Java Cluster - implementacja, zastosowanie oraz zagadnienia wydajnościowe

Piotr Pawłowski
Instytut Informatyki
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska
e-mail: P.S.Pawlowski@elka.pw.edu.pl

***Streszczenie*—Artukuł podzielony jest na trzy zasadnicze części, których celem jest kompleksowe opisanie rozwiązania jakim jest Java Cluster. Pierwsza z nich dotyczy implementacji samego rozwiązania i stanowi wstęp, kóry pozwala na lepsze zrozumienie materiału znajdującego się w dalszych częściach. Znajduje się też tam opis wielu poprawek dotyczących koncepcji samego problemu w stosunku do projektu pierwotnego. Kolejną zasadniczą częścią którą omawia artukuł są kwestie zwiazane z zastosowaniem stworzonego rozwiązania. Skupiono się tutaj głównie na tych problemach obliczeniowych dla których Java Cluster powinien dać dobre wyniki wydajnościowe. Omówiona jest też charakterystyka problemów, dla których proponowane rozwiązanie się nie sprawdza. Na bazie analizy tych założeń stworzona jest część trzecia, która omawia problemy wydajnościowe, zasygnalizowane zostaną też możliwosci potencjalnych poprawek, których autor w tej wersji rozwiązania nie zrealizował.** [1]

I. Wstęp

Java Cluster powstaje jako rozwiązanie mające na celu ułatwienie obliczeń równoległych. W wielu dziedzinach nauki i techniki coraz cześciej spotykamy się z dużymi, bądź wręcz wielkimi problemami obliczeniowymi. Coraz większa ilości danych do przetworzenia, sprawia że musimy częściowo zmienić swoje podejście do wielu zagadnień algorytmicznych. Jednocześnie pojawianie się wielordzeniowych procesorów powoduje że algorytmy sekwencyjne wcześniej czy później będą przepisywane na wersje równoległe, tak aby zapewnić maksymalne wykorzystanie procesora.Wynika z tego że wiele problemów obliczeniowych będzie przystosowanych do architektur równoległych. Przystosowanie to zaś nie jest trywialne, bowiem w wielu przypadkach wymaga zmiany podejścia do problemu, innej organizacji struktur danych i samych algorytmów.

Java Cluster jest rozwiązaniem, które ma pozwolić na tworzenie algorytmów obliczeniowych w taki sposób jak byśmy pisali je na procesory wielordzeniowe, z tą rożnicą że całe obliczenia będą wykonane w ramach odzielnych maszyn składających się na klaster. Jednym z głowych założeń podczas projektowania było stworzenie mechanizmu pozwalającego na łatwe pisanie kodu aplikacja, mechanizmu, który pozwoli się skupić programiście głównie na koncepcji samego rozwiązania oraz implementacji w przyjemnym i bezpiecznym środowisku programistycznym. Zupełnie niezasadne wydaje się że w wielu rozwiązaniach tego typu programista musi jawnie dbać o komunikację, pomiędzy procesorami (tak umównie nazwijmy maszyny liczące nasze zadanie obliczeniowe). Taki podejście nie tylko powoduje spore zaśmiecanie kodu zbędnymi wywołaniami ale naraża nas na wprowadzanie błędów. Jednocześnie pełna automatyzacja i zabranie użytkownikowi możliwości ingerencji w pewne zagadnienia techniczne też nie jest dobrym pomysłem, głównie dlatego że często odbija się to na wydajności rozwiązania, a ta w tym przypadku jest rzeczą kluczową. W związku z tym podczas tworzenia Java Cluster skupiono się głównie na tym aby stworzyć rozwiązanie, które będzie kompromisem jeżeli chodzi o powyższe parametry. Zrezygnowano z konieczności podawania wprost gdzie i kiedy ma zachodzić komunikacja, jednak pozostawiono możliwość definiowania co ma być synchronizowane oraz co ma być migrowane na inny węzeł. Aby osiągnąć powyżej opisane cele konieczna była modyfikacja JDK na poziomie jądra maszyny wirtualnej oraz bibliotek standardowych, co odpowiada drugiej i trzeciej warstwie od dołu na rysunku numer jeden.

