzajami pamięci. Pamięć operacyjna i pamięć masowa oparta o SSD służyły jako dwa poziomy pamięci podręcznej w dostępie do plików zlokalizowanych na wolniejszych macierzach dyskowych złożonych z napędów HDD (*Hard Disk Drive*). Przykładem takich architektur może być dwupoziomowy dostęp do danych ze zintegrowanym systemem plików w pamięci RAM [118] czy rozwiązania takie jak struktury pamięci podręcznej, oparte o nośniki SSD [32, 34]. Główną zaletą technologii wykorzystywanej do budowy pamięci operacyjnej jest jej bajtowe adresowanie, duża przepustowość i niskie opóźnienia w dostępie do danych, podczas gdy często wolniejsze i operujące na blokach pamięci masowe cechuje duża pojemność i trwałość zapisu.

Naukowcy w wielu oddziałach badawczych szukają układów pamięci, które połączy-

MOST WIEDZY Downloaded from mostwiedzy.pl

łyby zalety pamięci masowej z zaletami pamięci operacyjnej i pozwoliły na zbudowanie tak zwanej pamięci uniwersalnej. Taka pamięć powinna być adresowana bajtowo, nieulotna, ale równocześnie stosunkowo szybka i pojemna. Na przestrzeni ostatnich lat pojawiło się wiele technologii, które pretendowały do budowy takich układów, na przykład pamięć PCM (*Phase-Change Memory*) czy MRAM (*Magnetoresistive Random-Access Memory*). Przeszkody technologiczne i wysokie koszty produkcji uniemożliwiały jednak ich szerokie, komercyjne zastosowanie [54]. W 2015 roku firmy Intel i Micron ogłosiły, że posiadają technologię, która w najbliższym czasie pozwoli na stworzenie układów, łączących zalety pamięci operacyjnej i urządzeń opartych o SSD [39]. Oprócz zapowiedzi odbyły się także demonstracje prototypów [16]. Inne korporacje, takie jak IBM czy Samsung, również prowadzą badania w tym zakresie, okresowo publikując notatki prasowe informujące o postępach. W najbliższych latach (według zapowiedzi być może już w 2019 roku [40]) powinniśmy spodziewać się na rynku pierwszych urządzeń opartych o bajtowo adresowaną pamięć NVRAM (*Non-Volatile Random-Access Memory*).

Taka pamięć znajdzie wiele zastosowań w aplikacjach równoległych wysokiej wydajności. Ze względu na spodziewaną pojemność powinna poprawić szybkość wykonywania algorytmów o dużej złożoności pamięciowej, a trwałość zapisu pozwoli zmniejszyć prawdopodobieństwo utraty danych w wypadku awarii. Pojawiają się także mniej oczywiste sposoby wykorzystania pamięci NVRAM, na przykład przyspieszenie tak zwanego checkpointingu, czyli cyklicznego tworzenia punktów kontrolnych aplikacji, z których w razie awarii można odtworzyć jej stan [20].

Możliwe jest również, że pamięć NVRAM pozwoliłaby poprawić wydajność operacji na plikach w HPC. Intuicja podpowiada, że skoro nowe pamięci mają rozszerzać możliwości istniejących bez wprowadzania dodatkowych ograniczeń, wystarczającą zmianą powinno być wykorzystanie pamięci NVRAM w zastępstwie SSD. Okazuje się jednak, że optymalizacje stosowane w systemach plików i oprogramowaniu pośredniczącym są dostosowane ściśle do obecnie stosowanych architektur sprzętowych, a efekt takiej modyfikacji będzie co najwyżej nieznaczny [65, 69]. Wynika stąd, że dotychczas zaproponowane metody nie wykorzystują wszystkich zalet nowych rodzajów pamięci. Należy więc przeprowadzić odpowiednie badania w celu opracowania nowego rozwiązania, dostosowanego do spodziewanej specyfikacji urządzeń i pokazać potencjalny zysk z zastosowania nowego podejścia. Niniejsza rozprawa wykaże, że adresowana bajtowo pamięć NVRAM zastosowana w środowisku klastrowym umożliwia zwiększenie wydajności wybranych aplikacji równoległych, w szcze-

MOST WIEDZY Downloaded from mostwiedzy.pl

gólności opartych o MPI I/O.

1.1 Teza i cele pracy

Teza pracy brzmi: zastosowanie bajtowo adresowanej pamięci NVRAM w środowisku klastrowym umożliwia zwiększenie wydajności wybranych aplikacji równoległych wykorzystujących MPI I/O.

Do celów pracy należy:

- zaproponowanie autorskiego rozwiązania zgodnego z interfejsem MPI I/O w zakresie rozproszonej pamięci podręcznej opartej o bajtowo adresowaną pamięć nieulotną o dostępie swobodnym,
- przykładowa implementacja wyżej opisanego rozwiązania,
- opracowanie aplikacji demonstracyjnych wraz z dyskusją odnośnie do możliwych zastosowań i ograniczeń,
- przeprowadzenie serii eksperymentów, a także określenie warunków, w których proponowane rozwiązanie daje zyski wydajnościowe w stosunku do konfiguracji tradycyjnej.

W tezie rozprawy sprecyzowano interfejs dostępu do pliku oraz, pośrednio, standard zgodnie z którym musi być stworzona aplikacja, czyli MPI (*Message Passing Interface*). Chociaż badania pokazują, że w klastrach HPC do obsługi plików wykorzystywane jest przeważnie API (*Application Programming Interface*) systemu operacyjnego (w niektórych klastrach jest to 95% aplikacji) [67], to pozostałe aplikacje stosują MPI I/O lub rozwiązania oparte o MPI I/O, co pozwala osiągać znacznie większą wydajność pracy z plikiem [67]. Jeśli twórcom aplikacji zależy na zwiększeniu szybkości pracy na plikach, zastosowanie oprogramowania pośredniczącego staje się koniecznością. Dodatkowo mechanizmy zaimplementowane w systemie operacyjnym są mechanizmami ogólnymi, a nie dedykowanymi aplikacjom wysokiej wydajności, ich optymalizacja wiązałaby się więc najpierw z pracochłonnym przeniesieniem optymalizacji obecnych w oprogramowaniu pośredniczącym. Co więcej, opracowanie rozwiązania w dodatkowej warstwie zgodnej z MPI pozwala nie tylko na pełne uniezależnienie się od konkretnego systemu plików, ale również na wykorzystanie istniejących optymalizacji obecnych w implementacjach MPI I/O jednocześnie z zaproponowanym rozwiązaniem. Wobec powyższych argumentów, uszczegółowienie tezy do

MOST WIEDZY Downloaded from mostwiedzy.pl

aplikacji wykorzystujących interfejs MPI I/O nie powinno być uznawane za ograniczenie zakresu rozprawy.

Chociaż w wielu publikacjach związanych z wydajnością operacji na plikach podaje się jedynie przepustowość systemu zmierzoną testami syntetycznymi, w tezie niniejszej rozprawy przez wydajność rozumiane są nie tylko wyniki otrzymane przy użyciu benchmarków, ale głównie badana jest przez mierzenie sumarycznego czasu działania aplikacji. Uzasadnieniem takiej decyzji jest duży wpływ schematu dostępu do pliku na ogólną wydajność. Ponadto, zaprezentowane autorskie rozwiązanie jest częścią aplikacji i korzysta z tych samych zasobów, z których korzysta aplikacja – jego wydajność jest więc silnie skorelowana ze sposobem działania samej aplikacji. Testowanie wydajności operacji na plikach przy wykorzystaniu aplikacji demonstrujących jest również obecne w cytowanej literaturze [23, 114].

W tezie pracy nie zawężono właściwości aplikacji, których wydajność należy zwiększyć, żeby ją udowodnić. Mają być to jednak aplikacje wykorzystujące MPI I/O, zasadnym więc wydaje się założyć, że będą to aplikacje, dla których dostęp do pliku jest powodem problemów wydajnościowych, albo przynajmniej zajmuje zauważalną część czasu działania aplikacji. Dodatkowo, mając na celu zwiększenie wartości prowadzonych badań, należy spróbować stworzyć rozwiązanie dedykowane jak najszerszej grupie aplikacji.

1.2 Struktura rozprawy

Poniżej opisano przyjętą strukturę rozprawy.

- W bieżącym rozdziale krótko przedstawiono tematykę rozprawy, zdefiniowano tezę i cele pracy oraz omówiono te elementy tezy, które mogłyby się wydawać niejasne.

- Rozdział 2 omawia typy pamięci wykorzystywane w obecnych komputerach, ideę pamięci uniwersalnej, nowe technologie, mogące w przyszłości pomóc w stworzeniu takiej pamięci, a także bajtowo adresowaną pamięć NVRAM, która może trafić na rynek już niebawem i jest tematem niniejszej rozprawy.

- Rozdział 3 krótko wprowadza do operacji na plikach w obliczeniach wysokiej wydajności i omawia szerzej stosowane w tym zakresie optymalizacje. Zawiera również omówienie najważniejszych cech standardu MPI.

MOST WIEDZY Downloaded from mostwiedzy.pl

- Rozdział 4 pokazuje bieżący stan nauki w zakresie zastosowania pamięci omówionej w rozdziale 2 do zwiększenia wydajności aplikacji klastrowych, w tym w szczególności do optymalizacji operacji na plikach.

- Rozdział 5 prezentuje autorskie rozwiązanie pamięci podręcznej opartej o pamięć NVRAM. Opis skupia się na zastosowanej architekturze, pracy w środowisku podatnym na awarie, monitorowaniu aplikacji i potencjalnych ograniczeniach rozwiązania. W rozdziale poruszone są również główne problemy implementacyjne.

- Rozdział 6 zawiera rezultaty eksperymentów przeprowadzonych na aplikacjach demonstrujących wraz z omówieniem wyników. Przetestowane aplikacje to dwa benchmarki, program przetwarzający duże obrazy, wyznaczanie potęgi grafu, przeszukiwanie dwuwymiarowej mapy i symulacja zachowania tłumu oparta o uproszczone podejście agentowe.

- Rozdział 7 podsumowuje część teoretyczną i zaprezentowane rozwiązanie wraz w wynikami testów w kontekście tezy i celów niniejszej rozprawy.

- Rozdział 8 wyznacza dalsze możliwe kierunki prac.

1.3 Publikacje

Treści zawarte w rozprawie pokrywają się częściowo z tematyką następujących publikacji tworzonych lub współtworzonych przez autora rozprawy:

1. *A Parallel MPI I/O Solution Supported by Byte-addressable Non-volatile RAM Distributed Cache* [77] – nakreślenie problemu i pierwsza wersja rozwiązania prezentowanego w rozprawie;

2. *A Fail-Safe NVRAM Based Mechanism for Efficient Creation and Recovery of Data Copies in Parallel MPI Applications* [78] – omówienie możliwości wykorzystania nieulotności zastosowanej pamięci do zwiększenia niezawodności aplikacji i pierwszy projekt modułu mającego na celu dbanie o bezpieczeństwo danych;

3. *Three levels of fail-safe mode in MPI I/O NVRAM distributed cache* [72] – dalszy rozwój modułu odpowiedzialnego za ochronę danych przed utratą w przypadku awarii;

MOST WIEDZY Downloaded from mostwiedzy.pl

4. *Distributed NVRAM Cache – Optimization and Evaluation with Power of Adjacency Matrix* [74] – wdrożenie optymalizacji rozwiązania i weryfikacja wydajności przy użyciu aplikacji wyznaczającej n-tą potęgę grafu;

5. *A Solution to Image Processing with Parallel MPI I/O and Distributed NVRAM Cache* [75] – zaprezentowanie dalszych optymalizacji i kolejnej aplikacji demonstrującej możliwości rozwiązania – aplikacji pozwalającej przetwarzać obrazy o dużych rozmiarach;

6. *Multi-agent large-scale parallel crowd simulation* [76] – przedstawienie bardziej skomplikowanego przykładu zastosowania zaproponowanego rozwiązania, czyli agentowej symulacji zachowania tłumu;

7. *Multi-agent large-scale parallel crowd simulation with NVRAM-based distributed cache* [71] – rozszerzenie aplikacji symulującej zachowanie tłumu, kolejne optymalizacje rozwiązania oraz położenie większego nacisku na zabezpieczenie danych przed utratą w przypadku awarii;

8. *NVRAM as Main Storage of Parallel File System* [69] – weryfikacja potencjalnego zysku wydajnościowego z wdrożenia naiwnego podejścia do zastosowania pamięci NVRAM w rozproszonym systemie plików;

9. *Checkpointing of Parallel MPI Applications Using MPI One-sided API with Support for Byte-addressable Non-volatile RAM* [20] – poruszenie tematu innego interfejsu wchodzącego w skład MPI, One-sided API, również w kontekście zabezpieczenia danych przed utratą przy wykorzystaniu pamięci NVRAM;

10. *Using Redis supported by NVRAM in HPC applications* [70] – nieco inne spojrzenie na zastosowanie NVRAM w HPC, bazujące na bazie danych typu klucz-wartość;

11. *Recommendations for Writing Parallel Libraries with C and MPI* [73] – bardziej techniczne omówienie problemów i potencjalnych rozwiązań implementacji biblioteki zgodnej ze standardem MPI.

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 2

Adresowana bajtowo pamięć nieulotna o dostępie swobodnym

2.1 Pamięć w komputerach obecnie

Koncepcja nowoczesnego komputera, opisana przez Alana Turinga już w 1936 roku [107], zawierała dwa podstawowe elementy: element przetwarzający i pewną formę pamięci. Nie zmieniło się to do dnia dzisiejszego. Elementem przetwarzającym jest zazwyczaj procesor, a w zależności od przeznaczenia stosowane są obecnie różne układy pamięci. Szeroki wachlarz technologii i odmienne zastosowania doprowadziły do stworzenia całej gamy typów pamięci, pojawiła się więc konieczność jej usystematyzowania i klasyfikacji. Poniższa lista prezentuje wybrane cechy pamięci, które pomagają ją scharakteryzować, wraz z przykładami [100]:

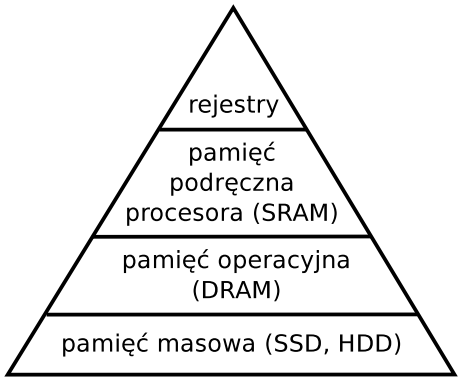
- miejsce w konstrukcji komputera – pamięć wewnętrzna (przykładowo: pamięć podręczna procesora, rejestry, pamięć operacyjna) i zewnętrzna (przykładowo: pamięć HDD i SSD),
- sposób dostępu – tracąca na znaczeniu pamięć o dostępie szeregowym (przykładowo: pamięć taśmowa) i pamięć o dostępie swobodnym (przykładowo: pamięć SSD),
- ulotność – pamięć ulotna (przykładowo: pamięć podręczna procesora), pamięć nieulotna (przykładowo: pamięć HDD i SSD), pozwalająca utrzymać dane w urządzeniu po odłączeniu zasilania,
- adresowanie – pamięć adresowana blokowo (przykładowo: pamięć HDD i SSD), pamięć adresowana bajtowo (przykładowo: pamięć operacyjna),
- pojemność – stosowane są pamięci o rozmiarach różniących się rzędami wielkości, np.

MOST WIEDZY Downloaded from mostwiedzy.pl

kilobajty pamięci podręcznej procesora, gigabajty pamięci operacyjnej i terabajtowe pamięci masowe,

- wydajność – na wydajność pamięci składają się takie parametry, jak opóźnienie w dostępie do danych czy przepustowość.

Zaprezentowana lista nie wyczerpuje właściwości charakteryzujących różne rodzaje pamięci, ma jedynie na celu podkreślenie szerokiej rozpiętości parametrów stosowanych typów pamięci.



Rysunek 2.1: Uproszczony schemat hierarchii pamięci

W praktyce w obecnych komputerach instalowana pamięć tworzy tak zwaną hierarchię, w której z każdym kolejnym poziomem rośnie pojemność pamięci, spada natomiast jej wydajność. Najbliżej procesora zlokalizowane są rejestry i pamięć podręczna. Obecnie sama pamięć podręczna jest również podzielona na poziomy (L1, L2, L3), a jej pojemność to maksymalnie kilkadziesiąt megabajtów (niektóre procesory Intel Xeon charakteryzuje pamięć podręczna o wielkości do 60 MB[1]). Kolejna w hierarchii jest pamięć operacyjna z typowymi układami o rozmiarze rzędu kilku gigabajtów, chociaż w profesjonalnych zastosowaniach pojawiają się urządzenia z pamięcią liczoną w dziesiątkach gigabajtów (przykładowo, polski producent GOODRAM oferuje układy dedykowane platformom serwerowym o maksymalnej pojemności 64 GB[2]). W ramach jednego węzła często instalowane jest kilka układów pamięci operacyjnej, co zwiększa nie tylko ogólną pojemność, ale i przepustowość. Następny poziom zajmuje pamięć masowa, czyli pamięć, która umożliwia przechowywanie dużych ilości danych przez długi czas. Jest to również pierwszy poziom, w którym po-

[1] https://www.intel.pl/content/www/pl/pl/products/processors/xeon/e7-processors/e7-8893-v4.html

[2] http://www.goodram.com/pl/product/pamieci-dedykowane/moduly-serwerowe/

MOST WIEDZY Downloaded from mostwiedzy.pl

wszechnie stosowana jest pamięć trwała. W przypadku pamięci masowej opartej o SSD, pojemność pojedynczego urządzenia to najczęściej kilka terabajtów, chociaż najbardziej pojemne produkty na rynku cechuje pojemność rzędu kilkunastu terabajtów (na przykład urządzenie oferowane przez firmę Samsung o pojemności 16 TB[3]). Hierarchię pamięci prezentuje się zwykle w postaci piramidy, tak jak to zilustrowano na rysunku 2.1.

Różne właściwości pamięci na każdym poziomie hierarchii wynikają z zastosowania różnorodnych technologii. Pamięć podręczna procesora oparta jest o statyczną pamięć o dostępie swobodnym, czyli SRAM (*Static Random Access Memory*). Główną zaletą SRAM jest wysoka szybkość odczytu i zapisu (opóźnienia rzędu kilku nanosekund), a także brak konieczności cyklicznego odświeżania. Do jej wad zalicza się wysoki koszt w przeliczeniu na pojemność. Do realizacji pamięci operacyjnej wykorzystuje się układy dynamicznej pamięci o dostępie swobodnym, czyli DRAM (*Dynamic Random Access Memory*). Układy DRAM są tańsze od SRAM ze względu na możliwą do uzyskania gęstość zapisu, wymagają natomiast cyklicznego odświeżania i są wolniejsze od SRAM (dostęp do pamięci DRAM wiąże się z opóźnieniem rzędu około 50 - 100 nanosekund). Obecnie najlepszą wydajność pamięci masowej uzyskuje się, stosując dyski półprzewodnikowe, tzw. SSD, oparte o układy typu flash. Pamięć stosowana w SSD jest nieulotną pamięcią o dostępie swobodnym, jest również stosunkowo tania, natomiast jej szybkość jest mniejsza niż szybkość układów DRAM.

Pamięć podręczna procesora w popularnych architekturach nie jest bezpośrednio dostępna z perspektywy programisty. Twórcy oprogramowania bazują więc głównie na pamięci operacyjnej, jeśli wymagana jest szybka pamięć, albo korzystają z pamięci masowej, kiedy dane trzeba trwale zachować. Zdarzają się również sytuacje, w których pamięci operacyjnej jest zbyt mało – w takim przypadku do przeprowadzania obliczeń wykorzystywana jest również pamięć masowa. Przykładem ręcznego zarządzania dwoma typami pamięci może być program GraphChi, który w zoptymalizowany sposób potrafi przetwarzać grafy o rozmiarze większym niż pamięć operacyjna [56]. Alternatywnym podejściem są techniki zawarte w systemie operacyjnym, które pozwalają programiście użyć większej ilości pamięci operacyjnej niż fizycznie zainstalowano – w takim przypadku system operacyjny może przenieść rzadziej używane fragmenty pamięci operacyjnej do pamięci masowej. W celu utrzymania w takiej sytuacji jak najwyższej wydajności stosuje się różne algorytmy optymalizujące [55]. Pojawiają się również biblioteki ogólnego przeznaczenia,

[3] `http://www.samsung.com/us/dell/pdfs/PM1633a_Flyer_2016_v4.pdf`

MOST WIEDZY Downloaded from mostwiedzy.pl

które pozwalają wydajnie korzystać z pamięci masowej opartej o SSD, jakby była pamięcią operacyjną, takie jak biblioteka SSDAlloc [4]. Z drugiej strony, dostęp do pliku zlokalizowanego na pamięci masowej jest przyspieszany przy użyciu pamięci operacyjnej – w popularnych systemach operacyjnych sam fakt wywołania funkcji zapisu nie oznacza więc, że dane trafiły do pamięci. Aby się upewnić, konieczna jest synchronizacja przy użyciu funkcji synchronizujących.

Do powyższych zagadnień można dodać jeszcze pamięć, będącą częścią akceleratorów sprzętowych, która również często dzieli się na różne poziomy o różnych właściwościach. Jako przykład można podać urządzenia takie jak Intel Xeon Phi, w których do dyspozycji jest mniej wydajna pamięć trwała, szybka pamięć operacyjna i pamięć cache procesora. W przypadku kart graficznych hierarchia pamięci jest jeszcze bardziej skomplikowana – dla architektury Nvidia CUDA do dyspozycji programisty są rejestry oraz pamięć o typie local, shared, constant i global, różniące się nie tylko parametrami, ale także poziomem dostępu. Dodatkowo, korzystając z akceleratorów, konieczne może się okazać kopiowanie danych pomiędzy pamięcią komputera, a pamięcią akceleratora. Obecnie istnieją mechanizmy, które ułatwiają programiście zarządzanie tak skomplikowaną hierarchią, wykonujące część typowych zadań w sposób zautomatyzowany, takie jak Unified Memory dostępne na urządzeniach kompatybilnych z Nvidia CUDA. Przy ich użyciu trzeba się natomiast liczyć ze spadkiem wydajności w porównaniu do ręcznej implementacji zoptymalizowanej pod kątem konkretnego programu [42].

Powyższe przykłady jednoznacznie pokazują, że tworzenie wydajnych aplikacji przy wykorzystaniu wielopoziomowej hierarchii pamięci jest zagadnieniem trudnym. Potencjalne uproszczenie hierarchii mogłoby ten proces znacznie ułatwić.

2.2 Pamięć uniwersalna

Naukowcy w wielu oddziałach badawczych, jak i pracownicy związani z przemysłem elektronicznym, szukają układów pamięci, które połączyłyby zalety technologii wykorzystywanych w pamięciach masowych i pamięci operacyjnej. Taką pamięć określa się terminem pamięci uniwersalnej. W najbardziej optymistycznym scenariuszu nowe urządzenia łączyłyby zalety SRAM, DRAM i SSD: bajtowe adresowanie, nieulotność, wysoką szybkość i dużą pojemność przy stosunkowo niskiej cenie. Dodatkowo, taka pamięć powinna mieć nieograniczoną żywotność, rozumianą jako brak ograniczenia na liczbę cykli zapisu. Ko-

MOST WIEDZY Downloaded from mostwiedzy.pl

rzyści z wprowadzenia takich układów to z jednej strony większa wydajność komputerów, z drugiej prostszy model tworzenia oprogramowania. Trudno spodziewać się, że któryś z producentów wprowadzi nagle rewolucyjne urządzenie, spełniające wszystkie wyżej opisane kryteria, bardziej prawdopodobne jest raczej podejście iteracyjne. Niniejszy podrozdział ma na celu pokazanie, że w najbliższej przyszłości należy oczekiwać urządzeń, które będą o krok bliżej do pamięci uniwersalnej. Ze względu na fakt, że rozprawa dotyczy tematu pamięci traktowanej z perspektywy twórców oprogramowania jako urządzenie o pewnej charakterystyce, zrezygnowano ze szczegółowego opisu sposobu działania poszczególnych technologii.

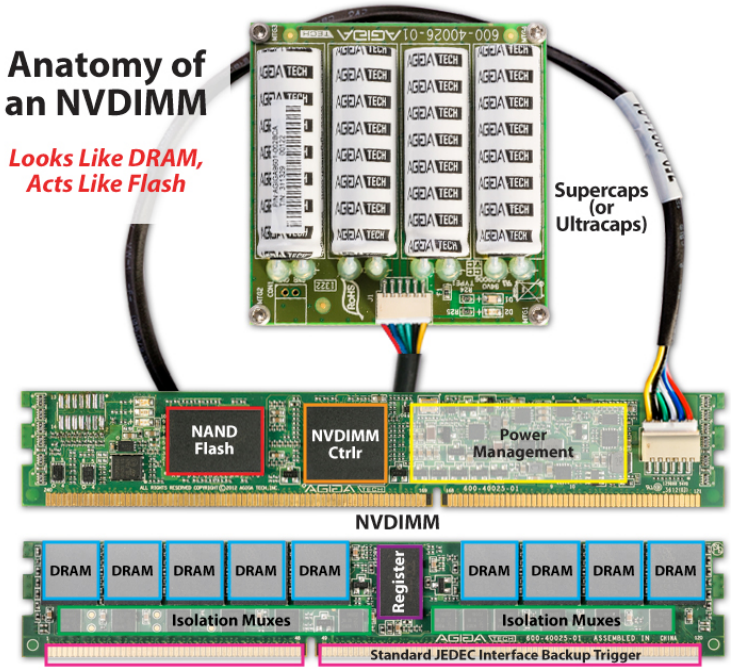
Temat pamięci uniwersalnej zaczął być głośny w 2005 roku, kiedy to obiecujące technologie wskazały czasopisma „Science" [2] i „Nature" [117]. W „Science" opisano dobre rezultaty prac nad pamięcią typu MRAM (*Magnetoresistive RAM*), która obecnie powinna być już na etapie prototypów w firmach takich jak Everspin [19] czy IBM [86]. Według materiałów dostępnych na stronie Everspin opóźnienie pamięci zarówno w przypadku odczytu, jak i zapisu, to jedyne 35 ns[4], czyli wartość o około trzy rzędy wielkości niższa od popularnych SSD. Artykuł umieszczony na łamach „Nature" promował tzw. *Ovonic memory*, teraz bardziej znaną w literaturze jako PCM (*Phase-Change Memory*) albo PCRAM (*Phase-Change RAM*). Chociaż nadal jest technologią obiecującą, obecnie wskazuje się na kilka problemów z nią związanych, które musiałyby być wyeliminowane, żeby pamięć została spopularyzowana [25]. Lista pozostałych technologii, które w przyszłości mogą pretendować do miana pamięci uniwersalnej, jest bardzo szeroka i obejmuje co najmniej kilkanaście różnych pozycji, takich jak FeRAM (*Ferroelectric RAM*), STT-RAM (*Spin-transfer torque*), ReRAM (*Resistive RAM*), DWM (*Domain-Wall Memory*) czy pamięci polimerowe [79].

Jednocześnie z rozwojem różnych koncepcji układów pamięci uniwersalnej wiele wysiłku wkłada się w specyfikację standardów, które miałyby obejmować taką pamięć. Organizacja JEDEC (*Joint Electron Device Engineering Council*), znana głównie z publikacji standardów dla pamięci takich jak DRAM, przygotowała specyfikację rodziny modułów NVDIMM (*Non-Volatile Dual In-line Memory Module*) [43]. Rodzina zawiera trzy standardowe typy układów:

- NVDIMM-F – układ wymagający sparowania modułu pamięci DRAM z modułem pamięci nieulotnej,

[4] `https://www.everspin.com/mram-technology-attributes`

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 2.2: Materiały marketingowe firmy AgigA Tech, ilustrujące nie tylko architekturę układów typu NVDIMM-N, ale również pokazujące konieczność zapewnienia modułom podtrzymywania zasilania na wypadek awarii *(źródło: agigatech.wpengine.com)*

- NVDIMM-N – pojedynczy moduł zawierający DRAM i półprzewodnikową pamięć trwałą,
- NVDIMM-P – moduł oparty o nowe rodzaje pamięci, w którym pamięć nieulotna może być bezpośrednią pamięcią operacyjną komputera.

W chwili pisania tej pracy na rynku można znaleźć moduły NVDIMM-N, na przykład produkowane przez Hewlett Packard Enterprise w układach o pojemności 8 GB i 16 GB[5]. Zasada działania takich układów jest stosunkowo prosta – tradycyjne operacje przeprowadzane są przy użyciu pamięci DRAM, a przy wystąpieniu ewentualnej awarii układ automatycznie kopiuje zawartość pamięci DRAM do pamięci półprzewodnikowej. Technologia radzi sobie również z brakiem zasilania, wykorzystując superkondensatory – albo wchodzące w skład płyty głównej, albo zorganizowane w formie dodatkowych modułów, takich jak zaprezentowany na rysunku 2.2, oferowanych na przykład przez firmę AgigA Tech[6]. Wadą obecnie produkowanych NVDIMM-N jest ich wysoka cena podyktowana między

[5] `https://www.hpe.com/us/en/servers/persistent-memory.html`
[6] `http://agigatech.wpengine.com/products/agigaram-nvdimms/`

MOST WIEDZY Downloaded from mostwiedzy.pl

innymi koniecznością instalacji redundantnej pamięci w ramach pojedynczego urządzenia, a niewielka pojemność nie pozwala na stosowanie modułów również w zastępstwie pamięci masowej.

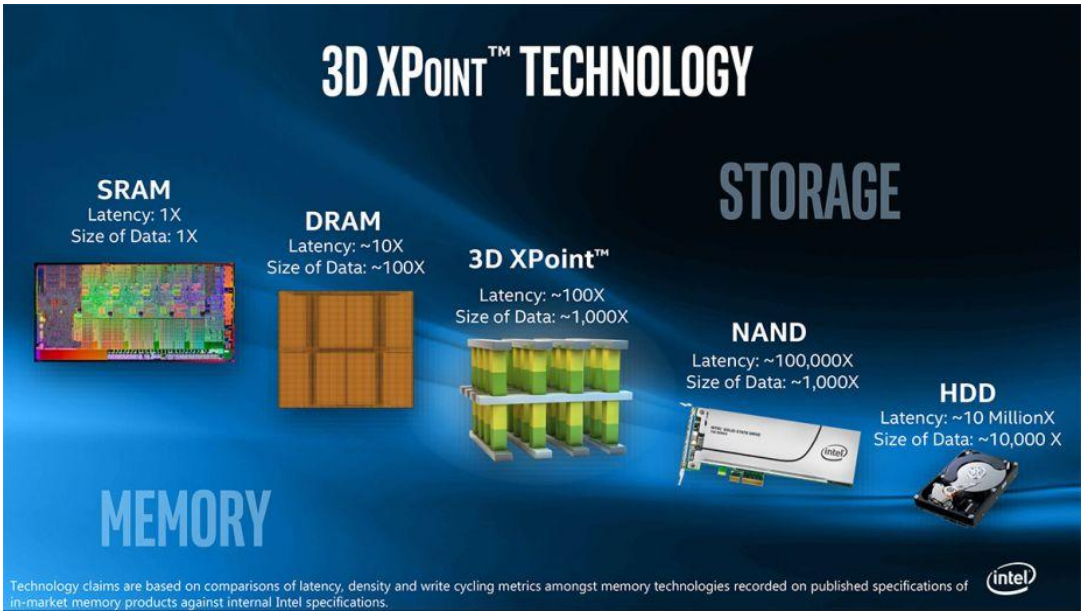
Wyraźnie widać, że ewentualnego dużego przełomu należy się spodziewać dopiero po pojawieniu się urządzeń zgodnych ze standardem NVDIMM-P. Dokładna specyfikacja NVDIMM-P nie jest jeszcze znana, mimo że oficjalne informacje mówiły o tym, że powinna pojawić się jeszcze w 2018 roku wraz ze specyfikacją układów DDR5 (*Double Data Rate*), czyli standardu dla pamięci operacyjnej [44]. W chwili pisania pracy brakuje kolejnych zapowiedzi ze strony JEDEC, a zdawkowe notatki na stronie organizacji zapewniają jedynie o trwających nadal pracach [45].

SNIA (*Storage Networking Industry Association*), inna organizacja zajmująca się standaryzacją, utworzyła grupę Persistent Memory and NVDIMM Special Interest Group [102]. Celem grupy jest przyspieszenie zaadaptowania modułów NVDIMM w rzeczywistych architekturach, między innymi przez koordynowanie prac nad standardami, promocję modelu programowania opartego o pamięci nieulotne, przygotowanie dokumentacji dla użytkowników czy specyfikację najlepszych praktyk. Do grupy należą między innymi Samsung, Intel, Micron i Western Digital. Powyższe inicjatywy sugerują, że badania nad nowymi układami pamięci są w zaawansowanej fazie.

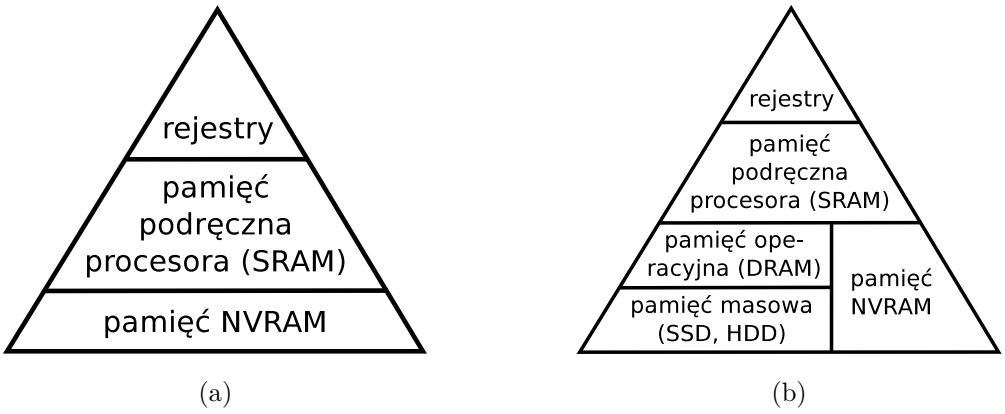
Obecnie duże nadzieje związane są z doniesieniami z firm Micron i Intel, które opublikowały materiały na temat technologii 3D XPoint [39]. Nowa pamięć, użyta w modułach NVDIMM, ma być trwała, zdecydowanie bardziej pojemna od układów opartych na DRAM (mówi się nawet o terabajtach pamięci instalowanej w pojedynczym węźle) i na tyle szybka, żeby umożliwić bezpośrednią współpracę z procesorem [93]. Od czasu ogłoszenia prac nad technologią cyklicznie pojawiają się coraz bardziej szczegółowe materiały, głównie w bazie IEEE [7, 10, 26, 89] i ACM [28]. Wielokrotnie prezentowane były również materiały marketingowe, takie jak te na rysunku 2.3, pokazującym spodziewane osiągi nowych układów. Niestety, na pierwsze urządzenia na rynku będzie trzeba jeszcze poczekać, nie ma też wiarygodnych informacji odnośnie do dat wypuszczenia produktu.

Kiedy pamięć uniwersalna powstanie, docelową konfiguracją będzie uproszczenie hierarchii pamięci w komputerach, tak jak to zilustrowano na rysunku 2.4a. W najprostszym wariancie można spodziewać się maszyn wyposażonych w układy typu NVDIMM-P zamiast osobnej pamięci DRAM i pamięci masowej. Zanim do tego dojdzie, czeka nas jednak prawdopodobnie okres przejściowy, w którym na rynku pojawią się urządzenia wyposażo-

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 2.3: Materiały marketingowe firmy Intel, pokazujące potencjalną pozycję układów NVRAM (tutaj opatrzonych nazwą technologii 3D XPoint) w hierarchii pamięci *(źródło: intel.com)*



Rysunek 2.4: Potencjalne możliwości kształtu hierarchii pamięci uzupełnionej o NVRAM

ne zarówno w układy DRAM i SSD, jak i w pamięć nieulotną o wydajności niższej niż DRAM i nie tak pojemną jak pamięć masowa. W niniejszej rozprawie rozpatrywany jest właśnie taki scenariusz, pokazany również na rysunku 2.4b.

Jedną z istotnych kwestii jest również umiejscowienie pamięci NVRAM w klastrze obliczeniowym. Największe możliwości oferuje konfiguracja, w której nowa pamięć instalowana jest we wszystkich węzłach klastra i taka konfiguracja jest jednym z założeń rozprawy. Z drugiej strony, może się okazać, że wysoka cena układów spowoduje konieczność

MOST WIEDZY Downloaded from mostwiedzy.pl

ograniczenia jej liczby i instalowania jedynie w wybranych maszynach, tworząc środowisko heterogeniczne. Podobna sytuacja miała miejsce w przypadku, kiedy pamięć masowa oparta o SSD zaczęła konkurować z dyskami HDD – w wielu rozwiązaniach była stosowana najpierw jedynie w wybranych węzłach serwerów systemu plików [32, 33, 34]. Należy jednak pamiętać, że zgodnie z wymienionymi standardami, spodziewane są układy oparte o moduły DIMM, których głównym zadaniem jest wspieranie obliczeń, a nie zastąpienie pamięci masowej. W obecnych klastrach obliczeniowych nadal dominują konfiguracje stosunkowo homogeniczne. Argumentem przemawiającym za instalacją nowych układów pamięci we wszystkich węzłach klastra mimo wysokiej ceny jest ich potencjalnie mniejsze zużycie energii w porównaniu do pamięci operacyjnej opartej o DRAM [57, 127].

Przeglądając materiały związane z rozwojem technik tworzenia i wykorzystania nowych rodzajów pamięci, dość szybko można zaobserwować problem ich nazewnictwa. Angielski termin *universal memory* jest przeważnie zarezerwowany dla pamięci idealnej, która łączyłaby zalety SRAM, DRAM i SSD, bez wprowadzania nowych wad. Istnieje również szeroka grupa typów pamięci określanych skrótem NVM (*Non-Volatile Memory*) lub NVRAM. W tej grupie powinny się znaleźć wszystkie pamięci nieulotne o swobodnym dostępie, niestety oba skróty nie określają sposobu adresowania. Co więcej, w praktyce termin NVM jest przeważnie kojarzony z urządzeniami SSD, podłączanymi do płyty głównej przy wykorzystaniu magistrali PCI Express i złącz SATA Express albo M.2. Źródłem tej nazwy jest prawdopodobnie specyfikacja interfejsu NVMe (*NVM Express*), który zastąpił starszy AHCI (*Advanced Host Controller Interface*), będąc od podstaw zaprojektowanym w celu optymalnego wykorzystania niskich opóźnień i równoległego sposobu dostępu do pamięci w ramach architektury nowoczesnych urządzeń SSD. W literaturze pojawia się też pojęcie NVMM (*Non-Volatile Main Memory*), termin sugeruje jednak całkowite zastąpienie nową pamięcią układów DRAM. Często używanym skrótem jest również NVDIMM (*Non-Volatile Dual In-line Memory Module*), natomiast nakłada on już ograniczenia związane z konkretną specyfikacją architektury urządzeń. Ostatnim znanym stosowanym terminem jest *Persistent Memory*, niestety mimo szerokiego użycia nie wskazuje on w żaden sposób na swobodny dostęp czy bajtowe adresowanie. W niniejszej pracy wykorzystywany będzie skrót NVRAM, należy jednak pamiętać, że za każdym razem powinien być on rozumiany jako bajtowo adresowana pamięć nieulotna o dostępie swobodnym, o wydajności wyższej niż wydajność urządzeń SSD i niższej niż wydajność pamięci DRAM, zainstalowana obok pamięci operacyjnej opartej o DRAM.

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 3

Obliczenia wysokiej wydajności i operacje na plikach

3.1 Wprowadzenie do I/O w HPC

Od lat obserwujemy szybki wzrost mocy obliczeniowej komputerów, który jest bezpośrednim następstwem zwiększania liczby tranzystorów w układach scalonych zgodnie z prawem Moore'a [94]. Z perspektywy algorytmów o wysokiej złożoności obliczeniowej, taki wzrost pozwala znacznie skrócić czas obliczeń, albo w tym samym czasie przetworzyć więcej danych. Sprawa wygląda jednak inaczej, jeśli spojrzymy na aplikacje przetwarzające dane wielkoskalowe (ang. *data-intensive computing*). Zdarza się, że takie programy pracują z ograniczoną wydajnością, ponieważ jednostki obliczeniowe przetwarzają otrzymane dane szybciej, niż nowe dane są im dostarczane. W optymalnie skonfigurowanym systemie szybkość wykonywania obliczeń powinna być zrównoważona szybkością dostarczania danych, na których te obliczenia mają być prowadzone.

