Usługi naukowe w chmurze

Wstęp

Scientific Computing był jedną z pierwszych aplikacji do obliczeń równoległych i rozproszonych. Do tej pory aplikacje naukowe pozostają jednymi z najbardziej intensywnych obliczeniowo i zainspirowały tworzenie infrastruktury obliczeniowej petaflopów, takiej jak Oak Ridge Jaguar i Los Alamos RoadRunner. Duża dedykowana infrastruktura sprzętowa stała się zarówno błogosławieństwem, jak i przekleństwem dla społeczności naukowej. Naukowcy są zainteresowani przetwarzaniem w chmurze z tego samego powodu, co firmy i inni profesjonaliści. Sprzęt jest dostarczany, utrzymywany i administrowany przez stronę trzecią. Abstrakcja oprogramowania i wirtualizacja zapewniają niezawodność i odporność na błędy. Stopniowane opłaty pozwalają na wieloskalowe prototypowanie i wykonanie. Zasoby chmury obliczeniowej to tylko kilka kliknięć i zdecydowanie najłatwiejsza, rozproszona platforma o wysokiej wydajności, do której można uzyskać dostęp. Nadal może istnieć dedykowana infrastruktura do nauki na ultraskalową skalę, ale chmura może z łatwością odegrać główną rolę w inicjatywie obliczeń naukowych. Naukowe przetwarzanie w chmurze to skomplikowany walc abstrakcyjnych modeli obliczeniowych, algorytmów naukowych i usług zwirtualizowanych. Z jednej strony algorytmy danych naukowych o dużej mocy obliczeniowej są implementowane na platformach programowania w chmurze, takich jak MapReduce i Dryad, podczas gdy z drugiej strony wykrywanie i wykonywanie usług implementuje szerszy obraz dzięki zależnościom produktów danych, łańcuchom usług i wirtualizacji. Chmura usług naukowych jest bardzo zwarta. Trudno jest dokonać znaczących odkryć naukowych na podstawie tylko jednego produktu danych. Jednak nawet pojedyncze produkty danych są wytwarzane z innych produktów, które z kolei wymagają jeszcze większej liczby różnych produktów do kalibracji. Łączenie usług w łańcuch jest niezbędne w chmurze naukowej, tak samo jak w przypadku chmury biznesowej. Jednak charakterystyczną cechą naukowego przetwarzania w chmurze jest przetwarzanie danych i eksperymentowanie; komputerowy słoń ukrywający się pod architekturą zorientowaną na usługi. Zarówno cykl życia usługi, jak i platformy przetwarzania są kluczowymi składnikami udanych obliczeń naukowych w chmurze. Omawiamy oba fronty z perspektywy atmosferycznego systemu przetwarzania w chmurze Service Oriented Atmospheric Radiances (SOAR). Upewniamy się również, że dotykamy wielu powiązanych technologii chmurowych, nawet jeśli niekoniecznie były one najlepiej dopasowane do naszego systemu SOAR.

Zarys

Chmura naukowa ma dwa końce, tylny i przedni. W następnej sekcji krótko opisujemy system promieniowania atmosferycznego zorientowanego na usługi (SOAR). Trzecia sekcja dotycząca paradygmatów programowania naukowego (zaplecze) opisuje, w jaki sposób platformy programistyczne wpływają na algorytmy naukowe. W czwartej sekcji omówiono Scientific Computing Services, które tworzą interfejs, opisując szczegółowo, w jaki sposób wirtualizacja usług wpływa na repozytoria naukowe. Odkryliśmy, że MapReduce i Dryad są bardzo skutecznymi platformami dla czegoś więcej niż nasze własne algorytmy. Podsumowujemy pracę własną i innych nad zastosowaniem tych paradygmatów do problemów związanych z nauką. Opisujemy również pięciofazowy cykl życia usług, ale z perspektywy zastosowań naukowych, i zajmujemy się niektórymi wyjątkowymi wyzwaniami, które odróżniają naukę od innych dziedzin usług.

Radiance atmosferyczne zorientowane na usługi (SOAR)

SOAR, wspólny projekt NASA, NOAA i UMBC, to skalowalny zestaw narzędzi usług internetowych, który zapewnia na żądanie złożone usługi siatkowe dla zestawów danych dotyczących promieniowania atmosferycznego z wielu czujników sondujących temperaturę i wilgotność. SOAR akceptuje dane wejściowe za pośrednictwem graficznego interfejsu użytkownika online (GUI) lub bezpośrednio z

innego programu. Serwer kolejkuje te żądania dla różnych złożonych usług danych naukowych w bazie danych, śledząc różne żądane przepływy pracy. Wykorzystuje duże zbiory danych zebrane przez NASA, NOAA i DOD. Te zestawy danych zawierają odczyty satelitarne temperatury i wilgotności z ostatnich trzech dekad. SOAR wykorzystuje chmurę Bluegrit z Uniwersytetu Maryland w hrabstwie Baltimore (UMBC) do stosowania transformacji danych, takich jak tworzenie siatki, próbkowanie, podzbiór i konwolucja, w celu generowania zestawów danych pochodnych z różnych radiancji atmosfery (Halem i in., 2009). Satelitarne instrumenty teledetekcji krążą wokół Ziemi synchronicznie ze słońcem, aby obserwować temperaturę, wilgotność oraz inne struktury i właściwości atmosfery. SOAR ułatwia eksperymenty zorientowane na klimat, zapewniając obliczenia i transformacje geoprzestrzenne. Stawia to SOAR w wyjątkowej pozycji, ponieważ musi łączyć się ze zdalnymi serwerami w celu pozyskiwania danych, ale jako pomocnik byłby dobrze umieszczony w jeszcze głębszym łańcuchu skomplikowanych eksperymentów naukowych. Rysunek 16.1 przedstawia schemat systemu wdrażania SOAR. Wykorzystuje zasoby obliczeniowe i danych chmury Bluegrit. Użytkownikami końcowymi mogą być indywidualni naukowcy lub inne centra usług danych i mogą korzystać z naszego interfejsu graficznego lub SOAP udostępnianego przez serwer WWW firmy Bluegrit. Żądania usług obejmują różne eksperymenty związane z klimatem, takie jak śledzenie ruchu chmur wschodnich lub generowanie obrazu planetarnego w wysokiej rozdzielczości. Management Server to sterownik zadań dla różnych podsystemów obliczeniowych. Odpowiada za planowanie zadań na różnych serwerach obliczeniowych. Zadania te obejmują precyzyjne geolokalizowanie siatki i dekompozycję według wartości osobliwych. Dane wejściowe dla różnych obliczeń usług mogą nie być dostępne lokalnie w momencie żądania. Management Server i Compute Blades muszą współdziałać z różnymi centrami danych NASA, aby uzyskać różne produkty danych do wymaganych obliczeń naukowych. Ponadto Management Server rutynowo planuje zadania w celu obliczenia i buforowania ogólnych wyników pośrednich, takich jak dzienne średnie radiancje w siatce.

Paradygmaty programowania naukowego

Jedną z największych przeszkód w uwolnieniu chmury do nauki jest zrozumienie jej paradygmatów obliczeniowych. Chmura zapewnia abstrakcję warstwy wykraczającą poza konfigurację samego systemu. Abstrakcja w chmurze zwykle wynika z rozproszonego oprogramowania pośredniczącego i scentralizowanego planowania zadań. Paradygmaty programowania wzmacniają oprogramowanie pośredniczące i zmienić sposób, w jaki programujemy; zmuszają nas do równoległego myślenia. Pozostała część tego rozdziału ma na celu nakłonienie naszych umysłów do zrozumienia, jak programować chmurę do zastosowań naukowych. Omawiamy dwie strategie programowania, MapReduce i Dryad, oraz różne problemy związane z nauką i sposoby ich implementacji w środowisku chmury. MapReduce to prosty paradygmat programowania dla rozproszonego przetwarzania w chmurze. Google uruchomił MapReduce jako rozwiązanie do przetwarzania równoległego dla swojego potoku indeksowania i szybko zdał sobie sprawę, że MapReduce jest przydatny do wielu innych zadań przetwarzania równoległego w zakresie pobierania danych z Internetu. Google obsługuje teraz tysiące aplikacji MapReduce. Chociaż zamierzonym celem MapReduce była analiza tekstu i uczenie maszynowe, jest ono również przydatne w wielu obliczeniach naukowych, pod warunkiem jednak, że spełniają one określone warunki . To sprawia, że MapReduce jest bardzo istotnym tematem dla obliczeń naukowych, ponieważ może ułatwić programowanie problemów, ale potencjalnie może jeszcze bardziej utrudnić trudne problemy, jeśli problem nie pasuje do paradygmatu. Dryad to elastyczny model programowania oparty na Directed Acyclic Graphs (DAG). Węzły reprezentują obliczenia, a krawędzie reprezentują kierunek przepływu danych. Dryad został opracowany przez Microsoft jako ogólny paradygmat problemów z przetwarzaniem w chmurze i jako alternatywa dla Google MapReduce. Deweloperzy Microsoftu szybko odkryli, że Dryad był znacznie bardziej elastyczny niż MapReduce, a jako dowód byli w stanie wdrożyć relacyjną bazę danych, Map and Reduce i wiele

innych paradygmatów oprogramowania, które są całkowicie zawarte w strukturze Dryad. Driada jest dobrze zaokrąglona i doskonale nadaje się do zadań wymagających dużej mocy obliczeniowej, dużej ilości danych, gęstych, rzadkich, połączonych i niezwiązanych. Dryad zapewnia bardzo elastyczne rozwiązanie i jest dobrą alternatywą dla problemów, które mogą nie pasować do prostszych paradygmatów, takich jak MapReduce. Z drugiej strony Driada jest stosunkowo skomplikowana i może nie być konieczna, gdy możliwe są łatwiejsze rozwiązania.

