Jan Kończak

Algorytmy odtwarzania stanu dla zreplikowanej maszyny stanowej z pamięcią ulotną i nieulotną

Streszczenie rozprawy doktorskiej

Promotor:
dr hab. inż. Paweł T. Wojciechowski, prof. PP

Poznań, czerwiec 2022

Wstęp

Po zobaczeniu komunikatu że wyszukiwarka internetowa czy strona mediów społecznościowych jest niedostępna większość ludzi automatycznie zakłada, że problem musi być po stronie dostępu do internetu – nieprzerwana dostępność poważnych usług w sieci jest brana za pewnik. Jednocześnie jest oczywiste, że każda taka usługa działa na wielu maszynach (tworzących *system rozproszony*), a te czasami się psują. Użytkownik usługi (*klient*) łączy się do jednej z maszyn, ale oczekuje że cały system odpowiednio zmieni swój stan w odpowiedzi na jego żądanie. Trudność pojawia się kiedy wielu klientów naraz wykonuje działania i maszyny tworzące usługę muszą się porozumieć – i to w sytuacji, gdy brak odpowiedzi od którejś może oznaczać zarówno jej awarię, jaki i przejściowe problemy z łącznością. Niektóre usługi mogą odpowiadać niespójnie klientom, wiele musi jednak zwracać odpowiedzi *silnie spójne*, czyli nieodróżnialne od takich jakie generowałaby usługa pracująca na jednej maszynie.

Zreplikowana maszyna stanowa (State Machine Replication, *SMR*) [Sch90] jest najpowszechniej stosowaną metodą budowy usług gwarantujących silną spójność pomimo awarii. SMR polega na wykonaniu przez każdą maszynę, nazywaną repliką, tych samych żądań (*komend*) w identycznej kolejności. W SMR każda replikowana usługa musi działać deterministycznie i tylko pod wpływem komend. Kluczowym problemem w SMR jest zachowanie *globalnego porządku* komend – do tego używa się gotowych rozwiązań problemu *konsensusu*, z których najpopularniejszym jest *Paxos* [Lam98]. Konsensus gwarantuje że ostatecznie wszystkie sprawne repliki wybiorą jedną i tą samą z zaproponowanych przez te repliki wartości. To pozwala na wybór kolejnych komend powtarzając konsensus dla kolejnych numerów.

Istniejące systemy SMR pozwalają na pracę systemu rozproszonego z prędkością ograniczona tylko przez wydajność maszyny stanowej, czyli replikowanej usługi (co jest górną teoretyczna granicą wydajności SMR) – jednak tylko przy założeniu, że repliki które ulegną awarii nie biorą udziału w dalszej pracy systemu. Jednak w praktyce konieczne jest odtwarzanie replik, czyli włączenie ich po usunięciu przyczyny awarii z powrotem do systemu. Istniejące rozwiązania SMR przy wsparciu odtwarzania wymagają trwałego zapisu komend przed odesłaniem odpowiedzi do klienta, przez co nawet w przypadku wykorzystania wydajnych dysków SSD do trwałych zapisów wydajność systemu SMR istotnie spada w wielu zastosowaniach [KA08, RST11, BSF+13].

Cele i wkład pracy

Rozprawa proponuje odpowiedzi na pytanie jak można zmniejszyć koszt wsparcia odtwarzania replik po awarii w systemie SMR opartym o algorytm Paxos. W tym celu wprowadza nowe sposoby wsparcia odtwarzania w algorytmie Paxos i pokazuje jak zbudować kompletny wydajny framework do budowy zreplikowanej maszyny stanowej wspierający odtwarzanie replik. Pierwsze z proponowanych rozwiązań, nazwane ViewSS i EpochSS, opierają się na założeniu, że w każdej chwili większość replik jest sprawna – identycznym jak założenie wymagane dla wysokiej dostępności systemów SMR. Pozostałe proponowane rozwiązania, mPaxos i mPaxosSM, wykorzystują wprowadzoną niedawno na rynek pamięć trwałą – pamięć o parametrach zbliżonych do DRAM która nie traci zawartości z zanikiem zasilania. Rozprawa porównuje również prototypowe implementacje proponowanych rozwiązań, zarówno pod kątem przepustowości bezawaryjnej pracy, jak i szybkości odtwarzania i wpływu odtwarzania na wydajność systemu.

