Planowanie przepływów pracy zorientowanych na usługi w chmurach przy użyciu podejścia adaptacyjnego opartego na agentach

Ponieważ coraz więcej użytkowników zaczyna używać chmur do wdrażania złożonych aplikacji i przechowywania zdalnych danych, rośnie potrzeba minimalizacji kosztów użytkowników. Ponadto wielu dostawców usług chmurowych zaczyna oferować specjalistyczne usługi, a tym samym pojawia się potrzeba wyboru najlepszej możliwej usługi pod względem czasu lub ograniczeń finansowych. Sprzedawcy, którzy nie tylko świadczą specjalistyczne usługi, ale także preferują stosowanie własnych zasad planowania i często wybierają własne strategie negocjacyjne. Aby jeszcze bardziej skomplikować, złożone aplikacje składają się zwykle z mniejszych zadań (np. aplikacji przepływu pracy) zaaranżowanych razem przez silnik kompozycji. W tym wysoce dynamicznym i nieprzewidywalnym środowisku systemy wieloagentowe wydają się być jednym z najlepszych rozwiązań. Agenci są domyślnie niezależni, ponieważ działają w swoim najlepszym interesie, kierując się własną polityką, ale także współpracują ze sobą, aby osiągnąć wspólny cel. W ramach planowania przepływu pracy cel jest reprezentowany przez minimalizację całkowitego kosztu użytkownika. Tu przedstawiono niektóre z wyzwań, jakie napotykają systemy wieloagentowe podczas planowania zadań. Przedstawiono również rozwiązania tych problemów oraz przedstawiono prototypową platformę do planowania wieloagentowego.

Wstęp

W ostatnich latach Cloud Computing (CC) stał się wiodącym rozwiązaniem w dziedzinie Distributed Computing (DC). W przeciwieństwie do Grid Computing brakowało otwartej wizji przezwyciężenia pewnych podstawowych problemów, w tym przejrzystego i łatwego dostępu do zasobów, kwestii licencyjnych lub politycznych, braku obsługi wirtualizacji lub skomplikowanych w użyciu architektur i narzędzi użytkownika końcowego. Chmury stały się głównym wyborem dla dostawców usług głównie ze względu na ich wsparcie dla wirtualizacji i podejścia zorientowanego na usługi. W chmurach prawie wszystko może być oferowane jako usługa. Doprowadziło to do pojawienia się kilku paradygmatów, w tym oprogramowania jako usługi (SaaS), infrastruktury jako usługi (IaaS) lub platformy jako usługi (PaaS). Ponieważ coraz więcej użytkowników zaczyna używać chmur do przechowywania lub wykonywania swoich aplikacji, systemy te stają się podatne na problemy związane z obciążeniem. Problem jest jeszcze trudniejszy, gdy rozważamy złożone zadania, które wymagają dostępu do usług dostarczanych przez różnych dostawców chmury, z których każdy ma własną politykę wewnętrzną. Wybór optymalnej/najszybszej usługi do konkretnego zadania staje się w tym przypadku istotnym problemem, ponieważ czasami użytkownicy płacą za czas spędzony na korzystaniu z usług bazowych. W konsekwencji planowanie zadań w usługach staje się jeszcze trudniejsze, ponieważ w środowiskach chmurowych każdy członek korzysta z własnych zasad i nie jest zobowiązany do przestrzegania zasad zewnętrznych. Kończymy z pakietem usług od różnych dostawców, które muszą być ze sobą zaaranżowane, aby uzyskać pożądany rezultat w określonym przedziale czasu. Utrzymanie realizacji w tym przedziale minimalizuje koszty produkcji i klienta. Ponieważ wybór usługi wymaga pewnych negocjacji między dostawcami, jednym z najprostszych i najprostszych rozwiązań jest wykorzystanie rozproszonych agentów, którzy pełnią role dostawców usług i klientów. W artykule przedstawiono podejście agentowe do problemu harmonogramowania zadań w chmurach. Rozwiązano dwa główne problemy: znajdowanie zasobów w chmurze i organizowanie usług od różnych dostawców chmury w celu osiągnięcia wspólnego celu. Zakłada się, że istnieją pewne ograniczenia dotyczące terminów i kosztów. Mimo że nacisk kładzie się na zadania przepływu pracy, można również obsługiwać zadania niezależne. W tym celu najpierw przedstawiamy rozwiązania problemu harmonogramowania zadań w chmurach. Następnie przedstawiamy zagadnienia dotyczące harmonogramowania zadań w środowiskach zorientowanych na usługi (SOE) wraz z kilkoma szczegółami dotyczącymi

harmonogramowania przepływu pracy. Podano również szczegółowy przegląd architektury platformy planowania opartej na agentach rozproszonych, która może dostosowywać się do zmian zasobów. Na koniec przedstawiono konkretny prototyp eksperymentalny i pewne wnioski.

Powiązane prace nad DS Scheduling

Wykonano wiele pracy w zakresie planowania zadań w systemach rozproszonych (DS). Prace te można podzielić na wyspecjalizowane algorytmy planowania (SA) dla chmur oraz systemy zarządzania zasobami (RMS), które wykrywają usługi i przydzielają im zadania. Poniżej pokrótce przedstawiamy niektóre z głównych prac dotyczących zarówno SA dla CC, jak i RMS dla DS. Jeśli chodzi o rozwój wydajnych SA dla DS, dobrym miejscem inspiracji okazała się natura. Ostatnie artykuły, takie jak Lorpunmanee, Sap, Abdullah i Chompooinwai oraz Ritchie i Levine, próbują uporać się z problemem planowania zadań, oferując metaheurystyki inspirowane wzorcami zachowań zaobserwowanymi w koloniach mrówek. Technika ta, zwana również optymalizacją kolonii mrówek (ACO), opiera się na fakcie, że mrówki w kolonii działają jako niezależni agenci, którzy próbują znaleźć najlepsze dostępne zasoby w swojej przestrzeni za pomocą globalnych technik wyszukiwania. Za każdym razem, gdy taki agent znajdzie zasób lepszy od już istniejącego, wyznacza do niego drogę za pomocą feromonów. Przyciągają one inne mrówki, które zaczynają korzystać z zasobu, dopóki nie znajdzie lepszego. W Banerjee, Mukherjee i Mahanti zaproponowano podejście oparte na ACO do inicjowania dystrybucji obciążenia usługi w chmurach. Symulowane wyniki w Google Application Engine (2010) i Microsoft Live Mesh (2010) wykazały niewielką poprawę przepustowości usług w chmurze przy użyciu proponowanego zmodyfikowanego algorytmu ACO. Największą wadą ACO w porównaniu z innymi podejściami jest to, że nie jest zbyt efektywne, gdy rozważa się planowanie dynamiczne. Powodem tego jest to, że zmiana harmonogramu wymaga dużo czasu, zanim optymalny scenariusz zostanie osiągnięty dzięki intensywnemu szkoleniu prowadzonemu w wielu iteracjach. Ponieważ DS są zarówno nieprzewidywalne, jak i niejednorodne za każdym razem, gdy zauważona zostanie zmiana, cały system musi zostać ponownie przeszkolony. Ten proces, który może trwać kilka godzin. Duży przedział czasu ponownego uczenia jest nie do zaakceptowania, gdy zadania są planowane w ramach ograniczeń terminów, ponieważ planowanie może trwać dłużej niż rzeczywiste wykonanie zadania. Poprawę tego można osiągnąć, łącząc czasochłonne wyszukiwanie globalne z wyszukiwaniem lokalnym w przypadku drobnych zmian w DS. Jednak zdefiniowanie pojęcia drobnych zmian jest nadal kwestią otwartą. Artykuł dotyczy harmonogramowania zadań High Performance Computing (HPC) w chmurach. Zużycie energii jest ważne zarówno pod względem kosztów użytkowania, jak i emisji dwutlenku węgla. Proponowany meta-scheduler uwzględnia takie czynniki, jak koszty energii, poziom emisji dwutlenku węgla, wydajność procesora i przepływy zasobów przy wyborze odpowiedniego centrum danych należącego do dostawcy chmury. Zaprojektowana heurystyka planowania oparta na energii wykazuje znaczny wzrost oszczędności energii w porównaniu z innymi politykami. Większość prac dotyczących RMS opierała się na założeniu stosowania ich w sieciach, a nie w chmurach. Można to wyjaśnić dwoma faktami. Po pierwsze, istnieje wiele podobieństw między chmurą a siatką, a RMS opracowany dla jednego typu może również dobrze działać w przypadku drugiego. Drugi jest związany z wiekiem, a ponieważ gridy pojawiły się wcześniej niż chmury, większość rozwiązań została opracowana dla tych pierwszych. Niemniej jednak kilka systemów RMS zorientowanych na siatkę można by przystosować również do pracy w chmurach. Jednym z przykładów jest Cloud Scheduler (2010). Pozwala użytkownikom skonfigurować maszynę wirtualną (VM) i przesyłać zadania do puli Condor. Maszyna wirtualna zostanie zreplikowana na komputerach i wykorzystana jako kontener do wykonywania zadań. Poniżej przedstawiamy niektóre z najbardziej znanych przykładów RMS dla DS w ogóle. Godne uwagi przykłady obejmują Globus-GRAM, Nimrod/G, Condor, Legion, NetSolve i inne. Wiele z tych rozwiązań używa stałych aparatów zapytań do wykrywania i publikowania zasobów i nie opiera się na zaletach oferowanych przez agentów rozproszonych. System ARMS stanowi przykład RMS opartego na