MapaReduce

Oryginalny artykuł MapReduce autorstwa Deana i Ghemawata (2004) bardzo zwięźle opisuje paradygmat programowania MapReduce w następujący sposób. Obliczenie pobiera zestaw wejściowych par klucz/wartość i tworzy zestaw wyjściowych par klucz/wartość. Użytkownik biblioteki MapReduce wyraża obliczenia jako dwie funkcje: Map i Reduce. Map, napisany przez użytkownika, pobiera parę wejściową i tworzy zestaw pośrednich par klucz/wartość. Biblioteka MapReduce grupuje razem wszystkie wartości pośrednie skojarzone z tym samym kluczem pośrednim I i przekazuje je do funkcji Reduce. Funkcja Reduce, również napisana przez użytkownika, akceptuje klucz pośredni I oraz zestaw wartości dla tego klucza. Łączy ze sobą te wartości, tworząc możliwie mniejszy zestaw wartości. Zazwyczaj tylko zero lub jedna wartość wyjściowa jest tworzona na wywołanie funkcji Reduce. Wartości pośrednie są dostarczane do funkcji redukcyjnej użytkownika za pomocą iteratora. To pozwala nam obsługiwać listy wartości, które są zbyt duże, aby zmieścić się w pamięci.

System czasu wykonywania, który implementuje MapReduce, obsługuje szczegóły planowania, równoważenia obciążenia i odzyskiwania błędów. Implementacja MapReduce przez Google wykorzystuje rozproszony system plików Google, w którym magazyny danych i umieszczanie danych w tym systemie plików są budowane w oparciu o założenie, że terabajtowe zestawy danych będą rozprowadzane na tysiącach dysków włożonych do węzłów komputerowych. W tego rodzaju systemie często zdarzają się awarie sprzętu i strategie replikacji stają się ważne. W przypadku awarii sprzętu, system ponawia zadania tylko na podstawie uszkodzonego sprzętu, aby nie musiał powtarzać całego procesu. Główną ideą implementacji MapReduce jest to, że wykonuje ona obliczenia, w których znajdują się dane, aby zoptymalizować opóźnienie sieci podczas przenoszenia danych. Badacze Google opracowali niedawno wsparcie programistyczne dla rozproszonej struktury danych, zwanej BigTable, która zapewnia możliwości podobne do tych w systemach bazodanowych, podczas gdy MapReduce jest czysto funkcjonalną notacją do generowania nowych plików ze starych. Dane zapisane w tabeli przez użytkowników są następnie przechowywane i zarządzane przez systemy. Chociaż BigTable nie zapewnia pełnego zestawu operacji obsługiwanych przez relacyjne bazy danych, zapewnia równowagę między ekspresją a możliwością skalowania dla bardzo dużych baz danych w środowisku rozproszonym. Framework open source Hadoop to implementacja MapReduce firmy Apache zapewniająca abstrakcję programistom, aby mogli pisać aplikacje, które mogą uzyskiwać dostęp do ogromnych danych i przetwarzać je w rozproszonym systemie plików. Zasadniczo platforma dystrybuuje dane i przetwarza funkcje map/sort/shuffle/reduce równolegle lokalnie w każdym węźle. W ten sposób można uzyskać masowo równoległe przetwarzanie za pomocą klastrów składających się z tysięcy węzłów. Ponadto wspierające środowisko wykonawcze zapewnia przezroczystą odporność na błędy poprzez automatyczne duplikowanie danych między węzłami oraz wykrywanie i ponowne uruchamianie obliczeń, które nie powiodły się w konkretnym węźle. Hadoop staje się większy i lepszy, aby dostarczać oprogramowanie typu open source do niezawodnego, skalowalnego, rozproszonego przetwarzania. Obejmuje podprojekty, takie jak HDFS, rozproszony system plików, który zapewnia wysoką przepustowość dostępu do danych aplikacji, HBASE to skalowalna, rozproszona baza danych, która obsługuje ustrukturyzowane przechowywanie danych dla dużych tabel i jest implementacją Google BigTable. Niestety, obecna wersja 0.20.0 pozwala tylko na pliki tekstowe i nie obsługuje

binarnych plików wejściowych, wspólnego formatu przetwarzania danych naukowych. Apache obiecuje zaktualizowaną wersję w przyszłym wydaniu. Można dodać inne etapy, aby rozszerzyć paradygmat MapReduce, takie jak sortowanie i tasowanie w implementacji Hadoop. Jak widać, paradygmat ma tylko dwie określone przez użytkownika funkcje: Mapuj i Zmniejsz. Świetnym sposobem na zapoznanie się z MapReduce jest przykład. Liczenie słów jest przykładem kanonicznym, z pseudokodem podanym przez Deana i Ghemawata

```
map(String key, String value):

// key: document name

// value: document contents

for each word w in value:

EmitIntermediate(w, "1");

reduce(String key, Iterator values):

// key: a word

// values: a list of counts

int result = 0;

for each v in values:

result += ParseInt(v);

Emit(AsString(result
```

Funkcja map emituje każde słowo plus powiązaną liczbę wystąpień (w tym prostym przykładzie tylko „1”). Funkcja Reduce sumuje wszystkie zliczenia wyemitowane dla określonego słowa. Tak więc, wykonując Map, aby zgłosić wystąpienie każdego słowa, a następnie zmniejszyć, aby zsumować liczbę wystąpień każdego słowa, wynikiem tego przykładu jest liczba słów dla każdego odrębnego słowa w dokumencie.

Tworząc nowe i różne funkcje Map i Reduce, MapReduce może być używany do rozwiązywania wielu problemów oprócz liczenia słów. Przetwarzanie może być wykonywane równolegle, ponieważ zarówno funkcje Map, jak i Reduce mogą być wykonywane równolegle. Mapa działa równolegle na każdym elemencie wejściowym. Reduce działa równolegle na oddzielnych grupach KV dla każdego odrębnego klucza.

Scalanie MapReduce

Yang, Dasdan, Lung-Hsiao i Parker (2007) wprowadzili ulepszenie funkcji Map-Reduce o nazwie Map-Reduce-Merge. To ulepszenie umożliwia lepszą obsługę złączeń w wielu heterogenicznych bazach danych w porównaniu z użyciem Map-Reduce. Autorzy zwracają uwagę, że Map-Reduce jest dobry dla jednorodnych baz danych. Omawiają problem z wydajnym wykonywaniem złączeń w wielu heterogenicznych bazach danych i wspominają, że Pike, Dorward, Griesemer i Quinlan (2005) wskazują, że istnieje spory brak dopasowania między mapami-Reduce i takimi złączeniami. Yang i inni (zwracają również uwagę na znaczenie operacji bazodanowych w wyszukiwarkach. Opisali również, w jaki sposób można zastosować funkcję Map-Reduce-Merge do relacyjnego przetwarzania danych Map-Reduce jest opisany w Yang et al. (2007) w następujący sposób:

Dla każdej pary (klucz,wartość) Map tworzy listę par formularza (klucz „wartość”) tak, że (klucz'',wartość'') jest jedną z tych par i tworzy listę wartości wartość'' Map-Reduce-Merge opisano w Yang et al. (2007) w kontekście rodowodów. W trybie Map-Reduce-Merge, mapa jest modyfikowana tak, aby działała na każdej linii oddzielnie. Reduce jest modyfikowane tak, aby działały na każdej linii oddzielnie, a następnie modyfikowane w celu utworzenia listy par (klucz'', wartość''') zamiast listy wartości wartość'''. Ponadto jako trzeci krok dodawane jest scalanie. Scal jest stosowany do wyjścia Zmniejsz w dwóch liniach. Z listy wartości skojarzonych z kluczem w jednym rodowodzie oraz z listy wartości skojarzonych z kluczem w innym rodowodzie, Reduce tworzy listę par formy (klucz '''', wartość ''''). Wszystkie pary stworzone przez Reduce tworzą nowy rodowód.