Algorytm Paxos, framework SMR

W rozprawie algorytm Paxos jest omówiony zarówno w wersji znanej z literatury [Lam98], jak i w wersji dostosowanej na potrzeby SMR i algorytmów odtwarzania. Celem algorytmu Paxos jest wybranie przez repliki identycznych wartości dla kolejno numerowanych *instancji*. Algorytm Paxos przebiega w dwóch fazach: *wyboru lidera* i *głosowania*. W pierwszej kandydat na lidera (replika p) wysyła do wszystkich replik (rozgłasza) wiadomość z wybranym przez siebie i większym niż mu znane *numerem głosowania* (numer b; numery głosowań są statycznie rozdzielone między repliki). Jeśli replika q odbierze taką wiadomość i nie wysłała jeszcze żadnej wiadomości z numerem wyższym niż b, to odsyła do p informacje o swoim ostatnim głosie w każdej z instancji. Po otrzymaniu odpowiedzi od większości replik p staje się liderem. Po wyborze lidera rozpoczyna się druga faza, powtarzana dla każdej kolejnej instancji do czasu aż lider nie ulegnie awarii. Jako lider, p wybiera wartość z głosowania o najwyższym numerze (jeśli wie o wcześniejszym głosowaniu) lub dowolną komendę od klienta i rozgłasza ją do wszystkich replik. Po otrzymaniu propozycji repliki rozgłaszają swoja zgodę na proponowaną wartość o ile nie wzięły udziału w wyborze kolejnego lidera. Algorytm Paxos gwarantuje, że jeśli większość replik zgodziła się na wartość c w głosowaniu o numerze b, to w każdym głosowaniu o numerze $b' > b$ może być głosowana tylko wartość c. Oznacza to, że z chwilą gdy większość replik zagłosowała za wartością c w głosowaniu b osiągnięty został konsensus. Poprawność algorytmu Paxos bazuje na dwóch *przyrzeczeniach* replik: 1) replika która odpowiedziała liderowi głosowania b przyrzeka nie odpowiadać na wiadomości z głosowań starszych niż b 2) replika która głosowała na wartość c w głosowaniu b przyrzeka informować każdego kandydata na lidera o swoim głosie.

Do budowy systemu SMR, poza algorytmem konsensusu i maszyną stanową, potrzeba jeszcze części odpowiedzialnej za połączenie klientów (wysyłających komendy), konsensusu (porządkującego komendy) i maszyny stanowej (wykonującej komendy i generującej odpowiedź). Rozprawa proponuje odpowiedni *framework* dla realizacji tego celu, przedstawia konkretną architekturę i struktury danych, gwarantującą ponad wymagane minimum semantykę jednokrotnego wykonania komendy w sytuacjach gdy w wyniku awarii klient wysyła komendę ponownie. Taki opis frameworku pozwala na wskazanie (w późniejszej części rozprawy) w proponowanych algorytmach korzystających z pamięci trwałej które konkretnie struktury danych należy przechowywać w pamięci trwałej.

Paxos, podobnie inne algorytmy konsensusu, dla poprawności potrzebuje pamiętać ostatni głos w każdej instancji, by w razie długotrwałej utraty łączności replika mogła po odzyskaniu połączenia *nadgonić* stan systemu. Aby ograniczyć wykorzystanie pamięci w SMR konieczne jest periodyczne tworzenie *migawki* maszyny stanowej (co wiąże się z kosztem obliczeniowym), by w razie potrzeby przesłać migawkę zawierającą stan po wykonaniu komend zamiast (dowolnie) dużej liczby komend.

Pamięć trwała (pmem)

Pamięć trwała (*pmem*) określa pamięć operacyjną której zawartość nie jest tracona po zaniku zasilania. Z założenia taka pamięć ma być adresowana bajtowo, mieć niskie opóźnienia, wysoką przepustowość i wytrzymywać dużą ilość zapisów. O ile prace nad taką pamięcią trwają od wielu lat, pierwszy dostępny na rynku produkt pojawił się w 2019 roku jako Intel Optane DC Persistent Memory module (DCPMM). Moduły DCPMM używają fizycznego i elektrycznego interfejsu DDR4 i zmodyfi-