Żeby ustrzec się problemu w ramach pojedynczego węzła, stosowane są między innymi coraz szybsze układy pamięci, większe pamięci podręczne czy konfiguracje, pozwalające uzyskać wyższą wydajność w związku ze zwielokrotnieniem układów pamięci. Techniki te nie zależą od poziomu, na których zlokalizowane są dane. Dla pamięci operacyjnej wyższą wydajność układu można uzyskać wybierając nowe układy o większej przepustowości [91], stosując większą pamięć podręczną procesora, co zredukuje liczbę operacji kierowanych do pamięci operacyjnej [83], albo używając kilku układów w konfiguracji wielokanałowej (ang. *multi-channel*) [83]. Dla pamięci masowej analogicznie: ostatnimi czasy duży skok wydajności można zaobserwować po zmianie nośnika pamięci masowej z HDD na SSD [48], szybszy dostęp do pliku ułatwia pamięć podręczna, którą system plików tworzy w pamięci operacyjnej [53], a przykładową konfiguracją poprawiającą wydajność przez użycie wielu

MOST WIEDZY Downloaded from mostwiedzy.pl

dysków jest macierz RAID (*Redundant Array of Independent Disks*) [22]. Do rozwiązań bazujących na sprzęcie dochodzi jeszcze szereg optymalizacji bazujących na oprogramowaniu.

W obliczeniach wysokiej wydajności, oprócz danych zlokalizowanych w pojedynczym węźle, przetwarzane są dane współdzielone przez wszystkie węzły w klastrze. Takie dane są często zorganizowane w formie zwykłych plików obsługiwanych przez system plików. Operacje na plikach zazwyczaj nazywane są w systemach wysokiej wydajności terminem I/O (ang. *input/output*), mimo że ogólnie termin ten ma duże szersze znaczenie. W HPC wykorzystywane są głównie równoległe systemy plików, nazywane również PFS, które pozwalają na jednoczesny dostęp do pliku realizowany przez wiele procesów na wielu węzłach. PFS, dzięki swojej równoległości, powinien być skalowalny w taki sposób, żeby zwiększając liczbę serwerów obsługujących system plików, mógł sprostać rosnącym wymaganiom ze strony klastrów obliczeniowych. Analogicznie więc do optymalizacji na innych poziomach (pojedynczy procesor, pojedynczy węzeł), mamy do czynienia ze zwielokrotnieniem podstawowych jednostek składowych dostarczających pamięci do systemu. Oczywistym jest również to, że wpływ na szybkość przetwarzania danych ma także wydajność pojedynczych serwerów systemu plików – zarówno jeśli chodzi o zdolności obliczeniowe (procesor, pamięć DRAM), parametry pamięci masowej (HDD, SSD, zastosowanie macierzy), jak i sieci łączącej poszczególne węzły serwerowe oraz serwery systemu plików z węzłami obliczeniowymi (Ethernet, Infiniband).

Tak samo analogicznie do pamięci podręcznej procesorów, przyspieszającej dostęp do danych zlokalizowanych w pamięci operacyjnej, i pamięci podręcznej systemu operacyjnego, przyspieszającej dostęp do plików zlokalizowanych w pamięci masowej, dostęp do plików zlokalizowanych na serwerach zewnętrznych może być przyspieszony przy użyciu zasobów lokalnych konkretnego węzła. Takie optymalizacje mogą być zarówno częścią systemu plików, jak i być od niego niezależne i wchodzić w skład danej aplikacji. Żeby współpraca aplikacji i systemu plików była jak najbardziej wydajna, optymalizacje wdrożone w aplikacji powinny skutkować zorganizowaniem komunikacji pomiędzy aplikacją a serwerem PFS zgodnie z listą najlepszych praktyk [18, 113], do których włącza się między innymi:

- unikanie żądań małych fragmentów plików,
- unikanie żądań nieciągłych,

MOST WIEDZY Downloaded from mostwiedzy.pl

- unikanie dostępu do losowych lokalizacji,
- agregowanie mniejszych żądań w większe.

Stosowanie wszystkich praktyk może się wiązać z dużym narzutem programistycznym, powstało więc oprogramowanie pośredniczące, które ułatwia wydajną komunikację z systemem plików.

Najczęściej stosowaną technologią, działającą w warstwie pośredniczącej, jest MPI I/O, czyli podzbiór funkcji standardu MPI, stworzony w celu wydajnej obsługi plików [6, 18, 67]. Samo MPI jest standardem definiującym interfejs programistyczny oraz protokół komunikacyjny, ułatwiający tworzenie aplikacji równoległych w środowisku klastrowym z pamięcią rozproszoną. Celem MPI jest nie tylko wydajność i skalowalność, ale również przenośność – kod stworzony zgodnie ze standardem może być skompilowany i uruchomiony na wielu różnych platformach. Chociaż dokumentacja skupia się na wykorzystaniu API przy użyciu języka C, C++ i Fortran, jest ono również z powodzeniem stosowane w programowaniu na innych, uznanych za bardziej nowoczesne platformach, takich jak Python[1] czy Java[2]. Funkcjonalnie, MPI pozwala między innymi na:

- komunikację pomiędzy procesami przy użyciu przesyłanych wiadomości, zarówno w konfiguracji jeden nadawca – jeden odbiorca, jak i na wiadomości zbiorowe,
- synchronizację procesów, na przykład przy użyciu bariery,
- definiowanie własnych, również złożonych typów danych,
- komunikację jednostronną, w której proces ma dostęp do zdalnej pamięci zlokalizowanej w innym węźle,
- stosowanie wielu wątków w ramach procesów i dynamiczne zarządzanie procesami,
- dostęp do plików w ramach wszystkich procesów rozproszonej aplikacji.

Jako że MPI jest jedynie standardem, żeby wykorzystać opisane możliwości w praktyce, należy posłużyć się jedną z jego implementacji. Obecnie dostępnych jest wiele implementacji w pełni realizujących założenia standardu, do najpopularniejszych należą otwartoźródło-

[1] `https://mpi4py.readthedocs.io/`
[2] `http://mpj-express.org/`

MOST WIEDZY Downloaded from mostwiedzy.pl

we projekty, takie jak MPICH[3], MVAPICH[4], Open MPI[5], czy rozwiązania zamknięte, przykładowo Intel MPI Library[6]. Poszczególne implementacje różnią się między innymi wewnętrzną architekturą, co może wpływać na wydajność niektórych operacji, a także stosowanymi technikami optymalizacyjnymi oraz różnym poziomem wsparcia mniej popularnych rozwiązań sprzętowych.

W kontekście operacji na plikach, twórcy MPI zauważyli, że realizacja niektórych optymalizacji nie będzie możliwa przy wykorzystaniu standardowego interfejsu do operacji plikowych, oferowanego przez system operacyjny. Utworzono więc całą kolekcję funkcji, pozwalającą nie tylko na bezpośredni synchroniczny dostęp do wybranych fragmentów plików, ale zawierającą również między innymi [81]:

- operacje nieblokujące, pozwalające na asynchroniczną realizację dostępu,
- współdzielone wskaźniki do konkretnego obszaru w pliku,
- funkcje grupowe wywoływane z poziomu wszystkich procesów, które otwierały obsługiwany plik,
- mechanizm widoków odpowiedzialny za możliwość zdefiniowania podzbioru danych widzianych przez dany proces zgodnie z określonym schematem,
- opcjonalny tryb pracy „atomic”, pozwalający w łatwy sposób uniknąć konfliktów w dostępie kosztem wydajności.

W typowym scenariuszu pracy z MPI I/O, przetwarzanie pliku rozpoczyna się od jego otwarcia przez wszystkie procesy, które będą później miały do otwartego pliku dostęp. Podczas otwierania możliwe jest również przekazanie szeregu wskazówek, związanych między innymi ze wzorcem dalszej pracy z plikiem, co ma umożliwić odpowiednie dostosowanie zachowania implementacji MPI w celu maksymalizacji wydajności. Jedną z takich wskazówek, opisaną w specyfikacji standardu, jest parametr „access_style”, mogący przyjąć wartości takie jak na przykład „read_mostly” (procesy aplikacji będą głównie odczytywały dane z pliku), „sequential” (plik będzie czytany sekwencyjnie), czy „write_once” (aplikacja zapisze plik tylko raz). Ostateczna lista wspieranych wskazówek jest jednak definiowana przez implementację, a nie standard. Po otwarciu pliku opcjonalnie można

[3] `https://www.mpich.org/`
[4] `http://mvapich.cse.ohio-state.edu/`
[5] `https://www.open-mpi.org/`
[6] `https://software.intel.com/en-us/intel-mpi-library`

MOST WIEDZY Downloaded from mostwiedzy.pl

utworzyć widok, zawężający dane widziane przez konkretny proces. Dalej następuje zazwyczaj szereg operacji odczytu i zapisu. W standardowym trybie, dane zapisane przez jeden z procesów nie muszą być automatycznie widoczne z poziomu innych procesów – żeby to wymusić konieczne jest użycie funkcji synchronizującej, która dba również o to, żeby zawartość pliku została przeniesiona ze struktur pomocniczych do systemu plików. Podczas pracy, funkcje MPI mogą zwrócić jeden z szeregu zdefiniowanych błędów, które powinny być obsłużone przez twórcę aplikacji, jeśli nie mają dopuścić do niespójności danych lub nawet awarii programu. Przetwarzanie kończy się zamknięciem pliku, w którym również muszą wziąć udział wszystkie procesy aplikacji.

Standard MPI został zaprojektowany zgodnie z paradygmatem programowania proceduralnego, co oznacza, że interfejs jest przedstawiony w postaci zestawu funkcji. W przypadku operacji na plikach wyspecyfikowano funkcje odpowiedzialne za zarządzanie plikiem, dostęp do danych, konfigurację formatu w celu zapewnienia zgodności z innymi systemami, a także spójność operacji w środowisku rozproszonym. Kompletny interfejs jest bardzo przejrzyście i precyzyjnie opisany w dokumentacji MPI [81], w rozprawie zdecydowano się więc na pokazanie jedynie najczęściej używanych funkcji. Zarządzenie plikiem odbywa się przy użyciu:

- `MPI_File_open(comm, filename, amode, info, fh)` – otwieranie pliku przez zbiór procesów w ramach zdefiniowanego trybu;
- `MPI_File_close(fh)` – zamykanie przetwarzanego pliku;
- `MPI_File_delete(filename, info)` – usuwanie pliku;
- `MPI_File_set_size(fh, size)` – zmiana rozmiaru pliku;
- `MPI_File_preallocate(fh, size)` – alokacja pamięci, istotna w przypadku tworzenia nowego pliku lub zwiększania rozmiaru istniejącego;
- `MPI_File_get_size(fh, size)` – odczyt rozmiaru pliku;
- `MPI_File_set_info(fh, info)` – możliwość dodatkowego skonfigurowania parametrów pracy z plikiem, głównie w celu zwiększenia wydajności;

W kwestii dostęp do pliku zaproponowano aż 34 funkcje, które można podzielić względem trzech kryteriów:

- sposób adresacji w ramach pliku:

MOST WIEDZY Downloaded from mostwiedzy.pl

  - *explicit offsets* – wskazanie konkretnej lokalizacji interesujących nas danych w pliku,
  - *individual file pointers* – użycie osobnych wskaźników w ramach każdego z procesów,
  - *shared file pointer* – pojedynczy wskaźnik dla wszystkich procesów biorących udział w przetwarzaniu;

- tryb komunikacji:
  - *blocking* – blokujący (synchroniczny), w którym proces czeka na zakończenie operacji,
  - *nonblocking* – nieblokujący, dzięki któremu proces może kontynuować pracę, a operacje plikowe będą wykonywane w tle,
  - *split collective* – tryb, w którym daną operację podzielono na dwie funkcje – rozpoczynającą i kończącą;

- organizacja procesów podczas wykonywania operacji:
  - *noncollective* – tryb, w którym każdy z procesów może wywoływać funkcje niezależnie,
  - *collective* – przetwarzanie grupowe, w którym każdy proces pracujący z danym plikiem jest zobowiązany wywołać daną funkcję.

Lista funkcji, dla czytelności bez parametrów wywołania, została przeniesiona z dokumentacji MPI i zaprezentowana na rysunku 3.1. Wśród pozostałych istotnych i często wykorzystywanych funkcji warto wymienić dwie:

- `MPI_File_set_atomicity(fh, flag)` – pozwala włączyć lub wyłączyć tryb *atomic*, który kosztem wydajności potrafi zapewnić spójność pojedynczych operacji na plikach;

- `MPI_File_sync(fh)` – zapewnia przeniesienie wszystkich uprzednio zapisanych danych (na przykład z pamięci podręcznej) na fizyczne urządzenie.

Niekiedy, podczas przetwarzania plików w aplikacji opartej o MPI, wykorzystuje się również dodatkową warstwę pośredniczącą, nadbudowującą MPI I/O i określającą format obsługiwanych plików [18]. Przykładem takiej biblioteki może być HDF5 (*Hierarchi-*

MOST WIEDZY Downloaded from mostwiedzy.pl

positioning	synchronism	coordination	
		noncollective	*collective*
explicit offsets	*blocking*	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	nonblocking	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_IREAD_AT_ALL MPI_FILE_IWRITE_AT_ALL
	split collective	N/A	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
individual file pointers	*blocking*	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	nonblocking	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_IREAD_ALL MPI_FILE_IWRITE_ALL
	split collective	N/A	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
shared file pointer	*blocking*	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	nonblocking	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	N/A
	split collective	N/A	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

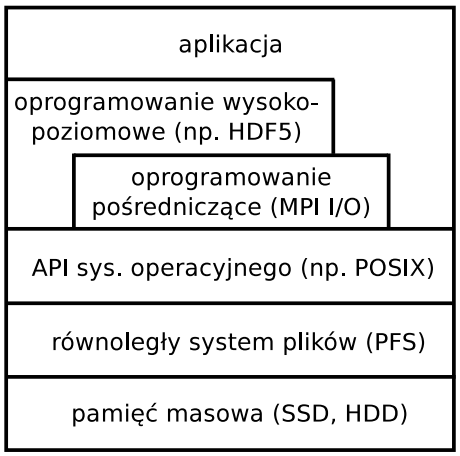
Rysunek 3.1: Podział funkcji MPI I/O odpowiedzialnych za dostęp do danych pliku *(źródło: MPI: A Message-Passing Interface Standard [81], prawa autorskie: University of Tennessee)*

cal Data Format)[7], który jest standardem, pozwalającym wykorzystać pojedynczy plik do przechowywania struktury hierarchicznej złożonej z grup, zbiorów danych w postaci macierzy o określonych wymiarach i atrybutów opisujących dwa poprzednie elementy. Innym przykładem takiego narzędzia jest PnetCDF (*Parallel NetCDF*)[8], będący równoległą wersją formatu plików NetCDF (*Network Common Data Form*)[9]. NetCDF jest chętnie wykorzystywany w aplikacjach meteorologicznych lub oceanograficznych i podobnie do HDF5 pozwala zorganizować dane w macierzach wzbogaconych o dodatkowe atrybuty na różnych poziomach. Zdaniem niektórych badaczy biblioteki nie wnoszą zbyt wiele w kontekście wydajności równoległego przetwarzania plików w HPC, ponieważ skupiają się tak naprawdę głównie na formatowaniu i organizacji danych [80] i z tego powodu nie będą szerzej omawiane w ramach rozprawy.

Podsumowując powyższe rozważania, w systemach wysokiej wydajności mamy do czynienia z wielopoziomową strukturą stworzoną do pracy z plikami, która nieco różni się od konfiguracji stosowanych w innych środowiskach. W literaturze wyróżnia się następujące

[7] `https://www.hdfgroup.org/solutions/hdf5/`
[8] `https://trac.mcs.anl.gov/projects/parallel-netcdf`
[9] `https://www.unidata.ucar.edu/software/netcdf/`

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 3.2: Schemat stosu technologicznego stosowanego w operacjach na plikach w obliczeniach wysokiej wydajności

warstwy oprogramowania HPC [66]:

1. rozproszony system plików zarządzający pamięcią masową,
2. interfejs, pozwalający na komunikację aplikacjom bądź bibliotekom narzędziowym z systemem plików,
3. opcjonalne oprogramowanie pośredniczące (ang. *middleware*), które z jednej strony może wystawiać interfejs stworzony z myślą o aplikacjach HPC, z drugiej strony zawierające szereg optymalizacji żądań do pliku; zazwyczaj stosowaną warstwą jest tutaj MPI I/O [66];
4. opcjonalna warstwa wysokopoziomowych narzędzi kierowanych niekiedy konkretnej klasie aplikacji,
5. aplikacja HPC pracująca na plikach wykorzystując API oferowane przez warstwę 2, 3 lub 4.

Powyższy schemat został również zilustrowany na rysunku 3.2.

W dalszej części rozdziału omówiono obecnie stosowane optymalizacje dostępu do plików w HPC, realizowane w warstwie rozproszonego systemu plików i oprogramowania pośredniczącego, a w szczególności MPI I/O, które jest tematem rozprawy. Zaprezentowane rozwiązania podzielono na dwie grupy – pierwsza zawiera optymalizacje niezależne od zastosowanej pamięci, a do drugiej zaliczono rozwiązania oparte o pamięć SSD, które mogą potencjalnie wskazać kierunek prac nad podejściem opartym o NVRAM. Dodatkowo,

MOST WIEDZY Downloaded from mostwiedzy.pl

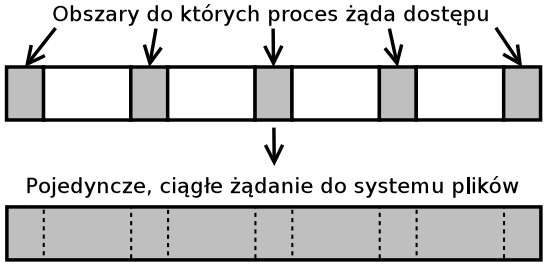
do osobnej sekcji wydzielono opis narzędzi pojawiających się pod terminem *burst buffer*, ponieważ z jednej strony są jedną z najbardziej popularnych i najczęściej stosowanych metod zwiększania wydajności I/O w ostatnich latach, a z drugiej mają wiele elementów wspólnych z zaproponowanym rozwiązaniem autorskim. Mechanizmy wysokopoziomowe i optymalizacje obecne w samych aplikacjach zostały pominięte ze względu na zawężone zastosowanie, natomiast nieliczne narzędzia wspierane pamięcią NVRAM zostały omówione w rozdziale następnym.

3.2 Optymalizacje niezależne od zastosowanej pamięci

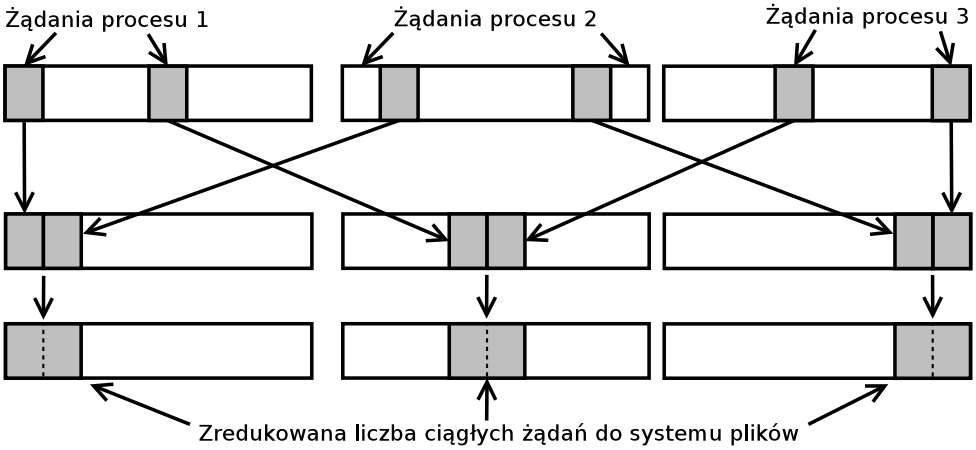
Wiele rozwiązań ma na celu poprawę wydajności operacji na plikach niezależnie od warstwy sprzętowej. Najbardziej popularną implementacją MPI I/O jest ROMIO, a jej dwie główne optymalizacje to *data sieving* i *two-phase I/O* [18, 105], oba zobrazowane schematycznie na rysunku 3.3b. *Data sieving* polega na zastąpieniu kilku nieciągłych żądań do pliku pojedynczym ciągłym żądaniem, a następnie rozdzieleniu pojedynczej odpowiedzi od systemu plików ponownie na kilka odpowiedzi. Ograniczeniem optymalizacji jest pamięć przeznaczona na jedno żądanie – jeśli nieciągłe żądania będą od siebie zbyt oddalone, ROMIO zrezygnuje z łączenia je w jedno. *Two-phase I/O*, jak sama nazwa wskazuje, przebiega w dwóch fazach. Faza pierwsza polega na reorganizacji serii żądań z różnych procesów MPI w taki sposób, żeby zebrać je w większe, ciągłe żądania, natomiast niekoniecznie ściśle związane ze źródłem żądania. Faza druga to odpowiednia dystrybucja odpowiedzi otrzymanej od sytemu plików w taki sposób, żeby każdy proces otrzymał dane, o które pytał. Obecnie ROMIO doczekało się wielu rozszerzeń, na przykład równoległej, wielowątkowej implementacji *two-phase I/O* [106].

Rozwiązaniem alternatywnym do ROMIO jest OMPIO, którego główną zaletą jest duża modularność [11]. OMPIO dzieli funkcjonalność dostarczaną przez MPI I/O na fragmenty nazywane frameworkami, z których każdy może być realizowany przez jeden z zestawu modułów. Podczas otwierania pliku inicjalizowane są wszystkie frameworki, a dla każdego z nich wybierany jest moduł, który powinien się najlepiej sprawdzić w obsłudze danego pliku. Dla przykładu, w sytuacji, w której komunikacja z systemem plików może się odbywać zgodnie z kilkoma protokołami, na przykład POSIX (*Portable Operating System Interface for Unix*) albo protokołem własnym, wybierany jest wariant potencjalnie bardziej wydajny. W zaprezentowanym rozwiązaniu utworzono pięć modułów, odpowiada-

MOST WIEDZY Downloaded from mostwiedzy.pl



(a)

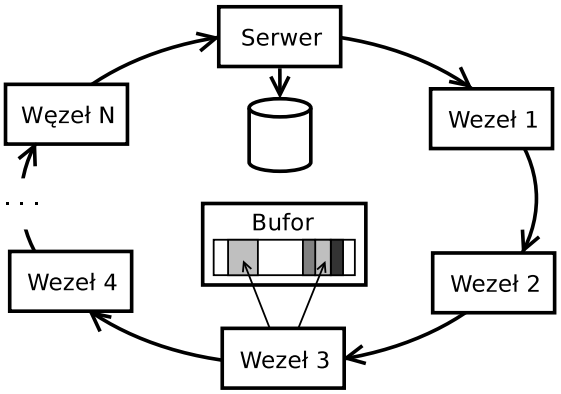


(b)

Rysunek 3.3: Schemat działania algorytmu *data sieving* (a) i *two-phase I/O* (b) stosowanych w popularnym rozwiązaniu ROMIO

jących kolejno za obsługę podstawowych operacji w systemie plików (utworzenie, otwarcie, zamknięcie, usunięcie), transfer danych z i do pliku (operacje odczytu i zapisu), funkcje zbiorowe (realizowane jednocześnie przez więcej niż jeden proces), pamięć podręczną i tryb współdzielonego wskaźnika pliku. Eksperymenty przeprowadzone przy użyciu benchmarka MPI Tile IO pokazały, że system z przykładową implementacją modułów był w stanie elastycznie dostosować się do pliku i kontekstu, w którym był otwierany, i zwiększyć wydajność pracy z plikiem w porównaniu do ROMIO [11]. Dużą wadą zaprezentowanego rozwiązania jest natomiast zgodność jedynie z jedną implementacją MPI, OpenMPI, a dostosowanie narzędzia do innych implementacji może być bardzo trudne ze względu na silne powiązanie architektury OMPIO i OpenMPI. Z tego samego powodu zrezygnowano z realizacji autorskiego rozwiązania jako dodatkowego modułu bądź rozszerzenia wybranej implementacji MPI I/O.

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 3.4: Architektura Catwalk-ROMIO; węzły obliczeniowe dodają kolejno żądania do bufora (na rysunku bufor jest obecnie obsługiwany przez węzeł nr 3)

Istnieją również mniej popularne implementacje MPI I/O, na przykład Catwalk-ROMIO [37], której główną ideą jest zorganizowanie wszystkich węzłów zgodnie z topologią pierścienia. W takiej topologii węzły przekazują między sobą bufor żądań, a każdy kolejny węzeł wypełnia bufor w taki sposób, żeby żądania zaczęły tworzyć większe, ciągłe bloki, jak pokazano na rysunku 3.4. Topologia pierścienia wydaje się być podatna na problemy ze skalowaniem przy większej liczbie węzłów ze względu na czas, jaki musi upłynąć, zanim wszystkie węzły zaktualizują bufor. Niestety, rozwiązanie było testowane maksymalnie dla szesnastu węzłów, nie pokazano również testów skalowalności.

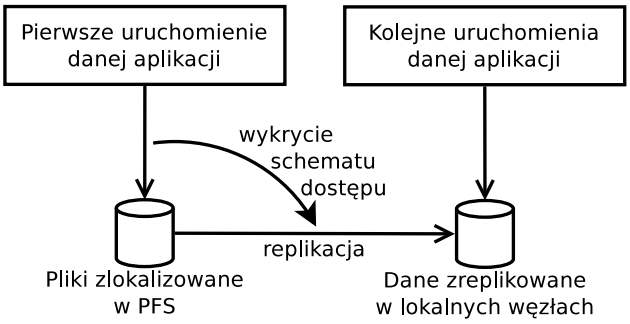
Omówione implementacje MPI I/O to rozwiązania dedykowane systemom operacyjnym zgodnym z POSIX, czyli obecnie głównie szerokiej rodzinie dystrybucji Linuksa oraz coraz mniej popularnym wariantom systemu Unix. Może się wydawać, że warto byłoby przyjrzeć się również implementacjom stosowanym w Microsoft Windows, okazuje się natomiast, że nie jest to obecnie system szeroko stosowany w HPC. Na wcześniej przywołanej liście top500, Windows pojawił się ostatnio w 2015 roku, w którym zarządzał tylko jednym superkomputerem z pięciuset najszybszych, a w całej historii listy nigdy zarządzał więcej niż czterema z pięciuset. Uzasadnia to niewielką liczbę prac naukowych, w których wspomniano o implementacjach MPI dedykowanych wyłącznie systemowi Windows (np. Microsoft MPI[10]) oraz brak badań na temat ogólnej wydajności takich implementacji, nie wspominając już o wydajności MPI I/O.

Niektóre rozwiązania nie tworzą własnej implementacji MPI I/O, a jedynie rozszerzają istniejącą. Twórcy biblioteki nazwanej APSM (*Asynchronous Progress Support for*

[10] `https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi`

MOST WIEDZY Downloaded from mostwiedzy.pl

MPI) pokazali, w jaki sposób rozszerzyć jedynie podzbiór funkcji MPI, w tym przypadku funkcje do komunikacji nieblokującej [115]. Od strony technicznej wykorzystano interfejs profilowania aplikacji PMPI (*MPI Profiling Interface*), który umożliwił przechwycenie wybranych funkcji i zrealizowanie ich w ramach biblioteki.



Rysunek 3.5: Schemat działania narzędzia PLDA

Jak wcześniej wspomniano, na wydajność aplikacji w znaczny sposób wpływa schemat żądań kierowanych do systemu plików. Można znaleźć rozwiązania, które potrafią dostosowywać się do sposobu pracy aplikacji z plikiem i modyfikować swoje zachowanie. Narzędzie PLDA (*Pattern-Direct and Layout-Aware*) potrafi zamienić zbiór nieciągłych żądań do systemu plików na pojedyncze żądanie ciągłe, a dodatkowo zreplikować wybrane fragmenty plików i przechowywać je w węzłach, w których będą miałby być użyte [120]. Proces w uproszczeniu ilustruje rysunek 3.5. Chociaż wyniki wydajnościowe rozwiązania są niezwykle korzystne (zwiększenie wydajności w niektórych przypadkach nawet o 970%), narzędzie wymaga najpierw wykonania kalibracji w postaci pojedynczego dodatkowego uruchomienia aplikacji, żeby zanalizować schemat pracy z plikiem. Oznacza to, że w przypadku aplikacji, w których schemat pracy zmienia się bardziej dynamicznie niż w syntetycznych benchmarkach, należy spodziewać się o wiele niższego zysku wydajnościowego.

We wstępie rozdziału omówiono najprostszą konfigurację oferującą dostęp do plików w HPC, gdzie komunikować się z systemem plików może każdy proces aplikacji. Istnieje jednak podejście organizowania węzłów zgodnie z bardziej skomplikowanymi topologiami [14, 41], w którym można separować część obliczeniową od części I/O, na przykład wydzielając odpowiednie węzły odpowiedzialne za komunikację z systemem plików czy stosując osobne izolowane sieci [104]. Niestety, ze względu na prostą konfigurację klastra testowego (wszystkie węzły, łącznie z węzłami serwera systemu plików, podłączone są do pojedynczego przełącznika), autorskie rozwiązanie zaproponowane w tej rozprawie nie optymalizuje dostępu do plików pod kątem zastosowanej topologii klastra. W przypad-

MOST WIEDZY Downloaded from mostwiedzy.pl

ku dużych superkomputerów o bardziej złożonej konfiguracji, odpowiednie dostosowanie zaproponowanego rozwiązania może znacząco wpłynąć na wyniki wydajnościowe – dalsze prace nad rozwiązaniem powinny wiec uwzględnić alternatywne topologie klastra.

Typową sytuacją, zwłaszcza w przypadku większych klastrów, jest jednoczesna obsługa kilku aplikacji przez jeden system plików [99]. Chociaż zagadnienie jest istotne, wykracza poza zakres rozprawy i również nie będzie w niej omówione.

Oprócz szeregu pomysłów na optymalizację zapisu i odczytu fragmentów plików, w jednym z artykułów poruszono kwestię obsługi metadanych [3]. Według autorów, dalszy rozwój rozproszonych systemów plików i skalowalnych aplikacji je wykorzystujących może potencjalnie doprowadzić do sytuacji, w której obsługa metadanych stanie się wąskim gardłem całego systemu. Chociaż eksperymenty przeprowadzono głównie dla operacji niewspieranych przez MPI I/O, takich jak zarządzanie katalogami, pojawiły się również scenariusze wchodzące w zakres rozprawy, na przykład problem otwierania pliku przez setki procesów. Treść artykułu wpłynęła na kształt autorskiego rozwiązania w dwojaki sposób. Z jednej strony, w celu odciążenia rozproszonego systemu plików zaprojektowano komunikację w taki sposób, żeby tylko jeden proces z danego węzła przesyłał żądania do PFS. Z drugiej strony, uwzględniono problem zarządzania metadanymi i dołożono wszelkich starań, żeby jak najbardziej zredukować ich rozmiar.

3.3 Optymalizacje bazujące na pamięci opartej o SSD

Spora część technik optymalizacji rozproszonych systemów plików i oprogramowania pośredniczącego bazuje na urządzeniach wyposażonych w pamięć SSD. Ze względu na pewne podobieństwa pamięci SSD do pamięci NVRAM, takie jak wyższa wydajność od dysków HDD czy swobodny dostęp, rozwiązania te zostały opisane nieco szerzej. W tym miejscu warto również wspomnieć o tak zwanych dyskach hybrydowych SSHD (*Solid-State Hard Drive*). Napędy SSHD to połączenie w jednej obudowie dwóch typów pamięci – HDD o większej pojemności i flash o mniejszej pojemności. Zasada działania urządzenia polega na zastosowaniu pamięci flash jako pamięci podręcznej – sterownik zwiększa wydajność urządzenia przechowując w szybszej pamięci najczęściej wykorzystywane dane. Ze względu na brak artykułów naukowych, nie omówiono jednak żadnego rozwiązania opartego o SSHD – autor podejrzewa, że jest to związane z potencjalnie niezwykle szybkim zużyciem modułu flash przy intensywnej pracy na dużych danych.

MOST WIEDZY Downloaded from mostwiedzy.pl

Badania wskazują na duży zysk wydajnościowy po prostej zmianie pamięci masowej rozproszonego systemu plików z HDD na SSD. W jednym z artykułów do testów wykorzystano konfigurację opartą o bliżej nieokreślony dysk HDD i urządzenie Intel Optane SSD, które bazuje na technologii 3D XPoint (czyli w teorii tej samej, która – zgodnie z poprzednim rozdziałem – ma pozwolić na stworzenie bajtowo adresowanej pamięci NVRAM) [116]. W klasterze testowym, złożonym z maksymalnie sześciu węzłów, uruchomiono cztery aplikacje weryfikujące przepustowość w operacjach odczytu i zapisu do pliku. Wyniki pokazały kilkukrotny wzrost przepustowości, zależny głównie od schematu dostępu do pliku w aplikacji. W rozprawie zdecydowano jednak, że jeśli pamięć NVRAM ma poprawiać wydajność aplikacji, to należy ją porównywać z konfiguracją opartą o SSD. Przy tak dużej liczbie badań zastosowania pamięci SSD w poprawie szybkości pracy z plikami lokalizowanymi na HDD, wykorzystanie pamięci NVRAM do optymalizacji rozwiązań opartych o HDD staje się zagadnieniem wtórnym.

Przykładem zastosowania architektury wielopoziomowej, gdzie mniejsza i szybsza pamięć (tutaj SSD) wykorzystywana jest jako pamięć podręczna w dostępie do pamięci wolniejszej (tutaj HDD) jest narzędzie iTransformer [125]. Twórcy iTransformera zrealizowali swoje rozwiązanie w formie modułu do kernela systemu Linux, więc jego dużą zaletą jest niezależność nie tylko od technologii warstwy pośredniczącej, ale również od oprogramowania rozproszonego systemu plików. Po zastosowaniu optymalizacji średni wzrost wydajności aplikacji testowych, w tym przypadku czterech benchmarków syntetycznych, wyniósł 35%. Taki wynik trudno porównać do rozwiązania przedstawionego w rozprawie – wydajność zaproponowanych mechanizmów zależy w dużym stopniu od sposobu działania aplikacji, co utrudnia wyznaczenie średniego przyrostu wydajności. Teoretycznie, dla większości eksperymentów wykonanych przy użyciu syntetycznych benchmarków przedstawionych w rozdziale 6, średni wzrost wydajności był większy niż dla narzędzia iTransformer i przekroczył 50%. W praktyce natomiast, jak to zostanie przedstawione dalej w rozprawie, spodziewany wzrost wydajności będzie wynikał ze sposobu dostępu do pliku w ramach aplikacji.

W wielu innych rozwiązaniach również stosuje się pamięć podręczną opartą o SSD w różnych wariantach integracji ze standardowym stosem technologicznym. Przykładowo, rozwiązanie iBridge, oferujące porównywalną wydajność, zostało zaimplementowane na poziomie rozproszonego systemu plików [126]. Chociaż takie rozwiązania wydają się potencjalnie możliwe do przeniesienia z optymalizacji zestawu pamięci HDD/SSD na zestaw

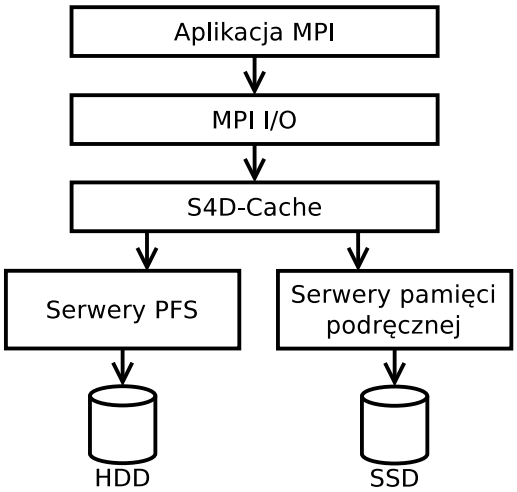
MOST WIEDZY Downloaded from mostwiedzy.pl

SSD/NVRAM, w pierwszym akapicie rozdziału 4.2 zostaną przedstawione badania, które pokażą, że proste zastąpienie pamięci SSD pamięcią NVRAM nie wpływa znacznie na wydajność aplikacji.

Narzędziem podobnym do wcześniej omówionego PLDA jest CARL (*Cost-Aware Region Level*) [33]. CARL to biblioteka działająca na styku MPI I/O i PFS, która również wymaga dodatkowego uruchomienia aplikacji w celu zebrania statystyk związanych ze schematem żądań kierowanych do pliku, a następnie potrafi odpowiednio rozdysponować fragmenty. Różnicą w stosunku do PLDA jest zmiana lokalizacji wybranych fragmentów plików – z węzłów lokalnych na wybrane serwery sytemu plików. Twórcy CARL-a założyli, że koszt urządzeń SSD jest dużo wyższy od dysków HDD i zaprojektowali swoje rozwiązanie w taki sposób, żeby z technologii SSD korzystały tylko niektóre serwery rozproszonego systemu plików. Po kalibracji przeprowadzonej dla wybranej aplikacji, narzędzie umieszcza częściej żądane fragmenty pliku na serwerach wyposażonych w nośniki SSD. Rozwiązanie doczekało się alternatywnego wariantu, w którym serwery pracujące na dyskach SSD nie zarządzają fragmentami plików, tworzą natomiast dodatkową warstwę pamięci podręcznej [34]. Wyniki prezentują się podobnie do rezultatów uzyskanych przez PLDA (poprawa wydajności w przypadku syntetycznych benchmarków nawet do 450%). Podobne są również wady rozwiązania, czyli konieczność co najmniej dwukrotnego uruchomienia aplikacji oraz wymaganie stałego schematu dostępu do pliku. Porównując wyniki biblioteki CARL z rozwiązaniem przedstawionym w rozprawie, biorąc również pod uwagę wcześniej wspomnianą dużą zależność wydajności od schematu dostępu do pliku, autorskie rozwiązanie dla specyficznych, korzystnych przypadków również było w stanie zaoferować kilkukrotny przyrost wydajności (dla syntetycznych benchmarków przedstawionych w rozdziale 6 nawet ponad 500%).

Nie wszystkie implementacje pamięci podręcznej przekierowujące wybrane żądania do serwerów zaopatrzonych w pamięć SSD wymagają kalibracji, żeby wykryć schemat dostępu do pliku. Przykładowo, biblioteka S4D-Cache przenosi najczęściej wykorzystywane fragmenty do szybszych węzłów już podczas działania aplikacji [32]. Koncepcję ilustruje rysunek 3.6 Chociaż poprawa wydajności nie jest tak duża jak w poprzednich rozwiązaniach, w niektórych przypadkach rozwiązanie potrafiło zwiększyć przepustowość sytemu plików nawet o około 60%. Interesująca okazała się również duża zależność wydajności rozwiązania od rozmiaru zainstalowanej pamięci SSD. Na podstawie wyników zaprezentowanych w artykule i podobnych, wyraźnie widać, że najbardziej korzystnym scenariuszem

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 3.6: Podstawowa koncepcja S4D-Cache, w której dodatkowa warstwa zlokalizowana pomiędzy MPI I/O a serwerem rozproszonego systemu plików może przekierować żądania do pamięci podręcznej

byłoby stworzenie takiej konfiguracji, w której szybka pamięć może zmieścić cały przetwarzany przez aplikację plik. W przypadku pamięci NVRAM, której spodziewana pojemność jest mniejsza od dostępnych układów pamięci SSD, trudno liczyć na taki rezultat, jeśli pamięć NVRAM miałaby być instalowana jedynie w serwerach systemu plików. W rozprawie założono jednak, że nowe układy pamięci miałyby być częścią każdego węzła klastra obliczeniowego, a w podstawowej konfiguracji rozmiar przetwarzanych plików jest mniejszy od sumarycznej ilości pamięci NVRAM w klastrze.

Nieco innym podejściem, w porównaniu do poprzednio opisanych rozwiązań, cechuje się LiU (*Lift it Up*) [38]. LiU to rozszerzenie rozproszonego systemu plików, w którym również część danych przeniesiono z nośnika HDD do SSD. Główną różnicą jest jednak metoda wybierania danych, które powinny trafić do szybszej pamięci. Twórcy narzędzia zauważyli, że dla obu wspomnianych nośników pamięci masowej, przepustowość przy sekwencyjnym odczycie i zapisie jest porównywalna, a największą różnicą jest opóźnienie w dostępie do pamięci. Podzielili więc żądanie na dwie części: głowę (ang. *head*), czyli początkowy fragment, i ciało (ang. *body*), czyli pozostałą część. Dzięki umieszczeniu głowy żądania w pamięci SSD uzyskano następujący efekt: kiedy żądanie trafia do systemu plików, opóźnienie w dostępie do danych zlokalizowanych na HDD jest kompensowane przez szybki dostęp do początkowego fragmentu danych żądania. Trudnością w takim podejściu jest natomiast wykrycie schematu dostępu do pliku i skuteczność zależna od rozmiaru

MOST WIEDZY Downloaded from mostwiedzy.pl

żądania. Ze względu na jedno z założeń rozprawy o poprawie wydajności aplikacji o nieznanym schemacie i rozmiarze żądań, podejście nie sprawdzi się w autorskim rozwiązaniu.