agentach. Używa PACE do przewidywania wydajności aplikacji, które są później używane jako dane wejściowe do mechanizmu planowania. W artykule przedstawiono podejście do planowania oparte na agentach dla wielu lokalizacji, składające się z dwóch odrębnych poziomów decyzyjnych, jednego globalnego, a drugiego lokalnego. Każdy z tych poziomów zawiera predykcyjny i reaktywny składnik do radzenia sobie z dystrybucją obciążenia i reagowania na zmiany w obciążeniach. Artykuł przedstawia podejście do równoważenia obciążenia sieci poprzez połączenie inteligentnych agentów i podejścia wieloagentowego. Każdy istniejący agent jest odpowiedzialny za obsługę planowania zadań w wielu zasobach w siatce. Podobnie jak u Sauera i innych, istnieje również hierarchia agentów, którzy współpracują ze sobą na zasadzie peer to peer w celu wspólnego celu, jakim jest znalezienie nowych zasobów dla swoich zadań. Ta hierarchia składa się z brokera, kilku koordynatorów i prostych agentów. Wykorzystując procesy ewolucyjne, SA są w stanie radzić sobie ze zmianami liczby zadań lub zasobów. Nimrod/G używa agentów do obsługi konfiguracji działającego środowiska, transportu zadania na miejsce, jego wykonania i zwrotu wyniku do klienta. Agenci mogą również rejestrować informacje pozyskane podczas wykonywania zadania, takie jak czas procesora, zużycie pamięci itp. Paper proponuje system, który z różnych modeli negocjacji, protokołów lub strategii może automatycznie wybrać najlepszy dla bieżących potrzeb obliczeniowych i zmian w środowisku zasobów. Czyni to, rozwiązując dwa główne problemy, z którymi DS musi się zmierzyć Cao i in.: skalowalność i adaptacyjność. Prace wykonane w Shen et al. tworzą architekturę, która wykorzystuje kilka wyspecjalizowanych agentów dla aplikacji, zasobów, żółtych stron i zadań. Na przykład pośrednicy pracy są odpowiedzialni za obsługę zlecenia od momentu jego złożenia i do czasu jego wykonania, a ich żywotność jest ograniczona do tego przedziału. Ramy oferują kilka modeli negocjacji między agentami pracy i zasobów, w tym protokołem kontraktowym, strategiami opartymi na aukcjach i teorii gier. AppLeS (Application-Level Scheduling) to przykład metodyki planowania adaptacyjnego, również opartej na agentach. Aplikacje korzystające z AppLeS mają wspólną architekturę i są planowane adaptacyjnie przez dostosowanego agenta planowania. Agent wykonuje kilka dobrze ustalonych kroków w celu uzyskania harmonogramu dla aplikacji: wykrywanie zasobów, wybór zasobów, wybór harmonogramu, wykonanie aplikacji i dostosowanie harmonogramu.

Problemy z planowaniem w środowiskach zorientowanych na usługi

Planowanie zadań wewnątrz SOE, takich jak chmury, jest szczególnie trudnym problemem, ponieważ istnieje kilka kwestii, z którymi należy się uporać. Obejmują one: szacowanie czasu wykonywania zadań i kosztów transferu; odkrywanie i wybór usług; negocjacje między klientami a różnymi dostawcami chmury; i zaufanie między zaangażowanymi stronami. W dalszej części omówimy każdy z tych problemów osobno.

Szacowanie czasu wykonywania zadań i kosztów transferu

Wiele SA wymaga pewnego rodzaju szacunków użytkownika w celu zapewnienia ulepszonych rozwiązań do planowania. Szacunki są albo szacowane przez użytkownika, albo generowane przy użyciu metod obejmujących profilowanie kodu, statystyczne określanie czasów wykonania, regresję liniową lub szablonowanie zadań. W zastosowaniu do SOE metody te mają zarówno zalety, jak i wady, jak pokazano w następnych akapitach. W SOE nie ma zbyt dużego wglądu w zasoby działające za usługą, a zatem użytkownikom trudno jest uzyskać informacje, które mogą pomóc im w prawidłowym oszacowaniu czasu działania. Szacunki podane przez użytkownika zależą od wcześniejszego doświadczenia użytkownika w wykonywaniu podobnych zadań. Użytkownicy mają również tendencję do przeceniania czasu wykonywania zadań, wiedząc, że harmonogramy na nich polegają. W takim przypadku planista, w zależności od heurystyki planowania, może odłożyć inne zadania z powodu błędnych informacji. Aby poradzić sobie z tymi scenariuszami, planiści mogą wdrożyć systemy kar, w których zadania należące do tych szkodliwych użytkowników byłyby celowo opóźniane. Czasami