kowanego logicznego interfejsu DDR oraz wymagają odpowiedniego wsparcia ze strony płyty głównej i procesora. Programy wykorzystujące pmem mogą *bezpośrednio z przestrzeni użytkownika* wykonywać trwałe zapisy. Zapis do pmem staje się trwały kiedy dane zostają przesłane z pamięci podręcznej procesora do kontrolera pamięci (zintegrowanego z procesorem we współczesnych procesorach). Dowolność momentu wysłania linii cache do pamięci przez procesor powoduje że zawartość pamięci po awarii zasilania może zawierać dane niespójne (w tym niezachowujące spójności pamięci podręcznej), stąd użycie pmem wymaga od programisty odpowiedniego tworzenia aplikacji [Sca20]. Rozwiązaniem ogólnego przeznaczenia jest, wspierane przez odpowiednie biblioteki programistyczne, dzielenie kodu na logiczne transakcje. Biblioteki w razie awarii i ponownego uruchomienia programu wycofują zmiany niezakończonych transakcji. Choć pmem ma parametry wydajnościowe znacznie bliższe pamięciom DRAM niż dyskom, jest jednak od pamięci DRAM mniej wydajny, stąd drugim istotnym zagadnieniem tworzenia aplikacji wykorzystujących pmem jest podział danych na te które mają znajdować się w pamięci trwałej i te, które mogą bądź muszą znajdować się w pamięci ulotnej.

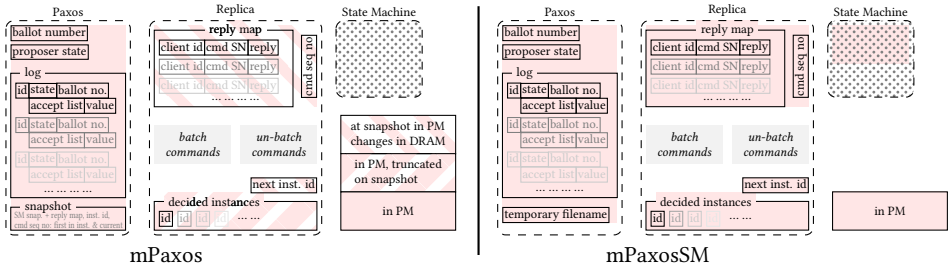
Istniejące metody wsparcia odtwarzania stanu dla SMR

Już artykuł wprowadzający Paxos [Lam89, Lam98] prezentuje sposób wsparcia odtwarzania uległych awarii replik: istotne danych są synchronicznie zapisie do pamięci która nie jest tracona przy awarii (*stable storage*). To rozwiązanie, na potrzeby rozprawy nazwane *FullSS*, jest w powszechnym użyciu [KA08, RST11]. [BDFG03] podaje metodę odtwarzania stanu zakładającą że przynajmniej większość replik nigdy nie ulegnie awarii oraz analizuje dla jakiej liczby procesów które mogą ulec awarii odtwarzanie jest możliwe w zależności od tego czy repliki korzystają ze stable storage. Proponowane w rozprawie ViewSS i EpochSS przyjmują bardziej praktyczne założenie że w każdej chwili większość replik działa. Konieczność przywracania do pracy replik które uległy awarii jest wskazywana w artykułach opisujących produkcyjne systemy oparte o Paxos [KA08, LMZ09, RST11]. Artykuły opisujące praktyczne zastosowania Paxos podkreślają też koszt periodycznego tworzenia migawek i transferu stanu potrzebnego do nadganiania. [BSF+13] proponuje optymalizacje zapisu na dysk zmniejszające koszt wsparcia odtwarzania i prezentuje sposób wykonania migawek pozwalający zmniejszyć wpływ ich tworzenia na przepustowość systemu.

W innych algorytmach konsensusu występują identyczne problemy dotyczące wpływu wsparcia odtwarzania na wydajność [OL88, LC12, HKJR10, OO14]. Godne uwagi są propozycje opracowane dla algorytmu Viewstamped Replication [LC12], które pozwalają na odzyskanie stanu z innych replik przy dodatkowych założeniach.

Pmem dostępne jest na rynku od niedawna, stąd nie pojawiły się jeszcze artykuły łączące pmem z Paxos czy SMR. Przed dostępnością pmem rozważano jedynie użycie go jako dodatkowy szybki i trwały cache w algorytmach konsensusu [RD17] oraz rozważano maskowanie awarii pamięci trwałej wykorzystując konsensus [DHL+18].