Dotychczas omówiono optymalizacje wysokopoziomowe w warstwie oprogramowania pośredniczącego albo systemu plików, jak również rozwiązania niskopoziomowe, implementowane jako rozszerzenie systemu operacyjnego. Należy zaznaczyć, że istnieją również metody przyspieszające pracę z plikami w HPC osadzone dużo bliżej fizycznej pamięci, w samym oprogramowaniu sprzętowym (ang. *firmware*) [46]. Nie wchodząc jednak w szczegóły konkretnej technologii, pozwalającej w przyszłości na zbudowanie pamięci NVRAM, opracowanie uniwersalnych optymalizacji na tak niskim poziomie byłoby niezwykle trudne.

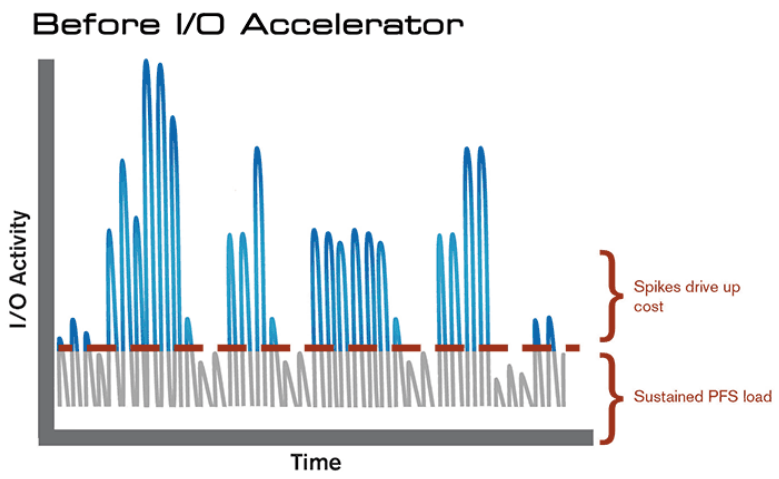
3.4 Rozwiązania typu *burst buffer*

Jak pokazano w cytowanej literaturze, żeby uniknąć zbytniego przeciążenia równoległego systemu plików stosuje się szereg różnych technik optymalizacyjnych, które pozwoliły wielu aplikacjom znacznie ograniczyć liczbę żądań i komunikować się z PFS jedynie w trzech przypadkach:

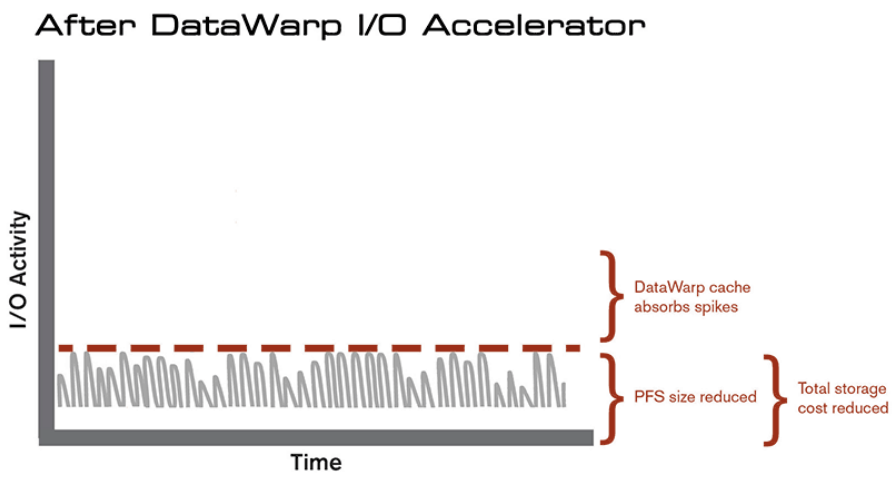
- inicjalizacji aplikacji lub kolejnych iteracji aplikacji, podczas których każdy proces wczytuje istotną dla przetwarzania porcję danych,
- zapisu przetworzonych danych, ponownie podczas każdej iteracji lub dopiero pod koniec cyklu życia aplikacji,
- zapisu kolejnych wersji stanu aplikacji, umożliwiających wznowienie obliczeń na wypadek awarii (ang. *checkpointing*) w przypadku aplikacji, w których zapis odbywa się rzadziej niż przy każdej iteracji.

Efektem jest często sytuacja, w której okresy stosunkowo niewielkiej intensywności operacji I/O przeplatają się z chwilowymi wybuchami (ang. *burst*) żądań, będących dla PFS sporym wyzwaniem. Technika *burst buffer* polega na utworzeniu w szybkiej pamięci buforów, pozwalających przyjąć chwilowe skoki obciążenia i zsynchronizować zgromadzone dane podczas okresu niskiej intensywności przyjmowanych żądań. Rysunek 3.7 ukazuje schematycznie intensywność operacji kierowanych do PFS w standardowej konfiguracji i architekturze rozszerzonej o *burst buffer*, w tym przypadku w rozwiązaniu DataWarp, proponowanym przez znaną z produkcji superkomputerów firmę Cray.

MOST WIEDZY Downloaded from mostwiedzy.pl



(a)



(b)

Rysunek 3.7: Materiały marketingowe firmy Cray pokazujące obciążenie przed (a) i po (b) zastosowaniu idei burst buffer zaimplementowanej w technologii DataWarp *(źródło: cray.com)*

W literaturze wyróżnia się dwa główne typy architektury *burst buffer*: instalowane w węzłach lokalnych i bufory współdzielone dostępne zdalnie [23]. W wersji lokalnej bufor pojawia się w każdym węźle obliczeniowym, a ogólna przepustowość operacji na plikach w optymalnych warunkach powinna rosnąć liniowo wraz z liczbą węzłów klastra [110]. Źródłem takiej zależności jest sposób pomiaru, stosowany w oprogramowaniu testującym, w którym kolejne procesy uruchamiane na kolejnych węzłach w chwilach dużego obciążenia korzystają wyłącznie z pamięci podłączonej lokalnie.

MOST WIEDZY Downloaded from mostwiedzy.pl

Podejście alternatywne zakłada, że buforów wyposażonych w szybką pamięć jest mniej niż węzłów obliczeniowych klastra. Tworzą one wtedy dodatkową warstwę, zlokalizowaną pomiędzy węzłami komunikującymi się z rozproszonym systemem plików, a systemem plików. Podejście może być stosowane w mniejszych klastrach bez wydzielonej warstwy węzłów odpowiedzialnych za komunikację z PFS, ale bardzo dobrze wpisuje się również w topologie znane z bardziej rozbudowanych środowisk, w których taka warstwa została użyta [64]. Wydajność aplikacji korzystających z *burst buffers* jest silnie zależna od liczby serwerów przeznaczonych na bufory, a także ich konfiguracji [111]. Pojawia się więc trudność w uzyskaniu jak najwyższej wydajności przy jednoczesnej redukcji kosztu rozszerzenia architektury, biorąc pod uwagę zarówno koszt zakupu, jak i utrzymania dodatkowych węzłów wraz z zużyciem ich energii. W praktyce w dużych klastrach można skorzystać z technologii, na którą składa się nie tylko oprogramowanie, ale i odpowiednio skonfigurowany sprzęt. Przykładami takich komercyjnych rozwiązań są wcześniej wspomniane DataWarp firmy Cray [35], albo Infinite Memory Engine, zaimplementowany przez firmę Data Direct Network [95].

Może się wydawać, że implementacja idei lokalnej wersji *burst buffer* powinna być dobrą podstawą do stworzenia autorskiego rozwiązania. Ideą *burst buffer* jest jednak poradzenie sobie wyłącznie z problemem chwilowego wzrostu intensywności żądań. Celem rozprawy jest natomiast próba zwiększenia wydajności aplikacji, nie określono jednak konkretnego schematu dostępu do pliku, więc rozwiązanie powinno być dopasowane również do aplikacji równomiernie obciążających system plików. Z pierwotną ideą wiąże się również sposób testowania implementacji *burst buffer*, który polega często na weryfikacji zachowania aplikacji podczas checkpointingu. Biorąc pod uwagę nieulotność pamięci NVRAM, można spróbować zaprojektować aplikacje, które nie będą korzystały z typowego checkpointingu, a zamiast tego zostaną napisane w taki sposób, żeby o spójność i ochronę danych przed utratą dbały mechanizmy automatyczne.

Istnieje również narzędzie typu *burst buffer* wykorzystujące pamięć NVRAM, zostanie ono jednak omówione w kolejnym rozdziale.

3.5 Podsumowanie

Podsumowując rozdział, wyraźnie widać, że temat optymalizacji przetwarzania plików w systemach wysokiej wydajności jest niezwykle istotny – według wielu badaczy obecnie

MOST WIEDZY Downloaded from mostwiedzy.pl

operacje I/O w klastrach są często wąskim gardłem całego systemu. W związku z tym pojawiają się badania, mające na celu zoptymalizowanie pracy z plikami w klastrze, wśród których większość zawiera również przedstawienie przykładowej implementacji zaproponowanego rozwiązania wraz z testami wydajnościowymi. Skrócone porównanie narzędzi opisanych w rozdziale zaprezentowano w tabeli 3.1. Wyraźnie widać, że metody optymalizacji są obecne na wielu poziomach przetwarzania: od poprawy mechanizmów systemu operacyjnego, przez rozszerzenia dedykowane rozproszonym systemom plików, po zwiększanie wydajności oprogramowania pośredniczącego, czyli przeważnie MPI I/O. Ze względu na to, że problemy z wydajnością pojawiają się przeważnie w sytuacji, w której z pliku korzysta wiele procesów aplikacji jednocześnie, większość optymalizacji bazujących wyłącznie na oprogramowaniu ma na celu reorganizację żądań, zanim trafią do systemu plików w taki sposób, żeby zredukować ich wolumen. Optymalizacje, które pojawiły się wraz z upowszechnieniem pamięci masowej opartej o SSD, to zazwyczaj różne warianty pamięci podręcznej, stosowanej ze względu na koszta głównie w serwerach PFS, często jedynie wybranych.

Projektując autorskie rozwiązanie na potrzeby rozprawy warto wykorzystać wnioski płynące z przytoczonej literatury. Techniki, które potencjalnie mogłyby potwierdzić tezę postawioną w rozprawie, powinny z jednej strony zmniejszać obciążenie rozproszonego systemu plików, z drugiej natomiast wykorzystać dodatkową pamięć w taki sposób, żeby unikać kosztownego czasowo sięgania do pamięci wolniejszej. Co więcej, skoro pomysły optymalizacyjne przedstawione w rozdziale zwiększały wydajność aplikacji, korzystnym wydaje się zaprojektowanie rozwiązania w taki sposób, żeby będąc zgodne z ogólnie przyjętymi standardami i interfejsami, mogło być stosowane w połączeniu z innymi narzędziami w dowolnej konfiguracji. Ważne jest również unikanie ograniczeń obecnych w przedstawionych narzędziach – rozwiązanie musi być skalowalne, niezależne od konkretnych elementów stosu technologicznego (systemu operacyjnego, implementacji MPI, systemu plików) oraz nie powinno wymagać kalibracji.

MOST WIEDZY Downloaded from mostwiedzy.pl

Tabela 3.1: Porównanie rozwiązań kompatybilnych z aplikacjami wykorzystującymi MPI I/O, zwiększających wydajność przetwarzania plików w HPC

Rozwiązanie	Poziom	Główna idea	Ograniczenia
ROMIO	implementacja MPI I/O	reorganizacja żądań	-
OMPIO	implementacja MPI I/O	modularyzacja	wyłącznie w OpenMPI
Catwalk-ROMIO	implementacja MPI I/O	topologia pierścienia	potencjalnie niska skalowalność
PLDA	rozszerzenie MPI I/O i PFS	reorganizacja żądań, replikacja danych w węzłach	konieczność kalibracji
APSM	rozszerzenie MPI I/O	żądania asynchroniczne	mechanizm wbudowany w większość implementacji
iTransformer	moduł jądra Linux	pamięć podręczna w SSD serwera PFS	SSD wyłącznie w serwerach PFS
iBridge	PFS	pamięć podręczna w SSD serwera PFS	SSD wyłącznie w serwerach PFS
CARL	rozszerzenie MPI I/O i PFS	reorganizacja żądań, wybrane serwery PFS z SSD	konieczność kalibracji, SSD wyłącznie w serwerach PFS
S4D-Cache	rozszerzenie MPI I/O	pamięć podręczna w SSD serwera PFS	SSD wyłącznie w serwerach PFS
LiU	rozszerzenie PFS	pamięć podręczna w SSD serwera PFS	SSD wyłącznie w serwerach PFS
Lokalne *burst buffers*	dodatkowa warstwa przed PFS	pamięć podręczna w SSD dodatkowej warstwy	dedykowane mechanizmom checkpointingu, niedostosowane do aplikacji
Zdalne *burst buffers*	dodatkowa warstwa przed PFS	pamięć podręczna w SSD dodatkowej warstwy	dedykowane mechanizmom checkpointingu, SSD wyłącznie w serwerach PFS, wymaga dodatkowych serwerów

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 4

Zastosowanie adresowanej bajtowo pamięci NVRAM w obliczeniach wysokiej wydajności

4.1 Optymalizacje obliczeń i zapobieganie utracie danych

Sam pomysł zastosowania pamięci NVRAM w obliczeniach wysokiej wydajności nie jest nowy. W 2012 roku, na fali doniesień o postępach nowych technologii, takich jak PCM, spróbowano ocenić potencjał pamięci NVRAM w aplikacjach operujących na dużych ilościach danych [108]. Badacze przygotowali oprogramowanie symulujące, które nakładało dodatkowe opóźnienie w dostępie do pamięci, a następnie skalibrowali je tak, żeby utworzyć pięć konfiguracji pamięci. Najmniej wydajna konfiguracja była porównywalna do pamięci SSD, a w najbardziej wydajnej zrezygnowano z dodatkowych opóźnień – jej wydajność była jednak zauważalnie niższa od pamięci DRAM ze względu na architekturę symulatora. Testy przeprowadzono na pojedynczym węźle przy użyciu aplikacji przeszukiwania dużych grafów wszerz, wynikiem referencyjnym była szybkość przetwarzania kolejnych węzłów grafu w wariancie, w którym cały graf zlokalizowany był w pamięci operacyjnej opartej o DRAM. Wyniki okazały się dość obiecujące: przy 256 wątkach wydajność aplikacji wahała się od około 50% referencyjnej wydajności dla najwolniejszej konfiguracji symulatora, do około 75% dla najszybszej konfiguracji. Badania potwierdziły więc, że pamięć NVRAM może znaleźć zastosowanie w aplikacjach przetwarzających duże dane, a w szczególności takie, które przekroczą rozmiar zainstalowanej pamięci operacyjnej. W niniejszej pracy weryfikacja możliwości użycia pamięci NVRAM również zostanie przeprowadzona przy użyciu symulatora.

Obecnie rozpatruje się wiele różnych scenariuszy potencjalnego użycia pamięci NVRAM ściśle w zastosowaniach HPC, a zależą one głównie od architektury pamięci w klastrze [109]. Inne rozwiązania będą stosowane w sytuacji, kiedy układy oparte o NVRAM będą zastę-

MOST WIEDZY Downloaded from mostwiedzy.pl

pować pamięć masową, występować w konfiguracji hybrydowej z pamięcią DRAM, albo nawet będą instalowane jedynie w serwerach odpowiedzialnych za I/O. Obszary, w których NVRAM powinien znaleźć zastosowanie, to między innymi wsparcie operacji I/O, tworzenie trwałych (odpornych na restart zarówno aplikacji, jak i całej maszyny) struktur danych czy zwiększenie niezawodności aplikacji. Dodatkowo, autorzy artykułu zauważają, że jeśli duża i trwała pamięć byłaby instalowana we wszystkich węzłach obliczeniowych, wówczas możliwe jest ograniczenie komunikacji i przetwarzanie większej ilości danych w ramach pojedynczego węzła. W artykule pojawia się również wątek zużycia energii – według przedstawionych badań DRAM odpowiada za około 30% do 60% zużycia energii, przez co, nawet jeśli pamięć NVRAM nie będzie oferowała wydajności porównywalnej z DRAM, może być chętnie stosowana w klastrach obliczeniowych ze względu na niższe koszta utrzymania.

Bardziej techniczne aspekty zastosowania pamięci NVRAM w HPC można znaleźć w artykule A. Rudoffa [93]. W swojej pracy dzieli on zastosowania nowego typu pamięci na takie, dla których nieulotność nie ma znaczenia, i takie, dla których nieulotność jest kluczowa. W pierwszej kategorii mieszczą się wszystkie te aplikacje, które skorzystają z nowej pamięci, jakby była nieco mniej wydajnym rozszerzeniem pamięci operacyjnej. Druga obejmuje zastosowania związane z ochroną danych przed utratą w przypadku awarii. W artykule zaproponowano również konkretne narzędzia, które można wykorzystać, żeby aplikacja tworzona już dziś wspierała pamięć NVRAM, gdy ta tylko pojawi się na rynku. Dla programistów zainteresowanych pamięcią ulotną wymieniono bibliotekę memkind[1], która pomaga zarządzać hierarchią pamięci. Do zastosowań związanych z nieulotnością zarekomendowano libpmemobj – bibliotekę, będącą częścią PMDK (*Persistent Memory Development Kit*[2]). Zgodnie z rekomendacją, narzędzia wchodzące w skład PMDK wykorzystano do zbudowania autorskiego rozwiązania opisanego w rozprawie.

Niektóre systemy bazujące na pamięci NVRAM, stworzone na potrzeby zastosowań HPC, oferują programiście interfejs, pozwalający zarządzać taką pamięcią jawnie, inne natomiast ukrywają pamięć NVRAM i traktują ją jako rozszerzenie pamięci operacyjnej. W jednym z artykułów przeprowadzono analizę możliwości zastosowania heterogenicznej pamięci operacyjnej – części opartej o DRAM, a części o symulowaną pamięć PCM – w aplikacjach HPC [88]. Do testów zaprojektowano algorytm zarządzający pamięcią, który wybierał odpowiednie miejsce do alokowania nowej strony pamięci i pozwalał na migrację

[1] `https://github.com/memkind/memkind`
[2] `http://pmem.io/pmdk/libpmem/`

MOST WIEDZY Downloaded from mostwiedzy.pl

stron pomiędzy dwoma typami pamięci. Eksperymenty przeprowadzono przy użyciu symulatora PCM, dla którego założono, że nowa pamięć będzie czterokrotnie wolniejsza od DRAM. Na szczególną uwagę zasługuje duża liczba i różnorodność przetestowanych aplikacji – było to pięć następujących aplikacji: modelowanie oceanu, przeprowadzenie popularnych obliczeń stosowanych w chemii, symulacje fizyki plazmy, symulacje fizyki cząstek i aplikacja realizująca obliczenia związane z astronomią i kosmologią. Wyniki pokazały duży potencjał konfiguracji hybrydowych. Węzły wyposażone w 1GB pamięci DRAM były jedynie od 20% do 60% wolniejsze niż takie, gdzie użyto maksymalnej wielkości pamięci DRAM (różnie w zależności od zużycia pamięci aplikacji, nawet do 32GB). W artykule dużą wagę przywiązano również do wpływu algorytmu na żywotność układów opartych o PCM. Może się okazać, że pamięć NVRAM będzie oferować stosunkowo niską jak na pamięć operacyjną żywotność, przez co opłacalnym będzie ograniczenie liczby zapisów do pamięci. Pokazano jednak, że wybierając odpowiednią politykę zarządzania pamięcią, możliwe jest ograniczenie liczby zapisów w zależności od aplikacji nawet dziesięciokrotnie, co wiązało się ze spadkiem wydajności jedynie o 10%. Rozwiązania zarządzające pamięcią NVRAM automatycznie posiadają tę zaletę, że będą mogły być stosowane również w istniejących aplikacjach oraz nowych, w których programiści nie chcą poświęcać czasu na implementację własnych mechanizmów. Ten sposób działania może być szczególnie istotny w chwili, kiedy pierwsze urządzenia oparte o NVRAM wejdą na rynek, dlatego też taki tryb pracy przewidziano dla rozwiązania autorskiego.

Przykładem konkretnego narzędzia, pozwalającego jawnie korzystać z pamięci NVRAM w HPC, jest system Phoenix [24]. Podstawowy zestaw funkcji biblioteki pozwala na tworzenie tak zwanych obiektów trwałych, które są zarządzane przez narzędzie w sposób przezroczysty dla programisty. Phoenix zarządza lokalną pamięcią DRAM, lokalną pamięcią NVRAM i pamięcią NVRAM zlokalizowaną w innych węzłach w taki sposób, żeby dostęp do obiektu był możliwie jak najszybszy, z drugiej strony natomiast, żeby dane obiektu były bezpiecznie zapisane na wypadek awarii. W praktyce mechanizm polega na inteligentnym kopiowaniu obszarów pamięci, w zależności od potencjalnej szansy na dostęp w najbliższym czasie. Wydajne tworzenie kopii bezpieczeństwa na zewnętrznym węźle było możliwe dzięki wykorzystaniu mechanizmu RDMA (*Remote Direct Memory Access*). Podstawowe API systemu Phoenix jest przykryte zestawem bibliotek, zaprojektowanych na wyższym poziomie abstrakcji. Autorzy artykułu omawiają głównie bibliotekę PHX-C/R (*checkpoint/restart*), która pozwala na zapisywanie kolejnych wersji stanu obiektów trwałych i ich odtwarzanie w razie potrzeby. Phoenix został przetestowany przy użyciu

MOST WIEDZY Downloaded from mostwiedzy.pl

oprogramowania symulującego pamięć NVRAM i trzech aplikacji demonstracyjnych (badania mikroturbulencji w urządzeniach typu tokamak, symulacji zjawisk pogodowych oraz rozwiązywania równań Naviera-Stokesa), które korzystały z narzędzia w celu cyklicznego zapisu stanu swoich obiektów. Wyniki testów zebrane na czterowęzłowym klastrze (każdy węzeł wyposażony w 20 GB RAM i 12 GB symulowanego NVRAM) były następujące: użycie biblioteki pozwoliło zapisywać kolejne wersje stanu obiektów nawet do około dwunastu razy szybciej, niż w sytuacji, w której użyto prostego kopiowania całych obiektów z pamięci DRAM do NVRAM. W artykule nie poruszono jednak tematu zdalnego dostępu do pamięci zarządzanej przez Phoenix – wygląda na to, że rozwiązanie nie zwiększało wydajności wymiany informacji pomiędzy rozproszonymi procesami aplikacji (pomijając tworzenie kopii zapasowej).

Inne podejście do tworzenia obiektów trwałych zaprezentowano w narzędziu SIMPO (*Scalable In-Memory Persistent Object*) [124]. Rozwiązanie wprowadza nowy model programowania, który klasyfikuje wszystkie funkcje na dwie grupy: odroczone (ang. *deferrable*) i natychmiastowe (ang. *instant*). Funkcje odroczone to tak zwane funkcje leniwie wartościowane (ang. *lazy evaluation*), które nie zwracają żadnej wartości i operują jedynie na trwałych obiektach zarządzanych przez SIMPO. Funkcje natychmiastowe muszą natomiast być wykonane od razu, ze względu na zwracaną wartość używaną w dalszym przetwarzaniu lub obliczenia przeprowadzane poza obszarem zarządzanym przez SIMPO. Taka klasyfikacja pozwoliła twórcom podzielić program na transakcje i zastosować optymalizacje pomagające w wydajny sposób przenosić dane pomiędzy pamięcią DRAM i NVRAM. Chociaż nowy model programowania może wydawać się dużą wadą, twórcy stworzyli narzędzie, które potrafi zakwalifikować funkcje do odpowiednich grup w sposób automatyczny. Drugie ograniczenie to język frameworka, ponieważ został on stworzony jedynie dla aplikacji napisanych w C++. Wydaje się jednak, że podejście mogłoby zostać zaimplementowane dla dowolnego języka. Wyniki wydajnościowe otrzymane przy użyciu symulatora pamięci NVRAM pokazały nawet ponad dwukrotne przyspieszenie w przypadku najbardziej korzystnych scenariuszy przetestowanego benchmarka. Narzędzie zostało także wykorzystane w rzeczywistych aplikacjach, rozszerzonych o mechanizm checkpointingu zapewniający bezpieczeństwo danych na wypadek awarii. Testy przeprowadzone dla uczenia maszynowego, mnożenia macierzy, kompresji i dekompresji danych oraz aplikacji rozwiązującej problem najkrótszej ścieżki w grafie pokazały pomijalny narzut związany z tworzeniem punktów przywracania. Jest to wynik porównywalny do narzutu na mechanizmy ograniczające skutki awarii zaproponowane w autorskim rozwiązaniu (rezultaty

MOST WIEDZY Downloaded from mostwiedzy.pl

eksperymentów są tematem rozdziału 6.9).

Jak wspomniano przy narzędziach Phoenix i SIMPO, ze względu na swoją nieulotność NVRAM jest wykorzystywany jako pamięć, będąca podstawą narzędzi tworzonych głównie z myślą o zabezpieczeniu aplikacji przed utratą danych [87]. Już w 2010 roku spekulowano, że wraz z nadejściem pamięci NVRAM może pojawić się możliwość zapisywania stanu aplikacji bez narzutu związanego z kopiowaniem danych [50]. W jednym z późniejszych badań, w których autor rozprawy również brał udział, przetestowano dwie techniki: checkpointing oraz podwójne buforowanie polegające na duplikacji używanej pamięci i naprzemiennym prowadzeniu obliczeń w jednym z obszarów, podczas gdy drugi przechowuje spójny stan aplikacji [20]. W rozwiązaniu zarządzanie pamięcią NVRAM odbywa się przy użyciu *One-sided API*, będącego częścią standardu MPI. Testy obejmowały dwie aplikacje, wykorzystujące technikę SPMD (*Single Program Multiple Data*) – implementację algorytmu PageRank oraz rozwiązywanie układu równań liniowych metodą gradientu sprzężonego. Eksperymenty przeprowadzono na klastrze złożonym z ośmiu węzłów, z których każdy był wyposażony z 32 GB pamięci RAM, z czego około 17 GB było wykorzystane w symulatorze NVRAM. Wyniki zebrano dla szeregu różnych wartości parametrów pamięci, takich jak przepustowość, opóźnienie w dostępie do danych oraz dodatkowy czas potrzebny na upewnienie się, że dane zostały trwale zapisane w pamięci urządzenia. Rezultaty pokazały, że jedna z metod, tj. podwójne buforowanie, pozwoliła na zwiększenie bezpieczeństwa przetestowanych aplikacji przy pomijalnym narzucie wydajnościowym.

Rozwiązań, które wykorzystują pamięć NVRAM do budowania systemów HPC tolerujących awarie (ang. *fault tolerant*) przy niewielkim spadku wydajności, jest obecnie więcej. Pierwszym przykładem może być metoda rozszerzania implementacji algorytmów w taki sposób, żeby dane trzymane w pamięci trwałej automatycznie tworzyły spójny stan podczas każdej iteracji [119]. Dla przetestowanych aplikacji: rozwiązywania równań liniowych, mnożenia macierzy i modelowania metodą Monte-Carlo, narzut związany z wprowadzeniem metody był w większości przypadków mniejszy niż 3%. Drugi przykład bazuje na pomyśle przetwarzania transakcyjnego, w którym do czasu potwierdzenia kolejnej transakcji możliwe jest odtworzenie spójnego stanu z końca poprzedniej [62]. W tym przypadku testy przeprowadzone dla sześciu aplikacji równoległych pokazały, że narzut rozwiązania jest porównywalny do checkpointingu z zapisem kompletnego stanu aplikacji co 100 000 iteracji. Aplikacje, które weszły w skład eksperymentów, to: modelowanie sieci bayesowskiej, rekonstrukcja sekwencji DNA, wykrywanie włamań do sieci komputerowych, poszukiwa-

MOST WIEDZY Downloaded from mostwiedzy.pl

nie wyjścia w trójwymiarowym labiryncie, przetwarzanie dużych grafów i implementacja algorytmu k-średnich. Niektóre rozwiązania idą jeszcze o krok dalej i pozwalają nie tylko na odtworzenie spójnego stanu danych, ale i całego procesu, na przykład po restarcie systemu operacyjnego [61]. Niestety, wymienione rozwiązania nie obsługują scenariusza awarii sprzętowej, w której zniszczeniu uległyby dane zapisane w pamięci NVRAM. W takim przypadku konieczna byłaby redundancja w zapisie danych, na przykład przez zdalny zapis na innym węźle.

Jeśli chodzi o narzędzia skupione na zdalnym rozproszonym dostępie do pamięci, jako przykład może posłużyć Megalloc, czyli biblioteka pozwalająca na użycie lokalnych pamięci NVRAM we wszystkich węzłach klastra jako jednej, dużej, rozproszonej pamięci współdzielonej [122]. Główną motywacją do takich badań jest nieznany rozmiar pierwszych urządzeń opartych o NVRAM – możliwe jest, że nie będą tak duże, żeby przechowywać pełną kopię potrzebnych danych. Co więcej, pamięć współdzieloną można traktować również jako interfejs komunikacji pomiędzy procesami w aplikacjach rozproszonych. Za wydajność rozwiązania Megalloc odpowiada nie tylko pamięć NVRAM, ale także technologia RDMA, która pozwala na szybki dostęp do zdalnej pamięci bez nadmiernego obciążania procesora. Zarządzanie pamięcią odbywa się na pojedynczym węźle zwanym masterem. Niestety, nie wiadomo, jak scentralizowane sterowanie wpływa na skalowalność rozwiązania, ponieważ testy przeprowadzono tylko dla czterech węzłów. Zgodnie z tezą pracy, autorskie rozwiązanie powinno działać wydajnie w środowisku klastrowym, a biorąc pod uwagę obecny rozmiar klastrów obliczeniowych, tworzenie rozwiązań opartych o centralne elementy sterujące niesie duże ryzyko niskiej skalowalności. Z tego powodu, projektując rozwiązanie, zasadnym wydaje się rozważenie najpierw możliwości implementacyjnych w kwestii architektury zdecentralizowanej. Innym przykładem rozwiązania, pozwalającego na wydajny dostęp do pamięci NVRAM zlokalizowanej na innym węźle, jest biblioteka Pyramid, która również korzysta z RDMA [121].

Część badań związanych z użyciem pamięci NVRAM w obliczeniach HPC skupia się na konkretnej dziedzinie problemu. Jeden z niedawno opublikowanych artykułów omawia wpływ parametrów pamięci na czas wykonania algorytmów grafowych [97]. Potencjał użycia pamięci NVRAM w takich aplikacjach zilustrowały wyniki – podczas gdy przetwarzanie grafów w całości zlokalizowanych w pamięci NVRAM wiązało się z około sześćdziesięcioprocentowym narzutem czasowym, dodatkowe użycie DRAM jako pamięci podręcznej pozwoliło zmniejszyć narzut do 5%. Badania takie jak to pozwolą w przyszłości oszacować

MOST WIEDZY Downloaded from mostwiedzy.pl

opłacalność rozszerzenia architektury pamięci o pamięć NVRAM i wybrać jej odpowiednie proporcje ilości w stosunku do pamięci DRAM. W innym artykule zaprezentowano użycie pamięci NVRAM do zwiększenia wydajności aplikacji HPC tworzonych w modelu MapReduce [112]. Lista aplikacji testowych była stosunkowo szeroka – zaproponowaną architekturę zweryfikowano między innymi dla algorytmów takich jak sortowanie, PageRank, algorytm centroidów czy zliczanie słów. Wyniki były silnie zależne od konkretnej aplikacji, a zysk wydajnościowy wahał się od około 10% do nawet około 60%.

W kontekście nowej hierarchii pamięci warto również wspomnieć o popularnym ostatnio temacie w dziedzinie HPC, czyli efektywności energetycznej. W jednym z badań podjęto próbę oszacowania wpływu zastosowania dodatkowej pamięci NVRAM na zużycie energii podczas wykonywania aplikacji przetwarzających duże ilości danych [103]. Jeśli chodzi o wybór technologii, badacze wykorzystali dane o najbardziej obiecujących, takich jak PCM, STTRAM i FeRAM. Wyniki pokazały, że użycie nowego typu pamięci może mieć na celu nie tylko zwiększenie wydajności, ale i zmniejszenie zapotrzebowania na energię elektryczną warstwy sprzętowej. Eksperymenty dotyczyły dwóch konfiguracji. W konfiguracji prostej NVRAM stosowany był jako pamięć operacyjna, natomiast DRAM wykorzystano jako pamięć podręczną. Konfiguracja hybrydowa polegała na pamięci NVRAM zainstalowanej obok pamięci DRAM. Eksperymenty przeprowadzono dla wybranych aplikacji z zestawu narzędzi NAS Parallel Benchmarks[3], zaproponowanego przez programistów NASA, zbioru benchmarków CORAL[4] i aplikacji przetwarzającej łańcuchy DNA. W przypadku konfiguracji hybrydowej, potencjalna oszczędność energii wyniosła średnio 42%, ale wiązało się to z dwudziestopięcioprocentowym narzutem na czas wykonania obliczeń. Wyniki otrzymane dla konfiguracji prostej to 21% potencjalnej oszczędności energii przy jedynie siedmioprocentowym narzucie wydajnościowym.

Podsumowując, potencjał NVRAM w aplikacjach wysokiej wydajności został już dawno zauważony, a obecnie trwają badania, mające na celu oszacowanie możliwości nowej hierarchii pamięci i zaprojektowanie rozwiązań, które wpłyną nie tylko na szybkość prowadzenia obliczeń, ale również na zminimalizowanie ryzyka utraty danych i efektywność energetyczną. W literaturze pojawiły się następujące pomysły zastosowania nowej pamięci:

- wykorzystanie pamięci NVRAM jako dużej pamięci operacyjnej, a pamięci DRAM jako pamięci podręcznej,

[3] `https://www.nas.nasa.gov/publications/npb.html`
[4] `https://asc.llnl.gov/CORAL-benchmarks/`

MOST WIEDZY Downloaded from mostwiedzy.pl

- wykorzystanie pamięci NVRAM do przechowywania danych niemieszczących się w pamięci operacyjnej,
- tworzenie kontenerów obiektów trwałych, które potrafią przetrwać awarie czy restarty aplikacji, a czasem nawet restart maszyny,
- implementacja mechanizmu checkpointingu,
- umożliwienie dostępu do pamięci NVRAM nie tylko z poziomu węzła, gdzie została zainstalowana, ale także przy użyciu sieci i techniki RDMA.

Przedstawione rozwiązania pokazują dwie strategie: zarządzanie nową pamięcią w sposób przezroczysty dla użytkownika i w sposób jawny na poziomie kodu źródłowego aplikacji. Chociaż drugie podejście jest bardziej elastyczne, budując autorskie rozwiązanie, zastosowano podejście pierwsze, które pozwoli zaoferować nową jakość szerokiej grupie aplikacji istniejących już w chwili pojawienia się urządzeń opartych o NVRAM na rynku.

4.2 Optymalizacje operacji I/O

W literaturze można również znaleźć próby wykorzystania pamięci NVRAM do wsparcia operacji na plikach w HPC. W jednym z artykułów badacze weryfikują wydajność zastosowania pamięci NVRAM między innymi jako pamięci masowej w równoległym systemie plików komunikującym się z aplikacją przy użyciu interfejsu MPI I/O [65]. W eksperymentach z użyciem MPI wykorzystano chętnie stosowany benchmark IOR[5] [6], który pozwala na konfigurowalną symulację nakładadających się lub losowych (ang. *interleaved or random*, skąd nazwa narzędzia) żądań do pliku. Do emulacji NVRAM posłużyło narzędzie *Persistent Memory Block Driver* (PMBD), co oznacza, że zastosowano blokowe adresowanie pamięci [13]. Symulator nie nakładał żadnych dodatkowych opóźnień, wiec wydajność pamięci (nie licząc narzutu sterownika) była porównywalna do wydajności DRAM. Wyniki wydajnościowe zebrano dla kilku różnych konfiguracji, z perspektywy rozprawy najbardziej istotne są konfiguracje klastrowe – klaster testowy wyposażony był w pięć węzłów, z czego jeden przeznaczono na serwer plików. Pomimo dużo lepszych parametrów symulowanej pamięci NVRAM w porównaniu do pamięci SSD, różnice w przepustowości przesyłania danych z i do pliku były nieznaczne i wynosiły maksymalnie kilka procent (nie podano pełnych wartości liczbowych, dane odczytane z wykresu). Co więcej, pomimo że każdy

[5] `https://github.com/hpc/ior`

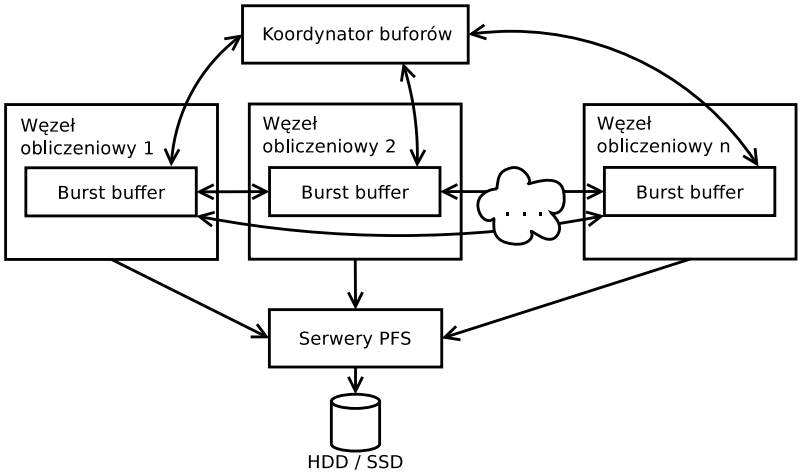
MOST WIEDZY Downloaded from mostwiedzy.pl

węzeł klastra był wyposażony w dwa procesory, a każdy z procesorów w sześć fizycznych rdzeni, podczas testów uruchomiono maksymalnie cztery procesy MPI na jednym węźle – nie wiadomo, jak wyglądałaby wydajność dla konfiguracji bardziej obciążających system plików. Z pracy można jednak wywnioskować, że proste zastąpienie nośnika pamięci masowej systemu plików z flash SSD na szybki NVRAM (tutaj porównywalny z DRAM) prawdopodobnie nie wpłynie znacząco na wydajność aplikacji w środowisku klastrowym. Do podobnych wniosków doszedł autor rozprawy, weryfikując, czy niewielkie zmiany w czasie dostępu do pamięci masowej mają duży wpływ na wydajność rozproszonego systemu plików [69]. Eksperymenty przeprowadzone na pięciowęzłowym klastrze i benchmarkach IOR oraz MPI Tile IO, uruchomionych w konfiguracji szesnastu procesów na każdym węźle, nie wykazały różnic w wydajności PFS większych niż 5%.

Kolejne rozwiązanie, również związane z I/O w HPC, nazwane Tinca, dodaje do systemu plików warstwę pamięci podręcznej opartej o NVRAM [114]. Główną ideą, stojącą za tym rozwiązaniem, jest wyeliminowanie narzutu na tak zwane księgowanie w systemie plików przez zapewnienie spójności pliku za pomocą wprowadzenia mechanizmu transakcji. Inną dużą zaletą narzędzia jest dopasowanie struktur pamięci podręcznej do bajtowego adresowania. Eksperymenty przeprowadzono wykorzystując symulację pamięci opartej o oprogramowanie wstawiające dodatkowe opóźnienia. Większość testów weryfikowała wydajność rozwiązania albo na pojedynczym węźle, albo przy użyciu benchmarków niepowiązanych z obliczeniami HPC, takich jak serwer poczty e-mail, albo internetowy serwer pośredniczący. Najbardziej interesującym z perspektywy tematu rozprawy był scenariusz polegający na uruchomieniu aplikacji TeraGen (generowanie dużych danych testowych) w środowisku klastrowym złożonym z czterech węzłów, korzystającym z systemu plików HDFS. Rezultaty zależały głównie od konfiguracji liczby replik HDFS. System Tinca był w stanie zredukować czas działania aplikacji od 29% do nawet 60%. W obliczu podobieństw narzędzia Tinca do założeń autorskiego rozwiązania zaprezentowanych w rozprawie, warto wskazać fundamentalne mechanizmy, które je odróżniają. Po pierwsze, Tinca działa w obrębie pojedynczego węzła przy założeniu, że zainstalowanie pamięci NVRAM jest konieczne jedynie w serwerze obsługującym system plików. Założeniem rozwiązania zaproponowanego w rozprawie jest zaopatrzenie wszystkich węzłów obliczeniowych klastra w pamięć NVRAM i pozostawienie architektury systemu plików bez zmian. Oznacza to, że nic nie stoi na przeszkodzie, żeby zaproponowane rozwiązanie połączyć z mechanizmami takimi jak Tinca, instalowanymi na serwerach PFS. Drugą różnicą jest zorientowanie na aplikacje HPC korzystające ze standardu MPI I/O – możliwe jest, że system Tinca

MOST WIEDZY Downloaded from mostwiedzy.pl

byłby korzystny również w przypadku takich aplikacji. Niestety w chwili pisania pracy nie jest on publicznie dostępny, co uniemożliwia ewentualne dostosowanie i testy. Poza tym architektura rozwiązania autorskiego zaproponowana w rozprawie została opublikowana wcześniej (połowa roku 2016) niż Tinca (koniec roku 2017).