użytkownikom trudno jest nawet podać szacunkowe czasy działania. Sytuacje te zwykle występują ze względu na charakter usługi. Rozpatrując dwa przykłady usług, jeden przetwarzający obrazy satelitarne i drugi rozwiązujący symboliczne problemy matematyczne, można wyciągnąć następujące wnioski. W pierwszym przypadku dość łatwo jest określić szacunkowe czasy działania na podstawie historycznych czasów wykonania, ponieważ zależą one od rozmiaru obrazu i wymaganej operacji. Drugi przypadek jest bardziej skomplikowany, ponieważ problemy matematyczne są zwykle rozwiązywane przez usługi udostępniające system algebry komputerowej (CAS). CAS to specyficzne aplikacje, które koncentrują się na jednej lub kilku dziedzinach matematycznych i oferują kilka metod rozwiązywania tego samego problemu. Wybór metody zależy od wewnętrznych kryteriów, których użytkownik nie zna. Prosty przykład podano, gdy rozważamy faktoryzację dużych liczb całkowitych (ponad 60 cyfr). Operacje te mają silne implikacje w dziedzinie kryptografii. W tym przypadku faktoryzacja n nie zależy od podobnych wartości jak n-1 lub n+1. Ponadto czas faktoryzacji nie jest powiązany z czasami wymaganymi do faktoryzacji n-1 lub n. Dlatego użytkownikom trudno jest oszacować czasy pracy w takich sytuacjach. Jak najbardziej doprecyzowanie pojęcia podobieństwa między dwoma zadaniami może być odpowiedzią na ten problem, ale w niektórych przypadkach, takich jak ten przedstawiony wcześniej, może to wymagać wyszukania identycznych zgłoszeń z przeszłości. Profilowanie kodu działa dobrze w przypadku zadań intensywnie korzystających z procesora CPU, ale nie radzi sobie z aplikacjami intensywnie przetwarzającymi dane, w których trudno jest przewidzieć czas wykonania przed odebraniem wszystkich danych wejściowych. Statystyczne oszacowania czasów wykonywania napotykają podobne problemy, jak profilowanie kodu. Szablony zostały również wykorzystane do przypisania szacunków zadań poprzez umieszczanie nowo otrzymanych zadań w już istniejących kategoriach. Do tworzenia szablonu wykorzystywane są ogólne cechy zadania, takie jak właściciel, aplikacja do rozwiązywania problemów, komputer użyty do przesłania zadania, rozmiar danych wejściowych, użyte argumenty, czas przesłania lub czas rozpoczęcia. Algorytmy genetyczne można następnie wykorzystać do przeszukiwania globalnej przestrzeni pod kątem podobieństw. Pomimo trudności w szacowaniu czasów działania istnieją SA, które w ogóle ich nie wymagają. Algorytmy te uwzględniają tylko obciążenie zasobów i przenoszą zadania tylko wtedy, gdy ich obciążenia stają się niezrównoważone. To podejście działa dobrze, a testy wykazały, że heurystyki planowania, takie jak Round-Robin dają wyniki porównywalne z innymi klasycznymi heurystykami opartymi na szacunkach czasu działania. W SOE problem dostarczania szacunków czasu pracy można przezwyciężyć innym ważnym aspektem związanym z kosztami obsługi, jakim jest terminowość wykonania. W tym przypadku nie ma znaczenia, jak szybko, jak wolno lub gdzie zadanie zostanie wykonane, o ile zostanie zakończone w określonym przedziale czasu. W związku z tym, przesyłając zadania w chmurze, użytkownicy mogą dołączyć ograniczenia dotyczące terminów zamiast szacowania czasu wykonywania do przepływów pracy lub zadań wsadowych i mieć nadzieję, że nie zostaną one znacznie przekroczone. Heurystyka planowania oparta na terminach jest szczególnie przydatna w przypadkach, gdy użytkownicy wynajmują usługi na określoną ilość razy. Z czasem wykonywania zadań powiązane są koszty transferu związane z przeniesieniem zadania z zasobu do innego. W SOE jest to problem, ponieważ zwykle niewiele lub nic wiadomo o fizycznej lokalizacji i trasie sieciowej do określonej usługi. W przypadku przenoszenia dużych ilości danych, takich jak zdjęcia satelitarne, nawet do kilkuset megabajtów, problemem staje się koszt transferu. Oprócz czasu potrzebnego na realokację danych pojawiają się problemy z licencjami i kosztami finansowymi. Zdarzają się przypadki, gdy zastrzeżone dane, takie jak zdjęcia satelitarne należące do określonych organizacji, nie mogą być przeniesione poza ich domenę (chmura). W takim przypadku przeniesienie do chmury, która zapewnia szybsze i/lub tańsze usługi przetwarzania obrazu, nie jest możliwe z powodu problemów licencyjnych. Realokacja zadań to coś więcej niż tylko przenoszenie zależnych danych. Chmury w dużej mierze opierają się na wirtualizacji i dlatego czasami w celu wykonania zadań trzeba stworzyć maszyny wirtualne o określonych cechach. W rezultacie podczas realokacji zadania cała maszyna wirtualna może wymagać

relokacji. Wiąże się to z kilkoma innymi problemami, takimi jak zatrzymywanie i wznawianie zadań wywłaszczeniowych lub ponowne uruchamianie zadań nieposiadających po ich bezpiecznym przeniesieniu. Problem kosztów transferu jest więc bardziej problematyczny niż na pierwszy rzut oka.

Wykrywanie i wybór usług

Usługi (oparte na SOAP , RESTful , Grid Services są ważną częścią systemów chmurowych. Pozwalają na udostępnienie oprogramowania, pamięci masowej, infrastruktury lub całych platform za pośrednictwem jednolitego interfejsu, który może być używany przez klientów zewnętrznych. Każdy dostawca usług udostępnia swoje usług dla ogółu społeczeństwa, aby mógł on z nich korzystać bezpłatnie lub za opłatą w celu rozwiązania określonego problemu.W tym morzu usług istnieje również ciągła potrzeba odkrywania odpowiednich usług do rozwiązania określonego zadania.Opis uniwersalny Rejestry wykrywania i integracji (UDDI) oferują rozwiązanie tego problemu.Każdy dostawca usług rejestruje swoje usługi w UDDI, który z kolei jest używany przez konsumentów usług do wyszukiwania określonych usług. Po znalezieniu takiej usługi jej interfejs można wykorzystać do przesyłania zadań i pobierania ich wyników.Po pomyślnym znalezieniu liczby możliwych kandydatów usług pozostaje problem wyboru najlepszego do zadania. W tym kierunku ważną rolę odgrywa heurystyka planowania, która na podstawie kilku kryteriów wybierze usługę, która z największym prawdopodobieństwem zminimalizuje koszty realizacji. Należy zauważyć, że w zależności od tego, czy heurystyka harmonogramowania jest adaptacyjna, czy nie, zadanie może zostać kilkakrotnie ponownie przydzielone przed faktycznym wykonaniem

Negocjacje między dostawcami usług

Negocjacje odgrywają ważną rolę w harmonogramowaniu zadań, gdy w rozwiązanie danego problemu zaangażowane są usługi z wielu chmur. Zazwyczaj negocjacje są powiązane z fazą wyboru usługi i obejmują żądanie harmonogramu dla określonej cechy usługi. Biorąc to pod uwagę, mniejsze koszty realizacji mogą być osiągnięte. Negocjacje mogą również obejmować decyzję, jakie dane/zadania mogą zostać przekazane do usługi oraz czy usługodawca może dalej wykorzystywać przekazane dane/zadania do własnych celów, czy też nie. Ponieważ w większości przypadków szczegóły dotyczące maszyny wirtualnej lub aplikacji, które są ujawniane jako usługa, są ukryte przed opinią publiczną, negocjacja wymaga wprowadzenia encji negocjatorów, które obsługują dyskusje dotyczące preselekcji w nazwie usługi/chmury. Zwykle ten etap jest realizowany przez jednego lub więcej agentów. Szczegóły dotyczące zaangażowanych agentów zostaną podane w sekcji 7.5. W zależności od wyniku negocjacji przyznawany jest dostęp do żądanej usługi lub rozpoczynają się nowe negocjacje z innym agentem.