Alternatywnym do odtwarzania sposobem przywracania do pracy replik które uległy awarii jest rekonfiguracja [LMZ10, LMZ09, LC12, JLM15, JM14], polegająca na dynamiczniej zmianie listy replik. O ile możliwość zmiany listy replik jest pożądana w praktycznych zastosowaniach, o tyle używanie tej funkcjonalności do obsługi awarii wymaga wykrycia awarii przez system (fałszywe podejrzenia awarii generują niepotrzebne koszty) i dostarczanie klientom aktualnej listy replik (repliki nieświadome usunięcia z grupy mogą spowalniać klientów).



Rysunek 1: Podział danych między pmem i DRAM w systemach mPaxos i mPaxosSM.

Proponowane metody wsparcia odtwarzania

W pierwszej kolejności w rozprawie proponowane są systemy mPaxos i mPaxosSM wykorzystujące pmem. Dzielą one dane na te które muszą przetrwać awarię i są trzymane tylko w pmem, i pozostałe które przechowywane są w zwykłej pamięci ulotnej. Dla kontrastu, FullSS duplikuje wybrane dane w pamięci (dla szybkiego dostępu) i na dysku (dla trwałości). System mPaxos trzyma na bieżąco wszystkie kluczowe dane algorytmu Paxos w pmem. Dane części odpowiedzialnej za zreplikowanie maszyny stanowej i obsługę klientów są przechowywane w pmem w wersji spójnej z ostatnią migawką maszyny stanowej, a zmiany od ostatniej migawki są przechowywane w pamięci ulotnej (Rys. 1). Dzięki temu możliwe jest odtworzenie stanu na podstawie zawartości pamięci trwałej bez zmian w replikowanej maszynie stanowej.

System mPaxosSM zakłada zmianę interfejsu między frameworkiem i maszyną stanową. W mPaxosSM maszyna stanowa musi utrwalać efekt wykonania każdej operacji w pamięci nietraconej przy awarii (np. pmem) przed zwróceniem odpowiedzi do frameworku, natomiast z interfejsu usunięto całkowicie tworzenie migawek i od maszyny stanowej wymaga się tylko wsparcia chwilowego wstrzymania pracy na czas dostępu frameworku do obszaru pamięci trwałej maszyny stanowej. W wyniku takich zmian system mPaxosSM nie potrzebuje wykonywać i utrwalać regularnych migawek stanu, a w porównaniu do systemu mPaxos przechowuje również dane części odpowiedzialnej za replikację na bieżąco w pamięci trwałej (Rys. 1). Pozwala to na dalsze skrócenie czasu odtwarzania, a brak konieczności regularnego wykonywania migawek pozwala poprawić sprawność maszyny stanowej, która cały czas może przetwarzać komendy bez poświęcania zasobów na wykonywanie migawek.

Pozostałe proponowane metody wspierania odtwarzania uległych awarii replik, ViewSS oraz EpochSS, zakładają że w każdej chwili większość replik jest dostępna. Z jednej strony takie założenie ogranicza zastosowanie omawianych algorytmów. W przypadku niespełnienia tego założenia zachowane są warunki poprawności, ale dalsza praca systemu nie jest możliwa, a dane mogą zostać utracone. Z drugiej strony systemy które tak wspierają odtwarzanie są nie gorsze wydajnościowo niż systemy bez jakiegokolwiek wsparcia dla odtwarzania, ponadto w praktycznych zastosowaniach systemów zreplikowanej maszyny stanowej dąży się do zagwarantowania takiego założenia jako koniecznego dla nieprzerwanej dostępności systemu.

Przy założeniu że w każdej chwili większość replik jest dostępna, odtwarzająca się replika p w ViewSS i EpochSS zaczyna odtwarzanie od poinformowania pozostałych replik o awarii i odtwarzaniu, równocześnie pozyskując w odpowiedzi informa-

cje jaki stan p potrzebuje przywrócić, a odtwarzanie kończy po odzyskaniu wskazanego stanu (używając mechanizmu nadganiania). Faza odtwarzania musi też zapewnić że wiadomości wysłane przed awarią przez p, a otrzymane przez inne repliki po zakończeniu procedury odtwarzania nie naruszą poprawności działania systemu.