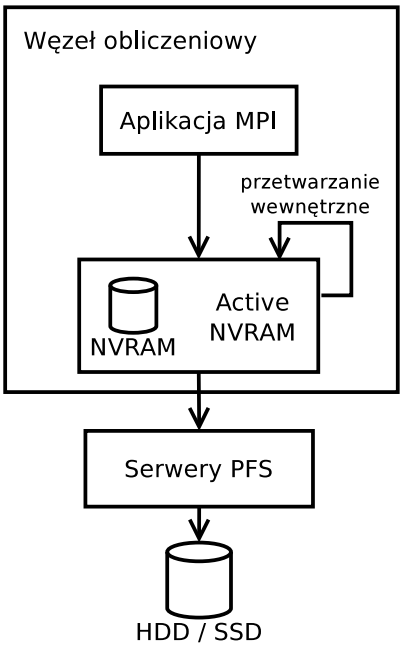


Rysunek 4.1: Wielowęzłowa architektura narzędzia CDBB

W 2017 roku zaprezentowano również rozwiązanie oparte o ideę *burst buffer* omówioną w poprzednim rozdziale. Autor narzędzia CDBB (*Collaborative Distributed Burst Buffer*) twierdzi, że standardowe implementacje *burst buffer* może cechować niski stopień wykorzystania zasobów, który nie był dotychczas problemem ze względu na niewysoką cenę pamięci opartej o SSD [23]. CDBB zakłada, że w pamięć NVRAM wyposażono każdy węzeł klastra, natomiast jej rozmiar jest stosunkowo mały. Rozwiązanie to jest dedykowane tworzeniu checkpointów i wprowadza dwa typy procesów, pomagające osiągnąć cel w sposób jak najbardziej wydajny: *CKPT writers* (procesy tworzące checkpoint), występujące w dużej liczbie i pojedynczy *BB coordinator* (koordynator buforów). Zadaniem *CKPT writer* jest zapis fragmentu danych w odpowiednim miejscu, uzgodnionym wcześniej z koordynatorem bufora. Koordynator natomiast przechowuje informacje o stanie wszystkich buforów i odpytany o docelową lokalizację, może udzielić jednej z trzech odpowiedzi: zlecić zapis w węźle lokalnym, w węźle zdalnym, albo bezpośrednio w PFS. Uproszczony schemat architektury rozwiązania zaprezentowano na rysunku 4.1. Podczas testów, gdzie zamiast pamięci NVRAM zastosowano zwykłą pamięć DRAM bez dodatkowych opóźnień, narzędzie potrafiło poprawić wydajność zapisywania stanów aplikacji nawet ośmiokrotnie. Niestety, autor nie zamieścił wykresów skalowalności, pojawiają się więc obawy, że pojedyncza instancja koordynatora może nie sprostać wzrastającemu obciążeniu. Pierwotnie

MOST WIEDZY Downloaded from mostwiedzy.pl

autorskie rozwiązanie również korzystało z pojedynczej jednostki zarządzającej, ale stosunkowo szybko stawała się ona wąskim gardłem całego systemu. Ponadto, przy checkpointingu żądania zapisu do pliku są zazwyczaj duże i ciągłe, co często niełatwo osiągnąć we wszystkich aplikacjach.

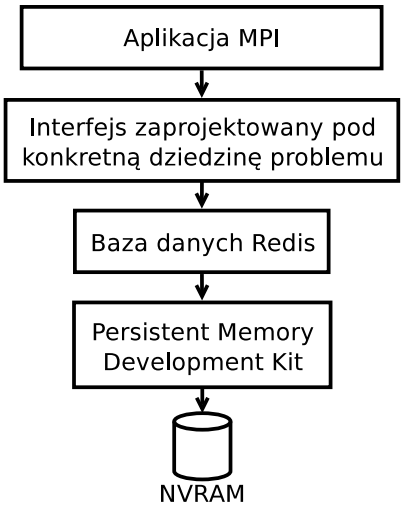


Rysunek 4.2: Rozwiązanie Active NVRAM, w którym oprócz buforowania żądań wychodzących z węzła, pojawia się również wewnętrzne przetwarzanie zgromadzonych danych

W literaturze można również spotkać hipotetyczne urządzenia, które łączą wsparcie dla operacji odczytu i zapisu danych do pliku z układem pozwalających na przetwarzanie tych danych. Active NVRAM to propozycja połączenia pamięci NVRAM z wbudowanym elementem obliczeniowym o niskim poborze mocy [49]. W praktyce, podczas testów, badacze wykorzystali symulację pamięci i pojedynczy rdzeń procesora o wydajności porównywalnej ze stosunkowo mało wydajnym rdzeniem Intel Atom. Eksperymentalna architektura składała się z szesnastowęzłowego klastra, każdy miał cztery rdzenie – trzy wykorzystane do typowych obliczeń, a jeden wchodził w skład symulowanego urządzenia Active NVRAM. Działanie systemu było następujące: kiedy jeden z procesów aplikacji MPI przeprowadzał zapis, oprócz danych przekazywał również zestaw meta-danych, definiujących sposób dalszego przetwarzania. Żądanie było obsługiwane przez Active NVRAM, który w pierwszej kolejności dbał o bezpieczne zapisanie danych w danym węźle, a następnie zarządzał żądaniami reorganizując je w taki sposób, żeby docelowa komunikacja z systemem plików

MOST WIEDZY Downloaded from mostwiedzy.pl

była jak najbardziej wydajna. Dodatkowo, metadane mogły zawierać instrukcje dalszego przetwarzania danych, w artykule pojawił się przykład sortowania zapisanych struktur albo checkpointingu. Rysunek 4.2 zawiera schemat prezentujący opisane podejście. Niestety, poprawa wydajności była możliwa jedynie w pewnych warunkach (wysoka częstotliwość zapisu czy duży rozmiar żądań), w artykule nie omówiono kwestii bezpieczeństwa danych w sytuacji awarii sprzętowej, a architektura wymaga zastosowania dodatkowego urządzenia Active NVRAM (o którym nie ma żadnych potencjalnych informacji w kwestii planów wprowadzenia produktu na rynek). Z drugiej strony, idea wydzielenia pojedynczego rdzenia do operacji związanych z obsługą pamięci NVRAM, obecna w symulatorze Active NVRAM, wydaje się oferować ciekawe możliwości, m.in. ograniczenie wpływu obciążenia generowanego przez aplikację na obsługę struktur w pamięci. Przy maszynach takich jak te wykorzystane w zaprezentowanych testach oznaczałoby to 25% niższą wydajność (jeden rdzeń z czterech przeznaczony na obsługę NVRAM), ale przy nowoczesnych stacjach serwerowych wyposażonych w co najmniej dwa wielordzeniowe procesory zdecydowano się wykorzystać pomysł w autorskim rozwiązaniu.



Rysunek 4.3: Stos technologiczny zaproponowany przez autora rozprawy, który zmienia klasyczne podejście związane z przetwarzaniem plików na wykorzystanie szybkiej bazy danych typu klucz-wartość, dostosowanej do pamięci NVRAM

Operacje wejścia/wyjścia w klastrach niekoniecznie muszą wiązać się z przetwarzaniem plików. Na fali popularności szybkich nierelacyjnych baz danych (NoSQL), wielokrotnie weryfikowano przydatność ich reprezentantów, takich jak bazy typu klucz-wartość (ang. *key-value*) w obliczeniach HPC [15]. Istnieją też systemy, które zostały zaprojektowane

MOST WIEDZY Downloaded from mostwiedzy.pl

z myślą o nowych rodzajach pamięci, na przykład rozwiązanie PapyrusKV [51]. PapyrusKV jest bazą stworzoną z myślą o obliczeniach HPC, napisaną przy użyciu standardu MPI, a jednym z jej założeń jest dostęp do dużej ilości pamięci NVRAM rozproszonej w ramach klastra. Tak jak inne systemy oparte o NVRAM, baza ta umożliwia zapisywanie kolejnych stanów aplikacji i dostarcza mechanizmy odtwarzania ich stanu. Innym przykładem użycia pamięci NVRAM w obliczeniach HPC może być architektura oparta o rozszerzenie do popularnej bazy Redis, pokazana na rysunku 4.3, zaproponowana przez autora rozprawy [70].

Tabela 4.1: Porównanie rozwiązań opartych o NVRAM dedykowanych operacjom I/O w HPC

Rozwiązanie	Główna idea	Zysk wydajnościowy	Ograniczenia
Zastąpienie pamięci masowej	NVRAM jako pamięć masowa	$< 10\%$	brak
Tinca	pamięć podręczna NVRAM w PFS	~ 30-60%	nie testowane dla MPI I/O
Active NVRAM	urządzenie łączące NVRAM i CPU	~ 0-380%	brak informacji o planach wdrożenia urządzenia
PapyrusKV, Redis	baza typu klucz-wartość	–	konieczność zmiany operacji plikowych na bazodanowe
CDBB	*burst buffer*	–	zysk wydajnościowy zmierzony jedynie dla checkpointów, potencjalne problemy ze skalowalnością

W tabeli 4.1 porównano opisane rozwiązania. Wyraźnie widać, że użycie NVRAM jako pamięci masowej jest możliwe, natomiast niezbyt opłacalne. Zmiana operacji plikowych na bazodanowe nie zawsze jest możliwa i wiąże się z dodatkowym narzutem programistycznym, a kilkusetprocentowe zyski wydajnościowe udało się uzyskać jedynie przy użyciu dedykowanego urządzenia. Za najbardziej obiecujące podejście uznano te związane z zaprojektowaniem nowych rozwiązań dostosowanych do specyfiki pamięci NVRAM ogólnego przeznaczenia, jak na przykład projekt Tinca.

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 5

Rozproszona pamięć podręczna dla aplikacji wykorzystujących MPI I/O

5.1 Definicja problemu

Przyjmijmy definicję czasu działania aplikacji ET (ang. *execution time*) jako pewną funkcję sprzętu HW (ang. *hardware*) na którym uruchomiono określone oprogramowanie SW (ang. *software*):

$$ET(HW, SW)$$

Należy przy tym zaznaczyć, że współcześnie bardzo trudno precyzyjnie wyznaczyć taką funkcję ze względu na zależność wydajności sprzętu i oprogramowania od wielu zazwyczaj pomijanych czynników, takich jak na przykład temperatura sprzętu, oprogramowanie sprzętowe optymalizujące zużycie energii, realizacja nieistotnych z perspektywy aplikacji zadań systemu operacyjnego czy różnice w kolejności wykonywanych rozkazów w środowisku równoległym.

W przypadku środowiska klastrowego, warstwę sprzętową można zamodelować jako graf, w którym krawędzie oznaczają łącza komunikacyjne L (ang. *communication links*), natomiast wierzchołki są zbiorem węzłów N (ang. *node*) – węzłów obliczeniowych, węzłów odpowiedzialnych za przechowywanie danych lub innych, takich jak przełączniki sieciowe (ang. *switch*):

$$HW = \langle N, L \rangle$$

W powyższym i kolejnych zapisach nawiasy ostrokątne „$\langle$” i „$\rangle$” oznaczają zbiór, natomiast nawiasy okrągłe „(” i „)” – sekwencję. Opisując węzeł klastra przyjęto, że największy wpływ na wydajność węzła mają jednostki przetwarzające PU (ang. *processing unit*), takie jak procesory czy akceleratory obliczeń, hierarchia pamięci rozumiana jako charakterystyka kolejnych poziomów pamięci M (ang. *memory*), oraz ich połączenia I

MOST WIEDZY Downloaded from mostwiedzy.pl

(ang. *interconnect*) realizowane przeważnie przez magistrale komunikacyjne:

$$n_i = \langle PU_i, M_i, I_i \rangle$$

gdzie: n_i – i-ty węzeł klastra

PU_i – podzbiór jednostek PU zainstalowany w i-tym węźle

M_i – podzbiór pamięci M zainstalowany w i-tym węźle

I_i – podzbiór magistrali I zainstalowany w i-tym węźle

Kluczowym parametrem jednostki przetwarzającej w kontekście czasu działania aplikacji jest moc obliczeniowa. W obliczeniach wysokiej wydajności określa się ją przeważnie w jednostce FLOPS, oznaczającej liczbę operacji zmiennoprzecinkowych, które urządzenie jest w stanie wykonać w ciągu sekundy. W przypadku pamięci, najważniejsze atrybuty to szybkość przesyłania danych oraz opóźnienie operacji zapisu i odczytu.

Jeśli chodzi o oprogramowanie, istotnymi elementami są: system operacyjny OS (ang. *operating system*), oprogramowanie pośredniczące MW (ang. *middleware*) oraz aplikacja APP (ang. *application*) korzystająca z API systemu operacyjnego i bibliotek pośredniczących:

$$SW = \langle OS, MW, APP \rangle$$

W kontekście rozprawy takim oprogramowaniem pośredniczącym są biblioteki standardu MPI, jak również proponowane rozwiązanie. Oprogramowanie pośredniczące w tradycyjnym stosie technologicznym bez wsparcia pamięci NVRAM będzie dalej oznaczane jako $MW_{\cancel{NVRAM}}$, podczas gdy oprogramowanie pośredniczące z elementami opartymi o pamięć NVRAM – MW_{NVRAM}.

Aplikacja w HPC jest natomiast pewnym zbiorem procesów P (ang. *process*), które – w ramach jednego lub więcej wątków ze zbioru T (ang. *thread*) wykonują sekwencje operacji ze zbioru możliwych operacji O (ang. *operation*). Takie operacje mogą polegać na przeprowadzaniu obliczeń (operacje ze zbioru oznaczonego jako CMP, ang. *computation*), komunikacji pomiędzy wątkami lub procesami (zbiór COM, ang. *communication*) oraz operacji wykonywanych na pamięci danego węzła, a w szczególności na plikach (zbiór MOP, ang. *memory operations*):

$$APP = \langle p_1, p_2, ..., p_j \rangle, p_k \in P$$

$$p_k = \langle t_1^k, t_2^k, ..., t_j^k \rangle, t_m^k \in T$$

$$t_m^k = (o_1^{km}, o_2^{km}, ..., o_l^{km}), o^{km} \in CMP \cup COM \cup MOP$$

gdzie: p_k – k-ty proces ze zbioru procesów P

t_m^k – m-ty wątek ze zbioru wątków T działający w ramach k-tego procesu

MOST WIEDZY Downloaded from mostwiedzy.pl

o_l^{km} – l-ta operacja m-tego wątku w ramach k-tego procesu

Korzystając z powyższego modelu, tezę rozprawy możemy zapisać następująco: istnieje takie oprogramowanie pośredniczące, które dodane do konfiguracji klastra wraz z dodatkową pamięcią NVRAM, pozwala zredukować czas działania wybranych aplikacji równoległych:

$$\exists_{SW_{\cancel{NVRAM}}} \exists_{HW_{\cancel{NVRAM}}} \exists_{MW_{NVRAM}} ET(HW_{\cancel{NVRAM}}, SW_{\cancel{NVRAM}}) > ET(HW_{NVRAM}, SW_{NVRAM}) :$$

$$HW_{\cancel{NVRAM}} = \langle N_{\cancel{NVRAM}}, L\rangle : \forall_{n_i^{\cancel{NVRAM}} \in N_{\cancel{NVRAM}}} n_i^{\cancel{NVRAM}} = \langle PU_i, M_i, I_i\rangle \wedge \{NVRAM\} \notin M_i$$

$$HW_{NVRAM} = \langle N_{NVRAM}, L\rangle : \forall_{n_i^{NVRAM} \in N_{NVRAM}} n_i^{NVRAM} = \langle PU_i, M_i \cup \{NVRAM\}, I_i\rangle$$

$$SW_{\cancel{NVRAM}} = \langle OS, c, APP\rangle$$

$$SW_{NVRAM} = \langle OS, MW_{NVRAM}, APP\rangle$$

5.2 Projekt rozwiązania

Najbardziej typową drogę przetwarzania żądań dostępu do pliku w aplikacji stworzonej w standardzie MPI prezentuje poniższa lista:

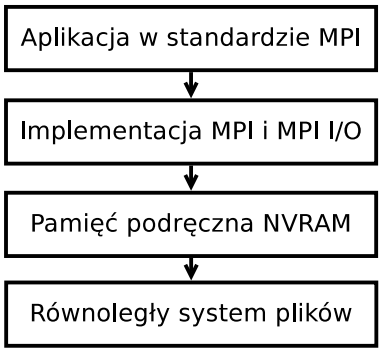
1. Program woła API MPI.
2. Żądanie realizowane jest przez implementację MPI I/O.
3. Implementacja MPI I/O optymalizuje pojedyncze żądanie lub grupę żądań.
4. Żądanie kierowane jest do systemu plików.

Konkretny stos technologiczny w takiej aplikacji będzie więc zawierał implementację MPI, implementację MPI I/O, będącą częścią implementacji MPI (obecnie w praktyce dla większości przypadków ROMIO[1]) oraz rozproszony system plików (np. OrangeFS[2], Lustre[3]). W celu pozostawienia elastycznej możliwości wyboru elementów składowych stosu technologicznego, proponowane rozwiązanie tworzy następną warstwę, umiejscowioną w kolejności przetwarzania pomiędzy aplikacją napisaną w standardzie MPI, a implementacją

[1] `http://www.mcs.anl.gov/projects/romio/`
[2] `http://www.orangefs.org/`
[3] `http://lustre.org/`

MOST WIEDZY Downloaded from mostwiedzy.pl

MPI I/O. Rysunek 5.1 ilustruje zależności pomiędzy komponentami. Wspomniana warstwa przechwytuje żądania kierowane z aplikacji do pliku przez funkcje MPI I/O, a następnie realizuje je, sięgając uprzednio do pamięci podręcznej umiejscowionej w pamięci NVRAM. Umiejscowienie rozwiązania w dodatkowej warstwie pozwala na jego stosowanie jednocześnie z innymi narzędziami optymalizującymi, na przykład opisanymi w cytowanej literaturze.



Rysunek 5.1: Ilustracja pozycji zaproponowanej pamięci podręcznej w kontekście stosu technologicznego aplikacji pracującej na plikach

Projektując rozwiązanie, należało przyjąć pewne założenia, głównie ze względu na brak gotowych układów opartych o pamięci NVRAM na rynku. Oczekiwane właściwości takich pamięci opisano w rozdziale 2, a dokładne szacunki wartości ich parametrów są przedmiotem rozdziału 6, w którym na początku szeroko omówiono konfigurację środowiska testowego. Jak już wspomniano wcześniej, założono również, że w nowe układy pamięci zostaną wyposażone wszystkie węzły wchodzące w skład klastra obliczeniowego. Biorąc pod uwagę rozmiary używanych plików, oczekiwane rozmiary pamięci NVRAM oraz liczbę węzłów w typowym klastrze HPC, uzasadnionym wydaje się również przyjęcie, że rozmiar pliku jest mniejszy od sumarycznego rozmiaru pamięci NVRAM w klastrze obliczeniowym. Dodatkowo, zakłada się również, że wszystkie węzły klastra są połączone ze sobą siecią Infiniband lub siecią 10 Gigabit Ethernet.

Tworzenie struktur pamięci podręcznej rozpoczyna się w chwili otwarcia pliku. Cały plik jest podzielony na n ciągłych fragmentów, gdzie n jest liczbą węzłów w klastrze. Każdą częścią pliku zarządza jednostka nazwana zarządcą pamięci podręcznej (ang. *cache manager*). Jednostka ta jest częścią aplikacji MPI, podczas instalacji tworzony jest pojedynczy zarządca w ramach każdego węzła. Po uruchomieniu zarządcy rezerwuje on odpowiednią ilość pamięci NVRAM i, przy użyciu funkcji MPI I/O, zaczytuje przyporządkowany mu

MOST WIEDZY Downloaded from mostwiedzy.pl

fragment pliku. Następnie oczekuje na kolejne żądania i wykonuje je synchronicznie w kolejności odbioru. Jedną z operacji realizowanych przez zarządcę jest żądanie zamknięcia pliku. Podczas obsługi operacji tego typu cały fragment pliku obsługiwany przez zarządcę zapisywany jest w systemie plików – poprzednie dane są nadpisywane. Po zakończonym zapisie zwalniana jest pamięć, a jednostka kończy swoje działanie.

Uruchomienie pojedynczego zarządcy na każdym z węzłów wiąże się z pewnym obciążeniem klastra. W przypadku klastrów złożonych z węzłów wyposażonych w procesory starszej generacji mogło to oznaczać znaczne obniżenie zasobów dostępnych dla aplikacji. Obecnie, kiedy w HPC powszechne jest stosowanie w jednej maszynie co najmniej dwóch procesorów wielordzeniowych, dodatkowe zużycie zasobów nie będzie aż tak odczuwalne. Przykładowo, testy rozwiązania wykonywano na klastrze wyposażonym w dwa ośmiordzeniowe procesory, więc w przypadku, kiedy jeden z nich był wyłącznie odpowiedzialny za zarządzanie rozwiązaniem, dostępna moc obliczeniowa spadła maksymalnie o około 6%.

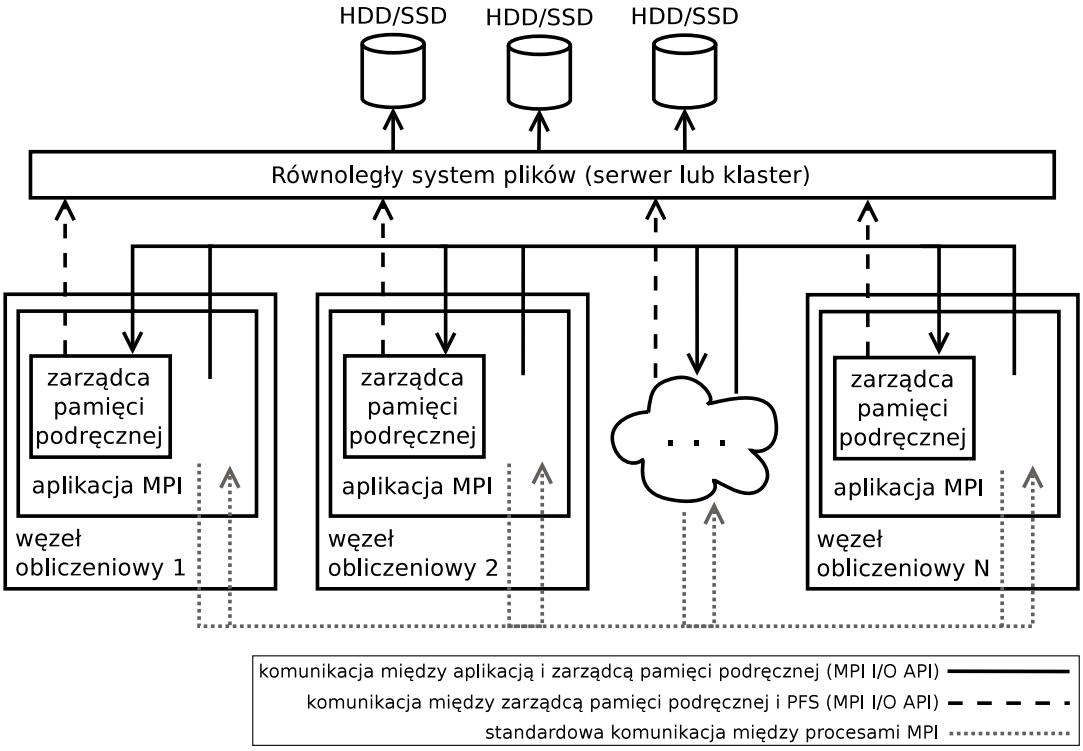
Kiedy procesy aplikacji chcą zrealizować operację dostępu do pliku, muszą skomunikować się z właściwym zarządcą pamięci podręcznej. Ze względu na deterministyczny przydział fragmentów plików do zarządców pamięci, aplikacja ma precyzyjne informacje w kwestii zarządcy odpowiedzialnego za dane, do których proces potrzebuje dostępu, więc zgłasza żądanie dokładnie do tego adresata, który je obsłuży. Istnieje pewne prawdopodobieństwo, że proces będzie zainteresowany fragmentem, którego części są obsługiwane przez kilku zarządców. W takim przypadku konieczne jest wysłanie większej liczby zapytań. Warto również zaznaczyć, że procesy aplikacji nie mają możliwości bezpośredniej komunikacji z systemem plików – wszystkie żądania muszą być obsłużone przez pamięć podręczną. Oznacza to, że z perspektywy aplikacji nie jest konieczne stałe utrzymywanie aktualnego stanu pliku w PFS.

W domyślnym trybie pracy MPI nie daje gwarancji nie tylko na spójność jednoczesnych zapisów, ale również na natychmiastową widoczność zapisywanych danych z poziomu aplikacji. O ile programiści aplikacji współbieżnych są zazwyczaj świadomi, że typowe operacje nie są atomowe i jednoczesny zapis do współdzielonego obszaru może skutkować niespójnością danych, o tyle brak widoczności zapisanych danych nawet po upewnieniu się, że odczyt jest poprzedzony synchronicznym zapisem zakończonym powodzeniem, może wydawać się nieintuicyjny. Pewnym rozwiązaniem jest jawne wywołanie funkcji synchronizującej, w przypadku niektórych aplikacji mogłoby to jednak oznaczać konieczność wywoływania jej przed większością operacji odczytu. Zdając sobie sprawę z opisanych ograniczeń, twórcy

MOST WIEDZY Downloaded from mostwiedzy.pl

MPI stworzyli opcjonalny tryb „atomic”, który pozwala na ich wyeliminowanie. Niestety wiąże się to zazwyczaj z obniżoną wydajnością operacji na plikach [60]. Dzięki temu, że fragmenty plików obsługiwane przez zarządców są rozłączne, zaproponowana pamięć podręczna gwarantuje pełną widoczność (z poziomu wszystkich procesów wszystkich węzłów) zmian wprowadzonych w pliku. Oznacza to brak konieczności wywoływania funkcji synchronizującej przed odczytem – jedynym zadaniem funkcji synchronizującej staje się więc uspójnienie stanu pamięci podręcznej z rozproszonym systemem plików. Ze względu na rozproszoną architekturę, wsparcie atomowości pojedynczych operacji nie jest dostępne w domyślnym trybie pracy.

Architektura rozwiązania przedstawiona jest na rysunku 5.2 i – z perspektywy konkretnego węzła – na rysunku 5.3.

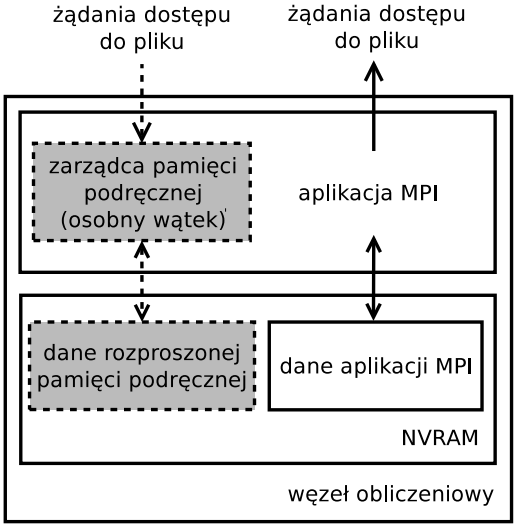


Rysunek 5.2: Schemat architektury rozwiązania. W celu zwiększenia czytelności nie uwzględniono konkretnych procesów MPI

Poniżej zaprezentowano główne cechy, które odróżniają zaproponowaną pamięć podręczną od typowych rozwiązań.

- Cały plik jest wczytywany do pamięci podręcznej. Jest to możliwe dzięki oczekiwanemu rozmiarowi pamięci urządzeń opartych o NVRAM. Rezultatem jest brak

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 5.3: Schemat architektury rozwiązania w obrębie pojedynczego węzła. Szare bloki oraz przerywane linie są przezroczyste z perspektywy programisty aplikacji

sytuacji, w której żądanie trafiłoby na jeszcze nie zaczytany fragment (ang. *cache miss*);

- Zmniejszono częstotliwość synchronizacji danych pomiędzy pamięcią podręczną a systemem plików. Wywołanie operacji synchronizacji jest często związane z obawą o utratę danych. W zaproponowanym rozwiązaniu bezpieczeństwo danych jest zwiększone ze względu na trwałość zapisu użytej pamięci, a także dzięki możliwości utworzenia redundantnej kopii danych, opisanej w dalszej części rozdziału;

- Zredukowano metadane do minimum. W blokowej organizacji pamięci podręcznej każdy blok jest zaopatrzony w szereg flag, mówiących między innymi o tym, czy blok był odczytywany i zapisywany. Opisywane rozszerzenie, wpierane przez bajtowe adresowanie pamięci, nie tworzy bloków (albo, inaczej mówiąc, tworzy jeden duży blok na każdy węzeł), co pozwala również na przesyłanie tylko tych danych, których dotyczy żądanie;

- Wyeliminowano potrzebę użycia centralnej jednostki zarządzającej pamięcią podręczną. Dzięki temu, że każdy zarządca jest odpowiedzialny za konkretny, znany fragment pliku, komunikacja pomiędzy zarządcami jest również zredukowana do minimum. Jest to szczególnie istotne w przypadku obliczeń HPC, gdzie dużą wagę przykłada się do skalowalności rozwiązania;

MOST WIEDZY Downloaded from mostwiedzy.pl

- Zoptymalizowano sposób komunikacji z rozproszonym systemem plików. Zgodnie z opisem zamieszczonym we wstępie, najbardziej wydajnym sposobem zapisu/odczytu pliku jest dostęp do jego dużych i ciągłych fragmentów. Dokładnie takie żądania są przesyłane podczas otwierania, zamykania i synchronizacji pamięci podręcznej z PFS. Dodatkowo, nagły wzrost intensywności żądań generowany w tych trzech scenariuszach idealnie odpowiada założeniom mechanizmów typu burst buffer, które potencjalnie powinno dobrze współpracować z zaproponowanym rozwiązaniem;
- Zrezygnowano z optymalizacji wprowadzających dodatkowe opóźnienia. Wiele rozwiązań zawiera dodatkowe optymalizacje dostępu do danych, takie jak zbieranie kolejnych żądań w celu złożenia ich w pojedyncze, większe. Przy minimalnych metadanych, ciągłej organizacji pamięci podręcznej i szybkiej sieci łączącej węzły klastra, zdecydowano się porzucić optymalizacje i obsługiwać kolejne żądania tak szybko, jak to tylko możliwe.

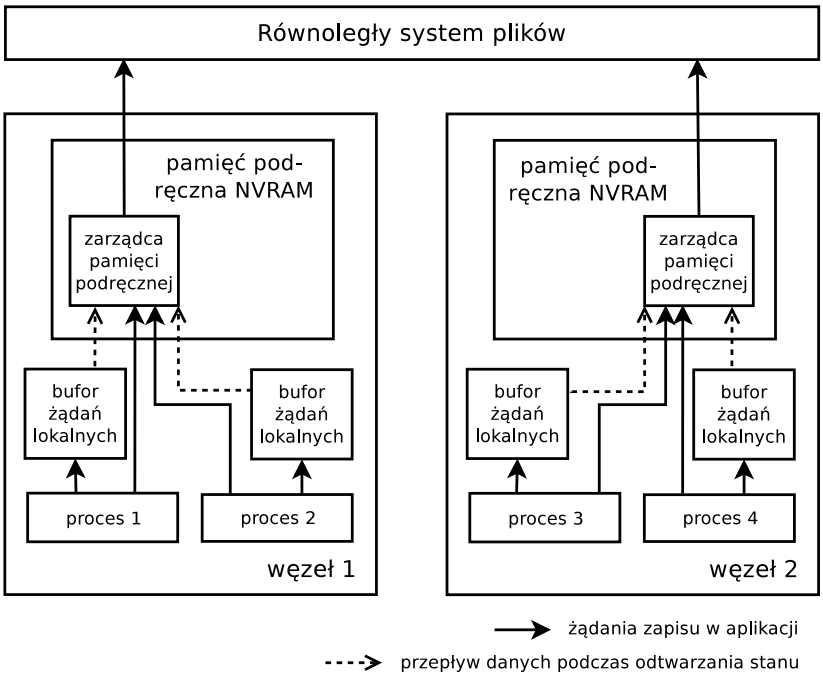
Pierwszy projekt rozwiązania z podstawową implementacją i ewaluacją wydajności został omówiony w artykule [77].

5.3 Praca w środowisku podatnym na awarie

Projektując rozwiązania dedykowane środowiskom klastrowym, w których występuje stosunkowo wysokie prawdopodobieństwo awarii, należy rozważyć zagadnienie współpracy takiego rozwiązania z mechanizmami zapobiegającymi utracie danych. Jest to szczególnie ważne w kontekście negatywnego wpływu na wydajność, który może się pojawić, jeśli przetwarzanie nie zostanie odpowiednio zoptymalizowane – szacuje się, że przy obecnym rozmiarze klastrów obliczeniowych narzut związany z zapewnieniem bezpieczeństwa danych może wynieść nawet ponad 60% czasu działania aplikacji [92]. W autorskim rozwiązaniu mechanizm odpowiedzialny za ochronę przed utratą danych zdecydowano się wbudować w struktury pamięci podręcznej. Wykorzystując trwałość zapisu pamięci NVRAM, rozwiązanie oferuje trzy poziomy bezpieczeństwa danych, opisane szerzej w artykułach [78, 72]. Każdy z poziomów wiąże się z pewnym narzutem czasowym, można je natomiast osobno konfigurować zgodnie z wymaganiami dla danej aplikacji. W bieżącym podrozdziale przedstawiono sposób działania mechanizmów. Ich wpływ na wydajność jest natomiast zbadany w rozdziale 6 zawierającym testy rozwiązania.

Na pierwszym poziomie rozwiązanie zapewnia spójność pliku z dokładnością do poje-

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 5.4: Przepływ danych w rozwiązaniu zapewniającym spójność pamięci podręcznej. Rysunek pokazuje także, że w przypadku awarii odtwarzana jest jedynie pamięć podręczna – ewentualny zapis odzyskanych danych w PFS musi być jawnie wywołany z aplikacji

dynczego żądania. W typowej aplikacji, jeśli awaria nastąpi podczas trwania procedury zapisu, może wystąpić sytuacja, w której w pliku znajdzie się jedynie część zapisywanych danych. Prezentowana biblioteka dostarcza funkcji naprawczych, które potrafią zagwarantować, że poprawnie zainicjalizowany zapis trafi w całości do przetwarzanego pliku:

- `recovery_data_create(data, offset, size, cache_path)` – tworzy kopię zapasową danych przed zapisem wraz z informacją o ich rozmiarze i pozycji w pliku (funkcja wykorzystywana głównie wewnętrznie);
- `recovery_data_remove(recovery_data_file_name)` – usuwa kopię zapasową po potwierdzeniu poprawnego zapisu; identyfikatorem kopi zapasowej jest nazwa pliku zwracana przez funkcję tworzącą kopię zapasową (funkcja wykorzystywana głównie wewnętrznie);
- `perform_data_recovery(file_handler, cache_path)` – pozwala odtworzyć spójny stan pliku na podstawie uchwytu do pliku i lokalizacji danych pamięci podręcznej (konieczne jawne wywołanie przez programistę w aplikacji).

MOST WIEDZY Downloaded from mostwiedzy.pl

Sam mechanizm działa zgodnie z poniższym algorytmem.

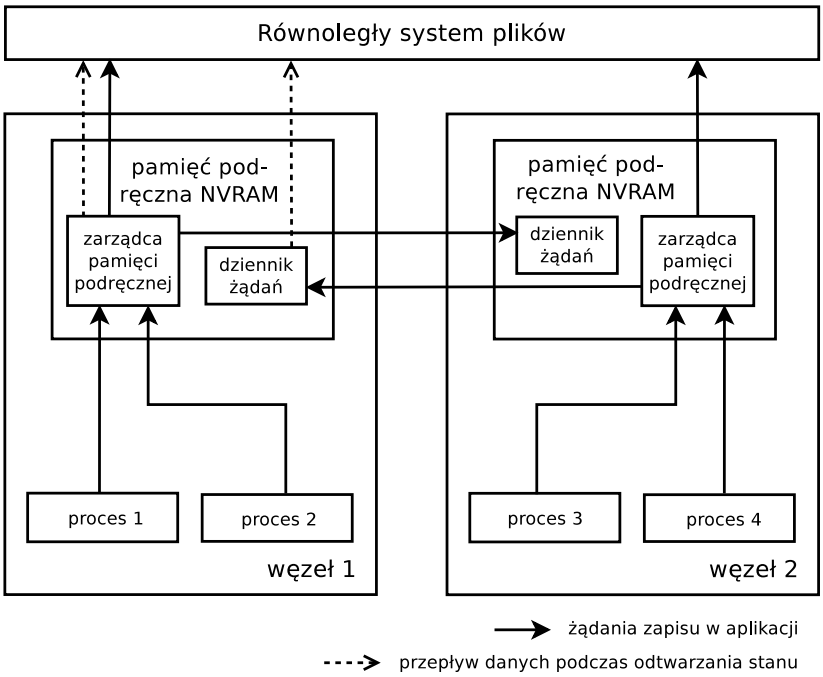
1. Wywołanie funkcji zapisu z perspektywy konkretnego procesu rozpoczyna się trwałym zapisem żądania w lokalnej pamięci NVRAM.
2. W kolejnym kroku żądanie przesyłane jest do jednego bądź kilku zarządców pamięci podręcznej i tam dalej przetwarzane.
3. Jeśli zapis żądania zakończył się powodzeniem, proces usuwa lokalną kopię żądania.
4. Wystąpienie awarii podczas zapisu spowoduje pozostawienie kopii żądania na węźle rozpoczynającym zapis. W takim przypadku, podczas odtwarzania pliku, na dane z pamięci podręcznej nakładane są wszystkie rozpoczęte, natomiast jeszcze nie ukończone żądania.

Na tym poziomie dane aplikacji chronione są w przypadku awarii spowodowanych błędem programistycznym, problemami z siecią czy środowiskiem. W przypadku awarii całego węzła, a w szczególności układu pamięci NVRAM, dane zostaną bezpowrotnie utracone. Mechanizm zachowania spójności pliku jest więc kierowany do małych klastrów, gdzie ryzyko awarii sprzętowej jest stosunkowo niskie. Architektura mechanizmu została pokazana na rysunku 5.4.

Powyższej wady pozbawiony jest drugi poziom, w którym kopia każdego żądania przetwarzanego przez zarządcę pamięci podręcznej jest także przechowywana w pamięci NVRAM sąsiedniego węzła lub węzłów. W tym trybie każdy węzeł uruchamia jednostkę odpowiedzialną za prowadzenie dziennika żądań zapisu, trzymanego w pamięci NVRAM. Żądania przechowywane są w dzienniku aż do synchronizacji pamięci podręcznej z systemem plików. W przypadku awarii, tak jak w rozwiązaniu opisanym w poprzednim akapicie, możliwe jest odtworzenie spójnego stanu pliku. Tym razem jednak dane odporne są także na awarie konkretnego węzła lub węzłów, jeśli zdecydowano się przechowywać więcej niż jedną redundantną kopię. Rysunek 5.5 ilustruje drugi poziom bezpieczeństwa danych w autorskim rozwiązaniu. Ze względu na ograniczenie pamięci NVRAM w ramach konkretnego węzła, w tym trybie konieczna jest cykliczna synchronizacja z systemem plików, która powoduje zwolnienie zajętej pamięci.

Trzeci poziom związany jest z działaniem funkcji synchronizującej. Synchronizacja danych pomiędzy aplikacją a systemem plików nie jest wyłącznie sposobem przekazywania informacji pomiędzy procesami. Drugim powodem synchronizacji jest obawa o utratę

MOST WIEDZY Downloaded from mostwiedzy.pl



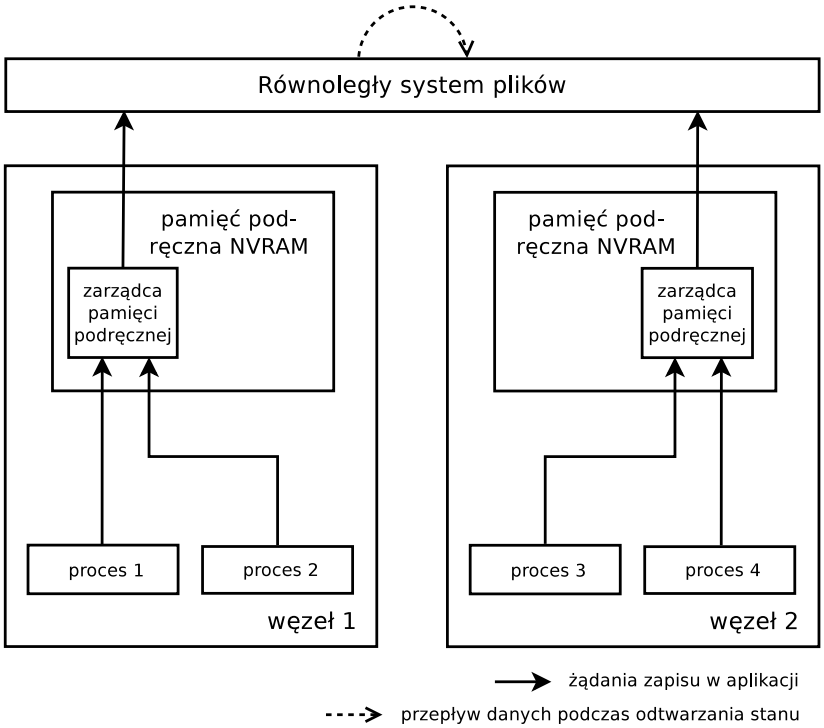
Rysunek 5.5: Przepływ danych w mechanizmie dziennika żądań zapisu. Na rysunku założono, że awaria wystąpiła na drugim węźle i nie był on w stanie brać udziału w dalszym przetwarzaniu

danych przechowywanych w pojedynczym węźle – równoległe systemy plików zazwyczaj stworzone są w taki sposób, żeby chronić dane przed awariami. W standardzie MPI dane należy uznać za trwale zapisane w pamięci urządzenia po zakończeniu działania funkcji `MPI_File_sync`, wywołanej ze wszystkich procesów, które otworzyły dany plik. Ważne jest również to, że sam fakt zapisu do pliku z poziomu pojedynczego procesu nie oznacza, że zapisane dane będą także widoczne dla pozostałych procesów. W tym celu również należy wywołać funkcję synchronizującą.