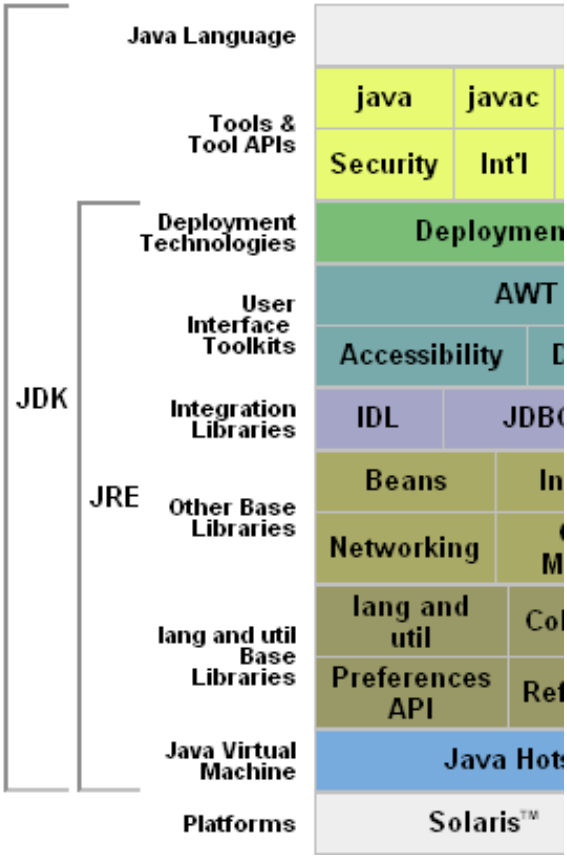
Java Cluster jest rozwiązaniem, które powstało w oparciu o Java 6 SE. Wybór ten był motywowany tym iż po pierwsze istniała potrzeba stworzenia takiego rozwiązania dla języka Java, które będzie dedykowane do obliczeń równoległych i mniej ogólne niż Java RMI, oraz tym że Sun zdecydował się na publikację kodu źródłowego całej platformy na licencji JRL (Java Research License).

II. Java Cluster - projekt i implementacja

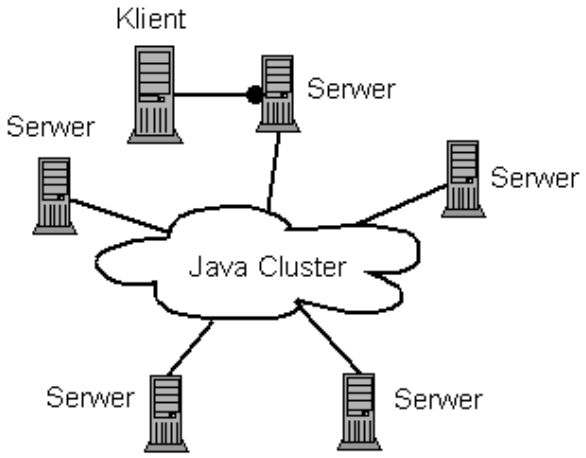
Architektura

Java Cluster jest rozwiązaniem, które pracuje w architekturze klient - serwer. Klientem jest maszyna, która inicjalizuje obliczenia, serwerem zaś maszyna, która te obliczenia będzie realizowała.Serwery tworzą strukturę grafu,

[1]Praca ta powstała w ramach seminarium dyplomowego na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2008 r.



Rysunek 1. JDK 6 - architektura systemu



Rysunek 2. Architektura

klient musi umieć połączyć się tylko z jednym serwerem aby poznać strukturę reszty sieci.

Serwery można w tym rozwiązaniu porównać do zdalnych procesorów zdalnych i to określenie będzie używane w dalszej części opracowania. Nie wszystkie obliczenia klienta muszą być migrowane, to programista decyduje co ma się wykonać na zdalnym procesorze a co na lokalnym. Migracja odbywa się poprzez umieszczenie kodu algorytmu w ciele głównej metody wątku. Wątek jest podstawowym bytem szeregowanym przez maszyne wirtualną, pozwalającym na współbieżność w przypadku wykonania algorytmu na jednym procesorze oraz równoległość w momencie kiedy pracujemy w środowisku klastra. Wątki, które mają podlegać migracji muszą dziedziczyć po klasie RemoteThread. Architektura klient - serwer jest intuicyjna i porządkuje rozwiązanie pod względem wykonywanych czynności. Java Cluster jedak dzili się jeszcze na JC-API (Java Cluster - API) i JC-NODE (Java Cluster - NODE),dwie dodatkowe warstwy, które wynikają z implementacji samego rozwiązanie.Można powiedzięć że podział na klient - serwer jest pionowy, zaś na JC-NODE i JC-API poziomy. Konsekwencją tego jest to że zarówno klient jak i serwer posiadają te dwie warstwy i z nich korzystają.