Zarówno w ViewSS i EpochSS każda replika utrzymuje w pamięci nietraconej w razie awarii (stable storage) jeden numer – w ViewSS jest to najwyższy znany numer lidera, w EpochSS liczba odtworzeń repliki. Podczas odtwarzania replika p wysyła ten numer do pozostałych replik, które odpowiadają najwyższym znanym numerem instancji i numerem lidera. Po zebraniu odpowiedzi od większości replik p wykonuje procedurę nadganiania aż osiągnie stan w którym wszystkie trwające instancje są przegłosowane, co kończy odtwarzanie. Dla radzenia sobie z wiadomościami wysłanymi przez p przed awarią we ViewSS pozostałe repliki muszą zmienić lidera przed wysłaniem odpowiedzi do p (o ile zmiana lidera nie nastąpiła od awarii p). W tym samym celu w EpochSS repliki zapamiętują liczbę odtworzeń p i w fazie wyboru lidera porównują ją z wartością zawartą w komunikatach tej fazy (które w EpochSS jako jedyne muszą być zabezpieczone przed wspomnianym problemem).

Porównanie działania proponowanych systemów

W rozprawie analizowany jest wpływ metod wsparcia odtwarzania na bezawaryjną pracę systemu jak i porównywane są kolejne kroki procesu odtwarzania po awarii.

System mPaxos w trakcie normalnej pracy niewiele różni się od zwykłego, niewspierającego odtwarzania systemu, jednak wymaga utrwalania danych w pmem. Poza niższą wydajnością pmem (w porównaniu z DRAM) oznacza to konieczność wymuszonego przesłania danych z pamięci podręcznej procesora do pmem oraz dodatkowe koszty wynikające z konieczności zachowania spójności danych w pmem na wypadek awarii (mogącej wydarzyć się w dowolnym momencie, również w trakcie logicznie niepodzielnego kroku algorytmu). System mPaxosSM przechowuje więcej danych w pmem, ale nie wymaga regularnego tworzenia migawek stanu, co redukuje nieodzowny w zwykłych systemach SMR koszt ograniczenia zużycia pamięci.

ViewSS wymaga zapisania do stable storage numeru głosowania przy każdej jego zmianie, co w trakcie bezawaryjnej pracy nie ma zwykle miejsca. EpochSS wymaga zapisu do stable storage tylko przy uruchamianiu repliki, zwiększa natomiast w niewielkim stopniu komunikaty fazy zmiany lidera. Zarówno ViewSS jaki i EpochSS nie zmieniają działania fazy zwykłego głosowania w porównaniu z systemem nie wspierającym odtwarzania, więc należy spodziewać się pełnego zachowania wydajności.

Koszt odtwarzania w systemach mPaxos i mPaxosSM zależy od tego, czy system wykonał postęp od awarii – jeśli tak, stan z pmem jest nieaktualny i po uruchomieniu replika potrzebuje wykonać nadganianie (identycznie jak w FullSS). Jeśli stan jest aktualny, systemy mPaxos i mPaxosSM nie muszą odtwarzać stanu algorytmu Paxos, gdyż ten znajduje się już w pamięci (pmem). Dla porównania, FullSS musi odbudować stan w pamięci na podstawie danych zgromadzonych w stable storage. W ViewSS i EpochSS repliki zawsze muszą wykonać nadganianie. Algorytm ViewSS może wymusić w trackie odtwarzania zmianę lidera, a więc wpłynąć negatywnie na wydajność systemu. Poza mPaxosSM każdy system (w tym mPaxos, nawet w przypadku gdy stan w pmem jest aktualny) musi wykonać żądania nieuwzględnione w migawce usługi. mPaxosSM korzysta zawsze z migawki bieżącego stanu, dzięki czemu liczba niewykonanych w niej żądań jest minimalna. Jednak w zamian, odtwarzanie w mPaxosSM jest opóźnione o czas odczytu danych z pamięci (czyli tworzenia migawki), a sama migawka może być większa niż w pozostałych systemach.

a)

Wielkość żądania	crash stop	FullSS	ViewSS	EpochSS
1024B (nasycona sieć)	38 087	1 871	38 200	38 209
128B (nasycony CPU)	158 288	11 164	157 317	159 192

b)

Wielkość żądania	crash stop	FullSS	ViewSS	EpochSS
1024B (nasycona sieć)	38 130	36 768	38 015	38 329
128B (nasycony CPU)	156 787	103 323	155 685	157 228

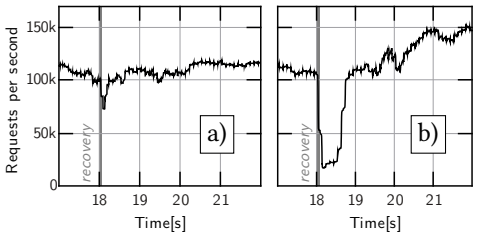
Tablica 1: Ilość żądań na sekundę wykorzystując a) HDD b) RAM dysk.