Pokonanie wewnętrznego harmonogramu zasobów

Ważnym problemem, który RMS muszą przezwyciężyć w SOE, jest wewnętrzny harmonogram używany przez dostawcę usług. Na ten program planujący nie ma wpływu ani nie jest pomijana interwencja z zewnątrz. W rezultacie mówi się, że planowanie między usługami jest realizowane przez meta-scheduler, który zajmuje się zadaniami na poziomie usługi, pozostawiając planowanie na poziomie zasobów wewnętrznym planistom wirtualnej organizacji (VO). Te wewnętrzne harmonogramy obsługują przydziały zadań w zależności od własnych zasad, a zatem nie ma gwarancji, że zadanie przesłane przez metaplanista zostanie wykonane w ramach ograniczeń kosztowych wynegocjowanych w momencie przesyłania. W wyniku negocjacji między metaplanistą a usługodawcą, ten ostatni mógłby próbować faworyzować zadanie, zwiększając jego priorytet. Działanie to leży w interesie usługodawcy, ponieważ może zostać ukarany, przy znacznie obniżonym zaufaniu do usług, za ciągłe przekraczanie narzuconych terminów. W konsekwencji przyszłe decyzje podejmowane przez metaplanera mogłyby zignorować usługę, a dostawca poniósłby straty kosztów. Uzyskujemy zatem symbiotyczną relację między metaplanistą a usługodawcą, która pozwala obu z nich na uzyskanie przewagi: zaufanie do

usługodawcy wzrośnie przy szybszym wykonywaniu zadań, a tym samym jego dochody wzrosną dzięki otrzymywaniu większej liczby zadań; a meta-scheduler wykona zadania szybciej, minimalizując koszty klienta, który je przesłał.

Zaufanie w środowiskach wielochmurowych

Podczas wykonywania zadań na zdalnych usługach wymagany jest pewien poziom zaufania między peerami. Wystawienie zaufania występuje z powodu wielu problemów, w tym podejścia blokowego usług oraz z powodu problemów z bezpieczeństwem. Usługom nie można ufać, ponieważ ich interfejsy działają jak czarne skrzynki, a zawartość można zmieniać bez powiadomienia. Zatem osoba żądająca usługi musi mieć pewność, że to, do czego uzyskuje dostęp, jest takie samo, jak to, co zostało ogłoszone przez usługę. Jeśli tak nie jest, maszyna wirtualna działająca za usługą nie byłaby w stanie rozwiązać danego zadania, powodując potencjalne straty kosztów ze względu na czas poświęcony na wybór usługi i przesłanie zadania. Kwestie bezpieczeństwa są również ważne i są ściśle związane z poprzednim problemem. Te problemy mogą dotyczyć zarówno zleceniodawcy usług, jak i usługodawcy. Ten pierwszy jest zwykle dotknięty, gdy przesyłane przez niego dane są wykorzystywane do innych celów niż te ustalone podczas negocjacji (np. klonowanie danych chronionych prawem autorskim). Na to ostatnie może mieć również wpływ wysyłanie do niego danych mających na celu szkodę dostawcy. Wyczerpujący wgląd w kwestie bezpieczeństwa w DS podano w wersji papierowej. Zaufanie zazwyczaj osiąga się za pomocą certyfikatów cyfrowych, takich jak certyfikaty X.509, które są powszechnie używane w przeglądarkach internetowych, bezpiecznych usługach poczty elektronicznej i systemach płatności elektronicznych. Korzystając z certyfikatów, klienci zazwyczaj proszą o taki certyfikat usługodawców w celu uzyskania dostępu. Usługi Web-SOAP i Grid-SOAP radzą sobie z kwestiami bezpieczeństwa przy użyciu standardu WSSecurity (WS Security, 2010). Umożliwia stronom dodawanie podpisów i nagłówków szyfrowania do wiadomości SOAP. Rozszerzenie WS-Security, WS-Trust (WS Trust, 2010), zajmuje się wydawaniem, odnawianiem i weryfikowaniem tokenów bezpieczeństwa lub relacji zaufania brokera między uczestnikami. Oprócz standardu WS-Security można również zastosować Transport Layer Security (TLS). Na przykład HTTPS może być używany do obsługi usług Web-SOAP, Grid-SOAP i RESTful.

Planowanie przepływu pracy

Przepływy pracy składają się z kilku zadań powiązanych ze sobą zależnościami danych/funkcjonalnych, które muszą być wykonane w określonej kolejności, aby osiągnąć cel problemu. Stosuje się je zwłaszcza w przypadkach, gdy problem można podzielić na mniejsze kroki, z których każdy jest wykonywany przez innego solwera, lub w naszym przypadku WS. W środowisku chmury użytkownicy zazwyczaj przesyłają swoje przepływy pracy do usługi, która organizuje wykonanie i zwraca wynik. Cokolwiek dzieje się poza interfejsem usługi, jest poza zasięgiem i niewidoczne dla klienta. Przepływy pracy można tworzyć za pomocą narzędzi graficznych lub bezpośrednio pisząc kod w obsługiwanym formacie, takim jak BPEL (WS-BPEL, 2010), YAWL itp. Po przesłaniu przepływu pracy za wykonanie zadań odpowiada silnik wykonawczy wysyłając je do odpowiednich WS. W naszym przypadku te WSs są zastępowane przez agentów planowania, którzy starają się planować zadania w najlepszej dostępnej usłudze, negocjując z innymi agentami. Po zakończeniu zadania jego wynik jest wysyłany z powrotem do silnika wykonawczego, który może przejść do następnego zadania i tak dalej. Ważnym problemem w tym łańcuchu komunikacyjnym jest powrót wyniku do silnika workflow. Aby rozwiązać ten problem, do każdego przesłanego zadania dołączany jest adres agenta odpowiedzialnego za VO, w którym znajduje się silnik. W ten sposób po zakończeniu realizacji wynik jest odsyłany bezpośrednio do agenta, który początkowo otrzymał zadanie. Można zauważyć, że nie podejmuje się żadnych wcześniejszych decyzji dotyczących harmonogramu, a zadania są planowane jeden po drugim, gdy stają się gotowe do zaplanowania. Jest to konieczne ze względu na dynamizm i nieprzewidywalność otoczenia. W artykule

omówiono ujednolicony model harmonogramowania zadań niezależnych i zależnych. Celem było umożliwienie SA dla niezależnych zadań, które można zastosować do przepływów pracy, gdy brane są pod uwagę dynamiczne środowiska i planowanie online. Chociaż to podejście jest odpowiednie, gdy potrzebne są globalne decyzje dotyczące planowania, zdarzają się przypadki, w których silnik przepływu pracy nie może łatwo uzyskać mapowania zadań do zasobów (Active BPEL, 2010) w czasie wykonywania. Zamiast tego można zastosować SA przepływu pracy, takie jak HEFT , Hybrid lub CPA. Jednak uwzględniają zadania w bieżącym przepływie pracy tylko wtedy, gdy potrzebne są decyzje dotyczące planowania lub zmiany harmonogramu. Algorytmy te zapewniają strategie planowania zadań przepływu pracy na heterogenicznych zasobach w oparciu o analizę całego grafu zadania. Za każdym razem, gdy przepływ pracy zostanie przesłany, zadania zostaną najpierw przypisane do zasobów, a dopiero potem rozpocznie się wykonywanie przepływu pracy. Negocjowanie zasobów odbywa się zatem przed środowiskiem wykonawczym. To statyczne podejście nie jest jednak odpowiednie dla środowisk o wysokiej dynamice (na przykład chmur), w których: nie można przewidzieć dostępności zasobów; zastrzeżenia są trudne do osiągnięcia; należy uzyskać globalną perspektywę; a ograniczenia terminów wymagają ciągłych negocjacji dotyczących zmiany harmonogramu. Poniżej przedstawiamy rozwiązanie oparte na agentach do planowania przepływów pracy. Tak zwani agenci planowania służą do negocjowania, planowania zadań i wysyłania odpowiedzi z powrotem do silnika przepływu pracy. Jego celem jest zapewnienie platformy do planowania przepływu pracy online, w której zadania są planowane tylko wtedy, gdy są gotowe do wykonania. Oznacza to, że zadanie, którego poprzednicy nie zakończyli realizacji, nie jest uważane za przekazane do realizacji.