Podczas takiej synchronizacji cała zawartość pliku jest domyślnie nadpisywana danymi z pamięci podręcznej. Drobna modyfikacja wystarcza więc, żeby nadpisywanie poprzedniej wersji pliku zastąpić tworzeniem kolejnej wersji. W ten sposób praca z plikiem stanie się wersjonowana, a w przypadku awarii możliwe będzie sięgnięcie do ostatniej wersji pliku bez konieczności odtwarzania go z pamięci podręcznej. Schemat mechanizmu jest pokazany na rysunku 5.6.

Możliwości, podobieństwa i różnice opisanych poziomów zebrano w tabeli 5.1. Zaprezentowane tryby zapewnienia bezpieczeństwa danych mogą być wykorzystane na dwa sposoby – albo jako zapewnienie spójności z funkcją wersjonowania przetwarzanych plików,

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 5.6: Przepływ danych podczas synchronizacji pamięci podręcznej z systemem plików. Podczas awarii, odtwarzany jest ostatni spójny stan aplikacji zapisany w PFS

albo jako alternatywa dla checkpointingu. Analogicznie do magazynów obiektów trwałych opisanych w rozdziale 4, jeśli w pliku przechowywany byłby cały stan aplikacji, autorskie rozwiązanie mogłoby w wystarczającym stopniu zastąpić mechanizm checkpointingu. W przypadku niektórych technik tworzenia aplikacji, między innymi popularnych master-slave czy SPMD, takie podejście jest stosunkowo łatwe do zaimplementowania ze względu na prostą możliwość podziału działania programu na iteracje. Dla bardziej skomplikowanych metod, takich jak metoda dziel i zwyciężaj, w której rekurencyjny podział problemu może stworzyć bardzo nieregularną strukturę procesów aplikacji, całkowite zastąpienie checkpointingu przez proponowane rozwiązanie może wymagać złożonej implementacji i nie jest zalecane.

MOST WIEDZY Downloaded from mostwiedzy.pl

Tabela 5.1: Różnice trzech poziomów zapewnienia bezpieczeństwa danych w zaproponowanym rozwiązaniu

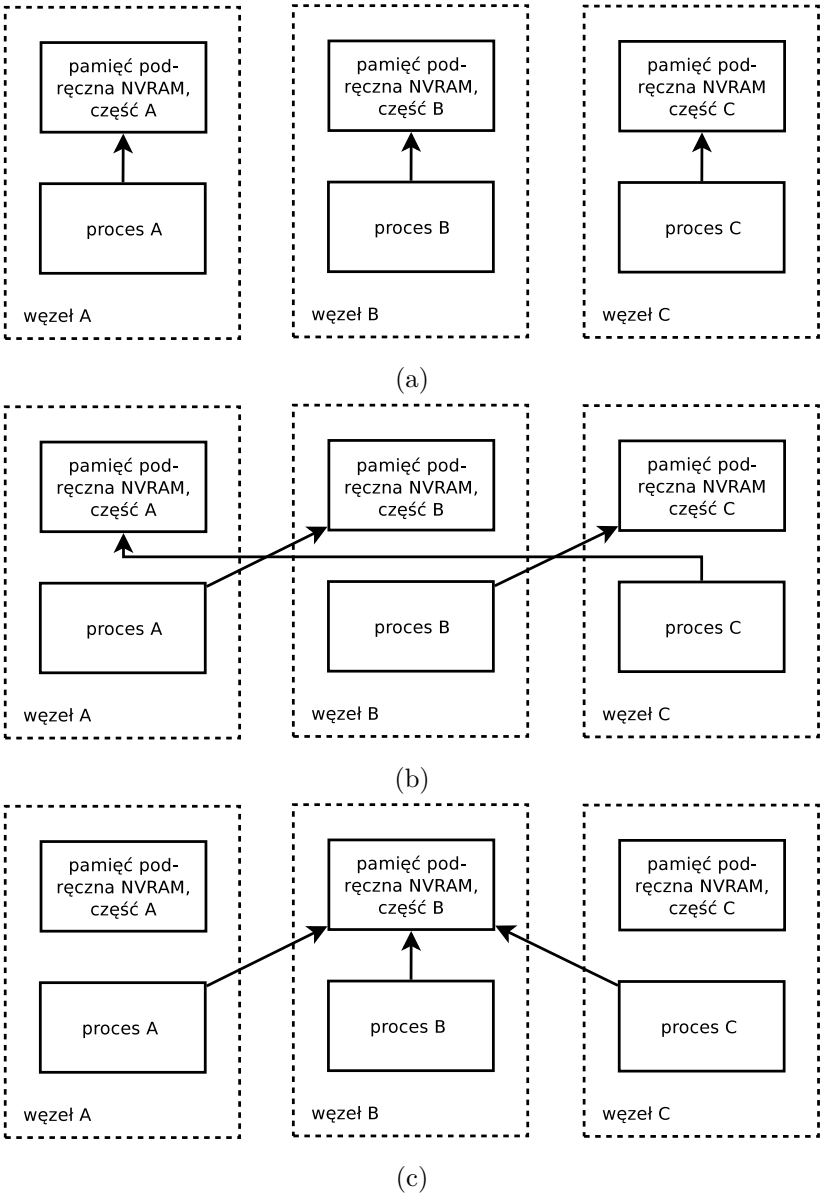
	Spójność w obrębie węzła	Dziennik żądań zapisu	Checkpointing przy synchronizacji
Narzut czasowy	pomijalny	niewielki, ale zauważalny	zależny od rozmiaru pliku i częstotliwości synchronizacji
Odporność na uszkodzenia	wszystkie usterki poza sprzętowymi	wszystkie usterki	wszystkie usterki
Czas odtwarzania	pomijalny	długi	długi
Dodatkowe wymagania	brak	częsta synchronizacja pliku	brak
Wersjonowanie	nie	nie	tak

5.4 Monitorowanie pamięci podręcznej

Przy zaproponowanej architekturze rozwiązania duży wpływ na wydajność będzie miało przyporządkowanie procesów do konkretnych węzłów, lokalizacja fragmentów pliku przetwarzanych przez procesy działające na danych węzłach, a także zależności czasowe pomiędzy żądaniami kierowanymi do konkretnych zarządców pamięci podręcznej. Najlepszą wydajność można osiągnąć przy redukcji komunikacji międzywęzłowej (komunikacja w ramach jednego węzła odbywa się przez pamięć współdzieloną, do komunikacji międzywęzłowej wykorzystywana jest sieć), należy przy tym także unikać nadmiernego przeciążania pojedynczych węzłów. Rysunek 5.7 ilustruje ten problem. W wariancie pokazanym na rysunku 5.7a mamy do czynienia z optymistycznym przypadkiem, w którym każdy z procesów współpracuje z lokalnym zarządcą pamięci podręcznej. Przykład zilustrowany przez rysunek 5.7b pokazuje sytuację w której żądania trafiają do zarządcy zlokalizowanego na innym węźle. W przeciętnym przypadku należy spodziewać się właśnie takiej sytuacji, ponieważ przy rosnącym rozmiarze klastra, szanse na to, że interesujące nas dane znajdują się na tym samym węźle maleją. Wariant pokazany na rysunku 5.7c to przypadek pesymistyczny, w którym dane pożądane przez wszystkie procesy zlokalizowane są w jednym węźle. Taka sytuacja może prowadzić do dużego obciążenia pojedynczego węzła i

MOST WIEDZY Downloaded from mostwiedzy.pl

upośledzenia wydajności całej aplikacji. Z drugiej strony, przy rosnącej liczbie węzłów w klastrze, maleje rozmiar fragmentu pliku obsługiwanego przed danego zarządcę, można więc przyjąć, że taka sytuacji jest specyficzna dla konkretnych aplikacji.



Rysunek 5.7: Trzy wersje tej samej aplikacji z różnym schematem dostępu do pliku

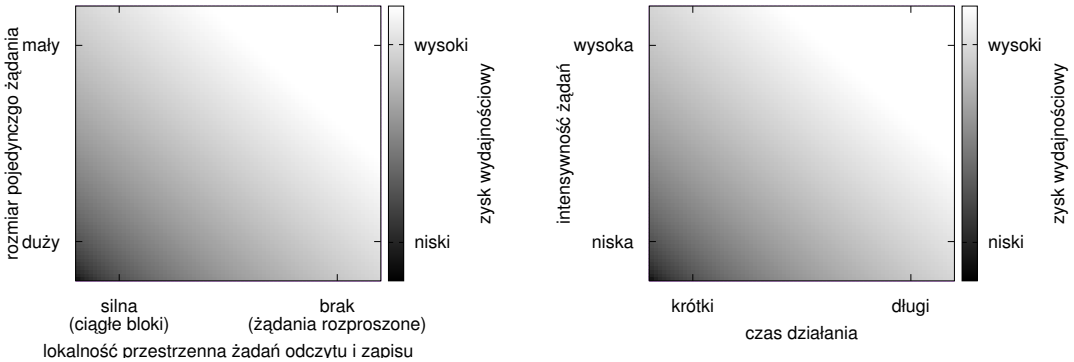
W celu ułatwienia twórcom aplikacji przeglądu sposobu pracy z pamięcią podręczną, jedną z części rozwiązania jest narzędzie przygotowujące informacje na temat działania zarządcy pamięci podręcznej. Dla wybranego interwału czasowego zbierane są statystyki, głównie odnośnie do typu żądania oraz jego źródła i celu. Dobranie odpowiedniego interwału gwarantuje pomijalny wpływ na wydajność aplikacji. Zebrane dane pozwalają

MOST WIEDZY Downloaded from mostwiedzy.pl

oszacować obciążenia konkretnych zarządców, porównywać je, a także badać ich zmienność w czasie.

5.5 Potencjalne ograniczenia

Z zaproponowanej architektury wynikają również możliwe ograniczenia. W bieżącym podrozdziale, oprócz ich wyspecyfikowania, zaznaczono okoliczności, w których mogą negatywnie wpłynąć na wydajność. Rysunek 5.8 pokazuje schematy działania aplikacji, dla których spodziewany zysk wydajnościowy z zaproponowanego rozszerzenia będzie największy, jak również takie dla których pojawia się niebezpieczeństwo upośledzenia wydajności w porównaniu do przetwarzania bez pamięci podręcznej.



(a) (b)

Rysunek 5.8: Diagramy prezentują charakterystykę docelowych aplikacji, które najbardziej skorzystają z zaproponowanego rozwiązania, jak również tych, które przy jego użyciu mogą zmniejszyć swoją wydajność

Pierwszym ograniczeniem może być długi czas otwierania, zamykania i synchronizacji pliku. Problem nie powinien jednak dotyczyć większości aplikacji HPC ze względu na ich długie działanie. W przypadku kiedy liczba odczytów i zapisów w ramach raz otwartego pliku jest stosunkowo duża, czas inicjalizacji i deinicjalizacji pamięci podręcznej nie będzie odgrywał znaczącej roli. Pomocne mogą się też tutaj okazać flagi zdefiniowanie w standardzie MPI, których obsługę również uwzględniono w rozszerzeniu:

- `MPI_MODE_CREATE` – pozwala na utworzenie pliku, więc w przypadku kiedy plik wcześniej nie istniał, wczytywanie pliku jest pomijane;

MOST WIEDZY Downloaded from mostwiedzy.pl

- `MPI_MODE_RDONLY` – tryb pozwalający wyłącznie na odczyt, pozwala ominąć synchronizację pamięci podręcznej podczas zapisu;
- `MPI_DELETE_ON_CLOSE` – użycie parametru skutkuje usunięciem pliku po przeprowadzeniu przetwarzania, co oznacza, że w tym przypadku synchronizacja pamięci podręcznej jest również pomijana.

Jednoczesne zastosowanie flag `MPI_MODE_CREATE` i `MPI_DELETE_ON_CLOSE` oraz brak wywołań funkcji synchronizującej będzie skutkować całkowitym pominięciem komunikacji z rozproszonym systemem plików. W takiej sytuacji pamięć podręczna może być traktowana jako rozproszona pamięć współdzielona, do której dostęp realizowany jest przez API MPI I/O.

Za drugie ograniczenie rozwiązania można uznać specyficzny tryb pracy z plikiem, który przyniesie największy zysk w porównaniu do działania bez użycia NVRAM. Wiele istniejących aplikacji jest zoptymalizowanych pod kątem możliwości typowych rozproszonych systemów plików, które preferują zapytania wysyłane z ograniczoną częstotliwością o stosunkowo dużych rozmiarach. Zdarza się, że takie optymalizacje nie są trywialne i wymagają dużego narzutu pracy programisty. Zaproponowane podejście kierowane jest raczej w stronę zapewnienia możliwości wydajnego operowania na małych, rozproszonych fragmentach pliku. Takie rozwiązanie nie wymusza na programiście implementacji wielkich buforów czy specjalnej reorganizacji danych. Z drugiej strony, patrząc na wysoce wyspecjalizowane oprogramowanie, zarówno warstwy PFS, jak i implementacji MPI I/O, w dodatku działających na wewnętrznych pamięciach podręcznych opartych o bardzo szybkie układy DRAM, trzeba liczyć się z tym, że w pewnych warunkach zaproponowane rozwiązanie nie będzie bardziej wydajnej od nierozszerzonego stosu technologicznego wyspecyfikowanego na początku rozdziału.

Trzecie ograniczenie związane jest ze specyficzną sytuacją, która może wystąpić podczas pracy z plikiem. Istnieje możliwość, w której duża grupa procesów jest zainteresowana dostępem do stosunkowo niewielkiego fragmentu pliku, zlokalizowanego w pamięci podręcznej pojedynczego węzła. W takiej sytuacji konkretny węzeł może zostać przeciążony, a czas odpowiedzi na żądanie ulegnie wydłużeniu. Pewną próbą rozwiązania problemu mogłaby być replikacja części danych, wiązałoby się to jednak z koniecznością implementacji szeregu mechanizmów zapewniających spójność zduplikowanych fragmentów. Pojedyncze próby zaprojektowania takiego rozwiązania zakończyły się niepowodzeniem związanym z ogólnym spadkiem wydajności, zagadnienie może być jednak bardziej szczegółowo zbadane

MOST WIEDZY Downloaded from mostwiedzy.pl

podczas dalszych prac rozwojowych.

5.6 Realizacja rozwiązania

5.6.1 Biblioteka libmpi-io-nvram

Oprócz dowiedzenia tezy założonej w tej rozprawie oraz osiągnięcia celów pracy, dołożono wszelkich starań, żeby zaproponowane rozwiązanie miało szerokie zastosowanie praktyczne. Biorąc pod uwagę dużą znajomość standardu MPI wśród programistów HPC, a także mnogość oprogramowania, które zostało napisane zgodnie z tym standardem, rozwiązanie zostało zaprojektowane w taki sposób, żeby z jednej strony było jak najbardziej przezroczyste dla programisty, z drugiej natomiast, żeby świadomy użytkownik mógł je odpowiednio konfigurować na poziomie kodu źródłowego. Stworzono więc bibliotekę `libmpi-io-nvram`, w której zaimplementowano pewien podzbiór funkcji MPI I/O, w skład którego wchodzą między innymi:

- `MPI_File_open(comm, filename, amode, info, fh)` – funkcja odpowiedzialna za otwarcie pliku, w zaimplementowanej bibliotece zawiera również logikę, związaną z inicjalizacją pamięci podręcznej, jednostek zarządców pamięci podręcznej oraz struktur odpowiedzialnych za zabezpieczenie danych przed utratą w przypadku awarii;
- `MPI_File_close(fh)` – funkcja odpowiedzialna za zamknięcie pliku, analogicznie do poprzedniej deinicjalizuje pamięć podręczną;
- `MPI_File_read_at(fh, offset, buf, count, datatype, status)` – funkcja pozwala na odczyt wybranego fragmentu pliku o zadanej długości;
- `MPI_File_write_at(fh, offset, buf, count, datatype, status)` – funkcja pozwala na zapis wybranego fragmentu pliku o zadanej długości;
- `MPI_File_sync(fh)` – funkcja synchronizująca, której wykonanie zapewnia, że wszystkie modyfikowane fragmenty pliku zostały trwale zapisane w systemie plików.

Pozostałe funkcje albo nie zostały zaimplementowane w opisywanej wersji rozwiązania (np. funkcje związane z adresowaniem przy użyciu współdzielonego wskaźnika, wsparcie dla widoków), albo zostały uznane za nieistotne z perspektywy tezy tej rozprawy (np.

MOST WIEDZY Downloaded from mostwiedzy.pl

funkcje zbiorowe, które w przypadku proponowanej implementacji nie zależą bezpośrednio od technologii zastosowanej pamięci).

Z punktu widzenia użytkownika biblioteki, w najprostszej konfiguracji wystarczy dołączyć plik nagłówkowy `file_io_pmem_wrappers.h`. W takim przypadku, każde wywołanie funkcji MPI będzie obsłużone przez implementację biblioteki libmpi-io-nvram. Dodatkowa, opcjonalna konfiguracja pamięci podręcznej odbywa się natomiast podczas otwierania pliku – standard MPI przewidział na konfigurację osobny parametr wywołania funkcji. Rozwiązanie jest również przezroczyste z perspektywy dalszej pracy z plikiem – każdy dostęp do pliku realizowany wewnątrz biblioteki wykorzystuje funkcje standardu MPI I/O. Oznacza to pełną kompatybilność rozszerzenia ze środowiskiem uruchomieniowym programu, ponieważ właściwy dostęp do pliku realizowany jest za pomocą tych samych narzędzi, niezależnie od tego, czy program korzysta z zaproponowanej pamięci podręcznej, czy nie.

Jedynym zewnętrznym narzędziem, użytym podczas realizacji rozwiązania, poza implementacją MPI i biblioteką pthreads, były biblioteki libpmem i libpmemblk z zestawu PMDK (*Persistent Memory Development Kit*)[4]. Zestaw PMDK jest kolekcją narzędzi, ułatwiających pracę z pamięcią NVRAM (nazywaną przez twórców terminem *persistent memory*), a biblioteka libpmem oferuje niskopoziomowy interfejs do zarządzania nią na zasadzie dostępu do pliku odwzorowanego w przestrzeni adresowej. Narzędzia wchodzące w skład zestawu są stworzone zgodnie z zaleceniami zaproponowanymi przez organizację SNIA [101], co zwiększa prawdopodobieństwo kompatybilności użytych narzędzi z urządzeniami mającymi pojawić się na rynku w najbliższej przyszłości. Samo PMDK bazuje na mechanizmie DAX (*Direct Access*)[5], który pozwala na bezpośredni odczyt i zapis do pamięci trwałej przy wykorzystaniu plików odwzorowanych w przestrzeni adresowej. Oznacza to, że na urządzeniu opartym o NVRAM musi być zainstalowany system plików wspierający DAX, a alokowana pamięć przyjąć formę plików. Obecnie zestaw bibliotek PMDK rozszerzający mechanizm DAX jest wspierany w systemach z rodziny Linux i Windows. Jeśli jednak okaże się, że nie zostanie przyjęty jako domyślne oprogramowanie zarządzające pamięcią NVRAM, autorskie rozwiązanie zostało napisane tak, żeby ewentualna wymiana biblioteki nie wpłynęła na kształt całego oprogramowania.

Architektura biblioteki została przewidziana w taki sposób, żeby zaoferować nie tylko możliwość parametryzacji rozwiązania, ale żeby pozwolić również na ewentualną obsługę

[4] `http://pmem.io/pmdk/libpmem/`

[5] `https://www.kernel.org/doc/Documentation/filesystems/dax.txt`

MOST WIEDZY Downloaded from mostwiedzy.pl

trybów innych niż pamięć podręczna. Przykładowo, w obecnej wersji biblioteki, oprócz pamięci podręcznej, będącej przedmiotem tej rozprawy, zaimplementowano także bardzo prosty i wydajny tryb optymalizujący pracę z plikiem umiejscowionym w lokalnej pamięci NVRAM, bez dostępu z poziomu innych węzłów. Technicznie mechanizm bazuje na strukturze, zawierającej zestaw wskaźników do funkcji zastępujących funkcje MPI I/O, przy czym dla każdego trybu tworzony jest osobny zestaw. Możliwość otwarcia różnych plików w różnych trybach pozwala tworzyć zaawansowane aplikacje, w których dostęp do każdego z nich będzie optymalizowany w różny sposób. W ramach testów posłużono się programem, którego wszystkie procesy pracowały na wspólnym pliku zarządzanym przez pamięć podręczną, a dodatkowo każdy proces z osobna wykorzystywał plik na lokalnej pamięci NVRAM jako magazyn swoich prywatnych danych tymczasowych. Obecnie, żeby wybrać tryb pracy inny niż domyślny, czyli pamięć podręczną, należy jawnie go wskazać podczas otwierania pliku. W kolejnych wersjach biblioteki należałoby spróbować opracować mechanizm, pozwalający na automatyczny wybór najlepszego rozwiązania w zależności od kontekstu.

Jednym z parametrów konfiguracyjnych pamięci podręcznej jest ścieżka do punktu montowania zasobu systemowego opartego o NVRAM. Podczas inicjalizacji biblioteka weryfikuje, czy dany zasób spełnia kryteria pamięci NVRAM, wykorzystując do tego funkcję `pmem_is_pmem(addr, len)` z zestawu PMDK. Jeśli funkcja potwierdzi poprawną weryfikację, wówczas możliwe jest wykorzystanie narzędzia zgodnie z opisem w ramach bieżącego rozdziału. Może się jednak zdarzyć, że do pamięci podręcznej podpięto zasób o innej charakterystyce i w takim przypadku, oprócz poinformowania o zaistniałej sytuacji w logach:

- dla pamięci ulotnej RAM rozwiązanie powinno być bardzo wydajne, nie będzie natomiast w stanie zadbać o zabezpieczenie danych przed utratą w przypadku awarii w inny sposób niż cykliczna synchronizacja z PFS (taki tryb działania opisano w jednej z publikacji [76]);
- dla pamięci opartej o HDD lub SSD rozwiązanie będzie niewydajne, nie tylko ze względu na wolniejszy dostęp, ale również na blokową organizację.

Jak wcześniej wspomniano, bibliotekę wyposażono w mechanizm monitorowania struktur pamięci podręcznej. W praktyce polega on na zapisaniu serii wcześniej zdefiniowanych zdarzeń do odpowiednich plików w ramach klastra, które następnie należy zebrać i przetworzyć, wyciągając interesujące statystyki, na przykład przy użyciu arkusza kalkulacyj-

MOST WIEDZY Downloaded from mostwiedzy.pl

nego. Ze względu na wydajność – monitoring rozwiązania nie może zauważalnie wpływać na czas działania aplikacji – jako format plików wybrano CSV (*comma-separated values*). Jeśli chodzi o typy plików i obsługiwane zdarzenia, można wyróżnić:

- pojedynczy plik zlokalizowany na pierwszym węźle klastra logujący globalne zdarzenia takie jak rozpoczęcie oraz zakończenie procedur otwarcia i zamknięcia pliku, synchronizacja pamięci podręcznej z PFS, konfigurowanie zarządców;
- po jednym pliku dla każdego zarządcy pamięci podręcznej, który zawiera następujące zdarzenia: komunikaty przychodzące i wychodzące, a w szczególności żądania dostępu wraz z nadawcą, lokalizacją i rozmiarem żądania, oraz szczegóły komunikacji z PFS.

Podczas monitorowania działania pamięci podręcznej przydatne mogą być również logi gromadzone na trzech poziomach szczegółowości (błędy, informacje ogólne oraz niskopoziomowe wpisy szczegółowe) i zestaw komunikatów na wypadek błędów, zdefiniowany zgodnie ze standardem MPI.

5.6.2 Główne problemy implementacji i rozwiązania

Do implementacji rozwiązania użyto języka C i standardu MPI. Oznacza to, że zaimplementowano podzbiór funkcji MPI, korzystając z samego MPI. Dzięki temu stworzono narzędzie uniwersalne, niezależne od architektury sprzętowej oraz systemu operacyjnego – popularne implementacje MPI wspierają różne konfiguracje sprzętowe (np. komunikację przy użyciu protokołów sieciowych, takich jak Ethernet, Infiniband, iWARP), działają również na wielu systemach operacyjnych (np. Linux wraz ze wszystkimi dystrybucjami, systemy z rodziny Unix, takie jak FreeBSD, platformy mniej powszechne takie jak OpenIndiana). Do budowania rozwiązania wykorzystano popularne narzędzia Autoconf i Make, więc rozwiązanie powinno być stosunkowo łatwe do zbudowania niezależnie od systemu operacyjnego i jego konfiguracji.

Implementacja rozwiązania w postaci biblioteki napisanej w standardzie MPI wiązała się z kilkoma trudnościami, wynikającymi z tej technologii, wielokrotnie omawianymi wcześniej w literaturze [81, 98, 36]. Niestety, część problemów do dziś nie została rozwiązana w oficjalnym standardzie, a twórcy bibliotek mogą się posiłkować albo narzędziami zewnętrznymi, albo opracować własne mechanizmy. Wśród takich problemów wymieniane są między innymi te zgromadzone poniżej.

MOST WIEDZY Downloaded from mostwiedzy.pl

Pierwszym problemem, znanym w literaturze, jest kwestia inicjalizacji środowiska MPI w ramach zewnętrznej biblioteki [98, 36]. Inicjalizacja środowiska MPI wymaga pewnej konfiguracji, a różne biblioteki mogą wymagać różnych parametrów konfiguracyjnych. W przypadku, kiedy aplikacja wykorzystuje wiele bibliotek napisanych w MPI, nie istnieje standardowy mechanizm pozwalający na wynegocjowanie między komponentami jednej, właściwej konfiguracji. Pełnym rozwiązaniem problemu jest wykorzystanie dodatkowego narzędzia zewnętrznego, oferującego mechanizmy, które pozwalają na implementację takiej logiki, na przykład PnMPI [96]. Z drugiej strony, takie rozwiązanie wymagałoby zastosowania wybranego narzędzia we wszystkich bibliotekach opartych o MPI używanych w aplikacji, co jest mało prawdopodobne w przypadku elementów niebędących częścią standardu. Ostatecznie pamięć podręczną wyposażono w prosty mechanizm pozwalający na osadzanie biblioteki napisanej w MPI w dowolnej innej takiej bibliotece, natomiast problem wykorzystania dwóch bibliotek MPI w aplikacji na tym samym poziomie pozostawiono do rozwiązania programiście takiej aplikacji.

Kolejny problem implementacyjny to brak mechanizmu zapytań o konfigurację obiektów MPI. W standardzie MPI część danych konfiguracyjnych przekazywana jest podczas inicjalizacji obiektów w strukturze `MPI_Info`. Niestety, po utworzeniu takiego obiektu (na przykład typu `MPI_File`) programista nie ma możliwości ustalenia wartości jego parametrów konfiguracyjnych. W zaproponowanym rozwiązaniu istotne parametry konfiguracyjne są przetwarzane już w chwili inicjalizacji obiektu, a następnie zapisywane w celu późniejszego wykorzystania.

Niekiedy wymagane jest rozszerzenie obiektu MPI o szereg dodatkowych informacji. Projektując rozwiązanie, zweryfikowano dwie metody powiązania utworzonego obiektu z dodatkowymi parametrami. Pierwsza bazuje na globalnej tablicy asocjacyjnej, w której każdemu obiektowi przyporządkowano zestaw dodatkowych wartości. Jej główną wadą jest konieczność zarządzania globalną strukturą, co może nie być trywialne w środowisku wielowątkowym. Rozwiązaniem wdrożonym w prezentowaną implementację pamięci podręcznej jest druga metoda, polegająca na opakowaniu obiektu MPI w strukturę zawierającą dodatkowe parametry. Za potencjalną wadę takiego podejścia można uznać konieczność przechwycenia wszystkich wołań funkcji MPI, operujących na obiekcie w celu opakowywania i rozpakowywania obiektu. W przypadku omawianej pamięci podręcznej nie jest to problemem, ponieważ przechwycenie żądań operacji na plikach było jednym z założeń implementacji rozwiązania.

MOST WIEDZY Downloaded from mostwiedzy.pl

Ciekawym problemem związanym z wydajnością jest powszechne stosowanie w implementacjach MPI sprawdzania odpowiedzi w nieskończonej pętli (tzw. *busy-waiting* albo *spinning*). W większości przypadków aplikacje pisane w standardzie MPI operują na stałej liczbie procesów i wątków, a najwydajniejsze konfiguracje tworzy się, przydzielając pojedynczy wątek lub proces do pojedynczego rdzenia procesora. Z tego powodu, typowym zachowaniem funkcji blokujących jest cykliczne sprawdzanie, czy komunikat nie nadszedł, tak szybko, jak to tylko możliwe. Oznacza to, że takie sprawdzanie skutkuje pełnym obciążeniem rdzenia procesora, mimo że funkcja nie realizuje żadnej właściwej logiki. W przypadku pamięci podręcznej i dynamicznego uruchomienia dodatkowych jednostek (zarządca pamięci podręcznej, zarządca dziennika żądań zapisu) przy standardowych ustawieniach dochodziło do sytuacji dużego konfliktu pomiędzy wątkami, w którym niekiedy przez dłuższy czas wątek obsługujący przychodzące żądanie nie był w stanie otrzymać wystarczającego czasu procesora ze względu na inny wątek, który w tym samym czasie zajmował się jedynie oczekiwaniem. W środowisku wielowątkowym taka sytuacja jest więc niedopuszczalna i wymaga odpowiedniej obsługi [90]. Najprostszym rozwiązaniem byłoby wyłączenie mechanizmu *busy-waiting* i jest to możliwe w niektórych implementacjach, na przykład w OpenMPI[6]. Niestety, w przypadku innych implementacji, takich jak MPICH[7], jest to niezalecane. Innym wyjściem z sytuacji byłoby zastąpienie komunikacji blokującej przez nieblokującą, ale w przypadku autorskiego rozwiązania nie było to możliwe ze względu na konieczność oczekiwania w pętli na konkretne żądanie. Ostatecznie problem rozwiązano, tworząc własną implementację funkcji blokujących przy użyciu nieblokujących i mechanizmu zwolnienia procesora przez zadany czas.

W przedstawionym projekcie rozwiązania jednostka zarządcy pamięci podręcznej jest niezależna od pozostałych procesów uruchomionych na danym węźle. Oznacza to konieczność utworzenia w aplikacji dodatkowego wątku lub procesu. Standard MPI oferuje obie możliwości, różnią się one jednak znacznie szczegółami technicznymi.

Tworzenie nowych procesów możliwe jest bezpośrednio za pomocą funkcji standardu:

- `MPI_Comm_spawn(command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes)` – pozwala na uruchomienie *maxprocs* instancji aplikacji MPI zlokalizowanej na ścieżce *command*, zwracając *intercomm* umożliwiający komunika-

[6] https://www.open-mpi.org/faq/?category=running#oversubscribing

[7] https://wiki.mpich.org/mpich/index.php/Frequently_Asked_Questions#Q:_Why_does_my_MPI_program_run_much_slower_when_I_use_more_processes.3F

MOST WIEDZY Downloaded from mostwiedzy.pl

cję pomiędzy procesami aplikacji MPI wywołującymi funkcję, a procesami powstałymi w wyniku wywołania funkcji;

- `MPI_Comm_spawn_multiple(count, array_of_commands, array_of_argv, array_of_maxprocs, array_of_info, root, comm, intercomm, array_of_errcodes)` – funkcja podobna do poprzedniej, pozwala jednak na większą dowolność w uruchamianiu różnych procesów; każda uruchomiona instancja aplikacji może się cechować inną ścieżką pliku uruchomieniowego, innymi przekazanymi parametrami, inną konfiguracją itp.

Teoretycznie każda implementacja MPI zgodna z drugą wersją standardu powinna na to pozwolić dla każdej wspieranej konfiguracji sprzętowej. W praktyce jednak okazuje się, że zestaw funkcji nie jest zbyt często wykorzystywany, a ich implementacja potrafi zawierać błędy – jako przykład można podać błąd implementacji jednej z funkcji w MVAPICH2 zgłoszony przez autora podczas prac nad rozwiązaniem[8]. Drugą wadą mechanizmu tworzenia nowych procesów w MPI jest ich potencjalnie niska wydajność, przez co technika jest niezalecana nawet w samej dokumentacji standardu [81]. Problemem może być również właściwy przydział procesów do węzłów (celem jest uruchomienie pojedynczego zarządcy pamięci podręcznej na każdym węźle) – taka możliwość nie jest częścią standardu i może być w różny sposób realizowana w różnych implementacjach MPI.

W przypadku wątków, do ich tworzenia należy wykorzystać mechanizmy systemu operacyjnego lub zewnętrznych bibliotek. Często wykorzystuje się w tym celu bibliotekę pthreads, wchodzącą w skład standardu POSIX rozpowszechnionego wśród systemów z rodziny Unix. Wsparcie wątków w implementacjach MPI może być mniej lub bardziej ograniczone, w praktyce jednak najpopularniejsze implementacje wspierają pełną integrację z wątkami. Oficjalny standard wyróżnia cztery poziomy wsparcia wątków:

- `MPI_THREAD_SINGLE` – jedyną poprawną konfiguracją jest pojedynczy wątek dla każdego procesu;
- `MPI_THREAD_FUNNELED` – aplikacja może uruchamiać wiele wątków dla każdego procesu, ale tylko wątek główny może korzystać z funkcji MPI;
- `MPI_THREAD_SERIALIZED` – wsparcie wielowątkowości z możliwością wywoływania funkcji MPI przez dowolny wątek, ale w ramach procesu dopuszczalne jest wywołanie wyłącznie jednej funkcji MPI jednocześnie;

[8] `http://mailman.cse.ohio-state.edu/pipermail/mvapich-discuss/2016-January/005901.html`

MOST WIEDZY Downloaded from mostwiedzy.pl

- `MPI_THREAD_MULTIPLE` – pełne wsparcie dla wątków bez ograniczeń.

Niestety, podejście oparte na wątkach jest również niezalecane, w tym przypadku z powodu trudności w zastosowaniu dodatkowych optymalizacji [81]. Pierwotnie w bibliotece libmpi-io-nvram zaimplementowano oba podejścia, ale ostatecznie zdecydowano się pozostać przy wątkach pthreads ze względu na większą dojrzałość implementacji MPI w ich obsłudze oraz ich szerszą popularność. Kwestia wydajności odgrywała przy wyborze drugorzędną rolę ze względu na wysoką zależność narzutu mechanizmu od sposobu działania aplikacji – niektóre aplikacje osiągały lepsze czasy przy użyciu wątków, w innych pod tym względem preferowane były procesy [73].

Wymienione problemy implementacyjne, wynikające głównie ze specyfiki standardu MPI, zostały omówione również w artykule [73].

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 6

Testy rozwiązania

6.1 Wprowadzenie do testów rozwiązania

W rozdziale zawarto szczegóły dotyczące środowiska testowego, charakterystykę aplikacji demonstrujących wydajność rozwiązania, opis przeprowadzonych eksperymentów, zgromadzone wyniki oraz ich omówienie. Celem testów było udowodnienie tezy poprzez wykazanie, że zastosowanie autorskiego rozwiązania, opartego o bajtowo adresowaną pamięć NVRAM, umożliwia zwiększenie wydajności wybranych aplikacji równoległych. Badając wydajność, porównano czas działania aplikacji uruchomionej przy użyciu standardowych narzędzi MPI I/O z wersją rozszerzoną o bibliotekę dodającą zaproponowaną pamięć podręczną. Do przeprowadzenia testów wykorzystano oprogramowanie napisane zgodnie ze standardem MPI – dwa syntetyczne benchmarki oraz cztery aplikacje użytkowe, realizujące następujące zadania:

- przetwarzanie dużych obrazów,
- wyznaczanie kolejnych potęg grafu reprezentowanego przez macierz sąsiedztwa,
- przeglądanie dwuwymiarowej mapy w poszukiwaniu określonych wzorców,
- symulowanie zachowania tłumu, na przykład podczas ewakuacji.

Ostatni zestaw eksperymentów wykorzystuje wcześniej opisane aplikacje do pokazania narzutu na czas działania, który jest związany z mechanizmami oferującymi możliwość odzyskania danych w przypadku awarii.

MOST WIEDZY Downloaded from mostwiedzy.pl

6.2 Środowisko testowe

Wszystkie testy przeprowadzono na dwóch klastrach: Lap06 i K2. Węzły klastra Lap06 były wyposażone w sprzętową emulację pamięci NVRAM. Klaster K2 zamiast NVRAM-u używał pamięci operacyjnej DRAM (mechanizm `tmpfs`) i z racji swojego rozmiaru był wykorzystywany do testów skalowalności aplikacji. W niniejszej rozprawie zaprezentowano jeden scenariusz testowy z wykorzystaniem K2. Konfigurację sprzętową i opis zainstalowanego oprogramowania zamieszczono odpowiednio w tabelach 6.1 i 6.2. Jeśli chodzi o system plików, wykorzystano Orange-FS, który jest bezpośrednim następcą popularnego PVFS2, stosowanego przy testowaniu większości nowych rozwiązań [6], a także w testach dużej grupy wcześniej omówionych optymalizacji MPI I/O [38, 126, 116, 125, 33, 32].

Tabela 6.1: Konfiguracja sprzętowa klastrów testowych

	Lap06	K2
Rozmiar klastra	6 węzłów obliczeniowych	96 węzłów obliczeniowych
Liczba węzłów PFS	2	3
CPU	2 x Intel Xeon E5-4620	2 x Intel Xeon E5345
DRAM	15 GB	8 GB
Sieć	40 Gb/s Infiniband, 10 Gb/s Ethernet	10 Gb/s Ethernet
Dane	SSD	HDD
Symulator NVRAM	17 GB, symulacja sprzętowa	brak

Tabela 6.2: Oprogramowanie użyte na potrzeby testów

	Lap06	K2
System operacyjny	CentOS 6.5	CentOS 6.5
Implementacja MPI	MPICH 3.2	MPICH 3.2
PFS	Orange-FS 2.9.6	Orange-FS 2.8.7

Sprzętowy symulator pamięci NVRAM został wypożyczony od firmy zewnętrznej, niestety jego szczegóły działania objęte są tajemnicą przedsiębiorstwa. Z punktu widzenia bieżącej rozprawy ważne jest, że symulator został szczegółowo przetestowany zgodnie z

MOST WIEDZY Downloaded from mostwiedzy.pl

metodą czarnej skrzynki (ang. *black-box testing*). Rezultaty potwierdziły poprawność działania symulatora, otrzymane wyniki pokryły się z oczekiwanymi.

Symulator rezerwował pewien wskazany obszar pamięci operacyjnej urządzenia i pozwalał na konfigurację trzech parametrów dla tego obszaru:

- dodatkowego opóźnienia dostępu – opóźnienia dodawanego do każdego żądania dostępu do pamięci (zarówno odczytu, jak i zapisu),
- dodatkowego opóźnienia zapisu – czasu, po którym dane w pamięci należy uznać za trwale zapisane,
- dzielnika przepustowości – parametru pozwalającego zredukować przepustowość zainstalowanej pamięci DRAM.

Nie dysponując rzeczywistym urządzeniem, które mogłoby posłużyć jako wzór konfiguracji symulatora, parametry należało wyznaczyć na podstawie materiałów udostępnionych przez producentów podobnych urządzeń. Nowe pamięci wielokrotnie były zapowiadane jako cechujące się wydajnością na poziomie wyższym od dysków SSD, ale niższym od pamięci DRAM.

W węzłach zainstalowano moduły DDR3-1600 o teoretycznej przepustowości 12,8 GB/s. Maszyny wyposażone są w technologię DDR3 triple-channel, pozwalającą zwielokrotnić przepustowość, w prezentowanym przypadku do teoretycznej wartości 38,4 GB/s. W praktyce, oprogramowanie symulatora wskazywało wartości około 37 GB/s. Nowoczesne dyski SSD, podłączane przy użyciu złącza M.2, oferują przepustowość sięgającą ponad 3 GB/s (najszybszy obecnie dysk na rynku Samsung SSD 960 PRO[1]). Zakładając instalację układów NVDIMM również w konfiguracji triple-channel, możemy się spodziewać przepustowości nieco wyższej niż 9 GB/s. Wykorzystany symulator obsługiwał jedynie całkowite wartości dzielnika przepustowości, domyślnie przyjęto więc czterokrotną redukcję przepustowości pamięci DRAM, co skutkowało praktycznymi osiągami na poziomie około 9 GB/s.

