ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2017
Seria: ORGANIZACJA I ZARZĄDZANIE z. 113 Nr kol. 1992

Bogumiła HNATKOWSKA
Politechnika Wrocławska
Wydział Informatyki i Zarządzania
Bogumila.Hnatkowska@pwr.edu.pl

Bartłomiej ZALAS
Politechnika Wrocławska
Wydział Informatyki i Zarządzania
bartekzalas@gmail.com

OPTYMALIZACJA SYSTEMU W OPARCIU O TESTY WYDAJNOŚCI

Streszczenie. Praca jest poświęcona problemowi optymalizacji systemu w oparciu o testy wydajności. Przyjęte podejście polega na dynamicznej zmianie konfiguracji systemu na podstawie charakterystyki ruchu sieciowego. Zadanie jest realizowane przez system APES, który monitoruje czasy wykonania żądań i na ich podstawie przełącza konfigurację serwera aplikacji oraz komponentów redundantnych. W artykule opisano architekturę systemu. Skuteczność zaproponowanego rozwiązania została potwierdzona eksperymentalnie. Otrzymane wyniki wskazują, że system APES jest w stanie skrócić czas wykonywania żądań w większości badanych przypadków. Poprawa wydajności wyniosła od 3% do 820%.

Słowa kluczowe: wydajność systemu, optymalizacja, testy wydajności, komponenty redundantne

SYSTEM OPTIMISATION BASED ON PERFORMANCE TESTS

Abstract. The work is devoted to the problem of system optimization based on performance tests. The proposed approach dynamically changes the configuration of the system due to the characteristics of network traffic. The task is implemented by APES which monitors application execution times and, based on them, adapts the configuration of the application server and redundant components. The article presents the architecture of the system. The proposed solution was tested in conducted experiments. The results have confirmed that the solution is able to shorten the time of request processing in most of the cases. The improvement was between 3% and 820%.

Keywords: system performance, optimization, performance tests, redundant components

1. Wprowadzenie

Optymalizacja systemu jest procesem mającym na celu zmniejszenie zużycia wymaganych zasobów, skrócenie czasu odpowiedzi i/lub zwiększenie przepustowości [4, 5]. Proces ten może być wspierany przez testy wydajności, które pozwalają monitorować bieżący stan systemu.

Celem pracy jest zaproponowanie architektury rozwiązania, pozwalającego na monitorowanie stanu systemu (aplikacji webowej hostowanej na serwerze aplikacji) z wykorzystaniem testów wydajności i automatyczne przełączanie konfiguracji serwera aplikacji oraz wybranych komponentów składowych rozwiązania tak, aby zwiększyć wydajność systemu. Zakłada się, iż aplikacja webowa będzie uruchamiana na maszynie wirtualnej Java, na serwerze aplikacji Spring Boot (w wersji 1.5.2) z wbudowanym kontenerem Apache Tomcat.

Rozwiązanie ma wesprzeć architektów/administratorów, którzy zwykle ustawiają parametry pracy systemu i jego środowiska ręcznie, kierując się doświadczeniem. Podstawowym problemem jest mnogość parametrów konfiguracyjnych. Typowymi, wpływającymi na wydajność aplikacji pisanych w języku Java są [9, 14]: maksymalny rozmiar puli wątków (ang. Max thread pool size), rozmiar puli połączeń z bazą danych (ang. Data connection pool size), rozmiar puli obiektów EJB (ang. EJB pool size), czy rozmiar sterty maszyny wirtualnej (ang. JVM heap size).

Pozostała część artykułu ma następującą strukturę. W rozdziale drugim przedstawiono podstawowe terminy, a w trzecim – przegląd istniejących rozwiązań z zakresu dziedziny problemu. W rozdziale czwartym opisano architekturę systemu APES (Automated Performance Evaluation System), którego zadaniem jest automatyzacja optymalizacji systemu w oparciu o testy wydajności. W rozdziale piątym przedstawiono sposób oceny poprawności działania systemu. W ostatnim rozdziale podsumowano wyniki prac oraz wskazano kierunki dalszego rozwoju.