JC-API (Java Cluster - API)

JC-API jest to zbiór słów kluczowych oraz klas języka Java, które są udostępnione programiscie aby mógł on w sposób świadomy tworzyć swoje algorytmy. Kluczowym założeniem przy tworzeniu JC-API było to aby do maksimum wykorzystać już istniejące w języku Java rozwiązania i co ważne zachować ich znaczenie. Dzięki takiemu podejściu możliwe jest też dużo łatwiejsze przeprojektowywanie algorytmów, które są już napisane w Javie pod nowe rozwiązanie.Kluczowymi elementami języka, które składają się na JC-API są:

- **synchronized**
- **transient**

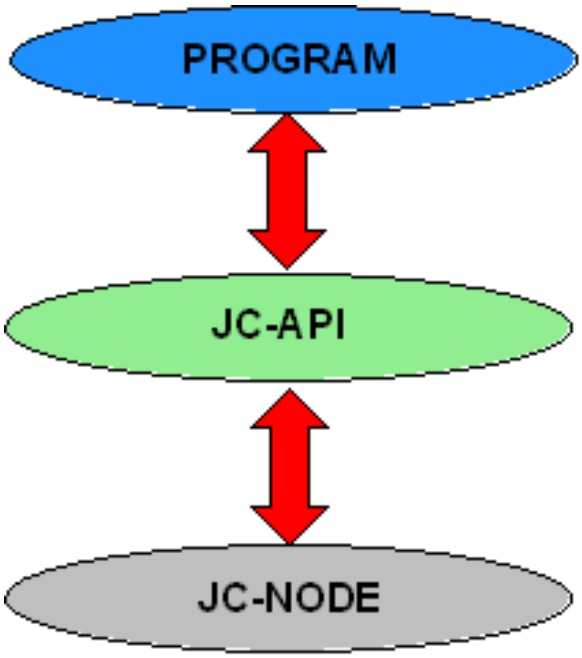
synchronized jest słowem kluczowym, którego użycie zapewnia synchornizację wątków w ramach całego klastra. Tak jak w przypadku wersji standardowej maszynu wirtualnej gdzie synchronizowane były lokalne wątki, sekcja krytyczna jest pozyskiwana w kontekście jakiegoś obiektu, monitora. Sekcja krytyczna jest też mechanizmem, który zapewnia nam spójność danych na których pracujemy, czyli przy założeniu że algorytm jest poprawnie napisany pracujemy zawsze na aktualnych danych. Aby zrozumieć dlaczego tak się dzieje należy się wgłębić w to w jakis sposób Java Cluster operuje na danych. Wiadomo bowiem że w środowisku klastrowym jeżeli chcemy pracować na wspólnych danych to odwoływanie się do jednej wspólnej kopii jest pomysłem dość kiepiskim. Jest tak tylko i wyłącznie ze względu na zagadnienia wydajnościowe, wiadomo bowiem że odowłanie do loklanej pamięci jest dużo szybsze niż do pamięci, która jest na innym komputerze. Z tego powodu zdecydowano się na replikację danych podczas migracji. Założono że każdy zdalny wątek pracuje na swojej kopii danych i ta kopia zawsze jest aktualna. Aktualność tej kopii zapewnia właśnie sekcja krytyczna, która dba aby w momencie kiedy wchodzimy do niej zaciągnąć aktualne wersje danych (w tym też obiektu na którym się synchronizujemy) oraz aby podczas wychodzenia z owej sekcji wysłać zaktualizowane dane.Każdy obiekt posiada swój identyfikator, który jest nadawany przez