Trudniejszym zadaniem było oszacowanie opóźnień w dostępie do pamięci i czasie potrzebnym na zapis. Obecnie, opóźnienia dla dysków SSD potrafią spaść poniżej 10 000 ns (np. dla urządzeń serii Intel Optane 900P[2]). Są to jednak czasy o trzy rzędy wielkości

[1] `http://www.samsung.com/semiconductor/minisite/ssd/product/consumer/ssd960/`

[2] `https://www.intel.com/content/www/us/en/products/memory-storage/solid-state-drives/gaming-enthusiast-ssds/optane-900p-series/900p-480gb-aic-20nm.html`

MOST WIEDZY Downloaded from mostwiedzy.pl

większe w porównaniu do pamięci DRAM, więc zdecydowanie zbyt długie, żeby korzystać z NVRAM bezpośrednio z pominięciem pamięci podręcznej procesora i pamięci DRAM [93]. Jeśli chodzi o parametry NVRAM używane w innych badaniach, były to opóźnienia w dostępie rzędu setek nanosekund [114, 122, 97]. Dodatkowo, w większości typów pamięci, czas zapisu do pamięci jest dłuższy od czasu odczytu. Biorąc pod uwagę powyższe, jak również ograniczone możliwości konfiguracyjne symulatora NVRAM, ustalono następujące wartości parametrów:

- dodatkowe opóźnienie dostępu o rząd wielkości większe od opóźnień występujących dla pamięci DRAM,
- dodatkowe opóźnienie zapisu o rząd wielkości większe od dodatkowego opóźnienia w dostępie.

Tabela 6.3 prezentuje dokładne domyślne wartości parametrów – jeśli w eksperymencie nie podano parametrów wprost, symulator został skonfigurowany zgodnie z parametrami z tabeli.

Tabela 6.3: Parametry symulatora NVRAM (tylko klaster Lap06)

Dodatkowe opóźnienie dostępu	600 ns
Dodatkowe opóźnienie zapisu	2000 ns
Dzielnik przepustowości	4

6.3 Benchmark ROMPIO

Chociaż aplikacje wspomniane w tezie pracy rozumiane są jako programy o praktycznym zastosowaniu, wydajność pracy z plikiem często mierzy się przy użyciu testów syntetycznych. Z tego powodu pierwsze przedstawione wyniki zgromadzono przy użyciu narzędzia ROMPIO – syntetycznego benchmarka zaprojektowanego między innymi do mierzenia wydajności MPI I/O[3]. Oprogramowanie powstało w Los Alamos National Laboratory i jest jedyną aplikacją demonstrującą, która nie została stworzona na potrzeby tej rozprawy. ROMPIO wybrano ze względu na wysoką konfigurowalność i licencję typu open source (obecnie kod źródłowy nie jest już publicznie dostępny).

[3] `https://www.osti.gov/biblio/1231008-rompio`

MOST WIEDZY Downloaded from mostwiedzy.pl

Przeprowadzone testy polegały na wielokrotnym zapisie lub odczycie paczki danych o zadanym rozmiarze. Rozmiar pliku dobierany był w taki sposób, żeby każde żądanie odnosiło się do innego obszaru w pliku. W eksperymencie wszystkie uruchomione procesy miały równy udział – każdy z nich miał za zadanie wysłać tę samą liczbę żądań. Szczegółowe parametry wspólne dla wszystkich testów były następujące:

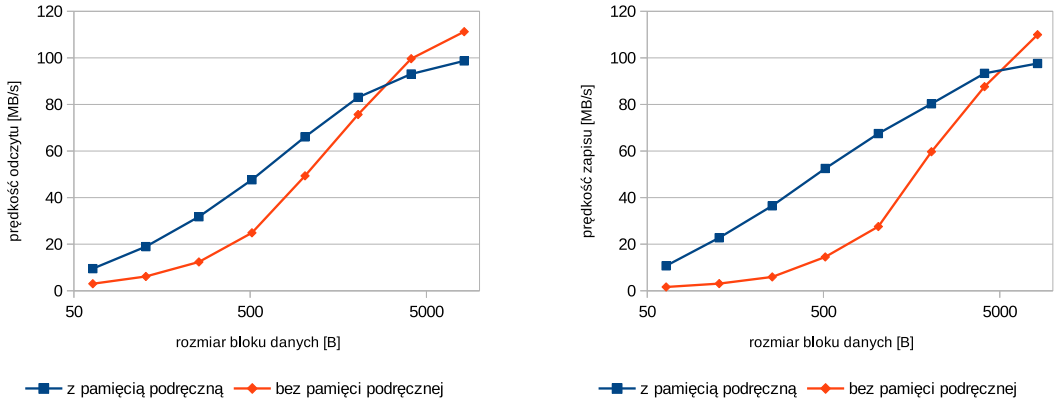
- klaster: Lap06,
- liczba procesów na każdy węzeł: 15,
- liczba żądań łącznie: 100 000.

Rysunek 6.1 pokazuje wpływ rozmiaru pojedynczej paczki danych na prędkość odczytu (wykres 6.1a) i zapisu (wykres 6.1b). W teście brało udział sześć węzłów. Wyniki otrzymane dla MPI I/O rozszerzonego o pamięć podręczną NVRAM pokazują, ze prędkości osiągnięte dla odczytu i zapisu są porównywalne, podczas gdy niezmodyfikowane MPI I/O dla niektórych rozmiarów bloków danych wyraźnie faworyzuje w tej kwestii odczyt. Zgodnie z obawami przedstawionymi w projekcie rozwiązania, zaproponowana pamięć podręczna poprawia wydajność dla paczek danych mniejszych od pewnego progu. W przedstawionej konfiguracji próg wyniósł 2 KB dla odczytu i 4 KB dla zapisu danych. Należy również pamiętać, że w teście rozmiar pliku był ściśle powiązany z rozmiarem bloku danych, co negatywnie wpływa na pamięć podręczną ze względu na duży narzut otwarcia i zamknięcia pliku – należałoby się spodziewać lepszych rezultatów dla testów ze stałym rozmiarem pliku.

Kolejny eksperyment, dla którego wyniki pokazano na rysunku 6.2, badał skalowalność prezentowanego rozwiązania. Podczas testów rozmiar bloku danych został ustawiony na 1 KB. Na wykresach ponownie widać podobną wydajność MPI I/O rozszerzonego o NVRAM, zarówno dla odczytu, jak i zapisu. Zauważalny jest również liniowy przyrost prędkości wraz z liniowym wzrostem liczby węzłów w klastrze, rozpoczynając od dwóch węzłów. W przypadku pojedynczego węzła, wydajność jest wyższa niż wynikałoby to z liniowej charakterystyki ze względu na brak komunikacji sieciowej. Benchmark testujący niezmodyfikowany stos technologiczny nie pokazał skalowalności – MPI I/O i PFS działają w tym przypadku niezależnie od liczby węzłów, natomiast zależnie od liczby żądań kierowanych do systemu, których w każdym scenariuszu testowym było tyle samo.

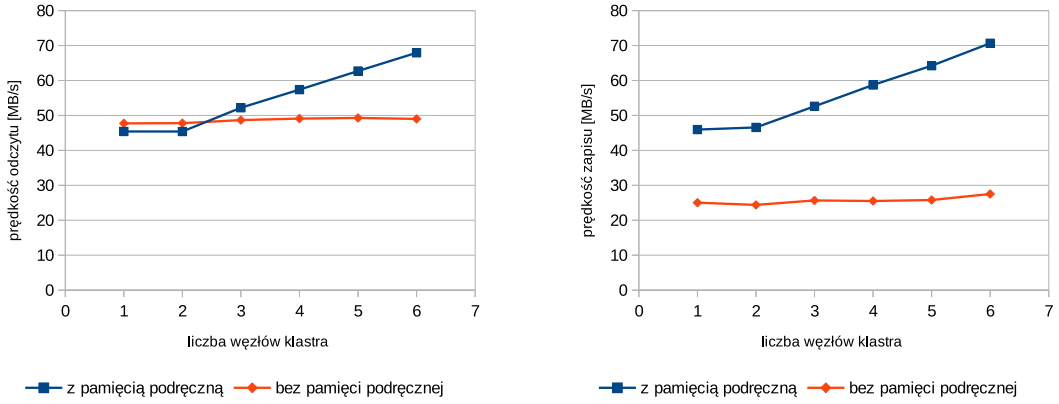
Testy wykonane przy użyciu ROMPIO pokazują, że dla pewnych scenariuszy (niewielki rozmiar żądania, klaster złożony z co najmniej kilku węzłów) proponowana pamięć pod-

MOST WIEDZY Downloaded from mostwiedzy.pl



(a) (b)

Rysunek 6.1: Benchmark ROMPIO – wpływ rozmiaru bloku danych na prędkość odczytu i zapisu



(a) (b)

Rysunek 6.2: Benchmark ROMPIO – wpływ rozmiaru klastra na prędkość odczytu i zapisu

ręczna oparta o NVRAM poprawia wydajność operacji na plikach. Wyniki potwierdzają również skalowalność zaprezentowanego rozwiązania.

6.4 Benchmark – błądzenie losowe

Drugą testowaną aplikacją jest benchmark stworzony na potrzeby rozprawy. W przeciwieństwie do ROMPIO, aplikacja, oprócz żądań odczytu i zapisu, wykonuje również serię obliczeń, co ma symulować działanie rzeczywistych aplikacji. Jest to istotne, ponieważ zaproponowane rozwiązanie jest częścią aplikacji i korzysta z tych samych zasobów. Żądania

MOST WIEDZY Downloaded from mostwiedzy.pl

nie odnoszą się do kolejnych, sąsiadujących fragmentów pliku – plik jest podzielony na bloki o zadanym rozmiarze, które traktuje się jak węzły grafu pełnego, a aplikacja wybiera kolejne węzły zgodnie z podejściem błądzenia losowego. Narzędzie działa zgodnie z algorytmem przedstawionym poniżej.

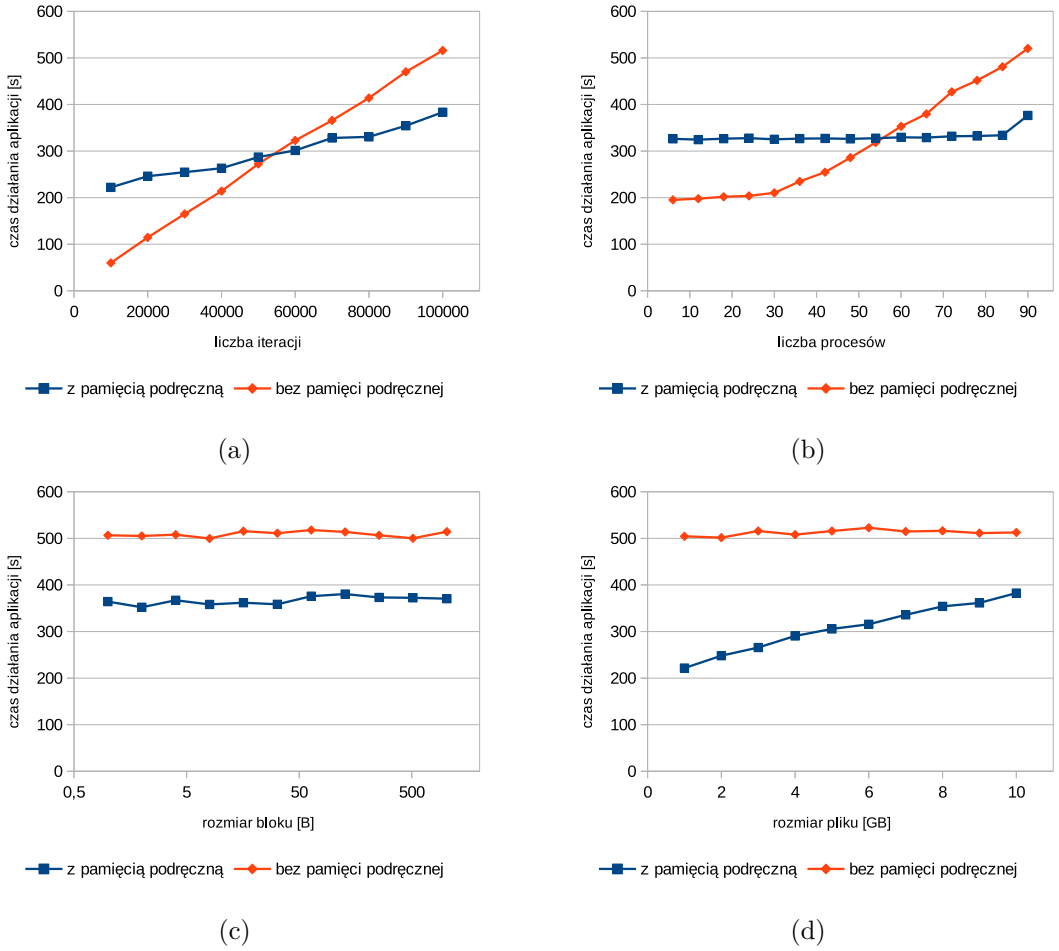
1. Każdy proces wybiera losowy blok pliku o zadanym rozmiarze i wczytuje go do pamięci.
2. Na danych wykonywana jest imitacja obliczeń – rodzaj obliczeń jest bez znaczenia, w testach, jeśli nie zaznaczono inaczej, użyto miliona iteracji problemu Collatza.
3. Każdy proces zapisuje blok do lokalizacji, z której został odczytany.
4. Przed rozpoczęciem kolejnej iteracji wszystkie procesy muszą zakończyć poprzednią iterację.

Do testów użyto klastra Lap06, a jeśli nie opisano inaczej:

- 15 procesów na każdy węzeł,
- 100 000 iteracji algorytmu,
- 10 GB pliku,
- bloku o rozmiarze 512 B.

Rysunek 6.3 przedstawia czas działania aplikacji dla różnych konfiguracji. Na wykresie 6.3a widać, że poprawa wydajności uzyskana dzięki zastosowaniu pamięci podręcznej opartej o NVRAM była możliwa dopiero w sytuacji wystarczająco długiego działania aplikacji. Związane jest to z narzutem na otwarcie i zamknięcie pliku. Wykres 6.3b pokazuje, że przy użyciu zaproponowanego rozwiązania aplikacja potrafiła poradzić sobie ze wzrostem intensywności odczytów i zapisów, o ile nie przekroczono pewnej liczby procesów. W omawianym eksperymencie wartość ta wyniosła czternaście procesów na każdy węzeł, co zgadza się z wartościami teoretycznymi – dwa zainstalowane procesory miały łącznie szesnaście rdzeni, w tym jeden rdzeń wykorzystywany był przez zarządcę pamięci podręcznej, jeden na bieżące zadania systemu operacyjnego. Jeśli chodzi o niezmodyfikowany stos technologiczny, zauważyć można rosnący czas działania aplikacji wraz ze wzrostem liczby żądań dostępu do pliku. Wykres 6.3c skupia się na różnym rozmiarze żądanego bloku

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 6.3: Benchmark – błądzenie losowe – wyniki eksperymentów

danych. W przeciwieństwie do testów przeprowadzonych przy użyciu ROMPIO, nie zaobserwowano wpływu rozmiaru żądania na czas działania aplikacji. Wyjaśnieniem jest tutaj mniejsza intensywność odczytów i zapisów – przedzielenie operacji na plikach fazą obliczeń pozwoliło uniknąć kolejkowania żądań. Ostatni zaprezentowany wykres 6.3d przedstawia wpływ rozmiaru pliku na czas działania aplikacji. O ile rozmiar pliku nie ma wpływu na niezmodyfikowane MPI I/O, o tyle w przypadku zaproponowanej pamięci podręcznej jest od niego liniowo zależny.

W opisanych eksperymentach dane testowe celowo wybrano w taki sposób, żeby pokazać charakterystykę aplikacji, dla której pamięć NVRAM zarządzana przez proponowane rozwiązanie zaczyna poprawiać wydajność. Wyraźnie widać, że dla pewnych scenariuszy czas działania aplikacji po rozszerzeniu o dodatkową pamięć podręczną uległ skróceniu. Kluczowa jest tutaj intensywność obliczeń, która musi zrekompensować narzut związany

MOST WIEDZY Downloaded from mostwiedzy.pl

z otwarciem i zamknięciem pliku. Sam narzut jest liniowo zależny od rozmiaru pliku.

6.5 Przetwarzanie dużych obrazów

Zadaniem kolejnej aplikacji było przetwarzanie obrazów o wysokiej rozdzielczości – w zaprezentowanych przypadkach testowych były to wielkości do około dwóch gigapikseli. Chociaż w pierwszej chwili taki rozmiar grafik może wydawać się mało praktyczny, kierowano się tutaj w pierwszej kolejności zastosowaniami naukowymi. Przykładami mogą być materiały produkowane przez teleskopy przekraczające rozdzielczość gigapikseli [47], albo obrazy utworzone z połączenia wielu mniejszych, takie jak te przygotowane przez NASA [85]. Zdarza się również, że bardzo duże obrazy prezentowane są ze względów marketingowych lub stają się swego rodzaju pokazem możliwości firm związanych z technologią przetwarzania fotografii. Z taką sytuacją mieliśmy do czynienia podczas prezentacji zdjęcia reklamowego o rozdzielczości 53 gigapikseli promującego firmę Bentley [123], albo niedawnej publikacji panoramy Szanghaju o rozdzielczości 195 gigapikseli [82]. Innym przykładem dużych obrazów (rozdzielczości dochodzące do gigapiksela) są cyfrowe reprodukcje dzieł sztuki, tak ja te utworzone i udostępnione na otwartej licencji w ramach projektu Google Art Project [27]. Uzasadniając potrzebę przetwarzania wysokorozdzielczych grafik, nie sposób nie wspomnieć o stale rosnącej rozdzielczości matryc w aparatach fotograficznych. Obecnie na rynku stosunkowo łatwo można znaleźć matryce o rozdzielczości przekraczającej 50 megapikseli [8], a w zastosowaniach profesjonalnych pojawiają się urządzenia, których producenci deklarują rozdzielczość efektywną na poziomie 400 megapikseli [31].

Duży rozmiar obrazów uzasadnia również wykorzystanie technik i urządzeń stosowanych w obliczeniach wysokiej wydajności, zamiast przeprowadzania obliczeń na pojedynczej maszynie. Prostą weryfikację tego założenia przeprowadzono przy użyciu najmniej złożonego obliczeniowo z przetestowanych algorytmów, czyli wykrywania krawędzi przy użyciu operatora Sobela. W eksperymencie porównano czas przetwarzania obrazu na klastrze testowym i laptopie średniej klasy (procesor Intel Core i7, 8 GB pamięci RAM, dysk SSD). Próby obróbki grafiki za pomocą oprogramowania oferującego interfejs graficzny, przeprowadzone przy użyciu programu GIMP[4], zakończyły się niepowodzeniem – obsługa aplikacji nie była możliwa po wczytaniu tak dużego obrazu, konieczne więc stało się wykorzystanie narzędzia pracującego w trybie tekstowym. Dla obrazów o rozmiarze prze-

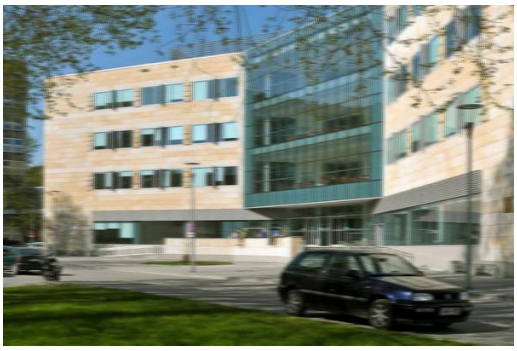
[4] `https://www.gimp.org/`

MOST WIEDZY Downloaded from mostwiedzy.pl

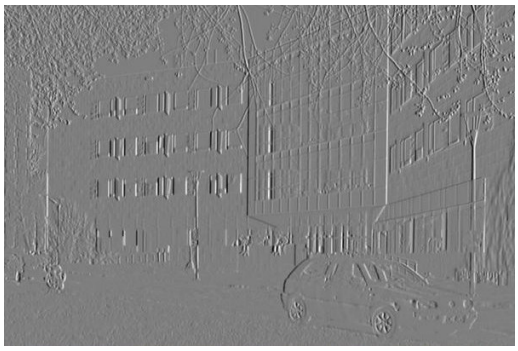
kraczającym 300 megapikseli, czas działania popularnej aplikacji ImageMagick[5] był około trzykrotnie dłuższy niż czas działania zaproponowanej aplikacji napisanej w standardzie MPI I/O (średnio 94 s lokalnie i 32 s na klastrze). A im bardziej złożone algorytmy przetwarzania i większy rozmiar obrazu, tym większy zysk z dalszego zrównoleglania obliczeń. Szersze uzasadnienie istotności tematu, szczegółowy opis architektury oraz kompletne testy wydajnościowe dla tej aplikacji można znaleźć w artykule [75].

(a) Zdjęcie niezmodyfikowane

(b) Rozmycie



(c) Rozmycie wieloprzebiegowe



(d) Operator Sobela

Rysunek 6.4: Zdjęcie oryginalne i przetworzone przy użyciu zaimplementowanych filtrów *(fotografia: Krzysztof Krzempek)*

Aplikacja traktuje plik obrazu tak jak macierz dwuwymiarową, w której każdy element reprezentuje trzy składowe koloru piksela. Macierz podzielona jest na podmacierze, a każda podmacierz przyporządkowana do konkretnego procesu MPI. Do takiej struktury podłącza się filtry operujące na danym fragmencie – na potrzeby eksperymentów zaimplementowano trzy filtry:

- rozmycie realizowane przez uśrednienie wartości sąsiadujących pikseli,

[5] `https://www.imagemagick.org/`

MOST WIEDZY Downloaded from mostwiedzy.pl

- wieloprzebiegowe rozmycie – filtr podobny do powyższego, jednak uśredniający wartość wielokrotnie,
- operator Sobela – popularny algorytm wykorzystywany do wykrywania krawędzie w obrazie.

Działanie zaimplementowanych filtrów prezentuje rysunek 6.4.

Tradycyjne podejście do przetwarzania obrazów zakładałoby, że w ramach węzła wczytywany jest pewien fragment obrazu do pamięci RAM na początku działania programu, następnie następuje właściwe przetwarzanie, a ostatecznie dane są zapisywane w pliku wynikowym. Takie podejście zakłada jednak, że rozmiar obrazu przekonwertowanego na macierz pikseli jest mniejszy od sumarycznej pamięci RAM zainstalowanej w klastrze. Sytuacja zmienia się, kiedy rozmiar pliku rośnie. Im większy rozmiar pliku, tym mniejsza część może zostać przetworzona w ramach jednej iteracji, rośnie więc liczba żądań odczytu i zapisu do pliku. Teoretycznie, przy odpowiednio dużym obrazie komunikacja z plikiem zaczyna być wąskim gardłem całego przetwarzania.

Przy tworzeniu tej aplikacji demonstrującej miano na celu pokazanie, że możliwe jest napisanie prostego programu, skupiającego się na logice właściwej problemowi, a wydajność pracy z plikiem pozostawić w głównej mierze zaproponowanej pamięci podręcznej. Odczyt i zapis przetwarzanych fragmentów odbywa się więc przy użyciu bardzo prostego, pojedynczego, liniowego bufora. Jeśli żądany fragment znajduje się w buforze, wartość pobierana jest bezpośrednio z niego, w przeciwnym razie bufor wypełniany jest blokiem danych rozpoczynającym się od zadanego fragmentu. Aplikację zoptymalizowano do obsługi sekwencyjnego przetwarzania większej liczby obrazów. Optymalizacja polega na nakładaniu się otwierania i zamykania jednego pliku z przetwarzaniem drugiego, co pozwala zniwelować narzut na wspomniane operacje, charakterystyczny dla zaproponowanej pamięci podręcznej.

Wszystkie eksperymenty oparte o przetwarzanie obrazów wykonano na klastrze Lap06, na każdym węźle uruchomiono piętnaście procesów MPI. Rysunek 6.5 pokazuje wyniki testów wydajnościowych.

Na pierwszym wykresie (6.5a) uzasadniono wybór wielkości bufora dla pozostałych testów. Eksperyment został przeprowadzony dla filtru rozmycia i obrazu o rozdzielczości 500 megapikseli. O ile sam rozmiar bufora nie miał znaczącego wpływu na działanie aplikacji z użyciem pamięci podręcznej opartej o NVRAM, o tyle nieodpowiednio ustawiony dra-

MOST WIEDZY Downloaded from mostwiedzy.pl

stycznie pogarszał wydajność w konfiguracji opartej o standardowy stos technologiczny. Uczciwe porównanie wymagało więc wybrania rozmiaru bufora korzystnego dla niezmodyfikowanego MPI I/O. Rozmiar bufora ustawiono więc na 512 KB, ponieważ zwiększanie jego rozmiaru powyżej tej wartości przestało mieć wpływ na czas działania obu wersji aplikacji.

Drugi wykres (6.4d) prezentuje wyniki uzyskane dla algorytmu wykrywania krawędzi i gigapikselowej grafiki. Dla dużych obrazów można zaobserwować ponad pięćdziesięcioprocentową redukcję czasu działania aplikacji po rozszerzeniu standardowej konfiguracji o zaproponowaną pamięć podręczną. Tak duży zysk wydajnościowy związany jest z wysoką częstotliwością zapisów i odczytów (sam algorytm nie jest trudny obliczeniowo), a także równomiernym obciążeniem wszystkich zarządców pamięci podręcznej (plik obrazu jest podzielony symetrycznie pomiędzy procesy).

Seria kolejnych trzech wykresów (6.5c, 6.5d, 6.5e) została umieszczona w celu omówienia wpływu poszczególnych parametrów pamięci NVRAM na ogólną wydajność aplikacji. Eksperymenty zostały przeprowadzone dla filtru implementującego wieloprzebiegowe rozmycie, proces rozmycia powtórzono dziesięciokrotnie. W testowanej aplikacji największy wpływ na wydajność miało opóźnienie w dostępie do danych. Zmniejszenie opóźnienia każdego żądania z 600 ns do 400 ns spowodowało spadek czasu działania aplikacji średnio o 6,5%. Ze względu na specyfikę algorytmu (więcej odczytów niż zapisów), zmiany dodatkowego opóźnienia zapisu do NVRAM nie wpłynęły tak znacznie na wydajność. Dwukrotne obniżenie dodatkowego narzutu na zapis zredukowało czas działania aplikacji średnio o 2%. Niewielki rozmiar żądań do pamięci związany jest również z niewielkim wpływem przepustowości pamięci na aplikację. W omawianym przypadku czterokrotne zwiększenie przepustowości wiązało się ze średnim zmniejszeniem czasu działania aplikacji jedynie o 2,5%. Biorąc pod uwagę duże różnice wartości parametrów pamięci NVRAM, można stwierdzić, że w przetestowanym zakresie parametry pamięci NVRAM mają stosunkowo niewielki wpływ na wydajność zaprezentowanej pamięci podręcznej.

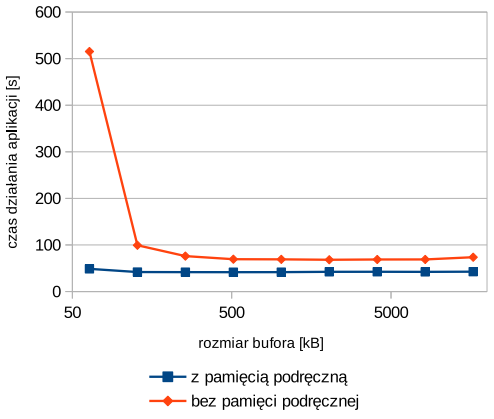
Ostatni zaprezentowany test omawianej aplikacji, pokazany na wykresie 6.5f, ilustruje sposób ograniczenia narzutu na otwieranie i zamykanie pliku. Aplikacja potrafi obsługiwać kolejne obrazy sekwencyjnie w taki sposób, w którym już podczas przetwarzania konkretnego obrazu, do pamięci podręcznej wczytywany jest następny. Należy liczyć się z tym, że stosując taką metodę, narażamy aplikację na negatywne skutki jednoczesnego dostępu do pamięci NVRAM przez dwa wątki zarządcy pamięci podręcznej (jeden wątek

MOST WIEDZY Downloaded from mostwiedzy.pl

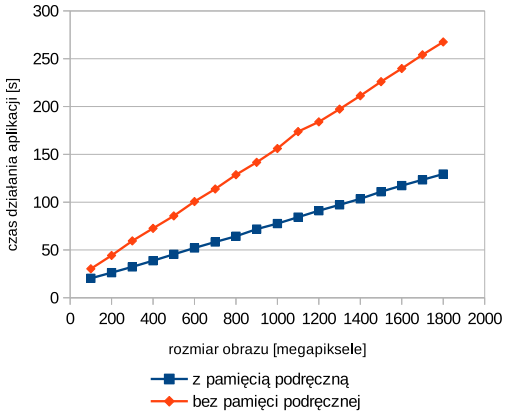
na jeden plik). W większości przypadków etap wczytania i zapisania całego pliku przy wykorzystaniu PFS jest jednak na tyle wąskim gardłem, że należy się spodziewać pozytywnego wpływu takiego mechanizmu na czas działania aplikacji. Dla testowanej aplikacji eksperymenty przeprowadzono korzystając z filtru dziesięciokrotnego rozmycia i obrazu o rozmiarze jednego gigapiksela. W omówionym scenariuszu, nakładanie się obliczeń z operacją otwarcia i zamknięcia pliku pozwoliło zwiększyć wydajność przetwarzania o około 10%.

Podsumowując eksperymenty, zaproponowana pamięć podręczna oparta o NVRAM znacznie zwiększyła wydajność aplikacji przetwarzającej obrazy. Parametry pamięci NVRAM w przetestowanym zakresie okazały się mieć stosunkowo niewielki wpływ na czas działania aplikacji. Narzut rozwiązania związany z operacją otwierania i zamykania pliku można w niektórych aplikacjach ograniczyć poprzez jednoczesne wykonywanie w tym czasie przetwarzania innego pliku, optymalizacja taka jest jednak możliwa jedynie w przypadku sekwencyjnego przetwarzania co najmniej dwóch plików.

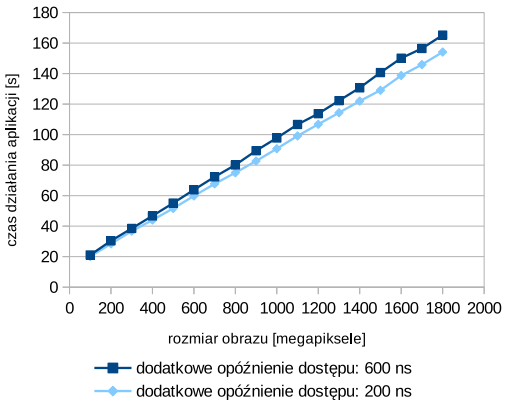
MOST WIEDZY Downloaded from mostwiedzy.pl



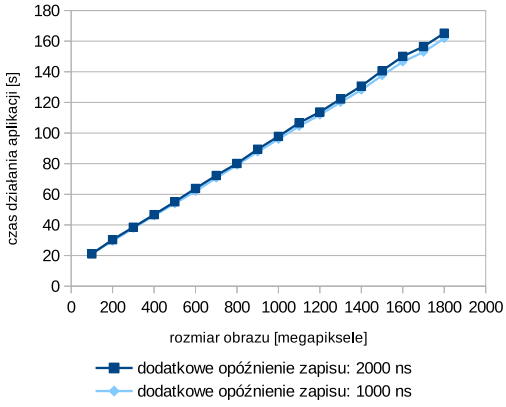
(a) Pojedyncze rozmycie



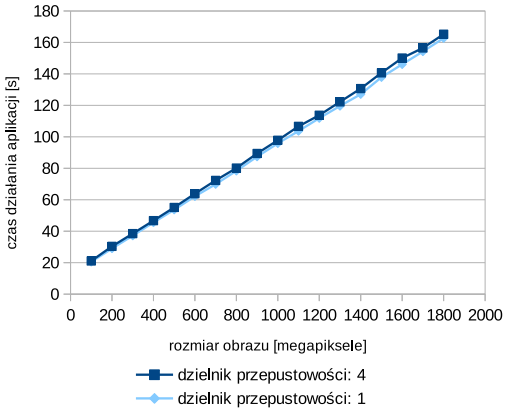
(b) Wykrywanie krawędzi



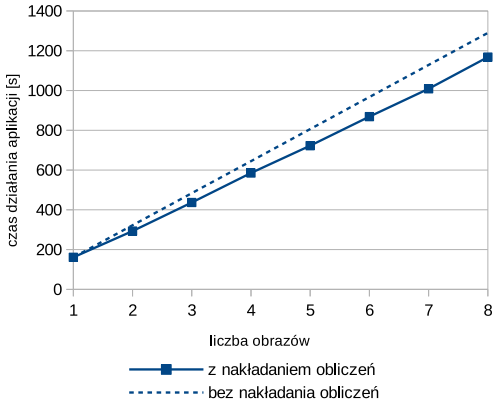
(c) Wieloprzebiegowe rozmycie



(d) Wieloprzebiegowe rozmycie



(e) Wieloprzebiegowe rozmycie



(f) Wieloprzebiegowe rozmycie

Rysunek 6.5: Przetwarzanie dużych obrazów – wyniki eksperymentów

MOST WIEDZY Downloaded from mostwiedzy.pl

6.6 Wyznaczanie potęgi grafu

Kolejną testowaną aplikacją jest wyznaczanie n-tej potęgi grafu. N-ta potęga grafu G to graf o takim samym zbiorze wierzchołków, dla których dwa wierzchołki łączy krawędź, jeśli ich odległość w grafie G wynosi co najwyżej n. Praktyczne zastosowanie algorytmu to między innymi wyznaczanie liczby trójkątów w grafie, znajdowanie długości najkrótszej ścieżki czy znajdowanie liczby ścieżek o zadanej długości, łączących dwa wierzchołki. Przenosząc problem grafowy do programu komputerowego, można go sprowadzić do wielokrotnego mnożenia macierzy sąsiedztwa takiego grafu przez samą siebie. Zakładając, że A jest macierzą sąsiedztwa, w macierzy A^n każdy element $a_{n(i,j)}$ reprezentuje liczbę ścieżek o długości n łączących wierzchołek i z wierzchołkiem j. Algorytm ilustrują poniżej macierze A, A^2 i A^3:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}, A^2 = A \cdot A = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}, A^3 = A \cdot A \cdot A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

Z powyższych macierzy można wyczytać między innymi następujące informacje:

- istnieją dwie ścieżki o długości 2, które łączą wierzchołek 4 z wierzchołkiem 3 ($a_{2(4,3)} = 2$),
- najkrótsza ścieżka, łącząca wierzchołek 3 z wierzchołkiem 1, ma długość 2 (najmniejsze n dla którego $a_{n(3,1)} > 0$ wynosi 2),
- liczba trójkątów w grafie wynosi 2 ($\frac{1}{3}\sum_{i=1}^{v} a_{3(i,i)} = 2$).

Jeśli chodzi o implementację mnożenia macierzy w środowisku równoległym, wykorzystano stosunkowo prosty algorytm Cannona [9]. Brak porównania zaimplementowanego podejścia z istniejącymi rozwiązaniami jest spowodowany:

- brakiem znanych rozwiązań w których komunikacja międzyprocesowa opiera się plik,
- powszechnym stosowaniem nieco innych algorytmów, takich jak SUMMA (*Scalable Universal Matrix Multiplication Algorithm*), który jest znany między innymi z implementacji w popularnym pakiecie LAPACK[6],

[6] `http://www.netlib.org/lapack/`

MOST WIEDZY Downloaded from mostwiedzy.pl

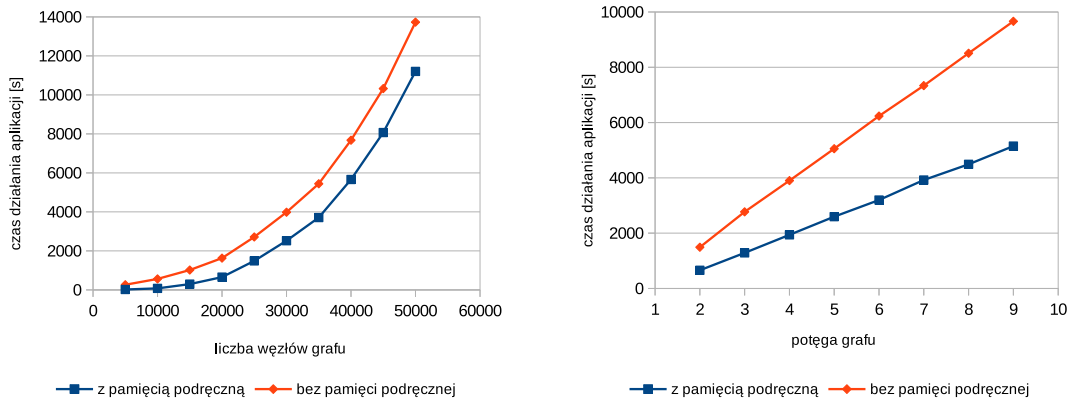
- wysoką wydajnością dojrzałych rozwiązań uzyskaną dzięki skomplikowanym optymalizacjom – zadaniem proponowanej pamięci podręcznej jest zwiększanie wydajności operacji plikowych, a nie konkurowanie z wysoce optymalnymi technikami wypracowanymi przez lata, które operacje plikowe ograniczają do minimum.

Bardziej szczegółowy opis aplikacji wraz z zastosowanymi optymalizacjami i kompletne testy wydajnościowe, obejmujące testy skalowalności, znajdują się w artykule [74].

Testy, dla których wyniki zaprezentowano na rysunku 6.6, przeprowadzono na klastrze Lap06 przy piętnastu procesach na każdym węźle. Rezultaty dla różnych rozmiarów grafu liczonych jako liczba węzłów (wykres 6.6a) wyznaczono dla grafu podniesionego do kwadratu. Wyniki dla różnych potęg grafu (wykres 6.6b) zebrano dla grafu zawierającego 20 000 węzłów.

Testowana aplikacja nie spełnia zalecanych właściwości, dla których zastosowanie pamięci podręcznej opartej o NVRAM gwarantuje poprawę wydajności. Teoretycznie, dla niskich potęg liczba żądań do pliku jest stosunkowo niewielka, przez co narzut na otwarcie i zamknięcie pliku powinien pochłaniać dużą część czasu działania aplikacji. Dodatkowo, procesy czytają dane dużymi blokami, a rozmiar bloku rośnie wraz ze wzrostem rozmiaru grafu. Pomimo takiej charakterystyki aplikacji, zaproponowane rozwiązanie poprawia wydajność nawet dla dużych grafów i niskiej potęgi, co pokazano na wykresie 6.6a. Dla grafu o rozmiarze 50 000 węzłów rozszerzenie MPI I/O zmniejszyło czas działania aplikacji o 18%, ogólnie redukcja czasu działania jest odwrotnie proporcjonalna do rozmiaru grafu. Jak już wcześniej wielokrotnie wspomniano, celem rozwiązania jest poprawa wydajności głównie podczas przetwarzania dużego wolumenu żądań o niewielkim rozmiarze. W przypadku dużych grafów, algorytm Cannona dzieli dane na stosunkowo duże, ciągłe fragmenty. Im większy graf, tym więcej danych przetwarzanych w ramach pojedynczej operacji, co skutkuje malejącą procentową redukcją czasu działania aplikacji przy rosnącym rozmiarze grafu. Wykres 6.6b ilustruje natomiast czas działania aplikacji dla kolejnych potęg grafu. Szereg optymalizacji wdrożonych do rozwiązania, takich jak na przykład zrównoleglenie otwierania i zamykania pliku, pozwolił utrzymać redukcję czasu działania aplikacji na podobnym poziomie niezależnie od potęgi grafu.