2. Przegląd literatury

2.1. Definicja podstawowych pojęć

Zgodnie z [12] wydajność jest związana z czasem odpowiedzi oraz przepustowością systemu pracującego pod pewnym obciążeniem i mającego pewną konfigurację. Obciążenie i konfiguracja to parametry wejściowe, podczas gdy czas/przepustowość to mierzalne parametry opisujące wydajność, tzw. wskaźniki wydajności (ang. key performance indicators). Innymi powszechnie stosowanymi wskaźnikami wydajności są [2, 12]:

wykorzystanie zasobów (ang. utilization) czy maksymalna liczba użytkowników obsługiwanych współbieżnie (ang. maximum users suport).

Pomiarów wskaźników wydajności dokonuje się najczęściej w ramach testów wydajności [3, 6] – w przypadku tej pracy – testów obciążeniowych.

Optymalizacja systemu jest procesem mającym na celu otrzymanie „najlepszych wyników w danych warunkach” [5]. Tutaj „najlepszy wynik” jest rozumiany jako najlepsza wydajność mierzona wskaźnikami wydajności (czas odpowiedzi, zużycie zasobów), natomiast „dane warunki” – jako aktualna konfiguracja systemu wraz z charakterystyką ruchu sieciowego.

Aplikacje webową można optymalizować na wiele sposobów zarówno po stronie klienta, jak i po stronie serwera. Artykuł dotyczy elementów konfigurowalnych po stronie serwera.

W zależności od architektury rozwiązania, aplikacja może być hostowana na kilku serwerach: serwerze aplikacji, serwerze bazy danych. W szczególności, optymalizować można operacje związane z dostępem do danych z poziomu serwera aplikacji, np. przez [1]:

- stronicowanie danych (ang. data paging)
- leniwe ładowanie (ang. lazy loading)
- grupowe wstawianie danych (ang. batched inserts)
- stosowanie pamięci podręcznej (ang. cache).

W pracy [9] autorzy proponują podejście do optymalizacji wydajności aplikacji składające się z kilku kroków:

1. Dla każdej wartości zmiennej konfiguracji ustal zakresy wartości.
2. Wygeneruj losową wartość z zakresu dla zmiennej konfiguracji.
3. Uruchom aplikację stosując wylosowaną wartość.
4. Zmierz wydajność aplikacji.
5. Powtarzaj kroki 2-4 aż wynik pomiarów wydajności będzie spełniać wymagania.

Zaproponowane podejście, ze względu na losowy dobór parametrów, jest mało wydajne. Jednakże, gdyby losowość zastąpić prostym systemem decyzyjnym, a sam proces zautomatyzować, podejście może być zaadoptowane do celów optymalizacji.

2.2. Istniejące podejścia do automatycznej optymalizacji wydajności

W pracy [10] autorzy zaproponowali mechanizm automatycznej optymalizacji systemu, w którym sterowali jednym parametrem – maksymalnym rozmiarem puli wątków – uzyskując poprawę wydajności o 37%. Parametr był dynamicznie zmieniany w zależności od aktualnego obciążenia serwera JBoss.

Bardziej ogólne podejście do optymalizacji systemu zostało zaproponowane w [11]. Autorzy proponują zbudowanie autonomicznego systemu, który w sposób ciągły monitoruje system i szuka sposobów poprawy wydajności.

Przykład realizacji idei autonomicznego systemu można znaleźć w pracy [8], w której opisano framework do automatycznego tuningowania. Framework składa się z dwóch modułów: monitorująco-diagnostycznego oraz adaptacyjnego. Część monitorująco-diagnostyczna zbiera dane o aktualnej wydajności aplikacji. Zebrane próbki są analizowane w kontekście sposobu wywołania metod i cyklu życia zdarzeń obiektów EJB i, w przypadku wykrycia problemów z wydajnością, zgłaszany jest alarm. W ramach jego obsługi moduł adaptacyjny próbuje rozwiązać problemy przez dobór odpowiednich komponentów programowych, których zastosowanie powinno obniżyć zużycie zasobów lub przyspieszyć wykonanie obsługi żądań.