Driada

Dryad to paradygmat programowania i struktura oprogramowania zaprojektowana wokół idei planowania zadań i przepływu danych. Programista musi utworzyć Kierowany Wykres Acykliczny (DAG), który reprezentuje zadanie przetwarzania. Węzły wykresu są obliczeniowe jądra działające na różnych procesorach, a krawędzie grafu reprezentują zależność przepływu danych. Każdy węzeł wykresu staje się dostępny do obliczeń, gdy tylko dostępne są wszystkie dane wejściowe. Scentralizowany menedżer zadań planuje dostępne węzły wykresów na bezczynnych maszynach. Maszyna wykonuje obliczenia jądra, a po zakończeniu węzeł przekazuje swoje dane wyjściowe do swoich dzieci, a maszyna ponownie staje się bezczynna. Węzły grafu potomnego stają się dostępne do obliczeń, gdy tylko wszystkie dane wejściowe są dostępne od jego zmarłych rodziców. Obliczenia są kontynuowane w ten sposób, dopóki cały DAG nie zostanie wykonany i program się zakończy. Isard i in. opisują swój system planowania zadań za pomocą zwięzłego diagramu.

Menedżer zadań (MZ) konsultuje się z serwerem nazw (NS), aby wykryć listę dostępnych komputerów. Utrzymuje wykres zadania i harmonogramy pracy z wierzchołkami (V), gdy komputery stają się dostępne przy użyciu demona (D) jako serwera proxy. Wierzchołki wymieniają dane za pośrednictwem plików, potoków TCP lub kanałów pamięci współdzielonej. Zacieniony pasek wskazuje wierzchołki w zadaniu, które są aktualnie uruchomione.

Częstym dylematem Driad jest to, że często istnieje więcej niż jeden DAG, który zadowoli obliczenie konkretnego problemu. Który DAG jest najszybszy? Czy należy wdrożyć bardzo dokładny DAG z wysokim stopniem równoległości, czy zgrubny DAG z niskimi kosztami planowania? Czasami wybór jest jasny, na przykład planowanie jednego węzła na maszynę, ale często wybór jest znacznie trudniejszy do zrozumienia na pierwszy rzut oka. Czasami najlepszy DAG zależy od projektu klastra obliczeniowego; projektowanie sieci i sprzętu we/wy może odgrywać kluczową rolę w określaniu potencjalnych wąskich gardeł w DAG przepływu danych. Te kwestie niskiego poziomu mogą zacząć kontrastować z filozofią, że chmura powinna być całkowicie oderwana od swojego podstawowego sprzętu. Dryad ułatwia programistom zabawę ze strukturą DAG, dopóki nie zaprojektują takiego, który działa wydajnie na ich maszynie docelowej. Ponadto Microsoft zaatakował dylemat polegający na ulepszeniu wykresu za pomocą szeregu automatycznych algorytmów przycinania i optymalizacji wykresów. Techniki te są wykonywane w czasie wykonywania w harmonogramie zadań, dzięki czemu mogą podejmować decyzje na podstawie aktualnych profili i dostępności zasobów. Jeden z takich algorytmów może sprawić, że DAG będzie bardziej zgrubny, hermetyzując mniejszy podgraf w pojedynczym węźle z wykonaniem szeregowym. Chociaż enkapsulacja sprawia, że system jest mniej równoległy, może znacznie poprawić wydajność w sytuacjach, w których wykres został zaprojektowany zbyt precyzyjnie. Inną techniką jest automatyczne hierarchizowanie redukcji danych. Może to znacznie poprawić wydajność, zmniejszając ilość danych przed wysłaniem pakietów do innych komputerów i między stojakami.

Geo-reprojekcja z teledetekcją

Siatka atmosferyczna i odwzorowanie geograficzne to doskonały przykład jednoprzebiegowego problemu naukowego, który jest dobrze dopasowany do paradygmatów programowania MapReduce i Dryad. Satelitarne instrumenty do teledetekcji mierzą promieniowanie ciała doskonale czarnego z różnych regionów Ziemi, aby określić prognozy pogody i klimatu, a także dostarczać modelom atmosferycznym surowe dane do asymilacji. Algorytm georeprojekcji jest jednym z pierwszych głównych kroków obliczeniowych w łańcuchu wszelkich satelitarnych prognoz atmosferycznych. Satelita obserwuje temperaturę powierzchni 316 K (43◦C ∼109◦F), która jest gorąca! Ale gdzie to jest? Nowy Meksyk? Libia? Georeprojekcja rozwiązuje to zadanie, tworząc siatkową mapę Ziemi ze średnimi obserwowanymi temperaturami lub radiancjami. Mierzony obszar na Ziemi jest funkcją pozycji instrumentu i kierunku, w którym jest obserwowany. Rysunek 16.3 jest schematem instrumentu satelitarnego NASA Atmospheric Infrared Sounder (AIRS). Satelita ma orbitę synchroniczną ze słońcem, podczas gdy Ziemia się obraca, więc na projekcji Lat Lon, wzór skanu stawu jest zilustrowany niebieskim paskiem po lewej stronie. Instrument mierzy wiele obserwacji podczas lotu, ponieważ czujnik szybko oscyluje w zakresie od –48,95 do 48,95 stopnia, wykonując 90 obserwacji co 2,7 sekundy. Mapa Ziemi jest jednolicie podzielona na kilka regionów lub komórek siatki. Celem jest zmierzenie temperatury lub promieniowania dla każdej komórki na siatce. Gdy satelita mierzy tę samą komórkę siatki więcej niż jeden raz, wynikowe temperatury są razem uśredniane.

Odwzorowanie geograficzne z teledetekcją z MapReduce

Program MapReduce do odwzorowania geograficznego jest podobny w strukturze do problemu kanonicznego liczenia słów, pod warunkiem, że zignorujemy szczegóły dotyczące geometrii obliczeniowej i optyki czujnika. Zamiast liczyć słowa, uśredniamy komórki siatki. Uśrednianie jest tylko trochę trudniejsze niż sumowanie, które z kolei jest tylko trochę trudniejsze niż liczenie. Program ma niezmienioną strukturę, ale nowe szczegóły.

```
map(int timestamp, Measurement measurements):

// key: timestamp

// value: a set of instrument observations

for each measurement m in measurements

Determine region r containing measurement m

EmitIntermediate(r, m.value);

reduce(int region, Iterator measurements):

// key: a region ID

// value: a set of measurement

// values contained in that region

double result = 0.0;

for each m in measurements:

result += m;

//divide by the total number of measurements
```

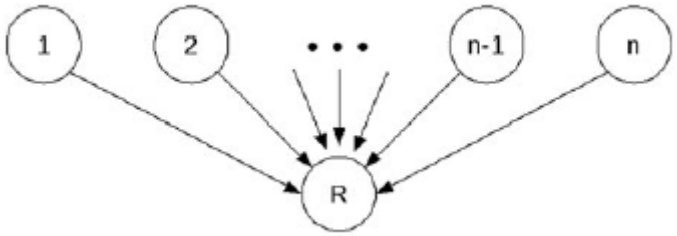
result = result / measurements.size;

Emit(result);

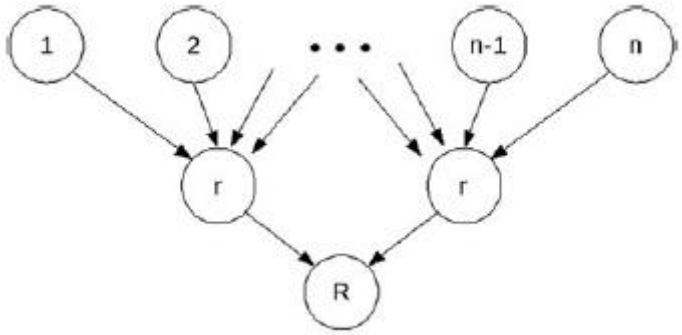
Cała optyka czujnika i geometria do określenia odpowiedniego obszaru są przesłonięte w powyższym pseudokodzie linią „Określ region r zawierający pomiar m”. W tym prostym przykładzie zakładamy, że na pomiar przypada tylko jeden region. Jednak w bardziej realistycznych projekcjach obserwacja może nakładać się na wiele regionów. W takim przypadku mapa musiałaby emitować częściowe pomiary dla każdego regionu, a redukcja pozostałaby niezmieniona. Zauważ, że ostatnim ważnym krokiem redukcji jest podzielenie przez liczbę pomiarów (wymiary.rozmiar). Podział ten przekształca rozłożone sumowanie w rozłożoną średnią, aby uzyskać średni pomiar regionu.

Georeprojekcja teledetekcyjna z Dryad

W przypadku Dryad zadanie zdalnej reprojekcji geograficznej może być obliczane nieco inaczej niż w przypadku MapReduce. Przyjmiemy, że siatka wyjściowa ma porównywalnie mniejszą rozdzielczość niż zbiór danych wejściowych. To założenie jest zwykle słuszne dla problemu, ponieważ sondujący instrument zazwyczaj obserwuje kilkakrotnie nakładające się obszary w ciągu kilku godzin lub dni obserwacji. Ponadto w przypadku zastosowań związanych z klimatem często nie są wymagane siatki drobnoziarniste, co pozwala na jeszcze dalszą redukcję danych. Problemy, które znacznie zmniejszają ilość danych, są zazwyczaj dobrze opisane za pomocą wykresu typu redukcyjnego. Podstawowy ogólny wykres redukcji pokazano na rysunku



Zauważ, że istnieje wiele wykresów redukcji, które dają taki sam wynik jak ten wymieniony powyżej. Przykładem jest ten przedstawiony na rysunku, który charakteryzuje się dwupoziomową hierarchią.