Ocena eksperymentalna

Proponowane metody, uznane za referencyjne algorytm FullSS oraz podejście niewspierające odtwarzania zostały zaimplementowane w oprogramowaniu JPaxos.

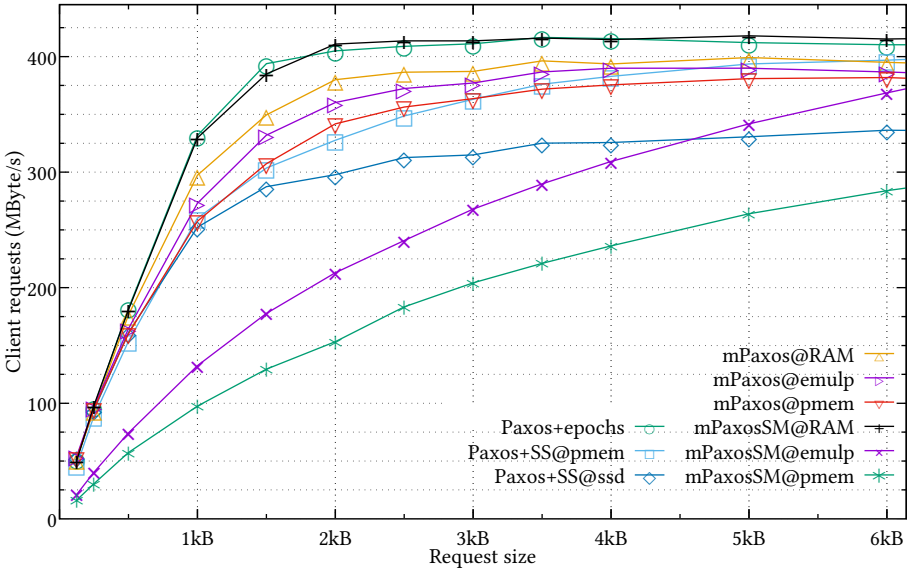
W wyniku przebiegu badań nad tematem i dostępności sprzętu ocena eksperymentalna została podzielona na dwa etapy: w pierwszym porównano system niewspierający odtwarzania, FullSS, ViewSS i EpochSS, w drugim porównano FullSS, EpochSS, mPaxos i mPaxosSM. W każdym etapie zbadano wpływ wybranej metody wsparcia odtwarzania stanu na przepustowość systemu w rutynowej pracy, wpływ awarii i odtwarzania na wydajność systemu, oraz czas trwania procesu odtwarzania, nadganiania stanu i wykonania nadgonionych komend na maszynie stanowej.

W pierwszym etapie porównanie wydajności systemu niewspierającego odtwarzania i opartych o ViewSS i EpochSS potwierdziło utrzymanie pełnej przepustowości tych systemów (Tab. 1). Przepustowość FullSS spada w różnym stopniu w zależności od charakteru obciążenia, będąc niższa nawet o ⅓ od wydajności porównywanych systemów dla małych żądań. Czas odtwarzania systemu opartego o FullSS okazał się mniejszy niż pozostałych dwóch, co pozwala szybciej przywrócić odporność systemu na kolejne awarie replik. Jednak po uwzględnieniu konieczności nadgonienia stanu EpochSS i ViewSS szybciej doprowadza replikę do pełnej sprawności. Sam proces odtwarzania stanu okazał się mieć zauważalny, choć nieznaczny wpływ na wydajność całego systemu, z pewnymi wyjątkami. W przypadku użycia magnetycznego dysku twardego (zamiast symulowanego wysokowydajnego dysku SSD) wymuszona przy odtworzeniu repliki przez algorytm ViewSS zmiana lidera doprowadziła do istotnego chwilowego spadku przepustowości (Rys. 2).



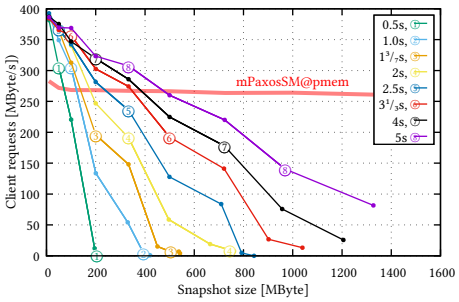
Rysunek 2: Wydajność ViewSS używającego: a) RAM dysk, b) HDD.