Komponenty programowe dobierane przez moduł adaptacyjny nazywane są komponentami redundantnymi/nadmiarowymi (ang. redundant components). Ich zastosowanie zostało szerzej opisane w [7]. Komponent redundantny jest tam definiowany jako obecność, w czasie wykonania, wielu wariantów komponentów dostarczających identyczne lub podobne usługi, lecz realizowane z wykorzystaniem innych strategii optymalizacyjnych. Do obsługi konkretnego żądania użytkownika przypisany jest tylko jeden z dostępnych komponentów redundantnych. Komponenty, realizujące różne strategie, mogą być dodawane/usuwane w czasie działania systemu. Ich dobór następuje na podstawie wiedzy na temat kontekstu, jak również istniejącego opisu komponentów. Na opis komponentów składa się zbiór atrybutów, których wartości określają, kiedy użycie komponentu jest sensowne.

3. Opis proponowanego rozwiązania

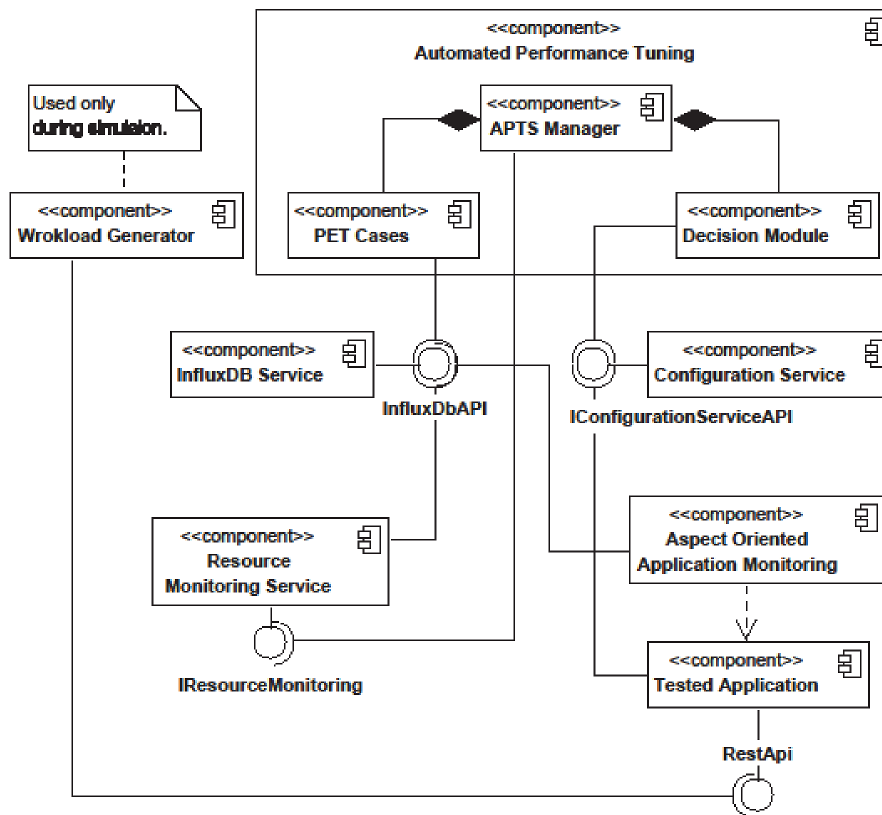
3.1. Założenia projektowe

Projektowany system APES powinien monitorować aktualne obciążenie systemu i przełączać parametry konfiguracyjne w zależności od tego obciążenia. Dodatkowym, ważnym wymaganiem jest, aby system mógł pracować w środowisku produkcyjnym, mając na nie minimalny wpływ.

3.2. Definicja architektury systemu

System APES ma budowę komponentową (patrz rys. 1). Podstawowymi komponentami są: generator obciążenia (Workload Generator), moduł tuningujący wydajność (Automated Performance Tunning), usługa bazodanowa (InfluxDB Service), usługa konfiguracyjna (Configuration Service), usługa monitorująca zużycie zasobów (Resource Monitoring Service), usługa monitorująca aplikację (Aspect Oriented Appication Monitoring). Na

diagramie, jako komponent, przedstawiono również aplikację, której wydajność chcemy poprawić (Tested Application).



Rys. 1. Komponentowa architektura systemu APES
Źródło: Opracowanie własne.

Generator obciążenia jest odpowiedzialny za symulację ruchu sieciowego (z wykorzystaniem biblioteki Gatling [13]) o zadanej charakterystyce, który jest kierowany do testowanej aplikacji. Moduł ten komunikuje się bezpośrednio z testowaną aplikacją, dzięki czemu może być łatwo wyłączony w środowisku produkcyjnym.