klienta, aktualizacja danych zachodzi tylko w przypadku kiedy wersją obietku sciągniętego jest większa niż wersja obiektu bieżącego. Operacja ta jest realizowana na poziomie deserializacji. Wynika z tego że wymiana danych w obrębie jednego wątku zachodzi dwa razy tyle ile jest sekcji krytycznych. Kolejnym słowem kluczowym jest **transient**, które pozwala na zadeklarowanie czy dany atrybut klasy ma być serializowany czy nie. Serializacja jak wiadomo jest procesem zamiany jakiegoś zbioru danych, obiektów w taką formę z której można je odtworzyć i przywrócić stan obiektu. Nie zawsze jednak pisząc nasz algorytm chcemy aby wszystkie zmienne były serializowane i siłą rzeczy migrowane na inny procesor. Dzięki temu mechanizmowi można w znaczący sposób przyśpieszyć wymianę danych. Kolejnym ważnym elementem JC-API jest zestaw klas, które pozwalają na tworzenie aplikacji równoległych.Z punktu widzenia programisty jedyną godną uwagi klasą jest `RemoteThread`. Klasa ta reprezentuje migrowany wątek i tylko wątki, których klasy dziedziczą z tej klasy są migrowane. Aby wydajnie i świadomie konstruować algorytmy równoległe istotna jest dla nas informacja na temat tego na ile procesorów możemy wysłać nasze obliczenia. Informacja ta musi być podawana w sposób dynamiczny i bazować na aktualnym stanie sieci. Aby zapewnić tą funkcjonalność do klasy java.lang.System dodano metodę getNodeCount(). Wywołanie tej metody powoduje synchroniczne odpytanie serwera/procesora o to z iloma innymi węzłami ma połączenie, wartość zwracana jest to własnie ta liczba plus jeden. Dodatkową informacją, którą możemy uzyskać jest węzeł/procesor. Dostęp do tego parametru uzyskujemy dzięki metodzie System.getNode(int nodeNumber). Wynikiem tej metody jest identyfikator węzła, kóry może nam posłużyć jako parametr klasy RemoteThread, co będzie oznaczało że wątek zostanie zmigrowany właśnie do tego węzła. W przypadku niepowodzenia migracji, całe obliczenia rozpoczynają się od początku. Ważne jest zauważenie tutaj że migracja wątków może odbywać się według dwóch strategii, pierwsza z nich polega na losowym romieszczeniu wątków na różnych procesorach, a to na jakie procesory obliczenia trafią nie jest istotne z punktu widzenia algorytmu. Druga zaś metoda pozwala na dokładne wskazanie procesora na którym dane obliczenia będą wykonywane. Informacja o węźle oprócz lokalizacji węzła zawiera informację o jego średnim obciążeniu. Pozwala to twórcy algorytmu na stosowanie różnych współczynników i finalnie profilowanie swojego algorytmu pod kontem systemu.

JC-NODE (Java Cluster - NODE)

JC-NODE jest to zestaw klas, które składają się na konstrukcję serwera oraz klienta. Jest to warstwa, która znajduje się poniżej JC-API. To właśnie wykorzystując JC-API wykonywany program komunikuje się z JC-NODE.

Podstawowym elementem JC-NODE po stronie klienta jest ThreadClient. Jest to element, który łączy się ze zdalnym procesorem, czyli naszym serwerem i rozpoczyna



Rysunek 3. Java Cluster - komunikacja

migrację.Każdy klient ma uniklany identyfikator w sieci.W momencie kiedy migracja jest możliwa przesyłany jest obiekt wątku i uruchamiany na zdalnym procesorze. Drugim niezwykle istotnym mechanizmem jeżeli chodzi o część kliencką jest serwer sekcji krytycznej. Warto zauważyć że jeżeli chodzi o synchronizację to właśnie klient, węzeł zlecający jest serwerem. A więc każde wejście do sekcji krytycznej powoduje odwołanie do serwera sekcji krytycznej. Serwer ten pozwala na zachowanie wielowejściowości, tzn. jeżeli dany wątek posiada sekcje krytyczną i chce do niej wejść to mu się to uda (w przypadku zwykłego mutexu doszło by do zakleszczenia).Część kliencka posiada zawsze najaktualniejsze dane i to z niej są one ściągane przez procesor zdalny w momencie wchodzenia do sekcji krytycznej oraz wgrywane w momencie wychodzenia z sekcji krytycznej.W momencie tworzenia nowego obiektu jest nadawany mu jednoznaczny identyfikator będący kolejnym numerem z sekwencji.Numer ten jest wspólny dla całej instancji maszyny wirtualnej po stronie klienckiej i jest nadawany w czasie wywoływania operatora tworzącego instancję obiektu.

Strona serwerowa JC-NODE jest to w zasadzie sieć powiązanych ze sobą procesorów, reprezentowanych przez klasę ThreadServer. Założeniem tej sieci jest to że każdy serwer zna lokację oraz średnie obciązenie wszystkich swoich sąsiadów. Stan sieci jest aktualizowany w czasie pracy w sposób dynamiczny, co powoduje że łacząc się do dowolnego serwera jesteśmy w stanie obserować jej obciążenie oraz działające wątki.Serwer jest jednostką autonomiczną, do jego działania nie jest