Rozproszona platforma planowania oparta na agentach w chmurach

Ponieważ chmury są nieprzewidywalne pod względem obciążenia zasobów i sieci, systemy muszą być w stanie dostosować się do nowych konfiguracji wykonywania, aby koszty ogólne nie zostały znacznie przekroczone. Systemy wieloagentowe (MAS) stanowią odpowiedź na ten problem, ponieważ opierają się na (pół)zdecentralizowanych środowiskach składających się z kilku wyspecjalizowanych agentów współpracujących ze sobą w celu osiągnięcia celu poprzez negocjacje. Podczas negocjacji każdy agent zachowuje egocentryczny punkt widzenia, starając się zminimalizować jego koszty. Chociaż jest to dobra opcja w przypadku bardzo dynamicznego DS, podejścia rozproszone wiążą się z dużym obciążeniem transferowym, ponieważ wymagają ciągłej aktualizacji od swoich partnerów w celu zachowania aktualnego globalnego widoku. W przeciwieństwie do tego, scentralizowane podejścia nie wymagają dużej komunikacji, ale ich szczyt wydajności jest maksymalizowany głównie w przypadku DS, które utrzymują względnie stabilną konfigurację. Zdecentralizowane rozwiązania oparte na agentach do planowania zadań pojawiają się również jako odpowiednie rozwiązania, gdy rozważamy federację wielu VO, z których każdy ma własne zasoby i wdraża własne zasady planowania. Wysyłanie zadań w takim środowisku wymaga współpracy między VO w celu wykonania ich z ograniczeniami, w tym terminami wykonania, obciążeniami, zależnościami IO itp. Agent obliczeniowy może być zdefiniowany przez elastyczność, zwinność i autonomię i jak przedstawiono w może działać jako mózg dla planowanie zadań wewnątrz infrastruktury multi-cloud. Agenci pozwalają zasobom działać jako autonomiczne podmioty, które samodzielnie podejmują decyzje w oparciu o wewnętrzną logikę. Ponadto inteligentny agent może być postrzegany jako rozszerzenie wcześniej podanej definicji poprzez dodanie trzech dodatkowych cech: reaktywności (agenci reagują na otoczenie), proaktywności (agenci podejmują inicjatywy kierowane celami) i zdolności społecznych (interakcje z innymi agentami) . Aby podejmować decyzje harmonogramowe, agenci muszą spełnić wszystkie poprzednie wymagania. Muszą szybko dostosować się do zmian w chmurze i komunikować się z innymi, aby znaleźć odpowiednią usługę do zadań, które wymagają szybszej realizacji. W kontekście planowania zadań

adaptacyjność agenta obejmuje obsługę zmian obciążenia lub dostępności zasobów. Poniżej przedstawiamy platformę planowania zorientowaną na agentów SOE.

Platforma planowania

Platforma planowania rozproszonych agentów składa się z kilku agentów współpracujących ze sobą w celu planowania zadań. Wewnątrz chmury składającej się z kilku dostawców usług (VO) agenci pełnią rolę negocjowania i osiągania porozumienia między rówieśnikami. Na podstawie heurystyki meta-schedulingu, wewnętrznego harmonogramu, wiedzy na temat usług, którymi zarządza oraz charakterystyki zadań, każdy agent będzie próbował negocjować relokację kilku zadań z/do niego. Próbując osiągnąć ten cel, agenci będą również próbowali zminimalizować globalny koszt związany z każdym przepływem pracy. Podejścia oparte na agentach mogą pozwolić każdemu dostawcy chmury na utrzymanie własnych wewnętrznych zasad planowania. Ponadto mogą również korzystać z własnych zasad planowania na poziomie metaplanowania. Decydując się na relokację zadań, każdy agent będzie kierował się własnymi regułami planowania i będzie próbował osiągnąć porozumienie w drodze negocjacji z resztą. Te aspekty pozwalają VO zachować autonomię i kontynuować funkcjonowanie jako niezależna jednostka w chmurze. Autonomia jest obowiązkowym wymogiem, ponieważ VO zwykle reprezentują firmy, które chcą zachować niezależność podczas świadczenia usług dla ogółu społeczeństwa. Każdy VO chcący ujawnić usługi umieszcza jednego lub więcej agentów w internetowym katalogu Yellow Pages, który może być przeszukiwany przez innych agentów chcących negocjować lepsze zasoby. Agenci mogą być projektowani jako jednostki modułowe. W ten sposób możemy dodawać nowe funkcjonalności do agentów bez konieczności tworzenia nowych typów agentów. Różni się to od poprzednich prac, które dotyczyły głównie hierarchii agentów. W ten sposób tworzymy superagenta, który stara się, aby zadania w swojej domenie otrzymywały najlepsze zasoby. Ponadto wyeliminowana jest potrzeba współpracy wielu agentów w celu obsługi tego samego zadania. Przykładami takich agentów są: agent wykonawczy, agent planowania, agent transferowy, agent interfejsu itp. W naszej wizji wszyscy wymienieni wcześniej wyspecjalizowani agenci stają się podmodułami wewnątrz każdego agenta. W ten sposób każdy agent będzie miał: moduł planowania, moduł komunikacyjny, moduł wykrywania usług i moduł wykonawczy. Suma wszystkich agentów tworzy meta-scheduler, który odpowiada za alokację zadań między usługami. Poniżej dzielimy agentów na dwie kategorie w zależności od tego, czy inicjują żądanie, tj. agenci żądający, czy odpowiadają na zapytanie, tj. agenci rozwiązujący. Podział ten nie wpływa na charakterystykę agenta i ma na celu jedynie zobrazowanie jego roli. Moduł komunikacyjny obsługuje dowolny rodzaj wymiany komunikatów z innymi agentami. Ułatwia również dialog między modułami, na przykład między modułem planowania a modułem wykrywania usług lub między modułem planowania a modułem executora. Moduł wykrywania usług umożliwia każdemu agentowi wykrywanie usług opublikowanych w UDDI znajdujących się w jego własnej domenie. Zazwyczaj każdy zasób lub dostawca wewnątrz VO, który chce zaoferować jakąś funkcjonalność ogółowi społeczeństwa, publikuje je jako usługi wewnątrz UDDI. Po opublikowaniu usługa może być używana przez agenta planowania podczas ponownego przydzielania zadań. Ten moduł nie służy do wykrywania usług poza domeną agenta. Powód takiego zachowania jest prosty: każda usługa poza jej domeną nie jest kontrolowana przez agenta, a zatem nie jest zaufana. Zaufanie do usług osiąga się poprzez negocjacje z innymi agentami. Moduł wykonawczy odpowiada za wywołanie usługi wybranej do realizacji zadania. Wywołanie usługi zazwyczaj osiąga się poprzez stworzenie klienta dopasowanego do interfejsu usługi. Tworzenie musi odbywać się dynamicznie w czasie wykonywania, ponieważ nie jest możliwe utworzenie listy prekompilowanych klientów na dysku ze względu na liczbę i różnorodność istniejących usług. Paper przedstawia interfejs API umożliwiający dostęp zarówno do usług opartych na SOAP, jak i usług Grid Services poprzez dynamiczne tworzenie klientów w oparciu o usługę WSDL (Web Service Description Language) (WSDL, 2010). Należy zauważyć, że moduł wykonawczy nie odpowiada za tworzenie jakiejkolwiek maszyny