Głównym elementem rozwiązania jest moduł tuningujący wydajność (Automated Performance Tuning), który składa się z trzech podmodułów:

- zarządcy APTS (APTS Manager) – jest odpowiedzialny za uruchomienie testów wydajności oraz raportowanie problemów wydajności do modułu decyzyjnego
- moduł decyzyjny (Decision Module) – jest odpowiedzialny za analizę wydajności i przełączanie parametrów konfiguracyjnych
- moduł analizy przypadków testowych (PET Cases) – reprezentuje scenariusze testów wydajności w postaci harmonogramów; zbiera wartości wskaźników wydajności z usługi bazodanowej i sprawdza je względem założonych warunków; w przypadku, gdy wartości nie mieszczą się w założonych zakresach – powiadamia o tym moduł decyzyjny za pośrednictwem zarządcy.

Usługa bazodanowa (InfluxDB Service) hostuje bazę danych InfluxDB [15], w której przechowywane są pomiary wskaźników wydajności testowanej aplikacji.

Usługa konfiguracyjna (Configuration service) jest komponentem, w którym jest przechowywana aktualna konfiguracja systemu. Usługa ta dostarcza REST API, pozwalającego sprawdzić aktualną konfigurację i ją zmienić.

Usługa monitorowania zasobów (Resource Monitoring Service) jest odpowiedzialna za dostarczanie informacji o bieżącym zużyciu zasobów na komputerze, na którym jest zainstalowana testowana aplikacja. Komponent ten jest kontrolowany przez zarządcę, który włącza/wyłącza proces monitorowania, wysyłając do usługi monitorowania odpowiednie żądania http.

Usługa monitorująca aplikację (Aspect Oriented Application Monitoring) dostarcza informacji na temat czasu wykonania poszczególnych metod przez aplikację testową. Realizacja usługi monitorującej z wykorzystaniem programowania aspektowego pozwoliła ją uniezależnić od testowanej aplikacji. Czasy wykonania metod są rejestrowane z usłudze bazodanowej.

4. Implementacja proponowanego rozwiązania

4.1. Parametry konfiguracyjne i komponenty redundantne

W celu potwierdzenia wykonalności i skuteczności zaproponowanego rozwiązania dokonano wyboru parametrów konfiguracyjnych oraz komponentów redundantnych, które będą przełączane przez usługę konfiguracyjną. Sensowność doboru tych elementów (ich wpływ na wydajność) potwierdzono osobnymi mini eksperymentami.

Zdecydowano o implementacji dwóch komponentów redundantnych (pamięć podręczna – cache, skumulowane wstawianie danych – batch insert) oraz sterowanie jednym parametrem konfiguracyjnym (rozmiar puli wątków – threads) – szczegóły są podane w tabeli 1.

Tabela 1

Opis komponentów redundantnych i parametrów konfiguracyjnych

Parametr	Możliwe wartości	Opis
Cache	NO_CACHE	Pamięć podręczna obsługiwana przez komponent NoCache
	CACHED	Pamięć podręczna obsługiwana przez komponent Cache
Batch	DIRECT	Wstawianie rekordów obsługiwane przez komponent Direct
	BATCHED	Wstawianie rekordów obsługiwane przez komponent Batched
Threads	T10	Serwer aplikacji używa 10 wątków
	T20	Serwer aplikacji używa 20 wątków
	T30	Serwer aplikacji używa 30 wątków

Źródło: Opracowanie własne.

Komponent Cache (zastosowano Spring Cache w wersji 4.3.7, z adnotacjami @Cacheable oraz @CacheEvict(allEntries = true)) wykorzystuje pamięć podręczną dla wyników metod,

podczas gdy komponent NoCache za każdym razem komunikuje się z bazą danych w celu odczytania wartości zwracanych do klienta. Podobnie, komponent Direct dodaje rekordy pojedynczo, podczas gdy Batched dodaje je w paczkach po 10 sztuk.