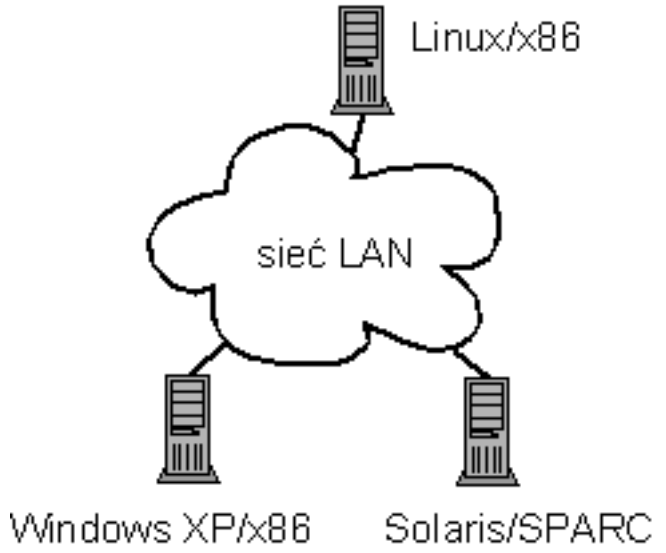
potrzebny żaden klient, serwer też nie zleca sobie żadnych zadań. Serwer pełni funkcję usługową względem klienta i stara się aby zadanie mu powierzone było wykonane w sposób optymalny. Optymalność oznacza że serwer nie podejmie się wykonywania żadnych obliczeń jeżeli w jego otoczeniu są serwery mniej obciązone. Obciążenie serwera jest liczone jako suma czasów wykonywanych wątków przez przez ilość procesorów. Im mniejszy współczynnik tym serwer jest mniej obciążony. Miara ta jest stosunkow prosta, możliwe jest że w przyszłości zostanie uzupełniona. W przypadku nie podjęcia się obliczeń, są one delegowane do najmniej obicążonego w danej chwili serwera. Delegacja jest procesem jednorazowym. Obsługa sytuacji wyjątkowych jeżeli chodzi o JC-NODE nie jest w tej wersji systemu specjalnie zawansowana. W przypadku gdy od systemu odłączy się klient i po pewnym czasie nie nawiąże z nami połączenia wszystkie obliczenia zostają zaprzestane. Jest to zrozumiałe gdyż skoro nie istnieje zleceniodawca dla którego realizujemy dane zadanie, to jak jest sens kontynuowania go. W przypadku gdy podczas obliczeń okaże się że nie ma komunikacji z serwerem wszystkie obliczenia są ponawiane.

Aby przesyłać dane pomiędzy klientem a serwerem konieczne jest zapisanie ich w zrozumiałym dla nich formacie. Proces ten nazywa się serializacją i pozwala na odtworzenie stanu obiektu z chwili zapisania, zserializowania go.W przypadku rozwiązania Java Cluster zdecydowano się na skorzystanie ze standardowego mechanizmu serializacji klas do wartości binarnych.Wybrano takie podejście ponieważ serializacja binarna jest stosunkowo szybka a co ważne jest silnie zintegorwana z samą maszyną. Wymusza to tylko jedną niedogodność dla programisty, jeżeli nie oznacza obiektów jako transient to muszą one implementować interfejs Serializable.

III. Java Cluster - zastosowanie

Java Cluster jest rozwiązaniem, które zostało stworzone z myślą o obliczeniach równoległych. Obliczenia równoległe mają zastosowanie w wielu dziedzinach nauki i techniki. Potrzeba zrównolegalania niektórych algorytmów jest związana z chęcią a niekiedy koniecznością przyśpieszenia obliczeń. Przykładowo proces wyliczania, analizy danych statystycznych do prognozowania pogody na następny dzień nie może zwyczajnie trwać dłużej niż dwadziescia cztery godziny. Oczywiście należy pamiętać że samo uruchomienie algorytmu w środowisku klastrowym nie oznacza gwałtownego przyśpieszenia, co więcej może spowodować że pojawią się spore opóźnienia. Warto zauważyć że zgodnie z prawem Amdahla jeżeli nasz algorytm posiada znaczącą ilość miejsc, których zrównoleglić się nie da, to samo dokładanie nowych procesorów nie spowoduje aż tak wielkiego wzrostu wydajności, gdyż te własnie miejsca będą wąskim gardłem. Przykładowo gdy jakaś część obliczeń, które są zrównoleglone zajmuje $t \times 100\%$, zostanie przyśpieszona n krotnie, to cały proces zostanie przyśpieszony tylko o $\frac{1}{(1-t)+\frac{t}{n}}$ razy. Aby zobrazować co z tego prawa wynika warto jest rozwiązać przykładowe zadanie. Załóżmy że mamy algorytm w którym 90% operacji da się zrównoleglić, zaś 10Przyśpieszmy teraz te operacje dziesięciokrotenie, dokładajać nowych 10 procesorów. W tym przypadku cały program przyspieszy dokładnie $\frac{1}{(1-0.9)+\frac{0.9}{10}} = \frac{1}{0.19} = 5.26$ razy. Gdy będziemy próbowali dołożyć sto procesorów przyśpieszenie wyniesie 9.17 raza.Wynika z tego jednoznacznie że jeżeli chcemy mysleć o obliczeniach równoległych to musimy znienić konstrukcję naszych algorytmów, samo dokładanie nowych jednostek obliczeniowych nie spowoduje gwałtownego wzrostu wydajności.Kluczowym słowem jest tutaj dekompozycja problemu, czyli takie przekształcenie algorytmu aby można było go podzielić na duże niezależne od siebie elementy. Dobrym przykładem, co prawda dość trywialnym jest dodawanie macierzy. Warto zauważyć że sam problem w zasadzie jest zdekomponowany od poczatku, bowiem polaga na tym iż wynikowa macierze jest tworzona poprzez wykonywanie niezależnych względem siebie dodawań. Taką operację można przykładowo zdekomponować względem wierszy, bądź grup wierszy. Jeżeli macierz podzielimy na pół i do każdego z dwóch porcesorów wyślemy odpowiednią połówkę, to przyśpieszenie obliczeń powinno wynosić w przybliżeniu sto procent. Wszystkie ewentualne straty będa wynikały z kosztów komunikacji między maszynami. Problem ten jest wręcz idelany jeżeli chodzi o obliczenia, bowiem nie ma żadnej synchronizacji obliczeń, co więcej dane są przesyłane tylko na początku i podczas zbierania wyników. Niestety w przypadku bardziej zaawansowanych algorytmów, problem nie jest aż tak trywialny i sama dekompozycja jest dość skomplikowanym zadaniem.