Częściowe węzły redukcji wyliczają częściowe centroidy, a następnie przekazują wynik do końcowego węzła redukcji R. To podejście jest bardziej równoległe, ponieważ istnieje więcej niezależnych węzłów pracujących nad wykonaniem części redukcji. Niestety, w tym podejściu hierarchicznym jest też więcej

narzutów, ponieważ istnieje więcej węzłów, które należy zaplanować. Poniższy pseudokod może być użyty do redukcji DAG opisanych w tej sekcji. Start() reprezentuje węzły 1-n, a Reduce() reprezentuje węzły r i R.

```
start():
// input0: instrument measurements
Measurement []measurements = ReadChannel(0);
// make an empty array of gridcell regions
Region []regions = new EmptyRegions();
//put each measurement in the region
for each measurement m in measurements
Determine region r containing measurement m
r.result += m;
r.count += 1;
//write the region array out to the channel zero
WriteChannel(newCentroids, 0);
reduce():
// input0-n: region arrays
Region [][]regions;
for every input channel i
regions[i] = ReadChannel(i);
//a single region array for the results
Region []results = new EmptyRegions();
//accumulate all of the regions together
for every input channel i
for every region j in regions[i]
results[j].result += regions[i][j].result
results[j].count += regions[i][j].count
//divide, to perform the averaging
for every region j in results
results[j].result /= results[j].count
//don't double divide if we reduce multiple times
results[j].count = 1
```

//We’re done, write results to output channel 0

WriteChannel(results, 0);

Klastrowanie K-średnich

Klastrowanie jest zasadniczym elementem wielu obliczeń naukowych, w tym klasyfikacji genów oraz symulacji fizyki ciała N. Celem klastrowania jest rozdzielenie wielu wielowymiarowych punktów danych na N grup na podstawie ich pozycji względem innych punktów w zestawie danych. Grupowanie K-średnich wykorzystuje koncepcję środka ciężkości lub średniej pozycji wszystkich punktów w tej grupie, aby zdefiniować skupienie. Początkowo punkty są pogrupowane losowo w klastry. K-średnie iteracyjnie udoskonala te klastry, aż do zbieżności do stabilnego klastrowania. K-Means używa centroidu skupień (średnia pozycja) do określenia grupowania skupień. Pojedyncza iteracja K-średnich wygląda następująco:

1. Oblicz centroid klastra (średnia ze wszystkich punktów) dla każdego klastra

2. Przypisz ponownie wszystkie punkty do klastra z najbliższym centroidem

3. Test na zbieżność

Klastrowanie K-Means z MapReduce

Klastrowanie K-Means to proces iteracyjny, który jest dobrym kandydatem do MapReduce. MapReduce będzie używany do obliczeń centroidu i klastrowania wykonywanych w każdej iteracji. Można wywołać MapReduce wewnątrz pętli „if a” (aż do zbieżności), aby obliczyć metody iteracyjne, takie jak grupowanie K-Means. Głównym powodem, dla którego K-Means jest rozsądnym kandydatem do MapReduce, jest to, że wykazuje ogromną niezależność danych. Obliczenie centroidu jest w zasadzie średnią rozłożoną, która jest małą odmianą sumy rozłożonej, czego przykładem jest kanoniczne liczenie słów. Rozproszone podsumowania wymagają prostych operacji redukcji listy. Ponowne przypisanie punktów do klastrów wymaga tylko aktualnego punktu i wszystkich centroidów klastrów; punkty mogą być przydzielane niezależnie od siebie. Poniżej znajduje się pseudokod dla klastrowania K-Means przy użyciu MapReduce. Funkcja Map pobiera jako dane wejściowe pewną liczbę punktów oraz listę centroidów i na tej podstawie tworzy listę częściowych centroidów. Te częściowe centroidy są agregowane w funkcji Reduce i używane w następnej iteracji MapReduce

map(void, {Centroid []centroids, Point []datapoints}):

// key: not important

// value: list of centroids and datapoints

Centroid []newCentroids;

Initialize newCentroids to zero

for each point p in datapoints

Determine centroid centroids[idx] closest to p

//accumulate the point to the new centroid

newCentroids[idx].position += p;

```
//we added a point, so remember the
//total for averaging
newCentroids[idx].total += 1;
for each centroid newCentroids[idx] in newCentroids
//send the intermediate centroids for accumulation
EmitIntermediate(idx, newCentroids[idx]);
reduce(int index, Centroid []newCentroids):
// key: centroid index
// value: set of partial centroids
Centroid result.position = 0;
//accumulate the position and total for a grand total
for each centroid c in newCentroids
result.position += c.position;
result.total += c.total;
//Divide position by total to compute
// the average centroid
result.position = result.position / result.total
Emit(result);
```

Klastrowanie K-Means z Driadą

K-Means można równie dobrze zaimplementować w paradygmacie i sposobie myślenia Driad. Głównym zadaniem programowania z paradygmatem Dryad jest zrozumienie przepływu danych w systemie. Ostre obserwacje dotyczące K-średnich są takie, że zazwyczaj każdy klaster ma wiele punktów. Innymi słowy, klastrów jest znacznie mniej niż punktów. Tak więc w każdej iteracji całkowita ilość informacji jest znacznie zmniejszona przy wyznaczaniu centroidów ze zbioru punktów. Redukcyjny DAG jest dobrym wyborem dla K-Means, ponieważ objętość danych jest zmniejszona. Poniższy pseudokod zostałby użyty do wykonania operacji redukcji grafu dla algorytmu K-średnich przy użyciu Driady. Na diagramach redukcji wykresów podanych w sekcji „Reprojekcje geograficzne z zdalnym wykrywaniem za pomocą Driady”, start() jest funkcją dla węzłów 1-n, a Reduce() jest funkcją dla węzłów r i R.

```
start():
// input0: Complete list of centroids from the prior run
Centroid []centroids = ReadChannel(0);
// input1: Partial list of datapoints
Point []datapoints = ReadChannel(1);
```

```
// make a new list of centroids
Centroid []newCentroids;
Initialize newCentroids to zero
for each point p in datapoints
Determine centroid centroids[idx] closest to p
//accumulate the point to the new centroid
newCentroids[idx].position += p;
//we added a point, so remember the
//total for averaging
newCentroids[idx].total += 1;
//send the intermediate centroids for accumulation
//channel zero is the only output channel we have
WriteChannel(newCentroids, 0);
reduce():
// input0-n: list of intermediate centroids
Centroid [][]newCentroids;
for every input channel i
newCentroids[i] = ReadChannel(i);
// make a list of result centroids
Centroid []results;
for every input channel i
results[i].position = 0;
//accumulate the position and total for a grand total
for each centroid c in newCentroids[i]
result[i].position += c.position;
result[i].total += c.total;
//Divide position by total to compute
// the average centroid
results.position = result.position / result.total
//don't double divide if we reduce multiple times
results.total = 1
```

//write our the results to channel 0

WriteChannel(results, 0);

Rozkład według wartości osobliwych

Dekompozycja wartości singularnej (SVD) może być również zrównoleglona z paradygmatami przetwarzania w chmurze. Cel SVD jest bardzo podobny do diagonalizacji macierzy. Należy opisać, jak macierz M może być reprezentowana jako iloczyn trzech macierzy w warunkach opisanych poniżej:

$$M = U\Sigma V^T$$

Gdzie M jest oryginalną macierzą m na n, U jest macierzą ortogonalną m na m, V^T jest macierzą ortogonalną n na n i jest macierzą diagonalną m na n. Ideą metody Jacobiego jest rozpoczęcie od tożsamości M = IMI i próba powolnego przekształcenia tej formuły w $M = U\Sigma V^T$ za pomocą serii obrotów mających na celu wyzerowanie elementów poza przekątną po jednym na raz. Niestety wyzerowanie jednego elementu może odzerować inny. Jeśli jednak to podejście zostanie wystarczająco powtórzone, macierz M zbiegnie się z macierzą diagonalną Σ . Ponieważ ta książka koncentruje się na przetwarzaniu w chmurze, a nie na algebrze macierzowej, nie będziemy szczegółowo omawiać formuł wymaganych przez metodę Jacobiego i pokrewne metody. Więcej informacji można znaleźć w. Algorytmy One Sided JRS i Jacobi, aby wyzerować pojedynczy element, wymagają modyfikacji dwóch wierszy w macierzy. Pojedyncze przeciągnięcie, dla każdej pary wierszy w macierzy, należy obliczyć iloczyn skalarny z innymi wierszami i użyć tej wartości do modyfikacji obu wierszy. W przypadku macierzy n-na-n istnieje n(n-1)/2 takich par wierszy. Dlatego naturalnym jest podzielenie macierzy na wiersze w celu równoległej implementacji. Rajasekaran i Song proponują podejście okrężne, w którym każda maszyna przechowuje dwa bloki, oblicza wszystkie pary wierszy w każdym bloku, a następnie oblicza wszystkie pary wierszy między tymi dwoma blokami. Następnie klocki są tasowane do innych maszyn, jak pokazano na rys. 16.6. Chociaż powyższy wzorzec dostępu do danych można zaimplementować bezpośrednio w systemach gridowych z przekazywaniem wiadomości, równie łatwo można go zaimplementować za pomocą paradygmatów przetwarzania w chmurze MapReduce i Dryad. Paradygmaty chmury nadal zapewniają dodatkowe korzyści, takie jak odporność na awarie i abstrakcja danych.