4.2. Mierzone wskaźniki wydajności

W celu umożliwienia sterowania elementami systemu, opisanymi w tabeli 1, zaimplementowano osobny framework, którego zadaniem jest pomiar i monitorowanie wybranych wskaźników wydajności. Są to:

- średni czas wykonania metod (MeanTime) – jeżeli czas wykonania jest większy niż 1.2s w ciągu ostatnich 10s framework generuje wartość EXCEEDED; wartość 1.2s została ustalona eksperymentalnie – przy niskim obciążeniu żądania były obsługiwane w czasie poniżej 1s
- liczba wstawień rekordów (InsertLevel) – framework monitoruje liczbę rekordów wstawionych w ciągu 1s przez testowaną aplikację do bazy danych; generuje wartość LOW gdy liczba wstawień/s w ciągu ostatnich 10s mniejsza niż 10; HIGH – w przeciwnym przypadku
- obciążenie procesora (CpuUsage) – framework monitoruje bieżącą zajętość procesora wyrażoną w procentach; generuje wartości: LOW – zajętość procesora poniżej 80%, HIGH – w przeciwnym przypadku.

4.3. Reguły decyzyjne

Moduł decyzyjny jest odpowiedzialny za ocenę aktualnego stanu i przełączenie systemu. W jego ramach zaimplementowano prosty system decyzyjny, na który składają się: zestaw reguł oraz silnik wnioskujący, którą akcję wykonać. Zdefiniowano ogółem 7 akcji: włącz batch (parametr Batch ustawiony na BATCHED), wyłącz batch (Batch ma wartość DIRECT), włącz cache (parametr Cache ustawiony na CACHE), wyłącz cache (Cache ustawiony na NO_CHACE), ustaw 10 wątków, ustaw 20 wątków, ustaw 30 wątków.

Przykładowe reguły decyzyjne są pokazane poniżej:

If InsertLevel = HIGH and Batch = DIRECT then Batch = BATCHED

If TrafficProfile = IMMUTABLE and Cache = NO_CACHE then Cache = CACHE

If MeanTime = EXCEEDED and Threads != T10 and CpuUsage = HIGH then Thread = T10

Ostatnia z reguł oznacza, że jeżeli czas wykonywania operacji jest długi, serwer obsługuje ponad 10 wątków, co powoduje duże zużycie procesora, to należy przełączyć liczbę wątków na 10.

5. Ocena proponowanego rozwiązania

5.1. Testowana aplikacja

Testowana aplikacja została zaimplementowana w części działającej po stronie serwera. Składa się na nią prosta baza danych przechowująca informacje o produktach, podzielonych na kategorie, oraz o związanych z produktami opiniach. Baza danych, posadowiona na systemie Postgres w wersji 9.6, została wypełniona losowymi danymi:

- kategorie produktów: 10000 elementów
- produkty: 20000 elementów
- opinie o produktach: 50000 elementów

Aplikacja dostarcza zestawu podstawowych usług do zarządzania danymi (REST API). Łącznie zaimplementowano 10 usług, w tym symulującą duże obciążenie procesora (obliczenie 40stej wartości ciągu Fibonacciego) oraz symulującą oczekiwanie na odpowiedź z pewnym okresem bezczynności podanym w milisekundach (bez obciążania procesora).

5.2. Środowisko testowe

System APES został zainstalowany na komputerze o następującej specyfikacji: procesor: Intel Core i3-3120M CPU 2.50GHz, 2 rdzenie, 4 logiczne procesory; pamięć fizyczna: 12 GB; system operacyjny: MicrosoftWindows 10 Education; wersja Java: 1.8.0_121.

Rozmieszczenie poszczególnych komponentów w węzłach zilustrowano na rys. 2. Zarządca APTS oraz generator obciążenia działają bezpośrednio na stacji klienckiej. Testowa aplikacja jest umieszczona na osobnej maszynie wirtualnej, podobnie jak usługa konfiguracyjna. Zabieg ten miał zmniejszyć wpływ systemu APES na testowaną aplikację.

Ponieważ zmiana liczby wątków wymaga restartu serwera, zdecydowano o umieszczeniu trzech niezależnych instancji testowanej aplikacji na osobnych instancjach Tomcat, obsługujących odpowiednio 10, 20 i 30 wątków, nasłuchujących żądań na innych portach. Wysyłaniem żądań do odpowiednich instancji, w zależności od przyjętej konfiguracji, zajmuje się dodatkowy komponent Dispatcher.