MOST WIEDZY Downloaded from mostwiedzy.pl



(a) (b)

Rysunek 6.6: Wyznaczanie potęgi grafu – wyniki eksperymentów

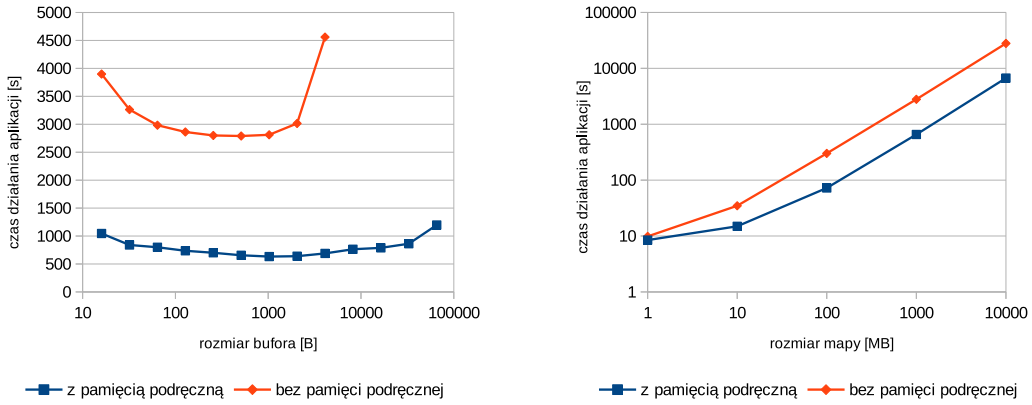
6.7 Przeszukiwanie dwuwymiarowej mapy

Przeszukiwanie dwuwymiarowej mapy to aplikacja, w której połączono równomierne, sekwencyjne przeglądanie pliku ze zintensyfikowanym dostępem w konkretnej lokalizacji. Zaimplementowany algorytm składa się z dwóch etapów. W pierwszym każdy proces przeszukuje swój fragment mapy w poszukiwaniu określonego wzorca. Jeśli znaleziono obszar spełniający zadane kryteria, proces zaczyna przeszukiwać jego najbliższe sąsiedztwo ograniczone określonym promieniem, zaznacza i zlicza wystąpienia wzorca. Podstawowym wyróżnikiem tej aplikacji jest wysoki stosunek liczby żądań odczytu do liczby żądań zapisu. Przykładem praktycznych zastosowań zaprezentowanego podejścia może być badanie rozprzestrzeniania się chorób w inteligentnym rolnictwie albo zastosowania militarne. Zasadność wykorzystania technik HPC do tego problemu jest podobna do przypadku wcześniej omówionego przetwarzania dużych obrazów – zrównoleglanie jest tym bardziej opłacalne, im bardziej złożony obliczeniowo jest algorytm przetwarzania konkretnego obszaru pliku. Aplikacja została także krótko omówiona i przetestowana w artykule [78].

Przeszukiwanie mapy jest drugą aplikacją opisaną w niniejszej rozprawie, która wykorzystuje bufor znacznie zwiększający wydajność tradycyjnego stosu technologicznego. W tym przypadku największy zysk z implementacji bufora przypada na pierwszy etap przetwarzania pliku. Testy przeprowadzono na klastrze Lap06 przy piętnastu procesach uruchomionych na każdym węźle. Wykres zaprezentowany na rysunku 6.7a uzasadnia ustawienie rozmiaru bufora na 512 B. Wykres pokazany na rysunku 6.7b ilustruje wy-

MOST WIEDZY Downloaded from mostwiedzy.pl

niki wydajnościowe aplikacji dla różnych rozmiarów plików. Odmienny wzorzec dostępu do pliku, w którym większość żądań to operacje odczytu bardzo wydajnie obsługiwanych w niezmodyfikowanym MPI I/O, nie zmniejszył zysku wydajnościowego z zaproponowanego rozwiązania. Dla największych rozmiarów plików (1 GB i 10 GB) redukcja czasu działania programu osiągnęła poziom 75%.



(a) (b)

Rysunek 6.7: Przeszukiwanie dwuwymiarowej mapy – wyniki eksperymentów

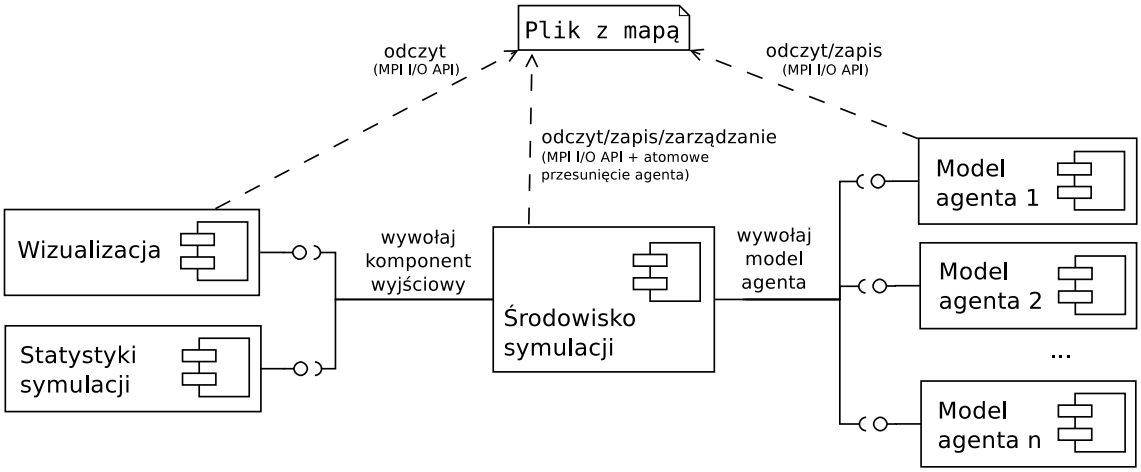
W tym przypadku tak dobre wyniki nie są jedynie skutkiem wysokiej wydajności zaproponowanego rozwiązania. Niewielka złożoność operacji działających na pobranych fragmentach danych spowodowała bardzo dużą częstotliwość żądań wysyłanych do systemu plików, a dwa węzły, na których uruchomiono PFS, nie były w stanie poradzić sobie z obciążeniem. W przedstawionej pamięci podręcznej obsługą wysokiego wolumenu żądań zajmują się wszystkie węzły klastra, co w omawianym scenariuszu pozwoliło rozładować obciążenie.

6.8 Symulacja zachowania tłumu

Omówiona aplikacja prezentuje agentowe podejście do symulacji zachowania tłumu. Wymaganiami, jakie przed nią zostały postawione, były nie tylko wysoka wydajność i skalowalność, ale również – tak jak w przypadku aplikacji do przetwarzania obrazów – prosty model programistyczny, skupiający się głównie na dziedzinie problemu. Utworzenie aplikacji dedykowanej na klastry jest w tym przypadku uzasadnione zarówno skalą problemu – tysiące agentów, duży obszar symulacji, wysoka rozdzielczość czasowa i przestrzenna – jak również stosunkowo wysoką skalowalnością systemów agentowych. Jeśli chodzi o projekt

MOST WIEDZY Downloaded from mostwiedzy.pl

rozwiązania, to zamiast tworzenia wyłącznie pojedynczej aplikacji, przygotowano szablon, który pozwala zdefiniować przez programistę własne reguły symulacji. Architektura aplikacji składa się z trzech typów komponentów: środowiska symulacji, modelu agenta i komponentu wyjściowego. Rysunek 6.8 pokazuje przykładową konfigurację komponentów wraz z zależnościami.



Rysunek 6.8: Architektura, pokazująca komponenty składowe aplikacji, która symuluje zachowanie tłumu

Środowisko symulacji to główny moduł aplikacji. Symulacja opiera się o mapę, złożoną z pojedynczych pól, reprezentujących fragment przestrzeni o wymiarach 40 cm na 40 cm, co według literatury powinno być minimalną przestrzenią zajmowaną przez pojedynczego agenta [29]. Każde pole ma stały rozmiar (w przypadku konfiguracji testowej były to 4 bajty) i służy do przechowania stanu złożonego z informacji, takich jak obecność agenta, jego właściwości widoczne dla innych agentów (na przykład wiek, płeć, flaga świadcząca o niepełnosprawności) czy typ pola (na przykład ściana budynku, zbiornik wodny, chodnik, trawnik). Środowisko symulacji nie definiuje w żaden sposób formatu pola, programista korzystający z zaprojektowanej architektury musi więc zarejestrować maski bitowe, które potrafią określić, czy pole jest wolne i można na nie przesunąć agenta. Mapa jest również jedynym sposobem na przekazanie informacji do innych agentów.

Pierwszym zadaniem środowiska symulacji jest załadowanie mapy. Program wczytuje plik zawierający mapę obszaru, a następnie przyporządkowuje do każdego procesu MPI pewną liczbę niezależnych agentów, które reprezentują ewakuujących się ludzi. Następnie, w kolejnych iteracjach, dla każdego agenta komponent wywołuje implementację modelu agenta, przekazując mu bieżącą lokalizację i stan, a następnie oczekuje na listę pozycji, na

MOST WIEDZY Downloaded from mostwiedzy.pl

które agent chciałby się przesunąć. Jeśli ruch jest możliwy (pole nie zostało przed chwilą zajęte przez innego agenta), moduł zmienia odpowiednio zapis o aktualnej pozycji. Po odpytaniu wszystkich agentów w miarę potrzeby uruchamiane są komponenty wyjściowe. Samo MPI I/O oferuje jedynie podstawowy zakres operacji na pliku. Wydajna implementacja symulacji wymagała jednak dodatkowej operacji, pozwalającej na przesunięcie agenta na konkretne miejsce (a więc zapisu pewnego fragmentu pliku) jedynie w sytuacji, kiedy wybrane miejsce nie zostało wcześniej zajęte przez innego agenta. Proponowana pamięć podręczna jest stosunkowo łatwo rozszerzalna, dopisano więc dodatkowo brakującą operację.

Model agenta to bezstanowy komponent, kontrolujący zachowanie pojedynczego agenta. Technicznie sprowadza się do funkcji wywołanej przez środowisko symulacji, do której przekazywana jest bieżąca lokalizacja agenta i jego stan. Dzięki temu, że stan przesyłany jest przez referencję, agent może zapisać w nim dowolne, istotne dla siebie informacje, które ponownie otrzyma w kolejnej iteracji. Każdy agent ogląda świat symulacji (w tym lokalizuje innych agentów) jedynie przez odczyt fragmentów pliku sąsiednich dla jego bieżącej lokalizacji. Jego wpływ na symulację ogranicza się do możliwości zmiany miejsca, albo zapisu pewnej części swojego stanu w pliku w taki sposób, żeby był widoczny dla innych agentów. Architektura aplikacji pozwala na zdefiniowanie dowolnej liczny modeli i przypisania każdemu dowolnej liczby agentów. W przedstawionych eksperymentach wykorzystano pojedynczą, bardzo prostą implementację modelu agenta: agent zmierza do celu w linii prostej i w miarę możliwości związanych z charakterystyką terenu i innymi agentami, omija napotkane przeszkody. Żeby symulować bardziej złożone operacje, przy każdym ruchu agent przez milisekundę wykonuje dodatkowe operacje.

Istotą zaimplementowanego symulatora była możliwość prześledzenia zachowania agentów podczas ewakuacji. Do zapisu danych wyjściowych symulacji służą komponenty wyjściowe, uruchamiane cyklicznie przez środowiska symulacji co określoną liczbę iteracji. Ze względu na to, że stan całej aplikacji przechowywany jest w pliku, parametrami wejściowymi komponentów są jedynie uchwyt do pliku i rozmiar mapy. Do cyklicznego zapisu kolejnych stanów całej mapy wykorzystano mechanizm synchronizacji pamięci podręcznej z PFS i jego tryb tworzenia przy każdej synchronizacji kolejnych wersji pliku. Rysunek 6.9 prezentuje fragment wizualizacji przykładowej symulacji testowej, powstały z połączenia pliku zlokalizowanego w PFS z odpowiadającą grafiką z mapą.

Szczegółowa architektura aplikacji i szerszy zestaw eksperymentów, ze szczególnym

MOST WIEDZY Downloaded from mostwiedzy.pl

Rysunek 6.9: Grafika prezentująca symulację zachowania tłumu; niebieskie punkty reprezentują agentów

uwzględnieniem zabezpieczenia danych przed utratą na wypadek awarii, zostały opisane w artykułach [71, 76].

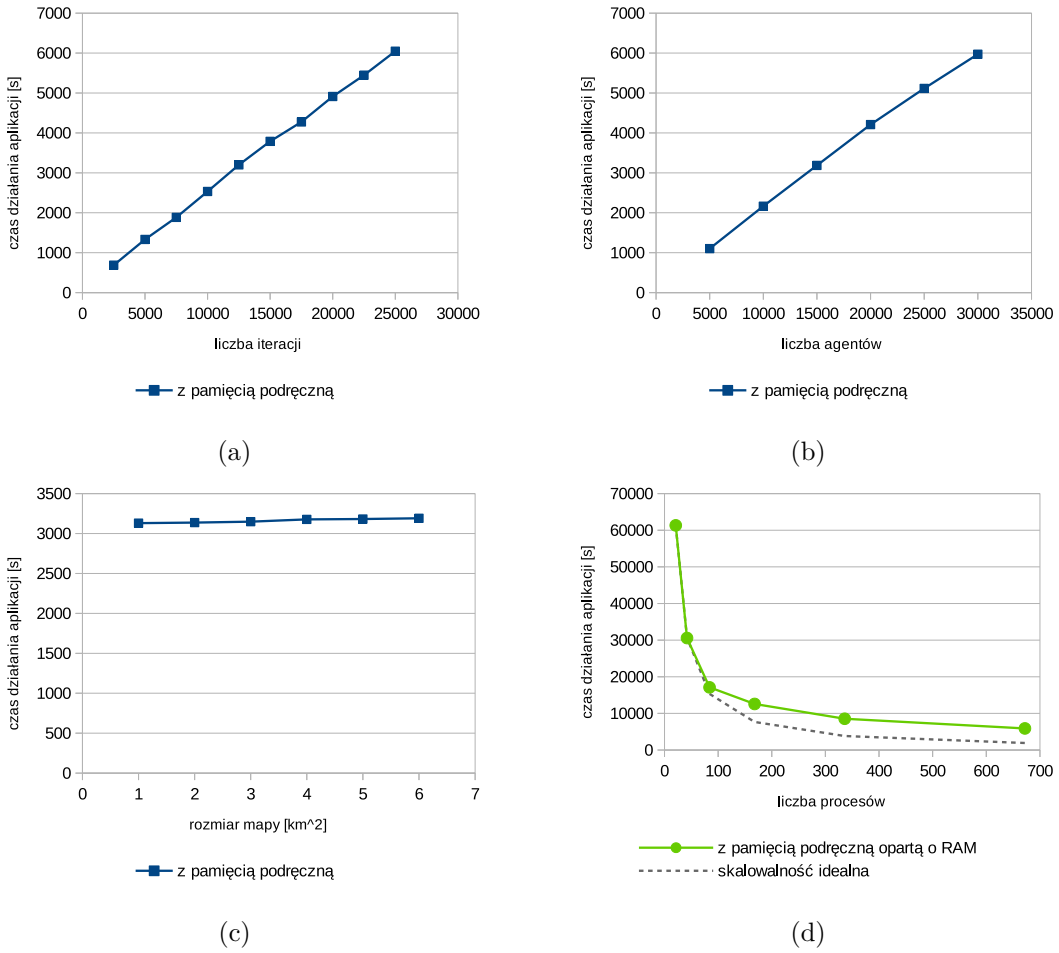
Wyniki dla aplikacji gromadzi rysunek 6.10. Trzy pierwsze eksperymenty przeprowadzono przy wykorzystaniu klastra Lap06 i piętnastu procesów na każdym węźle, czwarty test został uruchomiony na klastrze K2 i siedmiu procesach na każdym węźle. Jeśli parametr nie był zmienną w danym scenariuszu testowym, konfiguracja scenariuszy przedstawiała się następująco:

- 12 500 iteracji symulacji, co odpowiadało około godzinie symulowanej ewakuacji,
- mapa o rozmiarze 6 km^2, przedstawiająca fragment obszaru miasta Gdańsk,
- 15 000 agentów w przypadku testów na Lap06 i 60 000 agentów dla testów na K2, co odpowiadało rzeczywistej liczbie mieszkańców symulowanego fragmentu miasta.

Na wykresach widać, że czas działania aplikacji był liniowo zależny od liczby iteracji i liczby agentów. Niewielki wpływ zmiany rozmiaru mapy na wydajność spowodowany jest długim działaniem aplikacji w stosunku do wielkości pliku. Dobrą skalowalność rozwiązania pokazano przy użyciu klastra zawierającego około stu węzłów. Ze względu na dodatkową operację, zaimplementowaną na poziomie pamięci podręcznej, nie jest możliwe bezpośrednie porównanie wydajności aplikacji z niezmodyfikowanym MPI I/O. Ewentualna implementacja takiej operacji w standardowym stosie technologicznym wymagałaby albo bardzo niewydajnego sekwencyjnego dostępu do pliku, albo bardziej złożonej logiki,

MOST WIEDZY Downloaded from mostwiedzy.pl

co kłóciłoby się z koncepcją współpracy z plikiem w sposób wygodny dla programisty. Zdecydowano się więc porównać wyniki z istniejącymi systemami symulującymi zachowanie tłumu.



Rysunek 6.10: Symulacja zachowania tłumu – wyniki eksperymentów

W chwili tworzenia rozwiązania, typowe symulacje tłumu oparte o podejście agentowe były zdolne modelować w rozsądnym czasie zachowanie setek jednostek [52], niektóre publikacje opisywały rozwiązania, pozwalające na obsługę nawet 15 000 agentów [29]. W większości przypadków powierzchnia ewakuacji była ograniczona do części miasta albo budynku [29], zajmowano się też specyficznymi lokalizacjami takimi jak statki [5]. Zaproponowane rozwiązanie pokazywało stosunkowo prostą i modularną architekturę, opartą o przetwarzanie pliku, która pokazywała, jak przy uproszczonym podejściu agentowym symulować dziesiątki tysięcy agentów na obszarze małego miasta. Dużą zaletą opisanej metody jest też dobra skalowalność.

MOST WIEDZY Downloaded from mostwiedzy.pl

Najnowsze pozycje w temacie agentowych symulacji tłumu mogą się pochwalić wynikami bardziej korzystnymi od zaproponowanego rozwiązania – programy stworzone na potrzeby planowania ewakuacji podczas tsunami były zdolne przetworzyć od 25 000 agentów na obszarze 20 km^2 [68] do nawet 2 milionów na obszarze 81 km^2 [1]. Proponowana architektura w dalszym ciągu jednak pozostaje użyteczna ze względu na modularność, prostotę przetwarzania i duży nacisk położony na zabezpieczenie danych przed utratą w przypadku awarii.

6.9 Narzut mechanizmów ograniczających skutki awarii

Eksperymenty opisane w tym podrozdziale, związane z mierzeniem narzutu czasowego mechanizmów dbających o zabezpieczenie danych przed utratą, mają dwa główne cele. Zadaniem pierwszych trzech wykresów jest pokazanie, że rozwiązanie potrafi działać wydajnie w środowisku podatnym na awarie, co jest dobrze widoczne, zwłaszcza przy jednoczesnym porównaniu z aplikacją uruchomioną na środowisku bez zainstalowanej pamięci podręcznej. Drugi, bardziej obszerny zestaw testów, ma za zadanie pokazać wpływ parametrów konfiguracyjnych mechanizmów bezpieczeństwa danych na czas działania aplikacji.

Do testów wykorzystano aplikacje omówione już wcześniej: skonfigurowano je natomiast z innym zestawem parametrów, kolejno:

- dla błądzenia losowego (rysunek 6.11a, 6.11d, 6.11e i 6.11f) znaczenie zwiększono czas trwania pojedynczej iteracji, co pozwoliło wydłużyć czas działania aplikacji i zasymulować bardziej kosztowne obliczenia, lepiej uzasadniające potrzebę zastosowania mechanizmów ochrony danych przed awarią;
- aplikację wyznaczającą kolejne potęgi grafu (rysunek 6.11b) zmodyfikowano w taki sposób, żeby nadpisywała swoje poprzednie etapy obliczeń, ponieważ kolejne wersje pliku z macierzą można w razie potrzeby odtworzyć z kopii zapasowych;
- jako algorytm przetwarzający obrazy (rysunek 6.11c) wykorzystano wieloprzebiegowe rozmycie, zwiększono jednak liczbę przebiegów algorytmu, żeby wydłużyć czas działania aplikacji.

Na wykresach przez „potrójną ochronę danych” rozumiane jest włączenie wszystkich trzech wcześniej opisanych mechanizmów, zapewniających możliwość odtworzenia pliku w przypadku awarii. Jeśli nie zaznaczono inaczej, kopia zapasowa w ramach dziennika żądań

MOST WIEDZY Downloaded from mostwiedzy.pl

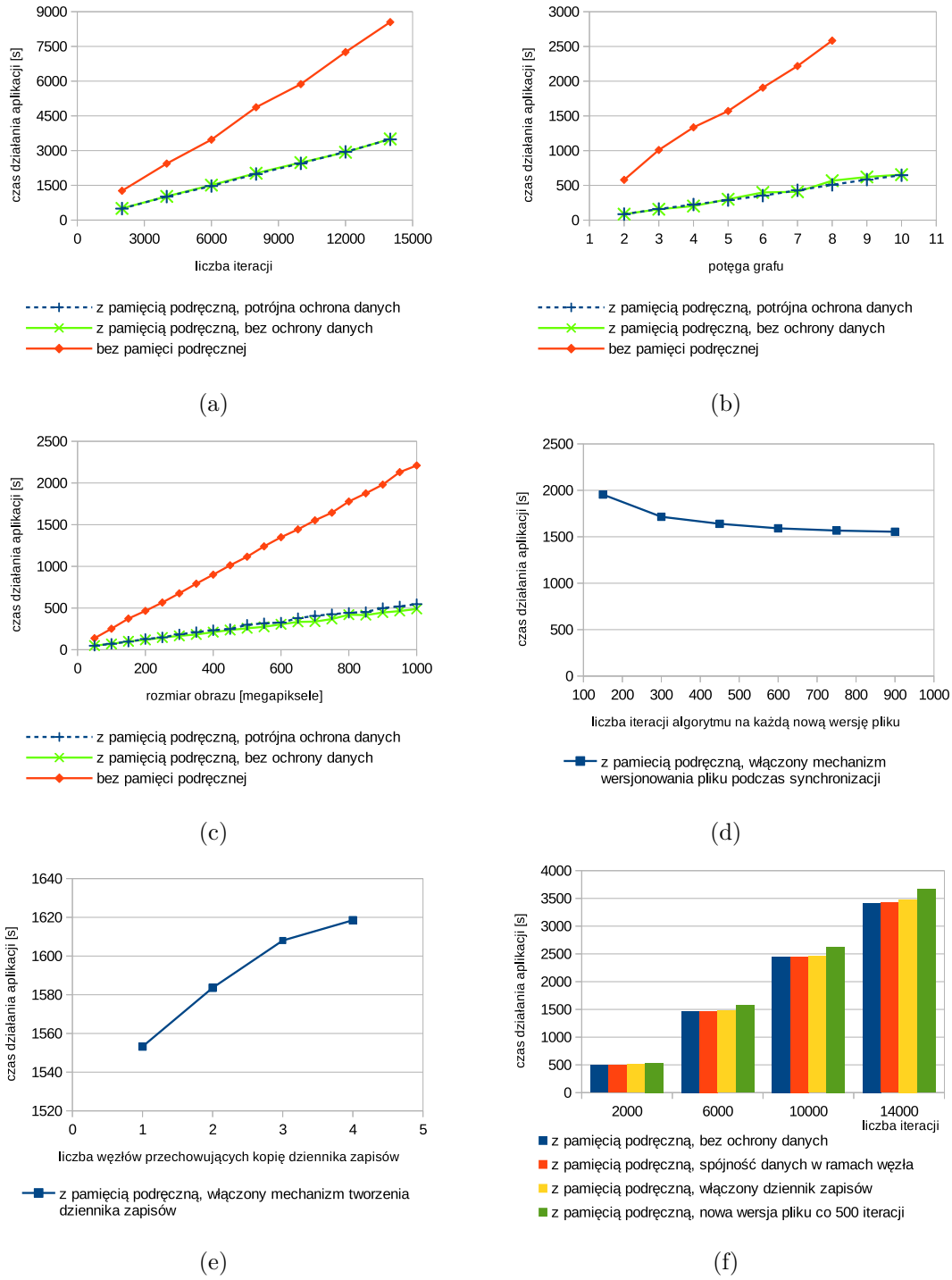
zapisów była przechowywana na jednym redundantnym węźle. Synchronizacja danych obywała się dla benchmarku, opartego o błądzenie losowe co 2000 iteracji, przy przetwarzaniu grafów po wyznaczeniu każdej kolejnej potęgi grafu, natomiast podczas pracy z obrazami po każdym kolejnym przebiegu filtru rozmycia.

Na rysunkach 6.11a i 6.11b wyraźnie widać, że dla wybranych konfiguracji narzut wszystkich trzech mechanizmów, dbających o ochronę danych, w przypadku awarii jest pomijalny. W przypadku trzeciej aplikacji, której działanie ilustruje wykres pokazany na rysunku 6.11c, narzut czasowy jest zdecydowanie obserwowalny – powodem takiego zachowania jest dużo wyższa intensywność operacji plikowych niż w przypadku poprzednich aplikacji. Przeprowadzone eksperymenty pokazały, że czas działania w przypadku aplikacji przetwarzającej obrazy wzrósł maksymalnie o około 20%, co można uznać za akceptowalne, zwłaszcza w porównaniu do wersji uruchomionej bez udziału zaproponowanego rozwiązania.

Rysunek 6.11d pokazuje wpływ częstotliwości tworzenia nowych wersji pliku na czas działania aplikacji. Początkowy szybki spadek pozwala na ustawienie stosunkowo częstego zapisu do pliku przy utrzymaniu wysokiej wydajności. Rysunek 6.11e ilustruje natomiast wzrost czasu działania aplikacji z każdym kolejnym węzłem przechowującym redundantną kopię zapasową przeprowadzonych zapisów. Chociaż wzrost może wydawać się duży, należy wziąć pod uwagę rozmiar klastra testowego – przy sześciu węzłach utrzymywanie trzech lub czterech dodatkowych kopii bezpieczeństwa wydaje się dość nadmiarowe. Wykres zamieszony na rysunku 6.11f pozwala porównać narzut czasowy trzech mechanizmów ochrony danych zaproponowanego rozwiązania. Wyraźnie widać, że największy narzut związany jest z tworzeniem nowych wersji pliku podczas synchronizacji, co jest spowodowane kosztowną czasowo komunikacją z rozproszonym systemem plików.

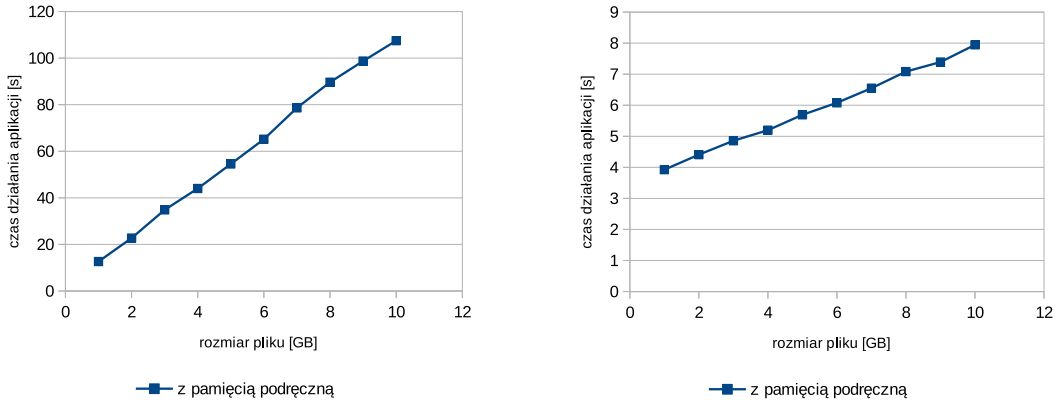
Kolejne dwa wykresy reprezentują pomiary narzutu na mechanizm tworzenia kolejnych wersji pliku i odtwarzania pamięci podręcznej z takiej wersji. Wyniki pokazują liniowy przyrost czasu tworzenia kopii (wykres 6.12a) i odtwarzania pamięci (wykres 6.12b) wraz z liniowym wzrostem rozmiaru pliku. Wyraźnie widoczne jest, że operacja odtwarzania danych jest szybsza niż operacja synchronizacji danych z systemem plików – procedura odtwarzania jest oparta o serię szybkich odczytów, podczas gdy procedura zapisu jest znacznie bardziej czasochłonna. Można tutaj również zaobserwować, że przy odpowiedniej konfiguracji systemu plików, czas działania jest ograniczony jedynie możliwościami zainstalowanego sprzętu. W analizowanym przykładzie szybkość odtwarzania wyniosła około

MOST WIEDZY Downloaded from mostwiedzy.pl



Rysunek 6.11: Wpływ mechanizmów ochrony danych na wydajność – wyniki eksperymentów

MOST WIEDZY Downloaded from mostwiedzy.pl



(a) (b)

Rysunek 6.12: Czas tworzenia kolejnych wersji pliku i odtwarzania pamięci podręcznej – wyniki eksperymentów

1 GB/s, co jest równe podwójnej prędkości odczytu dysku SSD (w klastrze użyto dwóch węzłów odpowiedzialnych za system plików). Podobna zależność pojawia się w przypadku zapisu kopii pliku.

Podsumowując, podczas eksperymentów weryfikujących wpływ na wydajność mechanizmów chroniącymi dane przed utratą podczas awarii, czas działania aplikacji w skrajnych przypadkach nie wydłużył się o więcej niż około 20% w porównaniu do wersji bez mechanizmów ochronnych. W wielu przypadkach jednak narzut był niezauważalny. Zdaniem autora takie wyniki są akceptowalne, zwłaszcza, że nawet w pesymistycznych przypadkach wykazano, że zastosowanie zaproponowanego rozwiązania opartego o pamięć NVRAM nadal zwiększa wydajności wybranych aplikacji w porównaniu do typowego stosu technologicznego. Podczas testów zmierzono również narzut czasowy na mechanizm tworzenia kopii pliku i odtwarzania pamięci podręcznej z takiej kopii podczas awarii. Czas działania obu procedur jest zdominowany przez możliwości sprzętowe pamięci zainstalowanej w PFS.

MOST WIEDZY Downloaded from mostwiedzy.pl

Rozdział 7

Podsumowanie i dalsze kierunki prac

7.1 Podsumowanie i wnioski

Podsumowując, w niniejszej pracy przeprowadzono analizę możliwości zwiększenia wydajności wybranych aplikacji korzystających z MPI I/O przy użyciu pamięci NVRAM.

Wstęp teoretyczny rozpoczęto od nakreślenia obecnie stosowanej hierarchii pamięci, opartej głównie o SRAM, DRAM i pamięć masową, wskazano również problem zarządzania tak skomplikowaną hierarchią. Następnie, jako rozwiązanie problemu, omówiono hipotetyczną pamięć uniwersalną, oraz szereg badań, które do stworzenia takiej pamięci mogą doprowadzić. Chociaż wskazano dwie konkretne technologie, które potencjalnie mogą w niedalekiej przyszłości posłużyć do zbudowania pamięci NVRAM, w dalszej części rozprawy odnoszono się do takiej pamięci bez konkretnych szczegółów technologicznych. Niestety, aktualny stan badań wskazuje na to, że w najbliższym czasie nie będzie możliwe wykorzystanie pamięci NVRAM jako pamięci operacyjnej, a hierarcha pamięci komputerowej zostanie dodatkowo rozbudowana o kolejny poziom.

Na podstawie dostępnej literatury omówiono również aktualnie stosowane metody pracy z plikami w klastrach obliczeniowych. Przedstawiono najbardziej popularny schemat architektury oparty o równoległy system plików, do którego mają dostęp wszystkie węzły klastra. Pokazano ograniczenia takich systemów plików oraz dobre praktyki umożliwiające unikanie spadków wydajności. Ze względu na narzut na prace projektowe i programistyczne, związane często z dużą złożonością implementacji wymienionych praktyk, pokazano szereg rozwiązań, które mają na celu zapewnienie wysokiej wydajność w sposób automatyczny. Wśród nich znalazły się rozwiązania zgodne z MPI I/O, czyli narzędzia do pracy z plikami dostosowane do potrzeb aplikacji wysokiej wydajności. Ze szczególną uwagą prześledzono metody oparte o zastosowanie pamięci masowej innej niż opartej o HDD. Pokazano również, że żadne z dostępnych rozwiązań nie próbuje wykorzystać wszystkich

MOST WIEDZY Downloaded from mostwiedzy.pl

spodziewanych zalet pamięci NVRAM instalowanej we wszystkich węzłach klastra obliczeniowego.

Analizę dostępnej literatury zamyka przegląd badań związanych z zastosowaniem pamięci NVRAM w aplikacjach wysokiej wydajności. Prowadzone analizy są od lat stosunkowo optymistyczne i wskazują na duży potencjał hierarchii pamięci rozszerzonych o NVRAM w HPC, związany nie tylko ze zwiększaniem wydajności, ale i zapewnieniem większej odporności na awarie czy mniejszym zużyciem energii w porównaniu do tradycyjnej architektury. Pomimo braku rzeczywistych urządzeń na rynku, wielu badaczy zdecydowało się opracować swoje rozwiązania i przetestować je przy użyciu symulatorów pamięci NVRAM. Pojawiły się więc rozwiązania ogólne, oferujące zarządzanie nową pamięcią w sposób przezroczysty dla użytkownika, jak i biblioteki oferujące interfejs, który pozwala zarządzać nią jawnie. Pokazano również po jednym przykładzie rozwiązania dedykowanego konkretnej dziedzinie (operacje na grafach) i konkretnej klasie algorytmów (MapReduce). W jednym z artykułów zarekomendowano również gotowe narzędzia, pomagające budować własne rozwiązania. Jeśli chodzi o kwestię zużycia energii, przytoczono jedno szczegółowe badanie, z którego wynikało, że nawet przy stosunkowo prostym automatycznym zarządzaniu pamięcią NVRAM możemy się spodziewać oszczędności rzędu 20% – 40% przy niewielkim narzucie czasowym. Niektóre z wymienionych rozwiązań, jak zastosowanie symulatora czy użycie rekomendowanych narzędzi, wdrożono również w autorskim rozwiązaniu.

Większość znanych rozwiązań HPC, opartych o NVRAM, dotyczy samego przetwarzania w aplikacji, w części badań rozpatrywano jednak zagadnienie bardziej zbieżne z tematem rozprawy, czyli operacje na plikach. W niektórych artykułach pojawił się wniosek, że proste zmiany parametrów nośnika pamięci (nawet przy parametrach pamięci NVRAM zbliżonych do DRAM) nie wpłynęły znacząco na wydajność aplikacji, co oznaczałoby potrzebę weryfikacji wydajności rozwiązań dedykowanych. Jedno z rozwiązań potwierdziło zwiększenie wydajności aplikacji klastrowych operujących na plikach, ale przetestowano jedynie jedną aplikację HPC (generator danych testowych), która nie jest zgodna ze standardem MPI, więc żadne z badań nie potwierdziły postawionej w rozprawie tezy.

W chwili rozpoczęcia prac nad zagadnieniem brakowało więc dowodów na to, że zastosowanie w klastrze dodatkowej pamięci, w dodatku znacznie wolniejszej od pamięci DRAM, umożliwia zwiększenie wydajności aplikacji wykorzystujących MPI I/O. Żeby potwierdzić tezę należało więc zaproponować i przetestować autorskie rozwiązanie. W tym

MOST WIEDZY Downloaded from mostwiedzy.pl

celu zaprojektowano i zaimplementowano rozproszoną pamięć podręczną opartą o układy NVRAM, zainstalowane w każdym węźle obliczeniowym klastra. Pamięć podręczna została umiejscowiona pomiędzy aplikacją a implementacją MPI I/O, co pozwoliło zachować wszelkie optymalizacje obecne w implementacjach MPI I/O i systemie plików. Ze względu na kompatybilność narzędzia z interfejsem MPI I/O, może być ono wykorzystywane w istniejących aplikacjach bez ich modyfikacji (konieczna jest jednak ponowna kompilacja). Tworząc rozwiązanie, zadbano nie tylko o wydajność operacji na plikach ogólnie, ale zbadano także wydajność pracy rozwiązania rozszerzonego o mechanizmy zabezpieczające przetwarzane dane przed utratą w przypadku awarii. Dodatkowo, dzięki zastosowanej architekturze rozszerzono również typowe implementacje o możliwość łatwego tworzenia kolejnych wersji danego pliku.

Do przetestowania pamięci podręcznej wykorzystano sprzętowy symulator pamięci NVRAM oraz sześć następujących aplikacji:

- benchmark ROMPIO,
- benchmark – błądzenie losowe,
- przetwarzanie dużych obrazów,
- wyznaczanie potęgi grafu,
- przeszukiwanie dwuwymiarowej mapy,
- symulację zachowania tłumu.

Zgromadzone wyniki porównano do konfiguracji klastra, niezawierającej pamięci podręcznej. Różnice w czasie wykonania aplikacji jednoznacznie pokazały, że zaproponowana pamięć podręczna umożliwia zwiększenie wydajności przetestowanych aplikacji. Eksperymentach miały również na celu pokazanie warunków, dla których użycie pamięci podręcznej będzie opłacalne – najbardziej korzystna konfiguracja z perspektywy zaproponowanego rozszerzenia to taka, w której:

- dostęp do pliku jest stosunkowo intensywny,
- czas działania aplikacji jest wystarczająco długi,
- rozmiar pojedynczych żądań jest stosunkowo mały,

MOST WIEDZY Downloaded from mostwiedzy.pl

- kolejne żądania odnoszą się do różnych fragmentów plików.

Powyższe warunki są jednak zazwyczaj spełnione dla aplikacji wysokiej wydajności, które przetwarzają duże ilości danych.

Podsumowując, zaproponowane rozwiązanie i przedstawione wyniki dowodzą tezę rozprawy – zastosowanie bajtowo adresowanej pamięci NVRAM w środowisku klastrowym umożliwia zwiększenie wydajności wybranych aplikacji równoległych wykorzystujących MPI I/O. Mimo że testy rozwiązania nie pokryły wszystkich możliwych wariantów przetwarzania plików, mogących pojawić się podczas działania oprogramowania stosowanego w obliczeniach wysokiej wydajności, to zaprezentowane aplikacje zostały wybrane w taki sposób, żeby były reprezentatywne dla szerokiego spektrum scenariuszy występujących w rzeczywistych zastosowaniach. Ze względu na niedostępność na rynku urządzeń w trakcie prowadzenia badań opisanych w rozprawie, eksperymenty przeprowadzono przy użyciu symulatora. Został on jednak gruntowanie przetestowany, a część testów powtórzono, badając wpływ jego parametrów konfiguracyjnych, symulujących parametry pamięci, na czas działania aplikacji. Na podstawie wyników przedstawionych w pracy możemy więc uznać tezę za udowodnioną.

Warto również podkreślić przyszłościowy charakter pracy. Obecnie jej główną wartością jest weryfikacja przydatności bajtowo adresowanej pamięci NVRAM przy przetwarzaniu plików. Jeśli natomiast, zgodnie z przewidywaniami opisanymi w rozdziale 2, układy oparte o taką pamięć pojawią się wkrótce na rynku, zaprezentowane podejście będzie mogło zostać wykorzystane nie tylko do redukcji czasu działania rzeczywistych aplikacji, ale również do oceny potencjalnego zysku wydajnościowego po wyposażeniu klastra obliczeniowego w takie urządzenia.

7.2 Dalsze kierunki prac

Udowodnienie tezy rozprawy w żaden sposób nie ogranicza dalszych możliwych prac związanych z tematem wykorzystania pamięci NVRAM w operacjach plikowych w systemach obliczeniowych wysokiej wydajności. Jednym z pierwszych zadań, które powinny zostać zrealizowane, jak tylko nowe układy pamięci trafią na rynek, jest zweryfikowanie na rzeczywistym sprzęcie zarówno autorskiego rozwiązania, jak i innych pomysłów opisanych w przytoczonej literaturze. W szczególności istotne będzie nie tylko zmierzenie rzeczywistego wpływu na wydajność rozwiązania zaproponowanego w rozprawie, ale również ponowna

MOST WIEDZY Downloaded from mostwiedzy.pl

próba weryfikacji wydajności rozproszonego systemu plików, operującego na NVRAM zamiast na obecnych nośnikach pamięci masowej, a także optymalizacji opartych o pamięci SSD ogólnie.

Drugim ważnym kierunkiem dalszych badań, który będzie możliwy po pojawieniu się rzeczywistych urządzeń, powinna być weryfikacja opłacalności zaproponowanej konfiguracji – w kontekście kosztów i zużycia energii. Możliwe jest, że pierwsza generacja urządzeń NVRAM będzie na tyle droga, że jednostki badawcze będą instalować nowe układy pamięci jedynie w wybranych węzłach, co uniemożliwi zastosowanie zaproponowanego rozwiązania. Z drugiej strony może się okazać, że wysoka efektywność energetyczna operacji przeprowadzanych w NVRAM-ie, połączona z odciążeniem serwerów rozproszonego systemu plików, pozwoli na zmniejszone koszty utrzymania klastra i skompensowanie ceny dodatkowej inwestycji w nową technologię. Niewykluczone również, że sam koszt, zarówno urządzeń, jak i zużytej energii, w przyszłości straci na znaczeniu wraz z coraz większą popularyzacją idei *green computing*, w której w przypadku superkomputerów największe znaczenie wydaje się mieć efektywność energetyczna.