Rozkład według wartości osobliwych za pomocą MapReduce

Wzorzec danych blokowych okrężnych opisany na powyższym rysunkuy można zaimplementować za pomocą MapReduce, ale z jednym zastrzeżeniem. Różnica polega na tym, że MapReduce woli kontrolować sposób dystrybucji danych na podstawie pary klucz/wartość bloku. W ten sposób klucz może być używany jako identyfikator maszyny wirtualnej, a nie identyfikator fizyczny. Każdy blok to para klucz-wartość. Operacja zmniejszania akceptuje dwie pary klucz-wartość (bloki) modyfikuje je i emituje obie z powrotem jako wyniki. Ta procedura redukcji mapy musi być wykonywana iteracyjnie aż do zbieżności.

map(int blockPos, Block block):

// key: the current position of the block

// value: 2D block

int newBlockPos;

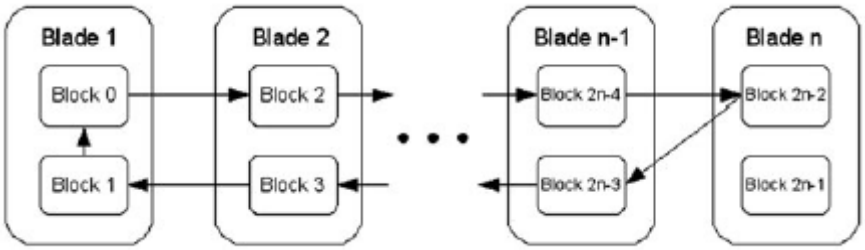
if (blockPos == 0)

```
newBlockPos = 1;
else if (blockPos = 2n-1)
newBlockPos = blockPos;
else if (blockPos == 2n-2)
newBlockPos = 2n-3;
else if (blockPos % 2 == 1)
newBlockPos += 2;
else // (blockPos % 2 == 0)
newBlockPos -= 2;
//key must be equal to the virtual machine ID.
//however, the slot (top or bottom) is also necessary
//to disambiguate the block slots
int machineID = floor(newBlockPos/2);
BlockValue blockVal;
blockVal.block = block;
blockVal.slot = newBlockPos % 2;
EmitIntermediate(machineID, blockVal);
reduce(int machineID, BlockValue (Marzouk & Ghoniem, 2005)
blockVals):
// key: virtual machine ID
// value: structure with the slot (top or bottom)
// and the block data
//use a convention arrang the blockvals by slot
if (blockVals[0].slot == 1)
swap (blockVals[0], blockVals (Alon et al., 1999);
//perform rotations in slot 0
Block block0 = blockVals[0];
for i=0 to block0.numRows-1
for j=i to block0.numRows-1
rotate(block0.row[i], block0.row[j]);
//perform rotations in slot 1
```

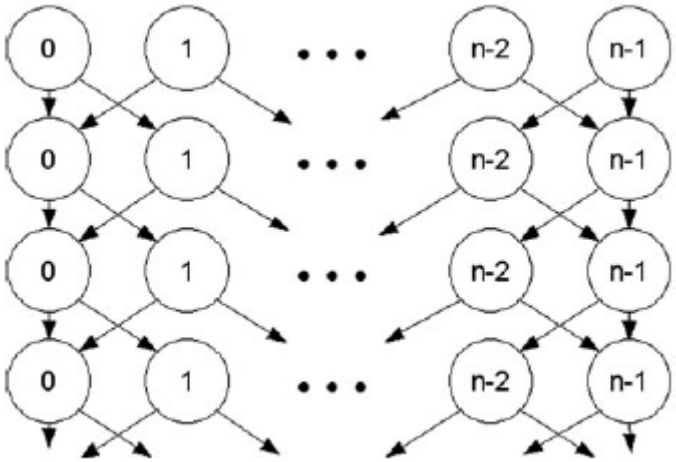
```
Block block1 = blockVals (Alon et al., 1999);

for i=0 to block1.numRows-1

for j=i to block1.numRows-1

rotate(block1.row[i], block1.row[j]);

//perform rotations across both slots

for i=0 to block0.numRows-1

for j=0 to block1.numRows-1

rotate(block0.row[i], block1.row[j]);

//remember the block position for the next iteration

int blockPos0 = 2*machineID;

int blockPos1 = 2*machineID + 1;

Emit(blockPos0, block0);

Emit(blockPos1, block1);
```

Rozkład według wartości osobliwych za pomocą Driady

Podobnie jak MapReduce, Dryad lubi kontrolować dystrybucję danych za pomocą planowania zadań. Z tego powodu równie ważne jest używanie identyfikatorów maszyn wirtualnych dla wzorca dostępu do danych SVD okrężnego. W przypadku Dryad węzły w DAG reprezentują maszyny wirtualne, a krawędzie reprezentują rozkład danych. Wykres na rysunku jest cykliczny.



Musi być acykliczny do użytku z Driadą. Aby to zrobić, musimy rozwinąć wykres. Niestety, wynikowy wykres miałby nieskończoną długość, ponieważ nie wiadomo, ile iteracji należy wykonać, aby zbieżność spadła poniżej pewnego progu błędu. Na szczęście wystarczy zrobić duży skończony graf i po prostu zakończyć wcześnie po zbieżności. Łączność wykresu jest nieco skomplikowana, aby osiągnąć wzór pokazany na rysunku.



Zauważ, że każdy węzeł ma dwa bloki: jeden na górze, a drugi na dole. Definiujemy, że kanał 0 odczytuje dane wejściowe z górnego bloku, a kanał wejściowy 1 odczytuje dane wejściowe z dolnego bloku. Określamy również, że kanał wyjściowy 0 zapisuje dane wyjściowe z górnego bloku, a kanał wyjściowy 1 zapisuje dane wyjściowe z dolnego bloku. W każdym kroku czasowym oba bloki są odczytywane, modyfikowane i zapisywane w odpowiednich kanałach.

Istnieje kilka przypadków narożnych dla łączności grafu. Przypadki te byłyby obsługiwane w konstrukcji grafu i dlatego nie są wymienione w pseudokodzie tej sekcji. Pseudokod w węźle do wykonywania rotacji bloków jest wymieniony poniżej.

node():

// input0: Block for top slot

Block block0 = ReadChannel(0);

// input1: Block for bottom slot

Block block1 = ReadChannel(1);

//perform rotations in slot 0

for i=0 to block0.numRows-1

for j=i to block0.numRows-1

rotate(block0.row[i], block0.row[j]);

//perform rotations in slot 1

for i=0 to block1.numRows-1

for j=i to block1.numRows-1

rotate(block1.row[i], block1.row[j]);

//perform rotations across both slots

for i=0 to block0.numRows-1

for j=0 to block1.numRows-1

```
rotate(block0.row[i], block1.row[j]);

WriteChannel(block0, 0);

WriteChannel(block1, 1);
```

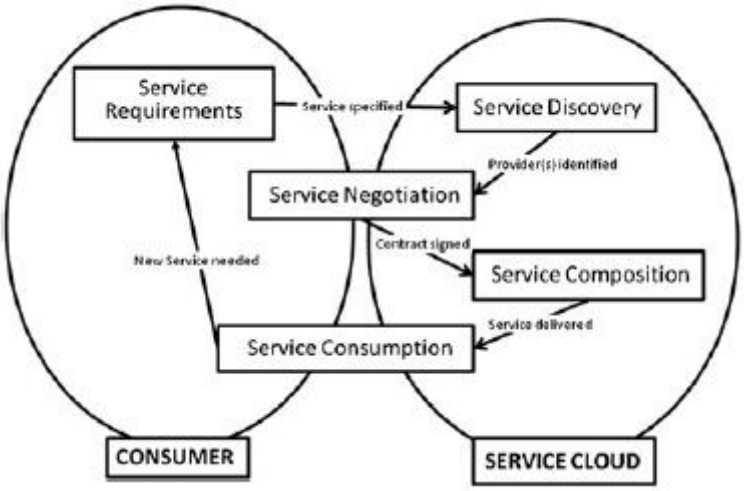
Dostarczanie usług w zakresie obliczeń naukowych w chmurze