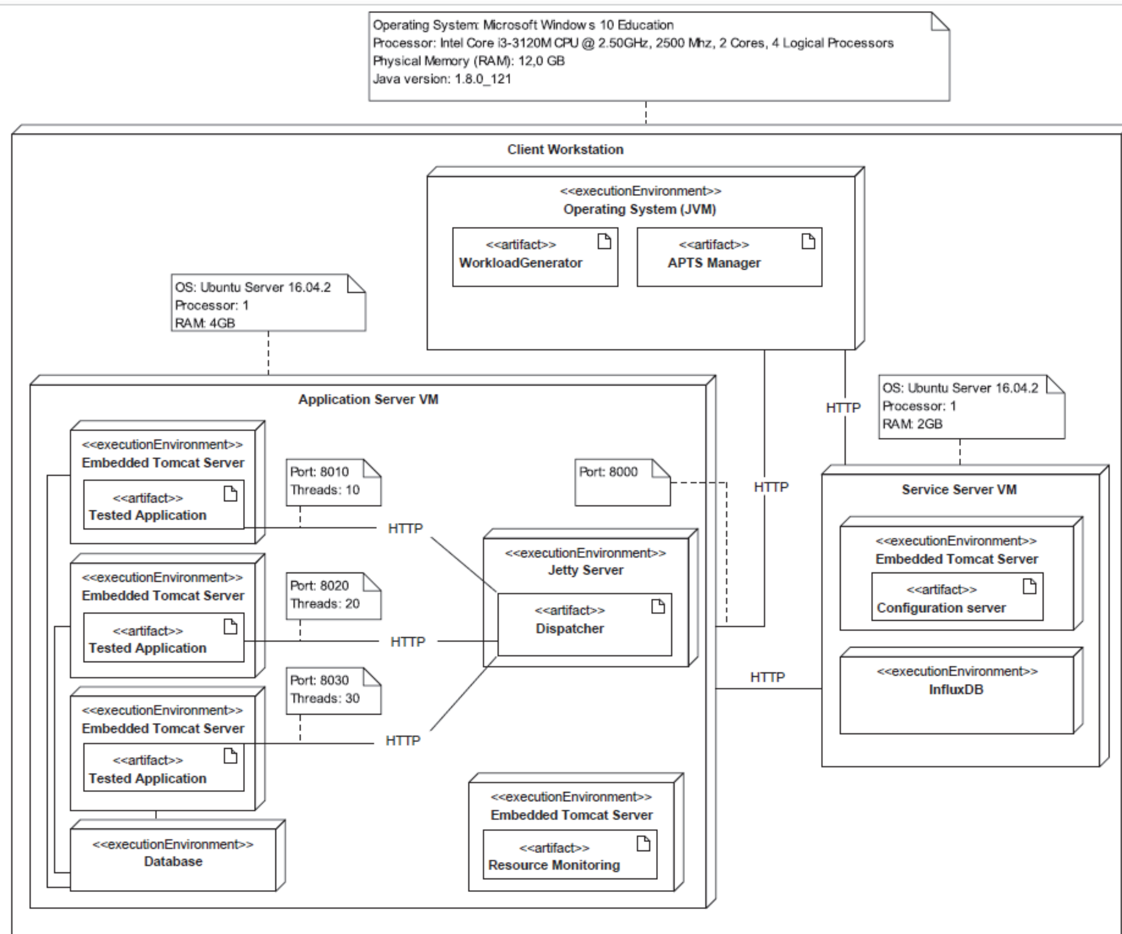
5.3. Scenariusze testowe

Przygotowano pięć bazowych scenariuszy testowych, z czego poniżej opisano jeden. Pierwsze trzy badały – przy specjalnie spreparowanym ruchu sieciowym – elementy konfiguracji w izolacji, czwarty – ich kombinację. Eksperymenty te potwierdziły poprawność

działania systemu APES. Ostatni eksperyment miał na celu sprawdzenie skuteczności działania systemu przy losowym ruchu sieciowym. Poniżej znajduje się opis jego przebiegu.

Przeprowadzono 2 niezależne typy eksperymentów dla następujących konfiguracji systemu:

1) konfiguracja domyślna (komponenty: NoCache, Direct, 20 wątków)
2) tryb adaptacji (system APES przełączal komponenty redundantne: Cache/NoCache, Direct/Batch oraz sterował liczbą działających wątków: 10, 20, 30).