Trzecim, dość oczywistym wątkiem badawczym, jest dalszy rozwój zaproponowanego rozwiązania. W rozprawie skupiono się na zwiększeniu wydajności aplikacji, które nie wykorzystywały pełnego potencjału implementacji MPI I/O. Dalsze prace powinny wziąć pod uwagę między innymi programy:

- zoptymalizowane pod kątem operacji zbiorczych, czyli wywoływanych ze wszystkich procesów klastra;
- korzystające z widoków sterujących widocznością danych z perspektywy procesu;
- wykorzystujące współdzielony wskaźnik do pliku.

Jak widać, MPI I/O ma bardzo szerokie możliwości, a dojrzałe rozwiązanie powinno zapewniać wysoką wydajność wszystkich możliwych operacji zdefiniowanych w standardzie. Należy również rozważyć bliższą integrację rozwiązania z istniejącymi optymalizacjami obecnymi w MPI I/O i części klienckiej systemu plików. Wartościowym wydaje się także wsparcie biblioteki w zakresie odpowiedniej obsługi różnych topologii klastrów obliczeniowych – można się spodziewać, że wykorzystanie zalet konkretnych konfiguracji powinno pozwolić na dodatkową redukcję czasu działania aplikacji czy zwiększenie niezawodności systemu.

MOST WIEDZY Downloaded from mostwiedzy.pl

Dalszy rozwój pamięci podręcznej to nie tylko inne obszary interfejsu MPI I/O, ale również próba rozszerzenia zaimplementowanych algorytmów. Jak pokazano w omówionej literaturze oraz na wykresach wydajności autorskiego rozwiązania, czas działania aplikacji jest zależny od schematu dostępu do pliku. Niektóre zaprezentowane narzędzia próbowały dostosować do takiego schematu swoje działanie, co często wymagało pierwszego uruchomienia programu w celu zaobserwowania pewnych prawidłowości. Interesującym tematem byłoby dostosowywanie się rozwiązania do aplikacji zrealizowane automatycznie, albo przez statyczną analizę kodu źródłowego uruchamianej aplikacji, albo dzięki predykcji dalszych operacji realizowanej już podczas wykonywania programu. Obecnie pojawia się również dużo rozwiązań projektowanych dla środowisk heterogenicznych – w przypadku tematu rozprawy mogłoby to oznaczać dostosowanie podejścia do środowiska, w którym w pamięć NVRAM zaopatrzono jedynie część węzłów obliczeniowych, albo do prowadzenia obliczeń jednocześnie na wielu klastrach.

Zgodnie z opisem przedstawionym w rozdziale 2, konfiguracja w której węzły są zaopatrzone jednocześnie w pamięć DRAM i NVRAM, to tylko jedna z możliwości, a najbardziej pożądaną konfiguracją byłoby uproszczenie hierarchii pamięci dzięki układom łączącym szybkość DRAM z nieulotnością i potencjalnie dużym rozmiarem NVRAM. Taka konfiguracja oznaczałaby konieczność ponownego przemyślenia koncepcji zaproponowanego rozwiązania, a być może zaprojektowania nowego, dostosowanego do nowych założeń.

Chociaż obecnie MPI I/O jest najbardziej popularną biblioteką pośredniczącą w dostępie do plików w klastrach obliczeniowych, trzeba mieć na uwadze, że standard MPI został stworzony w latach dziewięćdziesiątych i przez ten czas informatyka bardzo się rozwinęła. Obecnie na rynku istnieją narzędzia dużo bardziej nowoczesne, takie jak język Chapel[1] [17] stworzony w roku 2009 przez firmę Cray czy framework Apache Spark[2] [12], którego pierwsza wersja została wydana w 2014 roku. MPI bywa coraz częściej nazywany przestarzałym i niektórzy spodziewają się, że nowe rozwiązania będą zyskiwać na popularności [21]. A wraz ze spadającą popularnością MPI, należy również spodziewać się coraz mniej powszechnego użycia MPI I/O. Nowsze platformy również doczekały się warstw pośredniczących w wymianie danych z rozproszonymi systemami plików, np. w przypadku języka Chapel wydajność takiej warstwy jest porównywalna z wydajnością ROMIO, najbardziej popularniej implementacji MPI I/O [59], podobnie przypadku przykładowego rozwiązania dedykowanego platformie Spark [63]. Oznacza to, że w przyszłości należałoby

[1] `https://chapel-lang.org/`
[2] `https://spark.apache.org/`

MOST WIEDZY Downloaded from mostwiedzy.pl

również rozważyć dostosowanie rozwiązania również do innych, coraz szerzej stosowanych platform, a być może nawet stworzyć nowe rozwiązania od podstaw w taki sposób, żeby były w pełni zintegrowane z nowymi platformami.

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Spis rysunków

1.1 Grafika poglądowa ilustrująca wyzwania stawiane dzisiaj przed HPC 17

2.1 Uproszczony schemat hierarchii pamięci 24

2.2 Materiały marketingowe firmy AgigA Tech, ilustrujące nie tylko architekturę układów typu NVDIMM-N, ale również pokazujące konieczność zapewnienia modułom podtrzymywania zasilania na wypadek awarii *(źródło: agigatech.wpengine.com)* 28

2.3 Materiały marketingowe firmy Intel, pokazujące potencjalną pozycję układów NVRAM (tutaj opatrzonych nazwą technologii 3D XPoint) w hierarchii pamięci *(źródło: intel.com)* 30

2.4 Potencjalne możliwości kształtu hierarchii pamięci uzupełnionej o NVRAM 30

3.1 Podział funkcji MPI I/O odpowiedzialnych za dostęp do danych pliku *(źródło: MPI: A Message-Passing Interface Standard [81], prawa autorskie: University of Tennessee)* 39

3.2 Schemat stosu technologicznego stosowanego w operacjach na plikach w obliczeniach wysokiej wydajności 40

3.3 Schemat działania algorytmu *data sieving* (a) i *two-phase I/O* (b) stosowanych w popularnym rozwiązaniu ROMIO 42

3.4 Architektura Catwalk-ROMIO; węzły obliczeniowe dodają kolejno żądania do bufora (na rysunku bufor jest obecnie obsługiwany przez węzeł nr 3) . . 43

3.5 Schemat działania narzędzia PLDA 44

3.6 Podstawowa koncepcja S4D-Cache, w której dodatkowa warstwa zlokalizowana pomiędzy MPI I/O a serwerem rozproszonego systemu plików może przekierować żądania do pamięci podręcznej 48

MOST WIEDZY Downloaded from mostwiedzy.pl

3.7 Materiały marketingowe firmy Cray pokazujące obciążenie przed (a) i po (b) zastosowaniu idei burst buffer zaimplementowanej w technologii DataWarp *(źródło: cray.com)* 50

4.1 Wielowęzłowa architektura narzędzia CDBB 64

4.2 Rozwiązanie Active NVRAM, w którym oprócz buforowania żądań wychodzących z węzła, pojawia się również wewnętrzne przetwarzanie zgromadzonych danych 65

4.3 Stos technologiczny zaproponowany przez autora rozprawy, który zmienia klasyczne podejście związane z przetwarzaniem plików na wykorzystanie szybkiej bazy danych typu klucz-wartość, dostosowanej do pamięci NVRAM 66

5.1 Ilustracja pozycji zaproponowanej pamięci podręcznej w kontekście stosu technologicznego aplikacji pracującej na plikach 72

5.2 Schemat architektury rozwiązania. W celu zwiększenia czytelności nie uwzględniono konkretnych procesów MPI 74

5.3 Schemat architektury rozwiązania w obrębie pojedynczego węzła. Szare bloki oraz przerywane linie są przezroczyste z perspektywy programisty aplikacji 75

5.4 Przepływ danych w rozwiązaniu zapewniającym spójność pamięci podręcznej. Rysunek pokazuje także, że w przypadku awarii odtwarzana jest jedynie pamięć podręczna – ewentualny zapis odzyskanych danych w PFS musi być jawnie wywołany z aplikacji 77

5.5 Przepływ danych w mechanizmie dziennika żądań zapisu. Na rysunku założono, że awaria wystąpiła na drugim węźle i nie był on w stanie brać udziału w dalszym przetwarzaniu 79

5.6 Przepływ danych podczas synchronizacji pamięci podręcznej z systemem plików. Podczas awarii, odtwarzany jest ostatni spójny stan aplikacji zapisany w PFS 80

5.7 Trzy wersje tej samej aplikacji z różnym schematem dostępu do pliku 82

5.8 Diagramy prezentują charakterystykę docelowych aplikacji, które najbardziej skorzystają z zaproponowanego rozwiązania, jak również tych, które przy jego użyciu mogą zmniejszyć swoją wydajność 83

MOST WIEDZY Downloaded from mostwiedzy.pl

6.1 Benchmark ROMPIO – wpływ rozmiaru bloku danych na prędkość odczytu i zapisu 98

6.2 Benchmark ROMPIO – wpływ rozmiaru klastra na prędkość odczytu i zapisu 98

6.3 Benchmark – błądzenie losowe – wyniki eksperymentów 100

6.4 Zdjęcie oryginalne i przetworzone przy użyciu zaimplementowanych filtrów *(fotografia: Krzysztof Krzempek)* 102

6.5 Przetwarzanie dużych obrazów – wyniki eksperymentów 106

6.6 Wyznaczanie potęgi grafu – wyniki eksperymentów 109

6.7 Przeszukiwanie dwuwymiarowej mapy – wyniki eksperymentów 110

6.8 Architektura, pokazująca komponenty składowe aplikacji, która symuluje zachowanie tłumu 111

6.9 Grafika prezentująca symulację zachowania tłumu; niebieskie punkty reprezentują agentów 113

6.10 Symulacja zachowania tłumu – wyniki eksperymentów 114

6.11 Wpływ mechanizmów ochrony danych na wydajność – wyniki eksperymentów 117

6.12 Czas tworzenia kolejnych wersji pliku i odtwarzania pamięci podręcznej – wyniki eksperymentów 118

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Spis tabel

3.1 Porównanie rozwiązań kompatybilnych z aplikacjami wykorzystującymi MPI I/O, zwiększających wydajność przetwarzania plików w HPC 53

4.1 Porównanie rozwiązań opartych o NVRAM dedykowanych operacjom I/O w HPC 67

5.1 Różnice trzech poziomów zapewnienia bezpieczeństwa danych w zaproponowanym rozwiązaniu 81

6.1 Konfiguracja sprzętowa klastrów testowych 94

6.2 Oprogramowanie użyte na potrzeby testów 94

6.3 Parametry symulatora NVRAM (tylko klaster Lap06) 96

MOST WIEDZY Downloaded from mostwiedzy.pl

MOST WIEDZY Downloaded from mostwiedzy.pl

Bibliografia

[1] Aguilar Leonel, Lalith Maddegedara, Ichimura Tsuyoshi, Hori Muneo. On the performance and scalability of an HPC enhanced Multi Agent System based evacuation simulator. W: *Procedia Computer Science*, s. 937 – 947, 2017. International Conference on Computational Science, ICCS 2017, Zurych, Szwajcaria.

[2] Åkerman Johan. Toward a Universal Memory. W: *Science*, nr 308 (5721), s. 508 – 510, 2005.

[3] Alam Sadaf R., El-Harake Hussein N., Howard Kristopher, Stringfellow Neil, Verzelloni Fabio. Parallel I/O and the Metadata Wall. W: *Proceedings of the Sixth Workshop on Parallel Data Storage*, PDSW '11, s. 13 – 18, Nowy Jork, Stany Zjednoczone, 2011. ACM.

[4] Badam Anirudh, Pai Vivek S. SSDAlloc: Hybrid SSD/RAM Memory Management Made Easy. W: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, s. 211–224, Berkeley, Stany Zjednoczone, 2011. USENIX Association.

[5] Balakhontceva Marina, Karbovskii Vladislav, Sutulo Serge, Boukhanovsky Alexander. Multi-agent Simulation of Passenger Evacuation from a Damaged Ship under Storm Conditions. W: *Procedia Computer Science*, nr 80, s. 2455 – 2464, 2016. International Conference on Computational Science 2016, ICCS 2016, San Diego, Stany Zjednoczone.

[6] Boito F. Z., Inacio E. C., Bez J. L., Navaux P., Dantas M. A Checkpoint of Research on Parallel I/O for High Performance Computing. W: *ACM Computing Surveys*, nr 51, 2018. ACM.

[7] Bourzac Katherine. Has Intel created a universal memory technology? W: *IEEE Spectrum*, nr 54 (5), s. 9 – 10, 2017.

MOST WIEDZY Downloaded from mostwiedzy.pl

[8] Canon Inc. Canon EOS 5DS. Dane techniczne, 2019. `https://www.canon.pl/for_home/product_finder/cameras/digital_slr/eos_5ds/#specifications`.

[9] Cannon Lynn Elliot. *A cellular computer to implement the Kalman Filter Algorithm. Praca doktorska.* Montana State University, 1969.

[10] Cappelletti Paolo. Non volatile memory evolution and revolution. W: *2015 IEEE International Electron Devices Meeting (IEDM)*, Rozdziały 10.1.1 – 10.1.4, 2015.

[11] Chaarawi Mohamad, Gabriel Edgar, Keller Rainer, Graham Richard L., Bosilca George, Dongarra Jack J. OMPIO: A Modular Software Architecture for MPI I/O. W: *Recent Advances in the Message Passing Interface*, s. 81 – 89, Berlin, Niemcy, 2011. Springer Berlin Heidelberg.

[12] Chaimov Nicholas, Malony Allen, Canon Shane, Iancu Costin, Ibrahim Khaled Z., Srinivasan Jay. Scaling Spark on HPC Systems. W: *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, s. 97 – 110, Nowy Jork, Stany Zjednoczone 2016. ACM.

[13] Chen Feng, Mesnier Michael P., Hahn Scott. A protected block device for Persistent Memory. W: *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*, s. 1 – 12, 2014.

[14] Coteus Paul i in. Packaging the Blue Gene/L supercomputer. W: *IBM Journal of Research and Development*, nr 49 (2.3), s. 213 – 248, 2005.

[15] Cugnasco Cesare, Becerra Yolanda, Torres Jordi, Ayguadé Eduard. Exploiting Key-Value Data Stores Scalability for HPC. W: *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, s. 85 – 94, 2017.

[16] Cutress Ian. Intel's 140GB Optane 3D XPoint PCIe SSD Spotted at IDF, 2016. `http://www.anandtech.com/show/10604/intels-140gb-optane-3d-xpoint-pcie-ssd-spotted-at-idf`.

[17] Chamberlain Bradford L., Deitz Steven J., Figueroa Samuel, Iten David M., Stone Andrew. Global HPC Challenge Benchmarks in Chapel. 2008.

[18] Ching A., Coloma K., Li J., Liao W., Choudhary A. High-Performance Techniques for Parallel I/O. W: *Handbook of Parallel Computing: Models, Algorithms and Applications*, s. 166 – 189, 2001.

MOST WIEDZY Downloaded from mostwiedzy.pl

[19] de Charentenay Yann. STT-MRAM is moving to large scale commercialization (at last!). W: *2017 IEEE International Magnetics Conference (INTERMAG)*, s. 1 – 2, 2017.

[20] Dorożynski Piotr, Czarnul Paweł, Malinowski Artur, Czuryło Krzysztof, Dorau Łukasz, Maciejewski Maciej, Skowron Paweł. Checkpointing of Parallel MPI Applications Using MPI One-sided API with Support for Byte-addressable Non-volatile RAM. W: *Procedia Computer Science*, nr 80, s. 30 – 40, 2016.

[21] Dursi Jonathan. HPC is dying, and MPI is killing it, 2015. `https://www.dursi.ca/post/hpc-is-dying-and-mpi-is-killing-it.html`.

[22] Fan Bin, Tantisiriroj Wittawat, Xiao Lin, Gibson Garth. DiskReduce: RAID for Data-intensive Scalable Computing. W: *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, s. 6 – 10, Nowy Jork, Stany Zjednoczone, 2009. ACM.

[23] Fan Ziqi. *Improving Storage Performance with Non-Volatile Memory-based Caching Systems. Praca doktorska.* University of Minnesota, 2017.

[24] Fernando Pradeep, Kannan Sudarsun, Gavrilovska Ada, Schwan Karsten. Phoenix: Memory Speed HPC I/O with NVM. W: *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, s. 121 – 131, 2016.

[25] Fong Scott W., Neumann Christopher M., Wong H. S. Philip. Phase-Change Memory: Towards a Storage-Class Memory. W: *IEEE Transactions on Electron Devices*, nr 64 (11), s. 4374 – 4385, 2017.

[26] Foong Annie, Hady Frank. Storage As Fast As Rest of the System. W: *2016 IEEE 8th International Memory Workshop (IMW)*, s. 1 – 4, 2016.

[27] Google Art Project w zbiorach Wikimedia Commons. Gigapixel images from the Google Art Project, 2019. `https://commons.wikimedia.org/wiki/Category:Gigapixel_images_from_the_Google_Art_Project`.

[28] Greengard Samuel. Better Memory. W: *Commun. ACM*, nr 59 (1), s. 23 – 25, 2015.

[29] Gutierrez-Milla Albert, Borges Francisco, Suppi Remo, Luque Emilio. Individual-oriented Model Crowd Evacuations Distributed Simulation. W: *Procedia Computer Science*, nr 29, s. 1600 – 1609, 2014.

MOST WIEDZY Downloaded from mostwiedzy.pl

[30] Hadri Bilel. Introduction to Parallel I/O, 2011. https://www.olcf.ornl.gov/wp-content/uploads/2011/10/Fall_IO.pdf.

[31] Hasselblad Press Release. Hasselblad introduces the H6D-400c MS, 2018. https://www.hasselblad.com/press/press-releases/hasselblad-introduces-the-h6d-400c-ms/.

[32] He Shuibing, Sun Xian-He, Feng Bo. S4D-Cache: Smart Selective SSD Cache for Parallel I/O Systems. W: *2014 Ieee 34th Int. Conference On Distributed Computing Systems (ICDCS 2014)*, s. 514–523, 2014.

[33] He Shuibing, Sun Xian-He, Feng Bo, Huang Xin, Feng Kun. A cost-aware region-level data placement scheme for hybrid parallel I/O systems. W: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, s. 1 – 8, 2013.

[34] He Shuibing, Yang Wang, Xian-He Sun. Improving Performance of Parallel I/O Systems through Selective and Layout-Aware SSD Cache. W: *IEEE Transactions on Parallel and Distributed Systems*, nr 27 (10), s. 2940 – 2952, 2016.

[35] Henseler Dave, Landsteiner Benjamin, Petesch Doug, Wright Cornell, Wright Nicholas J. Architecture and design of cray datawarp. W: *Cray User Group CUG*, 2016.

[36] Hoefler Torsten, Snir Marc. Writing Parallel Libraries with MPI - Common Practice, Issues, and Extensions. W: *Recent Advances in the Message Passing Interface*, s. 345 – 355, Berlin, Niemcy, 2011. Springer Berlin Heidelberg.

[37] Hori Atsushi, Yamamoto Keiji, Ishikawa Yutaka. Catwalk-ROMIO: A Cost-Effective MPI-IO. W: *2011 IEEE 17th Int. Conference On Parallel Distributed Systems (icpads)*, s. 120 – 126, 2011.

[38] Huang Dachuan, Zhang Xuechen, Shi Wei, Zheng Mai, Jiang Song, Qin Feng. LiU: Hiding Disk Access Latency for HPC Applications with a New SSD-Enabled Data Layout. W: *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, s. 111 – 120, 2013.

[39] Intel Corporation. Intel and Micron Produce Breakthrough Memory Technology, 2015. http://newsroom.intel.com/community/intel_newsroom/blog/2015/07/28/intel-and-micron-produce-breakthrough-memory-technology.

MOST WIEDZY Downloaded from mostwiedzy.pl

[40] Intel Corporation. Reimagining the Data Center Memory and Storage Hierarchy, 2018. https://newsroom.intel.com/editorials/re-architecting-data-center-memory-storage-hierarchy/.

[41] Jain Nikhil, Bhatele Abhinav, Ni Xiang, Wright Nicholas J., Kale Laxmikant V. Maximizing Throughput on a Dragonfly Network. W: *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, s. 336 – 347, 2014.

[42] Jarząbek Łukasz, Czarnul Paweł. Performance evaluation of unified memory and dynamic parallelism for selected parallel CUDA applications. W: *The Journal of Supercomputing*, nr 73 (12), s. 5378 – 5401, 2017.

[43] JEDEC Press Release. JEDEC Announces Support for NVDIMM Hybrid Memory Modules, 2015. https://www.jedec.org/news/pressreleases/jedec-announces-support-nvdimm-hybrid-memory-modules.

[44] JEDEC Press Release. JEDEC DDR5 NVDIMM-P Standards Under Development, 2017. https://www.jedec.org/news/pressreleases/jedec-ddr5-nvdimm-p-standards-under-development.

[45] JEDEC Standards and Documents. Main Memory: DDR4 and DDR5 SDRAM, 2019. https://www.jedec.org/category/technology-focus-area/main-memory-ddr3-ddr4-sdram.

[46] Jung Myoungsoo, Choi Wonil, Srikantaiah Shekhar, Yoo Joonhyuk, Kandemir Mahmut T. HIOS: A Host Interface I/O Scheduler for Solid State Disks. W: *SIGARCH Comput. Archit. News*, nr 42 (3), s. 289 – 300, 2014.

[47] Kaiser Nick, Burgett William, Chambers Ken, Denneau Larry, Heasley Jim, Jedicke Robert, Magnier Eugene, Morgan Jeff, Onaka Peter, Tonry John. The Pan-STARRS wide-field optical/NIR imaging survey. W :*Proc. SPIE*, nr 7733 (14), 2010.

[48] Kang Seok-Hoon, Koo Dong-Hyun, Kang Woon-Hak, Lee Sang-Won. A case for flash memory ssd in hadoop applications. W: *International Journal of Control and Automation*, nr 6 (1), s. 201 – 210, 2013.

[49] Kannan Sudarsun, Gavrilovska Ada, Schwan Karsten, Milojicic Dejan, Talwar Vanish. Using Active NVRAM for I/O Staging. W: *Proceedings of the 2Nd International*

MOST WIEDZY Downloaded from mostwiedzy.pl

Workshop on Petascal Data Analytics: Challenges and Opportunities, PDAC '11, s. 15 – 22, Nowy Jork, Stany Zjednoczone, 2011. ACM.

[50] Kettering Brett M., Nunez James A. The role of non-volatile memory from an application perspective. W: *2010 IEEE Globecom Workshops*, s. 1921 – 1925, 2010.

[51] Kim Jungwon, Lee Seyong, Vetter Jeffrey S. PapyrusKV: A High-performance Parallel Key-value Store for Distributed NVM Architectures. W: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, s. 1 – 14, Nowy Jork, Stany Zjednoczone, 2017. ACM.

[52] Kobayashi Hiroyuki, Ishimoto Yutaka, Fujioka Masaki, Ishibashi Kenichi . A multi-agent evacuation simulator to design safe cities for high quality of life with computer clustering. W: *SICE, 2007 Annual Conference*, s. 3043 – 3046, 2007.

[53] Konishi Ryusuke, Amagai Yoshiji, Sato Koji, Hifumi Hisashi, Kihara Seiji, Moriai Satoshi. The Linux Implementation of a Log-structured File System. *SIGOPS Oper. Syst. Rev.*, nr 40 (3), s. 102 – 107, 2006.

[54] Kryder Mark H., Soo Kim Chang. After Hard Drives — What Comes Next? *Magnetics, IEEE Transactions on*, nr 45 (10), s. 3406 – 3413, 2009.

[55] Kumar Gyanendra, Tomar Parul. A Novel Longest Distance First Page Replacement Algorithm. nr 10, s. 1 – 6, 2017.

[56] Kyrola Aapo, Blelloch Guy, Guestrin Carlos. GraphChi: Large-Scale Graph Computation on Just a PC. W: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, s. 31 – 46, Hollywood, Stany Zjednoczone, 2012. USENIX.

[57] Kültürsay Emre, Kandemir Mahmut, Sivasubramaniam Anand, Mutlu Onur. Evaluating STT-RAM as an energy-efficient main memory alternative. W: *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, s. 256 – 267, 2013.

[58] Lang Samuel, Carns Philip, Latham Robert, Ross Robert, Harms Kevin, Allcock William. I/O Performance Challenges at Leadership Scale. W: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, nr 40, s. 1 – 12, Nowy Jork, Stany Zjednoczone, 2009. ACM.

MOST WIEDZY Downloaded from mostwiedzy.pl

[59] Larrosa Rafael , Asenjo Rafael, Navarro Angeles, Chamberlain Bradford L. A First Implementation of Parallel IO in Chapel for Block Data Distribution. W: *Applications, Tools and Techniques on the Road to Exascale Computing*, Advances in Parallel Computing, s. 447 – 454, 2017.

[60] Latham Robert, Ross Robert, Thakur Rajeev. Implementing MPI-IO Atomic Mode and Shared File Pointers Using MPI One-Sided Communication. W: *International Journal of High Performance Computing Applications*, nr 21 (2), s. 132 – 143, 2007.

[61] Li Xu, Lu Kai, Wang Xiaoping, Zhou Xu. NV-process: A Fault-tolerance Process Model Based on Non-volatile Memory. W: *Proceedings of the Asia-Pacific Workshop on Systems*, APSYS '12, s. 1 – 6, Nowy Jork, Stany Zjednoczone, 2012. ACM.

[62] Li Xu, Lu Kai, Zhou Xu. NV-TS: A Fault Tolerance Transaction System Based on Persistent Memory. W: *2012 International Conference on Computer Science and Electronics Engineering*, nr 2, s. 221 – 224, 2012.

[63] Liu Jialin, Racah Evan, Koziol Quincey, Canon Richard Shane. H5spark: bridging the I/O gap between spark and scientific data formats on HPC systems. *Cray user group*, 2016.

[64] Liu Ning, Cope Jason, Carns Philip, Carothers Christopher, Ross Robert, Grider Gary, Crume Adam, Maltzahn Carlos. On the role of burst buffers in leadership-class storage systems. W: *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, s. 1 – 11, 2012.

[65] Liu Wei, Wu Kai, Liu Jialin, Chen Feng, Li Dong. Performance Evaluation and Modeling of HPC I/O on Non-Volatile Memory. W: *2017 International Conference on Networking, Architecture, and Storage (NAS)*, s. 1 – 10, 2017.

[66] Luu Huong, Behzad Babak, Aydt Ruth, Winslett Marianne. A multi-level approach for understanding I/O activity in HPC applications. W: *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, s. 1 – 5, 2013.

[67] Luu Huong, Winslett Marianne, Gropp William, Ross Robert, Carns Philip, Harms Kevin, Prabhat Mr, Byna Suren, Yao Yushu. A Multiplatform Study of I/O Behavior on Petascale Supercomputers. W: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, s. 33 – 44, Nowy Jork, Stany Zjednoczone, 2015. ACM.

MOST WIEDZY Downloaded from mostwiedzy.pl

[68] Makinoshima Fumiyasu, Imamura Fumihiko, Abe Yoshi. Enhancing a tsunami evacuation simulation for a multi-scenario analysis using parallel computing. W: *Simulation Modelling Practice and Theory*, 2018.

[69] Malinowski Artur. NVRAM as Main Storage of Parallel File System. W: *Journal of Computer Science and Control Systems*, nr 9, s. 18 – 21, 2016.

[70] Malinowski Artur. Using Redis supported by NVRAM in HPC applications. W: *Computer Science (AGH)*, nr 18 (3), 2017.

[71] Malinowski Artur, Czarnul Paweł. Multi-agent large-scale parallel crowd simulation with NVRAM-based distributed cache. W: *Journal of Computational Science*, vol. 33, s. 83 – 94, 2019.

[72] Malinowski Artur, Czarnul Paweł. Three levels of fail-safe mode in MPI I/O NVRAM distributed cache. W: *Procedia Computer Science*, nr 136, s. 52 – 61, 2018. 7th International Young Scientists Conference on Computational Science, YSC 2018, Heraklion, Greece.

[73] Malinowski Artur, Czarnul Paweł, Matuszek Mariusz. Recommendations for Writing Parallel Libraries with C and MPI. Przesłany do recenzji.

[74] Malinowski Artur, Czarnul Paweł. Distributed NVRAM Cache – Optimization and Evaluation with Power of Adjacency Matrix. W: *Computer Information Systems and Industrial Management*, s. 15 – 26, 2017. Springer International Publishing.

[75] Malinowski Artur, Czarnul Paweł. A Solution to Image Processing with Parallel MPI I/O and Distributed NVRAM Cache. W: *Scalable Computing: Practice and Experience*, nr 19 (1), 2018.

[76] Malinowski Artur, Czarnul Paweł, Czuryło Krzysztof, Maciejewski Maciej, Skowron Paweł. Multi-agent large-scale parallel crowd simulation. W: *Procedia Computer Science*, nr 108, s. 917 – 926, 2017. International Conference on Computational Science, ICCS 2017, Zurych, Szwajcaria.

[77] Malinowski Artur, Czarnul Paweł, Dorożynski Piotr, Czuryło Krzysztof, Dorau Łukasz, Maciejewski Maciej, Skowron Paweł. A Parallel MPI I/O Solution Supported by Byte-addressable Non-volatile RAM Distributed Cache. W: *Position Papers of*

MOST WIEDZY Downloaded from mostwiedzy.pl

the 2016 Federated Conference on Computer Science and Information Systems, tom 9 *Annals of Computer Science and Information Systems*, s. 133 – 140. PTI, 2016.

[78] Malinowski Artur, Czarnul Paweł, Maciejewski Maciej, Skowron Paweł. A Fail-Safe NVRAM Based Mechanism for Efficient Creation and Recovery of Data Copies in Parallel MPI Applications. W: *Information Systems Architecture and Technology: Proceedings of 37th International Conference on Information Systems Architecture and Technology – ISAT 2016 – Part II*, s. 137 – 147, 2017. Springer International Publishing.

[79] Meena Jagan Singh, Sze Simon Min, Chand Umesh, Tseng Tseung-Yuen. Overview of emerging nonvolatile memory technologies. W: *Nanoscale Research Letters*, nr 9 (1), 2014.

[80] Mehta Kshitij, Gabriel Edgar, Chapman Barbara. Specification and Performance Evaluation of Parallel I/O Interfaces for OpenMP. W: *OpenMP in a Heterogeneous World*, s. 1 – 14, Berlin, Niemcy, 2012. Springer Berlin Heidelberg.

[81] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard Version 3.1, 2015. http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[82] Mogg Trevor. We could explore this astonishing 195-gigapixel panorama of Shanghai all day, 2018. `https://www.digitaltrends.com/news/check-out-this-astonishing-195-gigapixel-image-of-shanghai/`.

[83] Molka Daniel, Hackenberg Daniel, Schone Robert, Muller Matthias S. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System. W: *2009 18th International Conference on Parallel Architectures and Compilation Techniques*, s. 261 – 270, 2009.

[84] NASA High-End Computing Program. Lustre Best Practices, August 2015. `http://www.nas.nasa.gov/hecc/support/kb/lustre-best-practices_226.html`.

[85] NASA/ESA. Hubble's High-Definition Panoramic View of the Andromeda Galaxy, 2015. `http://www.spacetelescope.org/images/heic1502a/`.

[86] Nowak Janusz J. i in. Dependence of Voltage and Size on Write Error Rates in Spin-Transfer Torque Magnetic Random-Access Memory. W: *IEEE Magnetics Letters*, nr 7, s. 1 – 4, 2016.

MOST WIEDZY Downloaded from mostwiedzy.pl

[87] Patil Onkar, Hukerikar Saurabh, Mueller Frank, Englemann Christian. W: Exploring Use-cases for Non-Volatile Memories in support of HPC Resilience, 2017.

[88] Pavlovic Milan, Puzovic Nikola, Ramirez Alex. Data placement in HPC architectures with heterogeneous off-chip memory. W: *2013 IEEE 31st International Conference on Computer Design (ICCD)*, s. 193 – 200, 2013.

[89] Pawlowski J. Thomas. Memory as we approach a new horizon. W: *2016 IEEE Hot Chips 28 Symposium (HCS)*, s. 1 – 23, 2016.

[90] Protopopov Boris V., Skjellum Anthony . A Multithreaded Message Passing Interface (MPI) Architecture: Performance and Program Issues. W: *Journal of Parallel and Distributed Computing*, nr 61 (4), s. 449 – 466, 2001.

[91] Radulovic Milan Zivanovic Darko, Ruiz Daniel, de Supinski Bronis R., McKee Sally A., Radojković Petar, Ayguadé Eduard . Another Trip to the Wall: How Much Will Stacked DRAM Benefit HPC? W: *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, s. 31 – 36, Nowy Jork, Stany Zjednoczone, 2015. ACM.

[92] Rajachandrasekar Raghunath, Moody Adam, Mohror Kathryn, Panda Dhabaleswar K. A 1 PB/s File System to Checkpoint Three Million MPI Tasks. W: *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, s. 143 – 154, Nowy Jork, Stany Zjednoczone, 2013. ACM.

[93] Rudoff Andy. Persistent Memory: The Value to HPC and the Challenges. W: *Proceedings of the Workshop on Memory Centric Programming for HPC*, MCHPC'17, s. 7 – 10, Nowy Jork, Stany Zjednoczone, 2017. ACM.

[94] Schaller Robert. Moore's law: past, present and future. W: *IEEE Spectrum*, nr 34 (6), s. 52 – 59, 1997.

[95] Schenck Wolfram, El Sayed Salem, Foszczynski Maciej, Homberg Wilhelm, Pleiter Dirk. Early Evaluation of the "Infinite Memory Engine" Burst Buffer Solution. W: *High Performance Computing*, s. 604 – 615, 2016. Springer International Publishing.

[96] Schulz Martin, de Supinski Bronis R. PnMPI Tools: A Whole Lot Greater Than the Sum of Their Parts. W: *ACM/IEEE Supercomputing Conference (SC)*, s. 1 – 10. ACM, 2007.

MOST WIEDZY Downloaded from mostwiedzy.pl

[97] Shantharam Manu, Iwabuchi Keita, Cicotti Pietro, Carrington Laura, Gokhale Maya, Pearce Roger. Performance Evaluation of Scale-Free Graph Algorithms in Low Latency Non-volatile Memory. W: *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, s. 1021 – 1028, 2017.

[98] Skjellum Anthony, Doss Nathan E., Bangalore Purushotham V. Writing libraries in MPI. W: *Proceedings of Scalable Parallel Libraries Conference*, s. 166 – 173, 1993.

[99] Song Huaiming, Yin Yanlong, Sun Xian-He, Thakur Rajeev, Lang Samuel. Server-side I/O Coordination for Parallel File Systems. W: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, nr 17, s. 1 – 11, Nowy Jork, Stany Zjednoczone, 2011. ACM.

[100] Stallings William. *Computer Organization and Architecture: Designing for Performance*. Pearson, edycja 9, 2013.

[101] Storage Networking Industry Association. NVM Programming Model (NPM), 2017. `https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf`.

[102] Storage Networking Industry Association. Persistent Memory and NVDIMM Special Interest Group. `https://www.snia.org/forums/sssi/NVDIMM`.

[103] Suresh Amoghavarsha, Cicotti Pietro, Carrington Laura. Evaluation of emerging memory technologies for HPC, data intensive applications. W: *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, s. 239 – 247, 2014.

[104] Tessier François, Malakar Preeti, Vishwanath Venkatram, Jeannot Emmanuel, Isaila Florin. Topology-Aware Data Aggregation for Intensive I/O on Large-Scale Supercomputers. W: *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, s. 73 – 81, 2016.

[105] Thakur Rajeev, Gropp William, Lusk Ewing. Data sieving and collective I/O in ROMIO. W: *Frontiers '99 - Seventh Symposium On Frontiers Massively Parallel Computation, Proc.*, s. 182 – 189, 1999.

[106] Tsujita Yuichi, Yoshinaga Kazumi, Hori Atsushi, Sato Mikiko, Namiki Mitaro, Ishikawa Yutaka. Multithreaded Two-Phase I/O: Improving Collective MPI-IO Per-

MOST WIEDZY Downloaded from mostwiedzy.pl

formance on a Lustre File System. W: *2014 22nd Euromicro Int. Conference On Parallel, Distributed, Network-based Processing (pdp 2014)*, s. 232 – 235, 2014.

[107] Turing Alan Mathison. On Computable Numbers, with an Application to the Entscheidungsproblem. W: *Proceedings of the London Mathematical Society*, s. 230–265, 1937.

[108] Van Essen Brian, Pearce Roger, Ames Sasha, Gokhale Maya. On the Role of NVRAM in Data-intensive Architectures: An Evaluation. W: *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, s. 703 – 714, 2012.

[109] Vetter Jeffrey S., Mittal Sparsh. Opportunities for Nonvolatile Memory Systems in Extreme-Scale High-Performance Computing. W: *Computing in Science Engineering*, nr 17 (2), s. 73 – 82, 2015.

[110] Wang Teng, Mohror Kathryn, Moody Adam, Sato Kento, Yu Weikuan. An Ephemeral Burst-buffer File System for Scientific Applications. W: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, nr 69, s. 1 – 12, Nowy Jork, Stany Zjednoczone, 2016. IEEE Press.

[111] Wang Teng, Oral Sarp, Wang Yandong, Settlemyer Brad, Atchley Scott, Yu Weikuan. BurstMem: A high-performance burst buffer system for scientific applications. W: *2014 IEEE International Conference on Big Data (Big Data)*, s. 71 – 79, 2014.

[112] Wasi-ur Rahman, Islam Nusrat Sharmin, Lu Xiaoyi, Panda Dhabaleswar K. Can Non-volatile Memory Benefit MapReduce Applications on HPC Clusters? W: *2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems (PDSW-DISCS)*, s. 19 – 24, 2016.

[113] Wautelet Philippe. Best practices for parallel IO and MPI-IO hints, March 2015. `http://www.idris.fr/media/docs/docu/idris/idris_patc_hints_proj.pdf`.

[114] Wei Qingsong, Wang Chundong, Chen Cheng, Yang Yechao, Yang Jun, Xue Mingdi. Transactional NVM Cache with High Performance and Crash Consistency. W: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, nr 56, s. 1 – 12, Nowy Jork, Stany Zjednoczone, 2017. ACM.

MOST WIEDZY Downloaded from mostwiedzy.pl

[115] Wittmann Markus, Hager Georg, Zeiser Thomas, Wellein Gerhard. Asynchronous MPI for the Masses. W: *CoRR*, 2013.

[116] Wu Kai, Ober Frank, Hamlin Shari, Li Dong. Early Evaluation of Intel Optane Non-Volatile Memory with HPC I/O Workloads. W: *CoRR*, 2017.

[117] Wuttig Matthias. Phase-change materials: Towards a universal memory? W: *Nature materials*, nr 4, s. 265 – 266, 2005.

[118] Xuan Pengfei, Ligon Walter B., Srimani Pradip K., Ge Rong, Luo Feng. Accelerating big data analytics on HPC clusters using two-level storage. W: *Parallel Computing. Special Issue on 2015 Workshop on Data Intensive Scalable Computing Systems (DISCS-2015)*, nr 61, s. 18 – 34, 2017.

[119] Yang Shuo, Wu Kai, Qiao Yifan, Li Dong, Zhai Jidong. Algorithm-Directed Crash Consistence in Non-volatile Memory for HPC. W: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, s. 475 – 486, 2017.

[120] Yin Yanlong, Li Jibing, He Jun, Sun Xian-He, Thakur Rajeev. Pattern-Direct and Layout-Aware Replication Scheme for Parallel I/O Systems. W: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, s. 345 – 356, 2013.

[121] Yu Songping, Deng Mingzhu, Xing Yuxuan, Xiao Nong, Liu Fang, Chen Wei. Pyramid: Revisiting Memory Extension with Remote Accessible Non-Volatile Main Memory. W: *Security, Privacy, and Anonymity in Computation, Communication, and Storage*, s. 730 – 743, 2017. Springer International Publishing.

[122] Yu Songping, Xiao Nong, Deng Mingzhu, Xing Yuxuan, Liu Fang, Chen Wei. Megalloc: Fast Distributed Memory Allocator for NVM-Based Cluster. W: *2017 International Conference on Networking, Architecture, and Storage (NAS)*, s. 1 – 9, 2017.

[123] Zhang Michael. Bentley Used NASA Tech to Create This 53-Gigapixel Car Photo, 2016. `https://petapixel.com/2016/06/23/bentley-used-nasa-tech-create-53-gigapixel-photo-car/`.

[124] Zhang Mingzhe, Lam King Tin, Yao Xin, Wang Cho-Li. SIMPO: A Scalable In-Memory Persistent Object Framework Using NVRAM for Reliable Big Data Computing. W: *ACM Transactions on Architecture and Code Optimization*

MOST WIEDZY Downloaded from mostwiedzy.pl

[125] Zhang Xuechen, Davis Kei, Jiang Song. iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O. W: *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, s. 715 – 726, 2012.

[126] Zhang Xuechen, Liu Ke, Davis Kei, Jiang Song. iBridge: Improving Unaligned Parallel File Access with Solid-State Drives. W: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, s. 381 – 392, 2013.

[127] Zhou Ping, Zhao Bo, Yang Jun, Zhang Youtao. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. W: *SIGARCH Comput. Archit. News*, nr 37 (3), s. 14 – 23, 2009.

MOST WIEDZY Downloaded from mostwiedzy.pl