Istniejące metodologie rozwoju usług nie uwzględniają środowiska chmurowego, które obejmuje usługi tworzone na żądanie w krótkim czasie. Obecnie dostawcy usług decydują, w jaki sposób usługi są łączone i dostarczane konsumentom usług. Zwykle odbywa się to statycznie w procesie ręcznym. Istnieje potrzeba opracowania mechanizmów wielokrotnego użytku, zorientowanych na użytkownika, które pozwolą konsumentowi usługi określić pożądane ograniczenia związane z bezpieczeństwem lub jakością, a także automatyczne systemy po stronie dostawcy kontrolujące wybór, konfigurację i skład usług. Nie powinno to wymagać od konsumenta zrozumienia technicznych aspektów usług i składu usług. Radiance atmosferyczne zorientowane na usługi (SOAR) demonstrują wiele przykładów przepowiedzianego paradoksu. Klimatolodzy chcą badać profil atmosferyczny Ziemi i potrzebują obserwacji satelitarnych o odpowiedniej jakości do eksperymentów. Byłoby daremne, gdyby uczyli się każdego wymaganego kroku przetwarzania danych, włącznie z numerami wersji algorytmów i architekturą obliczeniową używaną do tworzenia wszystkich wymaganych danych. Zależy im jednak, aby taki łańcuch narzędzi był gdzieś dobrze udokumentowany. Gdyby kolega nie zgodził się z jego odkryciami po latach, kiedy wszystkie stare dane, algorytmy i sprzęt zostały zaktualizowane, czy naukowiec zna łańcuch narzędzi, który wyprodukował jego stare eksperymenty? Mówi się, że nauka bez odtwarzalności nie jest nauką, jednak w świecie, w którym dane i obliczenia są przekazywane w zawiłej chmurze, pochodzenie jest zbyt łatwe do zgubienia. Zaproponowaliśmy metodologię dostarczania zwirtualizowanych usług za pośrednictwem chmury (Joshi, Finin i Yesha, 2009). Cykl życia usługi IT w chmurze dzielimy na pięć faz. W kolejności wykonania są to wymagania, odkrycie, negocjacje, kompozycja i konsumpcja. Rysunek 16.8 ilustruje proponowany przez nas cykl życia usługi.

Wymagania serwisowe

W fazie wymagań usługi konsument wyszczególnia specyfikacje techniczne i funkcjonalne, które musi spełnić usługa naukowa. Definiując wymagania usługi, konsument określa nie tylko funkcjonalność, ale również niefunkcjonalność atrybutu, takie jak ograniczenia i preferencje dotyczące jakości danych, zgodności usług i wymaganych zasad bezpieczeństwa usługi. W zależności od kosztu i dostępności usługi, konsument może być podatny na kompromisy w zakresie jakości danych usługi. Na przykład prosta usługa dostarczająca obrazy Ziemi może dostarczać dane jako obrazy o różnej jakości rozdzielczości. W zależności od wymagań usługobiorcy mogą być zainteresowani obrazami o wysokiej rozdzielczości (wyższa jakość) lub mogą być zadowoleni z niższej rozdzielczości obrazu, jeśli skutkuje to niższymi kosztami usługi. Takie jednoznaczne opisy są przydatne nie tylko dla konsumenta usługi, ale także dla dostawcy. Na przykład koszt utrzymania jakości danych usługi można zoptymalizować w zależności od rodzaju jakości danych wymaganej w usłudze. Zaletą dla usługodawcy jest to, że nie będzie musiał utrzymywać danych o niższej jakości z taką samą wydajnością, jak dane o wyższej jakości; ale nadal będą mogli znaleźć konsumentów danych. Mogą rozdzielać dane na różne bazy danych i udostępniać je na żądanie. Ponieważ łatwość konserwacji jest kluczową miarą jakości, niskie zapotrzebowanie na konserwację danych serwisowych spowoduje poprawę jakości usługi. Faza zapotrzebowania na usługę składa się z dwóch głównych podfaz: specyfikacji usługi i żądania usługi. Specyfikacja usługi: W tej podfazie konsument określa szczegółowe specyfikacje funkcjonalne i techniczne potrzebnej usługi. Specyfikacja funkcjonalna opisuje szczegółowo, jakie funkcje/zadania powinna zautomatyzować usługa oraz akceptowalny poziom wydajności oprogramowania usługi i agenta usługi (tj. człowieka świadczącego usługę). Specyfikacje techniczne określają sprzęt, system

operacyjny, standardy aplikacji i zasady obsługi języka, których powinna przestrzegać usługa. Specyfikacje podają również dopuszczalne poziomy bezpieczeństwa, jakość danych i poziomy wydajności agenta serwisowego i oprogramowania serwisowego. Określono również szczegóły dotyczące zgodności usług, takie jak wymagane certyfikaty, normy, których należy przestrzegać itp. W zależności od wymagań specyfikacje mogą być tak krótkie, jak i szczegółowe. Rysunek 16.9 przedstawia wszystkie parametry jakości usług (QoS) dostępne dla użytkownika dla usługi sieciowej SOAR Gridded Average Brightness Temperature. GUI są dobrym, czytelnym dla człowieka uzupełnieniem języków opisu czytelnych dla komputera, używanych przez usługi sieciowe w stylu SOAP i REST. Parametry QoS zorientowane na użytkownika obejmują zmienną rozdzielczość i typ wyniku (obraz jpg lub hdf i pliki danych binarnych). Dostawcy usług samodzielnie podejmują decyzje dotyczące zasobów obliczeniowych i powiązanych parametrów. Na przykład pojedyncze zadanie usługi może być rozłożone na wiele węzłów, ale ten rodzaj przetwarzania rozproszonego nie jest konieczny w przypadku małych usług, które kończą się bardzo szybko. Od dostawcy zależy, w jaki sposób zaplanowane jest zadanie i jakie zasoby należy wyznaczyć. Zapytanie o serwis: Gdy konsumenci zidentyfikują i zaklasyfikują swoje potrzeby serwisowe, wydadzą „Request for Service" (RFS). Żądanie to można złożyć, kontaktując się bezpośrednio z dostawcami usług, np. za pomocą przycisku „Prześlij żądanie"



To bezpośrednie podejście omija wszelkiego rodzaju mechanizm wykrywania, ale kończy się niepowodzeniem, gdy konsument nie jest świadomy dostawcy usług. Alternatywnie, konsumenci mogą skorzystać z wyszukiwarki usług w chmurze, aby uzyskać żądaną usługę, o ile usługa jest zarejestrowana w jakimś silniku wyszukiwania. Wymóg usługi jest krytyczną fazą cyklu życia usługi, ponieważ określa „co" usługi. Jest to połączenie faz „planowania" i „zbierania wymagań" w tradycyjnym cyklu życia oprogramowania. W tej fazie konsumenci poświęcą maksimum wysiłku, dlatego zostało to w całości przedstawione w obszarze konsumenta na diagramie cyklu życia. Konsument mógłby zlecić kompilację specyfikacji technicznych i funkcjonalnych innemu dostawcy, ale odpowiedzialność za pomyślne zakończenie tej fazy spoczywa na kliencie, a nie na chmurze usług. Po wydaniu RFS wchodzimy w fazę wykrywania cyklu życia usługi.

Wykrywanie usług

W tej fazie dostawcy usług, którzy oferują usługi zgodne ze specyfikacjami wyszczególnionymi w RFS, są przeszukiwani (lub znajdowani) w chmurze. Wykrywanie jest ograniczone przez zdefiniowane atrybuty funkcjonalne i techniczne, a także przez politykę budżetową, bezpieczeństwa, jakości danych i agenta konsumenta. Jeżeli konsument wybierze opcję przeszukiwania chmury zamiast wysyłania RFS

do ograniczonej grupy dostawców, wówczas wykrywanie usług odbywa się za pomocą silnika wyszukiwania/wykrywania usług. Ten silnik wykonuje zapytanie dotyczące usług zarejestrowanych w centralnym rejestrze lub organie zarządzającym i dopasowuje domenę, typ danych, specyfikacje funkcjonalne i techniczne oraz zwraca wynik z dostawcami usług spełniającymi maksymalną liczbę wymagań wymienionych na górze. Faza odkrywania może nie zapewnić konsumentom pomyślnych wyników, dlatego będą musieli albo zmienić specyfikacje, albo zmienić swoje procesy wewnętrzne, aby móc korzystać z usługi, która najbardziej odpowiada ich potrzebom. Jeśli konsumenci znajdą dokładną usługę naukową w ramach budżetu, którego szukają, mogą zacząć korzystać z usługi. Jednak często konsumenci otrzymają listę dostawców, którzy będą musieli skomponować usługę, aby spełnić wymagania konsumenta. Konsument będzie wówczas musiał rozpocząć negocjacje z dostawcami usług, co jest kolejną fazą cyklu życia. Każdy wynik wyszukiwania zwróci również głównego dostawcę, który będzie negocjował z konsumentem. Zwykle będzie to dostawca, którego usługa spełnia większość specyfikacji wymagań.