wirtualnej wymaganej przez zadania. Zainicjowanie wszelkich wymaganych maszyn wirtualnych na podstawie opisu zadania zależy od zasobów usługi. Moduł planowania zajmuje się alokacją zadań do usług lub zadań do agentów. Ten moduł jest sercem platformy planowania opartego na agentach i opiera się na heurystyce planowania przy podejmowaniu decyzji. Każdy agent ma przypisane jedno lub więcej zadań. W zależności od heurystyki planowania może zdecydować się na wykonanie niektórych zadań na usługach zarządzanych przez agentów spoza swojej domeny. W ten sam sposób może decydować o przyjmowaniu nowych zadań od innych agentów. W zależności od zasad wdrożonych przez VO istnieją dwa możliwe scenariusze, z którymi może się zmierzyć moduł planowania. Pierwszy występuje, gdy agent nie ma informacji o zasobach, na których działają aplikacje, a widzi tylko interfejsy usług. Drugi to przypadek, gdy agent wie wszystko, co trzeba wiedzieć o podstawowych zasobach, tj. przepływie pracy, charakterystykach, topologii sieci itp. Oba te scenariusze są ważne w zależności od tego, jak agent zachowuje się, gdy zadanie musi zostać wykonane na jednym z jego usługi. Agent żądający przesyła zadanie bezpośrednio do usługi lub do agenta rozwiązującego. W dalszej części artykułu zajmujemy się tym drugim przypadkiem. Pierwsza opcja polega na pominięciu harmonogramu VO reprezentowanego przez agenta. Dzieje się tak, ponieważ zadanie będzie obsługiwane bezpośrednio przez wewnętrzny harmonogram zasobów. W konsekwencji dalsza optymalizacja planowania na poziomie metaplanowania byłaby utrudniona. Moduł planowania wewnątrz agenta implementuje heurystykę planowania zaprojektowaną do obsługi SOE. Heurystyka planowania może być postrzegana jako strategia wykorzystywana przez agenta do wyboru zasobu dla swoich zadań, podczas gdy interakcja z innymi agentami reprezentuje fazę negocjacji. Negocjacje przebiegają w oparciu o zasady zawarte w strategii. Aby negocjacje zakończyły się sukcesem, stosuje się wspólny język negocjacyjny i zestaw predefiniowanych zasad uczestnictwa. Każdy agent ma kilka usług, którymi zarządza. Dołączone są do nich kolejki zadań. W zależności od heurystyki harmonogramowania tylko określona liczba zadań może zostać wysłana (za pomocą modułu wykonawczego) do usługi w danym momencie. Po przesłaniu do usługi zadaniem wewnętrznego planisty jest przypisanie. Moduł planowania wewnątrz agenta implementuje heurystykę planowania zaprojektowaną do obsługi SOE. Heurystyka planowania może być postrzegana jako strategia wykorzystywana przez agenta do wyboru zasobu dla swoich zadań, podczas gdy interakcja z innymi agentami reprezentuje fazę negocjacji. Negocjacje przebiegają w oparciu o zasady zawarte w strategii. Aby negocjacje zakończyły się sukcesem, stosuje się wspólny język negocjacyjny i zestaw predefiniowanych zasad uczestnictwa. Każdy agent ma kilka usług, którymi zarządza. Dołączone są do nich kolejki zadań. W zależności od heurystyki harmonogramowania tylko określona liczba zadań może zostać wysłana (za pomocą modułu wykonawczego) do usługi w danym momencie. Po przesłaniu do usługi zadaniem wewnętrznego harmonogramu jest przypisanie .

Planowanie poprzez negocjacje

Centralną jednostką każdej platformy planowania opartej na agentach jest mechanizm planowania. Na podstawie jego zasad agenci podejmują aktywne/pasywne decyzje o przeniesieniu lub przyjęciu nowych zadań. Każda decyzja jest poprzedzona fazą negocjacji, w której agent żąda/otrzymuje informacje od innych agentów i na podstawie heurystyk harmonogramowania podejmuje decyzję, która oferuje akceptację. Negocjacje wymagają zarówno języka, jak i zestawu zasad uczestnictwa. W zależności od polityki VO i przestrzegania jej przez inne VO można zastosować wiele rodzajów negocjacji. Przykłady obejmują modele teorii gier, podejścia heurystyczne i rozwiązania oparte na argumentach. Dla języka komunikacji używanego przez naszą platformę opracowano minimalny zestaw lokucji:

• requestOffer(i,j,k): agent i wymaga oferty od agenta j na zadanie k. Zadanie k zawiera wszystkie informacje potrzebne do podjęcia decyzji o harmonogramie. Może to obejmować (jeśli są dostępne): szacowane czasy realizacji, szacunkowe koszty przelewu, terminy realizacji, wymagane nakłady itp.;

• sendOffer(j,i,k,p): agent j wysyła do agenta i ofertę ceny p na wykonanie zadania k. Cena p reprezentuje koszt wykonania zadania k na zasobie j. Mierzenie kosztów zależy od heurystyki planowania. Na przykład może reprezentować szacowany czas wymagany do wykonania zadania w usłudze należącej do agenta j;

• acceptOffer(i,j,k): agent i przyjmuje ofertę agenta j na wykonanie zadania k;

• sendTask(i,j,k): agent i wysyła do wykonania zadanie k do usługi świadczonej przez agenta j;

• odrzucaOffer(i,j,k): agent i odrzuca ofertę agenta j na wykonanie zadania k;

• requestTasks(i,j): agent i informuje agenta j, że chce wykonać więcej zadań;

• requiredDetails(i,j): agent i informuje agenta j, że potrzebuje więcej informacji na temat usług/zasobów, którymi zarządza. Dokładniej odnoszą się do szczegółów (na przykład URL WSDL) usługi proponowanej przez agenta j;

• sendDetails(j,i,d): agent j wysyła dostępne szczegóły do agenta i. Dane te zawierają wyłącznie publicznie dostępne dane wynikające z wewnętrznych polityk;

• informTaskStatus(i,j,k,m): agent i informuje komunikatem m agent j o statusie zadania k. Na przykład wiadomość może zawierać wynik wykonania zadania.