Rys. 2. Środowisko testowe – diagram rozmieszczenia
Źródło: Opracowanie własne.

Eksperyment dla każdej konfiguracji trwał 10 minut, w trakcie których generowane były żądania następujących kategorii: (1) wyszukanie kategorii produktu na podstawie id, (2) modyfikacja kategorii produktu, (3) dodanie nowej kategorii produktu, (4) generacja żądania obciążającego procesor, (5) generacja żądania oczekiwania z okresem bezczynności.

Każdy typ żądania był generowany na podstawie 4rech losowych parametrów: *UserLowerBoundary*, *UserUpperBoundary*, *StartAt*, *Duration*, gdzie:

- *UserLowerBoundary/UserUpperBoundary* – odpowiednio minimalna/maksymalna liczbę użytkowników generujących żądania
- *StartAt* – sekunda, w której rozpoczyna się generacja żądań danego typu

- *Duration* – okres (w sekundach) generacji żądań, przy czym *StartAt* + *Duration* nie może przekraczać 10 minut.

Ze względu na ograniczone zasoby komputerowe przyjęto, iż *UserLowerBoundary* oraz *UserUpperBoundary* są wartościami z przedziału 0-6 dla wszystkich typów żądań z wyjątkiem żądania obciążającego procesor, dla którego górną granicą jest 1.

Eksperyment powtórzono 10 razy. Uzyskane wyniki dla wszystkich 10 prób przedstawiono zbiorczo w tabeli 2.

Tabela 2

Ocena poprawy wydajności dla trybów konfiguracja domyślna/tryb adaptacji

Lp.	Konfiguracja	CPU [%]	CT/AT [ms]	I [%]
1	Domyślna	92	4,00	–
	Tryb adaptacji	93	3,50	14,29
2	Domyślna	43	2,80	–
	Tryb adaptacji	45	0,53	428,30
3	Domyślna	88	0,45	–
	Tryb adaptacji	69	0,53	-15,90
4	Domyślna	96	16,00	–
	Tryb adaptacji	76	3,40	370,59
5	Domyślna	67	3,40	–
	Tryb adaptacji	68	1,90	78,95
6	Domyślna	50	6,60	–
	Tryb adaptacji	19	0,88	650,00
7	Domyślna	86	2,20	–
	Tryb adaptacji	69	0,49	348,98
8	Domyślna	75	3,22	–
	Tryb adaptacji	54	0,35	820,00
9	Domyślna	72	4,20	–
	Tryb adaptacji	57	2,00	110,00
10	Domyślna	96	2,30	–
	Tryb adaptacji	95	1,40	64,29

Źródło: Opracowanie własne.

Poprawę wydajności systemu dla konfiguracji 2 w stosunku do konfiguracji 1 obliczono według wzoru:

$$I = \frac{CT - AT}{AT} * 100\% \qquad (1)$$

gdzie:

CT – średni czas obsługi żądania dla próbki kontrolnej (konfiguracja domyślna)

AT – średni czas obsługi żądania dla próbki badawczej (tryb adaptacji)

W 9-ciu na 10 prób system APES był w stanie poprawić wydajność systemu w stosunku do konfiguracji domyślnej. Jedynie w próbie 3ciej wydajność systemu się pogorszyła. W tym przypadku dłużej były obsługiwane żądania typu oczekiwanie procesora i wstawianie rekordów z powodu przełączenia systemu w tryb pracy z 10 wątkami (z 20 startowych). „Stare" żądania musiały być obsłużone na instancji serwera pracującej z 20 wątkami, podczas gdy nowe były kierowane do instancji pracującej z 10 wątkami. Przez krótki okres po

przełączeniu konfiguracji dochodziło do „przeciążenia" procesora i dłuższej obsługi samych wątków. System APES nie był w stanie skompensować straty w kolejnej fazie, gdyż żądania były generowane przez relatywnie krótki okres. W pozostałych przypadkach testowych uzyskano poprawę wydajności w zakresie od 14,29% do 820% w zależności od wygenerowanego ruchu sieciowego.