Obliczenia równoległe są dziedziną informatyki, która jest w dzisiejszych czasach dość rozwinięta, istnieje wiele centrów naukowych, które się w nich specjalizują. Obowiązującym standardem bibliotek do obliczeń równoległych w dzisiejszych czasach wydaje się być MPI (Message Passing Interface). Jest to rozwiązanie zaawansowane i popularne w środowisku naukowym. Java Cluster stara się stworzyć środowisko pracy pozwalające realizować podobne zadanie jednak bez wnikania w zbędne szczegóły, co więcej z założenia uruchomienie i konfiguracja klastra ma być błyskawiczna i wręcz natychmiastowa. To co jest istotną zaletą rozwiązania jest to że w gruncie rzeczy jesteśmy w dużej mierze niezależni od architektury sprzętu na którym uruchamiamy naszą aplikację równoległą. Może to być maszyna z procesorem Intela i systemem Windows XP ale też nic nie stoi na przeszkodzie żeby był to Sun Solaris z procesorami SPARC. Co więcej wszystkie te architektury mogą być połączone w jeden klaster i



Rysunek 4. Java Cluster - środowisko pracy

współpracować ze sobą. Uzycie określenie klaster nie jest przypadkowe i nie jest też błędne. O klastrach zwykło się mówić żę są to systemy homogeniczne, zaś gdzie w przypadku Intelowego komputera z Windows XP i Solarisa można mówić o klastrze, bardziej prezentuje się to jako grid obliczeniowy. Należy jednak zwrócić uwagę że klaster tworzy tutaj środowisko Java które przykrywa wiele różnic między tymi systemami. Przestają nas interesować kwestie dotyczące architektury systemu i środowiska na jakim on pracuje, ważne się staje tylko to że jest to aplikacja Javowa. Dzięki temu możemy wykorzystać każdą maszynę, którą mamy pod ręką do obliczeń równoległych. Jedyne co nas interesuje to poprawne skonstruowanie algorytmu.

Java Cluster jest rozwiązaniem dedykowanym do obliczeń równoległych, uruchamianie na platformie równoległej algorytmów, które wymagają bardzo częstej synchronizacji, która stanowi wąskie gardło nie jest wskazane przy tym rozwiązaniu. W przypadku takich problemów o wiele lepszym pomysłem jest napisanie algorytmu sekwencyjnego niż myślenie o równoległości. Należy pamiętać że sam proces synchronizacji jest dość kosztowny jeżeli chodzi o wydajność, co więcej powoduje że wszystkie procesory mogą czekać i w danym momencie wykonuje się tylko jedno zadanie. Tak jak to wynika z prawa Amdahla, nawet dokładając więcej maszyn, samo przyśpieszenie może być mało znaczące. Co więcej prawo Amdahla pomija koszty jakie są ponoszone przy komunikacji. Koszty te przy bardzo dużych zadaniach, gdzie musimy przesyłać spore ilościu danych mogą być znaczące. Należy więc najpierws skrupulatnie przeliczyć i przeanalizować algorytm pod kontem elementów blokujących zanim zadecydujemy się na realizację go w środowisku równoległym, takim jak Java Cluster.