Negocjacja usług

Faza negocjacji usługi obejmuje dyskusję i porozumienie między usługodawcą a konsumentem dotyczące dostarczonej usługi i kryteriów jej akceptacji. Dostarczona usługa jest określona przez specyfikacje określone w 400 D. Chapman et al. RFS. Akceptacja usługi jest zwykle oparta na umowach o poziomie usług (SLA), które zgadzają się usługodawca i konsument. Umowy SLA definiują dane usługi, tryb dostawy, szczegóły agenta, politykę zgodności, jakość i koszt usługi. Negocjując poziom usług z potencjalnymi dostawcami usług, konsumenci mogą wyraźnie określić ograniczenia dotyczące jakości usług (jakość danych, koszt, bezpieczeństwo, czas odpowiedzi itp.), których wymagają. Oczywiście naukowcy uwielbiają pracować z najdokładniejszymi i najdokładniejszymi dostępnymi danymi, dopóki nie napotkają z nimi praktycznych problemów. Drobny zestaw danych może zająć zbyt dużo pamięci, obliczenia mogą zająć zbyt dużo czasu, a być może nieodpowiednie algorytmy, takie jak analiza głównych komponentów, mogą wymagać niepraktycznej ilości pamięci RAM. Tego rodzaju problemy zmuszają naukowców do ponownego rozważenia poziomu jakości, którego faktycznie potrzebują do pożądanego eksperymentu. Umowy SLA pomogą w określeniu wszystkich takich ograniczeń i preferencji oraz będą częścią umowy o świadczenie usług między usługodawcą a konsumentem. Negocjacje wymagają informacji zwrotnej, a serwer musi być w stanie oszacować koszt związany z usługą i dostarczyć te statystyki konsumentowi przed wykonaniem usługi. Chociaż możliwe jest automatyczne negocjowanie przy użyciu algorytmów wspinaczkowych i powiązanych algorytmów optymalizacyjnych, SOAR był systemem usługowym przeznaczonym głównie dla użytkowników końcowych będących ludźmi. W zależności od usługi i QoS obliczenie żądanej transakcji SOAR może zająć sekundy, minuty lub godziny. Nie jest konieczne podanie dokładnego oszacowania czasu, ale należy przedstawić przynajmniej przybliżone przypuszczenie. Na przykład bardzo pomocne byłoby ostrzeżenie „Obliczanie tej transakcji może potrwać kilka godzin”. Oprócz ostrzegania użytkowników o usługach o wysokich kosztach, dostawca musi również wyjaśnić pewną strategię obniżania kosztów. Oczywiście zmniejszenie QoS obniży koszty, ale stopień może się różnić. Niektóre parametry usługi mogą mieć dramatyczny wpływ na koszt usługi, podczas gdy inne mają niewielki wpływ. Na przykład w usłudze SOAR Gridded Average Brightness Temperature „rozdzielczość” ma ogromny wpływ na wydajność, podczas gdy „typ” (format) nie ma dużego wpływu na wydajność. Niestety, jeśli konsument nie wie, które parametry zmienić, może losowo przełączać opcje lub całkowicie z frustracji zrezygnować z usługi. Chociaż algorytmy mogą z radością przełączać opcje systematycznie, ludzie woleliby wiedzieć, co robią. SOAR dokumentuje najbardziej znane sposoby obniżania kosztów. Możliwe są nawet wywłaszczające GUI, które oznaczają użytkownika, gdy tylko wybierze opcję o wysokim koszcie. Czasami usługodawca będzie musiał połączyć zestaw usług lub skomponować usługę z różnych elementów dostarczanych przez różnych usługodawców, aby spełnić wymagania

konsumenta. Na przykład usługi SOAR często muszą komunikować się z centrami danych NASA, aby uzyskać dane o wyższej rozdzielczości do przetwarzania. Faza negocjacji obejmuje również dyskusje, które główny dostawca usług prowadzi z innymi dostawcami komponentów. W przypadku, gdy usługi są świadczone przez wielu dostawców (usługa złożona), za strukturę usługi odpowiada główny dostawca łączący się z konsumentem. Główny dostawca będzie również musiał negocjować jakość usług (QoS) z dostawcami pomocniczymi, aby zapewnić spełnienie metryk umowy SLA. Tak więc faza negocjacji składa się z dwóch krytycznych podfaz, negocjacji umów SLA i negocjacji QoS. Jeśli istnieje potrzeba usługi złożonej, toczą się dyskusje iteracyjne między konsumentem a głównym dostawcą oraz głównym dostawcą i dostawcami komponentów. Produktem końcowym fazy negocjacji jest umowa o świadczeniu usług między konsumentem a głównym dostawcą oraz między głównym dostawcą a dostawcami części składowych (lub wtórnymi). Po zatwierdzeniu umowy serwisowej cykl życia przechodzi do fazy tworzenia, w której oprogramowanie serwisowe jest kompilowane i składane.

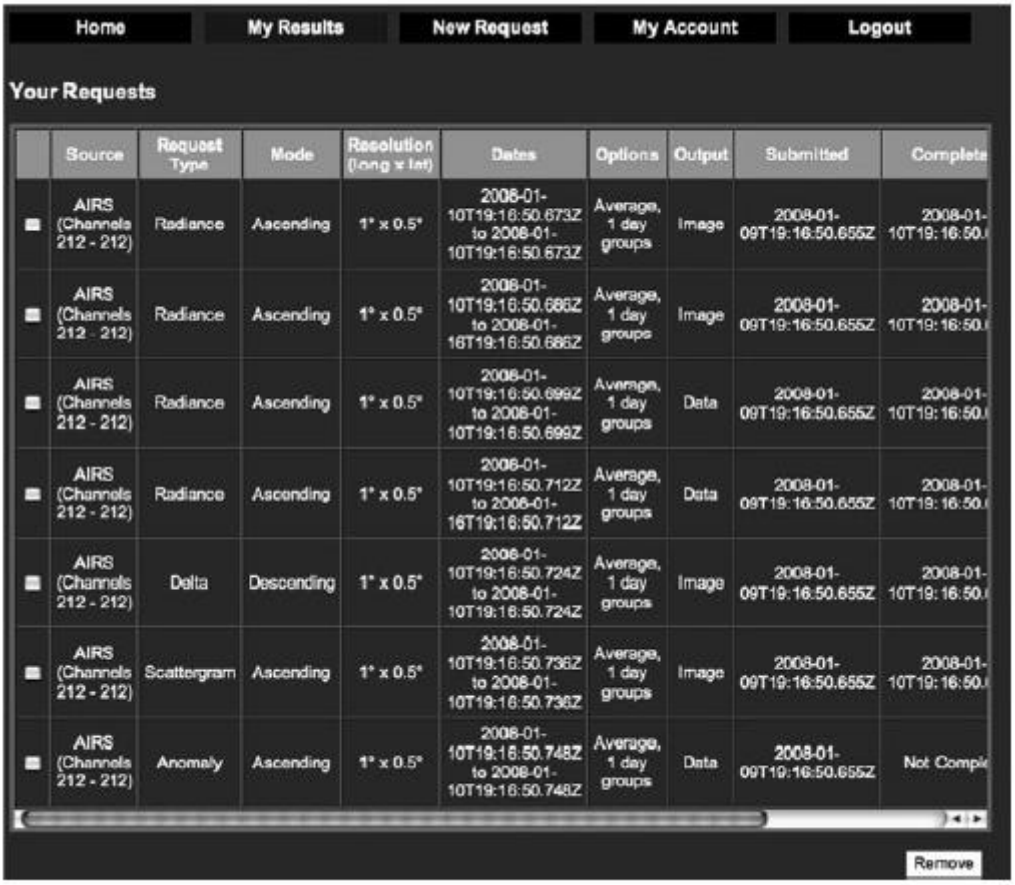
Skład usługi

W tej fazie co najmniej jedna usługa świadczona przez jednego lub więcej dostawców jest łączona i dostarczana jako jedna usługa. Choreografia usługi określa kolejność wykonywania elementów usługi. Często skład może nie być statycznym przedsięwzięciem, a nawet może zależeć od parametrów QoS użytych w RFS. Usługa średniej jasności w siatce SOAR jest doskonałym przykładem dynamicznego przepływu pracy kompozycji. Surowe dzienne obserwacje są buforowane w zgrubnej rozdzielczości, więc do obsługi takich żądań potrzebne jest tylko lokalne przetwarzanie równoległe. Jeśli jednak określone dane nie są dostępne w pamięci podręcznej, należy je uzyskać z centrów danych NASA Goddard i MODAPS. Taki jest przykład łączenia usług z zagranicznymi podmiotami NASA. W zależności od żądania to zdalne wykonanie może zająć godziny, aby obliczyć, a zatem wymaga wykonania asynchronicznego. Jednak dla porównania, zgrubne usługi lokalne można ukończyć w kilka sekund. Jest to doskonały przykład tego, jak specyfikacja QoS i RFS może mieć tak ogromny wpływ na wynikowy przepływ pracy, wydajność i zasoby. Wielokrotnie to, co jest reklamowane przez dostawcę jako pojedyncza usługa, może z kolei być zwirtualizowaną usługą złożoną z różnych komponentów dostarczanych przez różnych dostawców. Konsument musi wiedzieć, że usługa jest złożona wyłącznie do celów księgowych. Dostawca będzie musiał monitorować wszystkie inne usługi, od których jest zależny (takie jak usługi baz danych, usługi sieciowe itp.), aby upewnić się, że umowy SLA określone w poprzedniej fazie są przestrzegane i rejestrowane pod kątem odtwarzalności naukowej. Po skomponowaniu usługi jest ona gotowa do dostarczenia konsumentowi. Cykl życia wchodzi następnie w ostatnią fazę zużycia usług.