Testy z pierwszego etapu używały maszyny stanowej mającej minimalny wpływ na wydajność systemu. Drugi etap testów został przeprowadzony dla replik powielających magazyn klucz-wartość i na nowszym sprzęcie, wyposażonym w pamięć trwałą Intel Optane DCPMM. Dla miarodajnego porównania algorytmy FullSS i EpochSS również używały pamięci trwałej jako pamięć nietraconą przy awarii. W rutynowej pracy, po uwzględnieniu odpowiednich poprawek, przepustowość systemów kształtuje się w kolejności od najszybszego: EpochSS, mPaxos, FullSS, mPaxosSM (Rys. 3). Pierwsze trzy systemy są w stanie wykorzystać dostępną przepustowość najbardziej obciążonego łącza, a więc ich dalsze skalowanie ogranicza sieć. System mPaxosSM okazał się zauważalnie wolniejszy, z tego powodu został on poddany



Rysunek 3: Przepustowość dla różnych wielkości żądań.
(Paxos+epochs = EpochSS, Paxos+SS = FullSS)

profilowaniu. W jego wyniku można określić że przepustowość mPaxosSM jest ograniczana przez przepustowość zapisu do pamięci trwałej, oraz że liczby ani rozmiaru zapisów nie można zredukować. Poza szczegółową analizą wydajności systemów dla wybranego rozmiaru żądań przeprowadzono też ocenę pracy dla szerokiego zakresu rozmiarów żądań, jak również różnych ustawień tworzenia migawek. To pozwoliło na określenie dla jakich zastosowań koszt tworzenia migawek przewyższa spadek wydajności cechujący mPaxosSM, a więc na określenie dla jakich zastosowań mPaxosSM osiąga większą przepustowość od pozostałych systemów (przykład na Rys. 4).



Rysunek 4: Porównanie przepustowości mPaxosSM i FullSS dla żądań o rozmiarze 6kB i różnych ustawień migawek.

Analiza zachowania systemu w trakcie awarii i odtwarzania repliki dla systemów FullSS, EpochSS i mPaxos potwierdziła wyniki podobnej analizy wykonanej w pierwszym etapie eksperymentów, ponadto pokazała że mPaxos odtwarza się w czasie znacznie krótszym niż pozostałe, jednak proces późniejszego nadganiania przebiega wolniej niż dla FullSS (Tab. 2). Odtwarzanie repliki w mPaxosSM nie jest konieczne, bo wszystkie krytyczne dane przetwarzana znajdują się w trwałej pamięci operacyjnej. Konieczne jest, jak w każdym przypadku, nadgonienie stanu, które w mPaxosSM wymaga wykonania migawki na żądanie i w systemie z trzema replikami skutkuje istotnym spadkiem wydajności przez okres tworzenia takiej migawki. Dla pięciu replik spadek wydajności jest dużo niższy, bo poza repliką wykonującą migawkę i nad-

	Czas odtwarzana algorytmu Paxos	Czas nadganiania	
		algorytmu Paxos	wykonania żądań
FullSS	1.4s	4.6s	11s
EpochSS	1.7s		8s
mPaxos	0.4s	5.8s	16s
mPaxosSM	0.4s	2.3s	35s

Tablica 2: Czas potrzebny do odtworzenia repliki i nadgonienia stanu dla przykładowej aplikacji z migawką o rozmiarze 200MB i obszarem pmem o wielkości 512MB.

ganiającą stan w systemie pracują trzy repliki (a nie jedna) na które rozkłada się obsługa klientów. Z racji wspominanych limitów wydajnościowych pamięci trwałej przepustowość systemu mPaxosSM jest uzależniona od przepustowości wykonywania komend, co w procesie odtwarzania przełożyło się na znacznie wydłużony czas wykonania nadgonionych komend.