Ważnym elementem w przypadku konstruowania algorytmu jest też zastanowienie się nad tym w jaki sposób pobieramy dane do obliczeń. Bardzo częstą praktyką progrmiastów, która jest jak najbardziej godna pochwały jest otwieranie strumienia danych i zaczytywanie z niego kolejnych wartości. O ile takie podejście jest zasadne w przypadku pracy na loklanej maszynie, gdzie odczyt ze strumienia jest stosunkowo szybki, to w przypadku środowiska równoległego, należy pamiętać że odzczyt z tego strumienia może być wolny, gdyż źródło z którego czytamy nie znajduje się na loklanej maszynie. Problem zaczyna się robić w momencie kiedy istnieje proces, który pisze do tego strumienia z którego my czytamy. Wtedy oczywiście procesy czytające, muszą czekać na zakończenie zapisu, jeżeli chcemy liczyć na to że nasz algorytm da poprawne wyniki. Powowduje to że w pewien ukryty, bądź nie, sposób musimy się synchronizować na zasobie jakim jest strumień. Zdecydowanie nie jest to najlepszy pomysł, gdyż może wprowadzić znaczne opóźnienia. Aby rozwiązać ten problem dobrym pomysłem jest buforowanie danych do przeliczenia. Przykładowo jeżeli mamy do przeliczenia ogromny, zbór danych, które nie mają szans zmieścić się w pamięci, to wtedy dzielimy dane na te segmenty, które w pamięci mogą się zmieścić. Te właśnie segmenty stanowią podstawę do obliczeń na zdalnym procesorze. Co więcej w momencie kiedy dane te zostaną wysłane możemy pamięć przez nie zaalokowaną wykorzystać do zaciągnięcia danych potrzebnych dla następnego procesora. Podobny schemat ma miejsce w momencie zbierania wyników. Jedyna wada tego rozwiązania to sekwencyjne wysyłanie danych do liczenia i delegowanie zadań. Nie jest to jednak poważny problem bowiem jeżeli zadanie liczone na zdalnym procesorze jest dużo dłuższe niż sama komunikacja, co jest warunkiem koniecznym aby stosować Java Cluster, to wtedy z dużym prawdopodobieństwem wyniki zaczną wracać w tej samej kolejności co były wysyłane (czyli procesor, który pierwszy otrzymał dane skończy wcześniej) i straty wydajnościowe nie powinny być dużę. Oczywiście jeżeli pierwszy procesor jest dziesięć razy wolniejszy niż drugi to prawidłowość nie zajdzie w takim stopniu w jakim została opisana powyżej. Nie jest to jednak problem kluczowy bowiem nawet w pesymistycznym przypadku samo przesyłanie powinno stanowić stosunkowo mały procent czasu potrzebnego na obliczenie problemu.

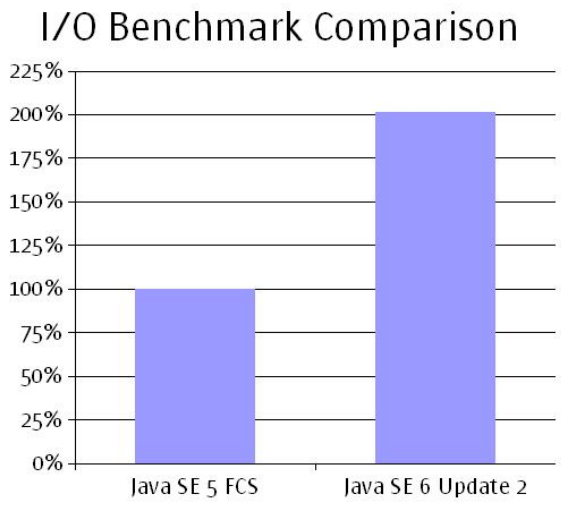
IV. Java Cluster - wydajność rozwiązania

Aby rozpocząć rozważania dotyczące wydajnosći rozwiązania należy najpierw omówić kwestie wydajnościowe związane z samym środowiskiem w jakim system został stworzony.Powszechnie panującym przekonaniem wśród programistów, projektantów systemów informatycznych jest to że Java jako środowisko jest wolna. Opinia ta wywodzi się w głównej mierze z pierwszych lat istnienia rozwiązania Suna na rynku informatycznym. Nikt nie zamierza się sprzeczać z tym że pierwsze wersje, głównie 1.0,