Zużycie i monitorowanie usług

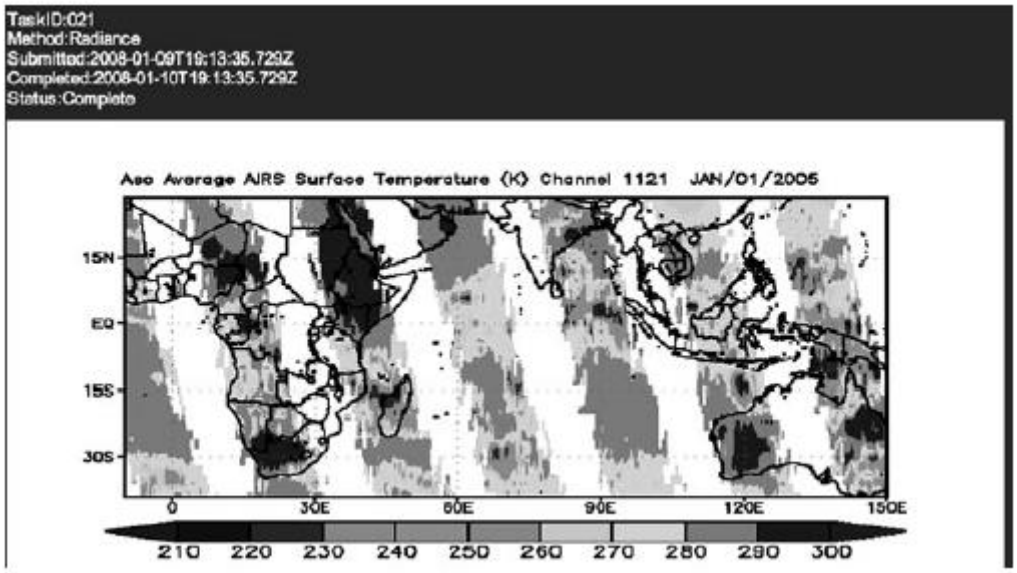
Usługa jest dostarczana do konsumenta w oparciu o tryb dostawy (synchroniczny/asynchroniczny, w czasie rzeczywistym, tryb wsadowy itp.) uzgodniony w fazie negocjacji. Po dostarczeniu usługi konsumentowi, za nią następuje zapłata. Konsument zaczyna wówczas korzystać z usługi. Ważną częścią fazy zużycia jest monitorowanie wydajności przy użyciu zautomatyzowanych narzędzi. W tej fazie konsument będzie potrzebował narzędzi umożliwiających monitorowanie jakości i zakończenie świadczenia usług, jeśli zajdzie taka potrzeba. Będzie to obejmować alerty dla ludzi lub automatyczne zakończenie w oparciu o polityki zdefiniowane przy użyciu ontologii związanych z jakością, które należy opracować. Podfaza monitorowania usług mierzy jakość usługi i porównuje ją z poziomami jakości określonymi w umowie SLA. Ta faza obejmuje zarówno obszar konsumencki, jak i chmurowy, ponieważ monitorowanie wydajności jest wspólną odpowiedzialnością. Jeżeli konsument nie jest zadowolony z jakości usługi, powinien mieć możliwość jej zakończenia i zaprzestania płatności za usługę. Jeśli usługa zostanie zakończona, konsument będzie musiał ponownie uruchomić cykl życia usługi poprzez

ponowne zdefiniowanie wymagań i wydanie RFS. Narzędzie do monitorowania wydajności używane w SOAR pokazano na rysunku.

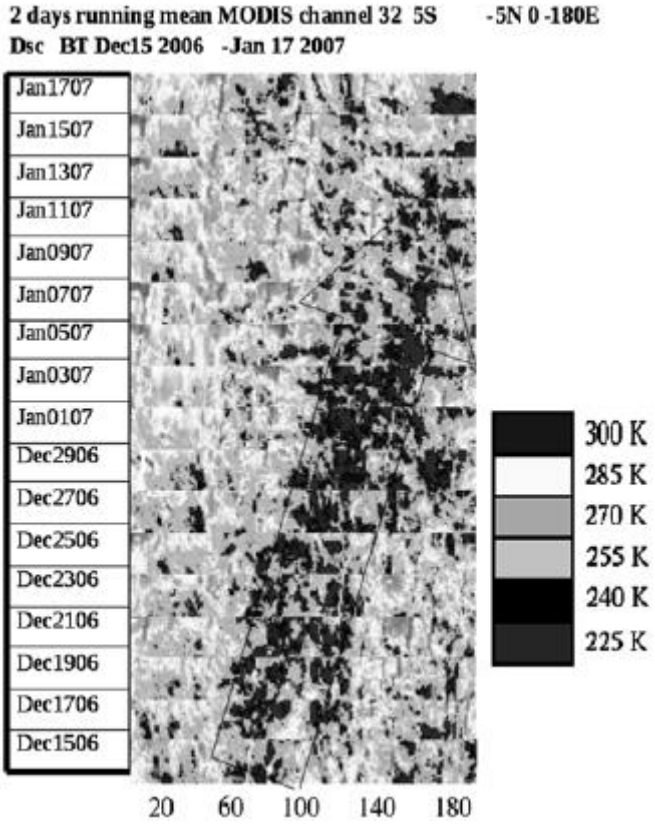


Narzędzie nie strasznie skomplikowane, ale bardzo skuteczne. Pokazuje nie tylko wykonane zadania, ale także te, które są w trakcie realizacji. Narzędzie zapewnia sygnaturę czasową do przesłania, dzięki czemu można łatwo monitorować, jak długo trwa żądanie. Użytkownik może wybrać zadanie i usunąć je, nawet jeśli jest w toku. Dzięki temu użytkownik może anulować zadania, które zajmują zbyt dużo czasu. Anulowanie zadań jest również dostępne za pośrednictwem protokołu SOAP i może być używane zarówno przez interfejs maszynowy, jak i interfejs użytkownika.

Przykład siatkowego obrazu temperatury jasności pokazano na rysunku



To przykład prostego wyniku wygenerowanego z systemu SOAR. Obraz temperatury jasności pochodzi z sondy atmosferycznej na podczerwień (AIRS) nad Oceanem Indyjskim w dniu 1 stycznia 2005 r. Wyświetlane paski wynikają z orbity biegunowej synchronicznej ze słońcem i zostałyby rozwiązane, gdyby żądanie danych klienta było dłuższe Kropka. Najcieplejsze regiony znajdują się w Afryce Południowej i Etiopii. Niebieskie kropki wokół Madagaskaru, Nowej Gwinei i innych miejsc są spowodowane chmurami. Tego typu wykresy, które bardzo wyraźnie pokazują chmury, można wykorzystać na diagramach Hovmollera (średnia ruchoma) do śledzenia ruchu chmur w czasie i monitorowania procesów, takich jak oscylacja Maddena Juliana, jak pokazano.



Podsumowanie/wnioski

Podsumowując, uważamy, że przetwarzanie w chmurze, a w szczególności paradygmat programowania równoległego MapReduce i jego uogólnienia, oferują nowe i ekscytujące podejście do obliczeń naukowych. Paradygmaty obliczeniowe i usługowe na pierwszy rzut oka mogą wydawać się sprzeczne z intuicją. Jednak po opanowaniu uwalniają korzyści, jakie zapewnia chmura w zakresie elastycznych zasobów obliczeniowych, zwirtualizowanych usług internetowych, abstrakcji oprogramowania i odporności na awarie. MapReduce i Dryad to paradygmaty, które mogą pomóc w przetwarzaniu danych naukowych na zapleczu, podczas gdy wykrywanie i dostarczanie usług zapewnia nie tylko interfejs, ale cały przepływ pracy obliczeniowej, w tym wybór danych i jakość usług. SOAR jest przykładem kompletnej naukowej aplikacji chmurowej i mamy nadzieję, że lekcje, które wyciągnęliśmy z tego i innych systemów, mogą pomóc innym w ich naukowych przedsięwzięciach