Zasady uczestnictwa są wymagane, aby zabronić agentom mówienia czegoś, czego nie wolno im powiedzieć w danym momencie. Negocjacja zaczyna się albo od prośby o więcej zadań od agenta j, albo od zapytania ofertowego na dane zadanie, które agent zdecydowałem się przenieść. Istnieje stałe połączenie między agentem aparatu przepływu pracy a agentem planowania odpowiedzialnym za VO, w którym działa aparat. To właśnie temu agentowi stawia się zadania na pierwszym miejscu. Po wysłaniu nowego zadania do tego agenta, jego obowiązkiem jest znalezienie i wynegocjowanie wykonania zasobu, który ma największą szansę na zminimalizowanie ograniczenia terminu. Agenty aparatów przepływu pracy są podobne do agentów planowania i mogą się z nimi komunikować. Nie mogą jednak planować zadań na zasobach. Ich jedynym celem jest zapewnienie interfejsu między silnikiem a platformą do metaplanowania. Podczas planowania przepływów pracy pojawia się ważny problem, który należy uwzględnić w fazie negocjacji. Biorąc pod uwagę wykonanie zadania w usłudze, która zapewnia wynik, który może być dalej wykorzystany tylko w usługach należących do tego samego VO, wszelkie inne możliwe rozwiązania poza tym VO byłyby ignorowane. Dlatego zadaniem agenta żądającego jest negocjowanie rozwiązania, które zmaksymalizuje zbiór wyszukiwania. Z tego powodu należy osiągnąć równowagę między najlepszym kosztem czasu w danym momencie a przyszłymi ograniczeniami. Na przykład wybór najszybszej usługi do wykonania zadania może zostać przekształcony w wybór usługi, która wykonuje zadanie szybciej i bez ograniczeń w korzystaniu z wyniku. W przypadku, gdy agent j zażądał więcej zadań od innego agenta i, ten drugi poprosi tego pierwszego o oferty dotyczące kosztów wykonania niektórych jego zadań. W tym momencie agent j odeśle do agenta i ofertę na wykonanie zadania. Na podstawie tej oferty poproszę o więcej szczegółów dotyczących dostępnych usług, które pozwolą mu podjąć właściwą decyzję: odrzuci lub zaakceptuje ofertę. W przypadku, gdy agent i zaakceptuje ofertę agenta j, zadanie zostanie przekazane do kolejki obsługi podlegającej temu agentowi. W zamian zakończy się komunikatem o statusie zadania. Po zakończeniu zadania wynik zostanie odesłany z powrotem do agenta przepływu pracy, który przekaże go do silnika. Silnik użyje wyniku do wybrania kolejnych zadań do zaplanowania i wykonania. W ramach

przedstawionego protokołu negocjacyjnego kluczowy element odgrywa moment złożenia wniosku o ofertę relokacji lub o nowe zadania. Ten moment w zasadzie oznacza początek negocjacji. W naszym artykule poruszono problem prawidłowego doboru momentu zapytania ofertowego. Proponowana heurystyka planowania uwzględnia ten moment realokacji i pokazano, że ma on bezpośredni wpływ na wynik harmonogramu. Aby rozszerzyć to podejście na SOE, w artykule przeanalizowano podejście oparte na terminach. Badanie opiera się na fakcie, że w SOE użytkownicy zazwyczaj chcą zminimalizować swoje koszty w odniesieniu do czasu użytkowania, a tym samym zapewnić termin wykonania każdego zadania w swoich przepływach pracy. Celem harmonogramu jest zminimalizowanie globalnego opóźnienia zadania, czyli różnicy między rzeczywistym czasem zakończenia zadania a podanym przez użytkownika czasem ostatecznym. Heurystyka planowania nosi nazwę DMECT (Dynamic Minimalization of Estimated Completion Time). Okresowo oblicza dla każdego zadania czas do ostatecznego terminu (TUD), lokalny czas oczekiwania (LWT) – czas od przypisania go do bieżącej kolejki usług – oraz całkowity czas oczekiwania (TWT) – czas od przesłania zadania . Na podstawie tych wartości podejmuje się decyzję, czy przenieść zadanie, czy nie, sprawdzając, czy TUD/TWT - LWT jest mniejsze niż 0, czy nie. Jeśli wartość jest mniejsza, podejmowana jest akcja requestOffer. Należy zauważyć, że przy podejmowaniu decyzji o relokacji zadania brane są pod uwagę wszystkie dostępne usługi. Są to zarówno wewnętrzne (część bieżącej domeny agenta), jak i zewnętrzne (uzyskane z zapytania requestOffer). W ten sposób każda istniejąca usługa ma szansę konkurować o zadania. Nietrudno zauważyć, że relacja relokacji będzie próbowała przenosić zadania szybciej w miarę zbliżania się ich terminu. W odpowiedzi na zapytanie requestOffer, każdy agent wykona akcję sendOffer, która poinformuje agenta żądającego o możliwych wyborach. Każda odpowiedź zazwyczaj zawiera koszt wykonania zadania w najlepszej usłudze. Jeżeli wstępne zapytanie zawierało również dolną granicę tego kosztu, zwracana jest lista usług oferujących lepsze ceny. Koszt planowania składa się z możliwych czasów realizacji w połączeniu z kosztami pieniężnymi. Na przykład na żądanie każdy agent obliczy szacowany czas wykonania każdej usługi i zwróci tylko te, które mają wartości mniejsze niż początkowo podany limit. Alternatywnie mógł zwrócić tylko najmniejszą wartość, zapewniając, że została złożona najlepsza dostępna oferta. W przypadku, gdy niemożliwe jest oszacowanie czasu wykonania z powodu niewystarczających danych lub polityk wewnętrznych, jako miarę można zastosować długość kolejki usług. W pokazaliśmy, że im mniejsza kolejka, tym prawdopodobieństwo, że szybciej wykonuje zadania. Po zebraniu wszystkich kosztów agent żądający wybierze najlepszy zgodnie z heurystyką harmonogramowania, tj. Najkrótszy czas wykonania w naszym przykładzie. Wszystkie inne oferty zostaną odrzucone. Po wybraniu zadanie zostanie wysłane do wybranego agenta rozwiązywania problemów, który umieści zadanie w kolejce usług, a wartość LWT dla przeniesionego zadania zostanie ustawiona na 0. W scenariuszu, w którym zadanie nie zostanie wykonane na nowo wybranej usłudze również, tj. TUD/TWT – LWT < 0, agent solvera wyśle zapytanie requestOffer do innych agentów, stając się w ten sposób nowym agentem żądającym dla tego zadania. Decyzja o tym, kiedy poprosić o nowe zadania, to kolejny ważny przypadek, który uruchamia proces negocjacji. W takim przypadku agent wysyła wiadomość requestTasks do wszystkich pozostałych agentów, informując ich o chęci przyjęcia zadań. Po wysłaniu tej wiadomości agenci zaczną wysyłać do niej requestOffers dla zadań, które chcą ponownie przydzielić. Od tego momentu negocjacje przebiegają podobnie jak w omawianym wcześniej przypadku. W zależności od polityki żądania nowych zadań mogą być wykonywane okresowo lub wtedy, gdy obciążenie usług pod nadzorem agenta spadnie poniżej określonego limitu. W zależności od polityki planowania, takie podejście polegające na aktywnym wyszukiwaniu nowych zadań może być nieefektywne. Na przykład w naszym scenariuszu wykorzystującym heurystykę DMECT takie żądanie nie miałoby żadnego efektu, dopóki co najmniej jedno zadanie nie przekroczy swojego limitu pozostawania w kolejce zasobów. Inne heurystyki planowania oparte na prostych technikach równoważenia obciążenia, takie jak ta przedstawiona w Frincu i in., (2009) mogą być bardziej odpowiednie dla tego scenariusza. W takich

przypadkach nie ma warunków uniemożliwiających migrację zadań między agentami. Gdy agent zdecyduje, że obciążenie jego usług spadło wystarczająco, można zażądać nowych zadań.