Wnioski

Rozprawa pokazuje, że dodanie wsparcia dla odtwarzania replik po awarii do systemów SMR opartych o Paxos nie musi wiązać się z istotnym spadkiem wydajności systemu w szerszej gamie zastosowań niż pozwalały na to dotychczasowe metody wsparcia odtwarzania. W systemach które potrzebują wsparcia również dla jednoczesnych awarii większości (w tym wszystkich) replik wykorzystanie pamięci trwałej redukuje narzut wydajnościowy związany ze wspieraniem odtwarzania, szczególnie dla niewielkich żądań (mPaxos). W systemach które zakładają że w każdej chwili większość replik jest dostępna odtwarzanie da się wspierać bezkosztowo (ViewSS, EpochSS). Ponadto, w zastosowaniach w których koszt periodycznego tworzenia migawek usługi jest wysoki użycie pamięci trwałej może poprawić wydajność nawet w porównaniu z podejściem niewspierającym odtwarzania (mPaxosSM). Poza wynikami dotyczącymi głównego tematu, rozprawa prezentuje też wnioski dotyczące pmem – wskazuje które parametry istniejącego sprzętu przyczyniają się do problemów wydajnościowych i wskazuje które funkcje dostępnych bibliotek dla pmem wymagają poprawy wydajności.

Ocena eksperymentalna prototypów systemów z proponowanymi metodami odtwarzania została przeprowadzona z wykorzystaniem sprzętu wypożyczonego przez Intel Polska.

Badania naukowe prezentowane w rozprawie są częścią projektu Persistent Datastore.
The Persistent Datastore project is supported by the Foundation for Polish Science, within the TEAM programme co-financed by the European Union under the European Regional Development Fund, grant No. POIR.04.04.00-00-5C5B/17-00.
Strona projektu: http://www.cs.put.poznan.pl/persistentdatastore

Bibliografia streszczenia rozprawy

[BDFG03] Romain Boichat, Partha Dutta, Svend Frølund, and Rachid Guerraoui. Deconstructing Paxos. *SIGACT News*, 34(1):47–67, March 2003.

[BSF+13] Alysson Neves Bessani, Marcel Santos, João Felix, Nuno Ferreira Neves, and Miguel Correia. On the efficiency of durable state machine replication. In *Proceedings of USENIX Annual Technical Conference, USENIX ATC '13*, June 2013.

[DHL+18] Huynh Tu Dang, Jaco Hofmann, Yang Liu, Marjan Radi, Dejan Vucinic, Robert Soulé, and Fernando Pedone. Consensus for non-volatile main memory. In *2018 IEEE 26th International Conference on Network Protocols, ICNP '18*, September 2018.

[HKJR10] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. ZooKeeper: wait-free coordination for Internet-scale systems. In *2010 USENIX Annual Technical Conference, USENIX ATC '10*, June 2010.

[JLM15] Leander Jehl, Tormod Erevik Lea, and Hein Meling. Replacement: Decentralized failure handling for replicated state machines. In *34th IEEE Symposium on Reliable Distributed Systems, SRDS '15*, September-October 2015.

[JM14] Leander Jehl and Hein Meling. Asynchronous Reconfiguration for Paxos State Machines. In *Distributed Computing and Networking - 15th International Conference, ICDCN '14*, January 2014.

[KA08] Jonathan Kirsch and Yair Amir. Paxos for System Builders: An Overview. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware, LADIS '08*, September 2008.

[Lam89] Leslie Lamport. The part-time Parliament. Technical Report 49, Systems Research Center, Digital Equipment Corp., Palo Alto, September 1989.

[Lam98] Leslie Lamport. The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.

[LC12] Barbara Liskov and James Cowling. Viewstamped Replication Revisited. Technical Report MIT-CSAIL-TR-2012-021, Computer Science and Artificial Intelligence Lab, MIT, July 2012.

[LMZ09] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical Paxos and primary-backup replication. In *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC '09*, August 2009.

[LMZ10] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, 2010.

[OL88] Brian M. Oki and Barbara H. Liskov. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC '88*, August 1988.

[OO14] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14*, June 2014.

[RD17] Amitabha Roy and Subramanya R. Dulloor. Cyclone: High availability for persistent key value stores. *CoRR*, abs/1711.06964, 2017.

[RST11] Jun Rao, Eugene J. Shekita, and Sandeep Tata. Using Paxos to build a scalable, consistent, and highly available datastore. In *Proceedings of the 37th International Conference on Very Large Data Bases, VLDB '11*, August-September 2011.

[Sca20] Steve Scargall. *Programming Persistent Memory: A Comprehensive Guide for Developers*. Springer Nature, 2020.

[Sch90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.