6. Podsumowanie

Celem pracy była prezentacja architektury systemu, który – na podstawie wyników testów wydajności – jest w stanie przełączać konfigurację systemu tak, aby czas odpowiedzi był jak najkrótszy. System jest autonomiczny, działa bez ingerencji człowieka, zgodnie ze zdefiniowanymi regułami decyzyjnymi. Jest również przygotowany do pracy w środowisku produkcyjnym. Komponenty odpowiedzialne za monitorowanie wydajności i optymalizację są posadowione poza tym środowiskiem.

System APES jest w pewnym stopniu zależny od testowanej aplikacji. Zakłada się, że optymalizowana aplikacja dostarczy usługi konfiguracyjnej, która przechowuje informację o aktualnie używanej konfiguracji i dostarczy API do jej zmiany w czasie działania. Ponadto, zakłada się, iż na tym samym węźle co aplikacja będą posadowione usługi związane z jej monitorowaniem.

Skuteczność systemu APES została potwierdzona eksperymentalnie. W ponad 90% przypadków udało się uzyskać znaczące skrócenie czasu obsługi żądań.

W przyszłości planowany jest rozwój systemu w dwóch obszarach. Pierwszym jest poszerzenie liczby parametrów konfiguracyjnych i komponentów redundantnych, którymi można sterować. Drugim jest zastąpienie prostego systemu regułowego mechanizmami uczenia maszynowego, np. siecią neuronową, która mogłaby rozpoznać ukryte wzorce w przychodzących danych i odpowiednio do tych wzorców dobierać właściwe komponenty redundantne.

Bibliografia

1. Chakraborty, A., Ditt, J., Vukotic, A., Machacek, J.: Pro Spring 2.5. Apress 2008, Berkeley, CA, p. 829–855.
2. Molyneaux I.: The Art of Application Performance Testing. O'Reilly Media 2015, Sebastopol USA.

3. Myers G.J. , Sandler C., Badgett T.: The art of software testing, Third Edition. JohnWiley & Sons, Inc. 2011, New Jersey USA.
4. Oaks S.: Java Performance: The Definitive Guide, O'Reilly Media 2014, Sebastopol USA.
5. Rao S.S.: Engineering Optimization Theory and Practice. John Wiley & Sons Inc. 2009, New Jersey USA.
6. Spillner A., Linz T., Schaefer H.: Software testing foundations, Fourth Edition, O'Reilly Media 2014, Sebastopol, California, USA.
7. Diaconescu A.: A framework for using component redundancy for self-adapting and self optimising component-based enterprise systems, [in:] Crocker R., Steele G.L. (eds.): Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '03). ACM, New York, NY, USA, p. 390–391.
8. Diaconescu A., Mos A., Murphy J.: Automatic the performance management in component based software systems, [in:] 1st International Conference on Autonomic Computing, ICAC 2004, IEEE Computer Society 2004, p. 214–221.
9. Raghavachari M., Reimer D., Johnson R.D.: The deployer's problem: configuring application servers for performance and reliability, [in:] Clarke L.A., Dillon L., Tichy W.F (eds.): Proceedings of the 25th International Conference on Software Engineering, IEEE Computer Society, p. 484–489.
10. Zhang Y., Qu W., Liu A.: Automatic Performance Tuning for J2EE Application Server Systems. [in:] Ngu A.H.H., Kitsuregawa M., Neuhold E.J., Chung J-Y, Sheng Q.Z. (eds.): Web Information Systems Engineering – WISE 2005: 6th International Conference on Web Information Systems Engineering, Springer Berlin Heidelberg 2005, Berlin, Heidelberg, p. 520–527.
11. Kephart J. O., Chess D.M.: The vision of autonomic computing. "Computer", Vol. 36, No. 1, 2003, p. 41–50.
12. Sarojadevi H.: Performance Testing: Methodologies and Tools. "Journal of Information Engineering and Applications", Vol. 1, No. 5, 2011.
13. Gatling Load and Performance testing - Open-source load and performance testing. *http://gatling.io*, (access: 10.08.2017).
14. GlassFish Server Open Source Edition, Performance Tuning Guide, https://glassfish.java.net/docs/4.0/performance-tuning-guide.pdf, Oracle, May 2013, (access: 10.08.2017).
15. InfluxData (InfluxDB) – Open Source Time Series Database for Monitoring Metrics and Events; *https://www.influxdata.com*, (access: 10.08.2017).