Szczegóły implementacji prototypu

W tej sekcji przedstawiamy niektóre aspekty implementacyjne prototypu platformy do harmonogramowania. Platforma opiera się na JADE jako platformie agenta oraz na silniku OSyRIS (OSYRIS WorkFlow Engine, 2010) do realizacji przepływu pracy. JADE ułatwia tworzenie aplikacji rozproszonych zgodnie z paradygmatem zorientowanym na agentów i jest w rzeczywistości wieloagentowym oprogramowaniem pośredniczącym zgodnym z FIPA (Foundation for Intelligent Physical Agents). Jest zaimplementowany w języku Java i zawiera wtyczkę Eclipse, która ułatwia proces tworzenia oprogramowania poprzez integrację narzędzi graficznych do programowania, wdrażania i debugowania. Ponadto JADE może być rozproszone na kilka zasobów, a jego konfiguracją można sterować za pomocą zdalnego graficznego interfejsu użytkownika. Agenci mogą w dowolnym momencie swobodnie migrować między tymi zasobami. JADE zapewnia również: standardową architekturę do planowania działań agenta; standardowy protokół komunikacyjny z wykorzystaniem języka komunikacji agenta (ACL); i umożliwia integrację wyższej funkcjonalności poprzez umożliwienie użytkownikom dołączania własnych modułów Prolog w celu uzasadnienia aktywności. Mimo że prosty model agentów JADE ułatwia programowanie, wymaga to znacznego nakładu pracy na uwzględnienie inteligencji, gdy wymagana jest złożona kontrola. Paper przedstawia rozszerzenie do JADE, w którym platforma jest rozszerzona o dwa typy agentów w celu utorowania drogi dla bardziej elastycznego systemu chmury agentów. Dwa typy agentów to: agent BeanShell odpowiedzialny za wysyłanie i wykonywanie zachowań pochodzących od innych agentów; oraz agent Drools odpowiedzialny za odbieranie i wykonywanie reguł pochodzących od innych agentów. Dla obu typów agentów oferowane są mechanizmy uwierzytelniania i autoryzacji. OSyRIS to silnik realizacji przepływu pracy inspirowany naturą, w którym reguły są wyrażane zgodnie z paradygmatem akcji warunku zdarzenia: zadania są wykonywane tylko wtedy, gdy wystąpią pewne zdarzenia i spełnione zostaną dodatkowe warunki opcjonalne. W OSyRIS zdarzenia reprezentują wykonanie zadań, a warunki są zwykle umieszczane na wartościach wyjściowych. Pojedyncza instrukcja jest używana cała reszta (split, łączenie, równoległość, sekwencja, wybór, pętla) wywodząca się z niej: LHS -< RHS | warunek, istotność, gdzie LHS (lewa strona) reprezentuje zadania, które należy wykonać przed wykonaniem zadań RHS (prawa strona). Silnik opiera się na metaforze chemicznej, w której zadania pełnią rolę cząsteczek, a regułami wykonania są reakcje. Do symulacji VO wykorzystaliśmy dwa klastry dostępne na uniwersytecie. Jeden składający się z 8 dwurdzeniowych węzłów Pentium z 4 GB pamięci RAM każdy (zwany VO1) a drugi ma 42 węzły z 8 rdzeniami i 8 GB pamięci RAM na każdy. Ten ostatni klaster jest podzielony na 3 ostrza (nazywane VO2, VO2 i VO3), każdy z 14 węzłami. Do każdego bloku dołączony jest jeden agent planowania, który zarządza uruchomionymi na nich usługami. Pojedynczy agent służy do zarządzania całym VO1. Węzły są sparowane, a każda para jest udostępniana przez usługę obsługiwaną przez agenta obsługującego zarządzającego VO. Agenci są rejestrowani w repozytorium żółtej strony. Do planowania zadań międzyagentowych używana jest heurystyka DMECT. Chociaż różne SA mogą być używane do planowania zasobów lokalnych, zdecydowaliśmy się na jeden: heurystykę MinQL. Scenariusz planowania przebiega następująco: gdy agent planowania otrzyma zadanie, dołącza je do jednej ze swoich kolejek usług. Zadania są odbierane w drodze negocjacji z innymi agentami lub bezpośrednio od agenta przepływu pracy. Protokół negocjacji wykorzystuje warunek relokacji DMECT SA. Każda usługa może wykonywać co najwyżej k instancji jednocześnie. Zmienna k jest równa liczbie procesorów w parze węzłów. Po wysłaniu do usługi zadanie nie może zostać odesłane do agenta, chyba że zostało to wyraźnie określone w heurystyce planowania. Zadania wysyłane do usług są planowane wewnątrz zasobu za pomocą MinQL SA, który wykorzystuje prostą technikę równoważenia obciążenia. Agenci planowania okresowo wysyłają do usługi zapytania dotyczące ukończonych zadań. Po znalezieniu

informacji w środku jest ona używana do zwrócenia wyniku agentowi odpowiedzialnemu za instancję przepływu pracy. To przekazuje informacje do silnika, który z kolei przekazuje kolejny zestaw zadań agentowi w celu zaplanowania. Aby zasymulować heterogeniczność chmury pod względem możliwości, usługi oferują różne funkcjonalności. W naszym przypadku usługi oferują dostęp zarówno do CAS, jak i metod przetwarzania obrazu. Ponieważ każdy CAS oferuje różne funkcje do rozwiązywania problemów matematycznych, tak samo usługa go ujawniająca. To samo dotyczy usług przetwarzania obrazu, które nie implementują wszystkich dostępnych metod w każdej usłudze.

Wnioski

W niniejszym opracowaniu przedstawiliśmy niektóre zagadnienia dotyczące harmonogramowania zadań, gdy oferowane są usługi różnych dostawców. Problemy takie jak szacowanie czasu pracy i kosztów transferu; wykrywanie i wybór usług; zaufanie i negocjacje między dostawcami w zakresie dostępu do ich usług; lub sprawienie, aby niezależny harmonogram zasobów współpracował z meta-schedulerem. Jak opisano, wiele istniejących platform planowania jest zorientowanych na siatkę, a programy do planowania w chmurze dopiero zaczynają się pojawiać. W konsekwencji wprowadzono podejście MAS do problemu planowania w chmurze. MAS zostały wybrane, ponieważ zapewniają większą elastyczność i są dystrybuowane z natury. Mogą również stanowić dobry wybór w scenariuszach planowania, w których wymagane są negocjacje między dostawcami. Negocjacje są szczególnie ważne w przypadku przepływów pracy, w których zadania muszą być wspólnie aranżowane i wykonywane w ściśle określonych terminach, aby zminimalizować koszty użytkownika. Wynika to z faktu, że dostawcy mają różne zasady dostępu i harmonogramowania, a zatem wybór najlepszej usługi do wykonania zadania z dostarczonymi danymi wejściowymi staje się czymś więcej niż prostym problemem związanym z realokacją. Prototypowy system wykorzystuje jeden rodzaj agentów, które łączą wiele funkcjonalności. Powstały meta-scheduler utrzymuje autonomię każdego VO w chmurze. Prezentowane rozwiązanie jest w trakcie rozwoju i planowane są przyszłe testy z wykorzystaniem różnych SA i konfiguracji platform.