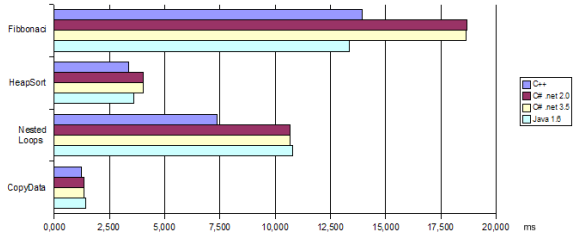
1.1 nie należały do demonów szybkości. Główną przyczyną tego stanu rzeczy było zastosowanie interpretacji jako głównego mechanizmu przetwarzania kodu. Oczywistym jest że intepretacja sama w sobie jest mało wydajna, gdyż obciąża procesor w zbytnim stopniu niż to jest potrzebne, głównie dlatego że konieczne jest ciągłe tłumaczenie kawałków kodu na istrukcje procesora nawet jeżeli te instrukcje były już wcześniej wykonywane. Sam proces tłuaczenia też może być zrealizowany na wiele sposobów. W dzisiejszej wersji maszyny wirtualnej stosowany jest tzw. just in time compiler. Kod programu napisany w javie jest kompilowany do wersji pośredniej zwanej bajtkodem. Sam bajtkod jest to kod natywny javy. Można go porównać do kodu maszynowego, jednak należy pamiętać że zawiera on wiele dość wysokopoziomowych instrukcji, odwołań. Bajtkod jest tym co przetwarza maszyna wirtualna, przy czym samo przetwarzanie działa w dość specyficzny sposób, bowiem konkretne instrukcje bajtkodu, są tłumaczone na natywny kod procesora. Tłumaczenie odbywa się dokładnie raz i w momencie ponownego odwołania do konkretnej instrukcji bajtkodu, jest automatycznie brana jej natywna reprezentacja.Dzięki temu sam proces przetwarzania programu jest o wiele szybszy, szczgólnie że z biegiem działania aplikacji różnice między programem napisanym w języku niżeszgo poziomu, jakim jest C/C++ się zmniejszają. Kolejnym zaskoczeniem jeżeli chodzi o samą maszynę wirtualną jest to że alokacja pamięci nie koniecznie musi być wolniejsza niż w przypadku wywołań systemowych. Oczywiście jednokrotne wywołanie alokacji jest wolniejsze, tutaj nie ma wątpliwości. Jeżeli weźmie się jednak pod uwagę program, który działa przez pewien czas to statystycznie alokacja i zwalnianie pamięci potrafi być nawet szybsza niż przy konwencjonalnych rozwiązaniach, nie mówiąc już o tym że o wiele bezpieczniejsza. To co jest faktycznie wolniejsze w przypadku Javy, to operacje wejścia - wyjścia, według wyliczeń potrafią być one do trzech, czterech razy wolniejsze. Jednak operacje te przyśpieszają z wersji na wersję.

Nie należy też ukrywać że operacje graficzne, jako w pewnym sensie wariant I/O są też wolniejsze, czego przykładem może być Java Swing (mimo iż w ostatnich latach dzięki usilnym staraniom developerów Suna, przyśpieszyła ona wielokrortnie, głownie dzięki silnemu wsparciu ze strony maszyny wirtualnej).

W przypadku obliczeń równoległych wydajność ma bardzo duże znaczenie. Należy jednak potraktować kwestie wydajnościowe w trochę inny sposób. Szybkość z jaką są wykonywane dodawania i mnożenia jest istotna, jednak nie jest moim zdaniem kwestia najważniejsza, szczególnie że w dzisiejszych czasach różnice między natywnymi platformami a maszynami wirtualnymi się zacierają. Najistotniejsze jest to aby dokonać takiego rozkładu problemu, żeby obliczenia miały szanse wykobywać się równolegle w prawdziwym tego słowa znaczeniu. Java Cluster nie jest jeszcze



Rysunek 5. Porównanie I/O dla Java 1.5 i Java 1.6



Rysunek 6. Języki programowania - porównanie wydajnościowe

rozwiązainem przetestowanym, wiele eksperymentów musi jeszcze być wykonanych aby można było dokonać pełnego porównania z innymi istniejącymi rozwiązaniami. Jednak to czego można się spodziewać nie powinno znacząco odbiegać od istniejących rozwiązań. Oczywistym jest że w stosunku do platform natywnych dla danej architektury będą pewne różnice wydajnościowe, głównie spowodowane przez samą architekturę maszyny wirtualnej. Jednak tak jak w każdym rozwiązaniu technologicznym istnieje kwestia kompromisu. Nawet jeżeli tracimy trochę na wydajności, która to strata będzie głównie dotyczyłą operacji wykonywanch sporadycznie, to zyskujemy na przenośnoći i bezpieczeństwie kodu, który tworzymy.

Jeżeli weźmiemy pod uwagę że Sun pracuje teraz nad wersja siedem maszyny wirtualnej, która będzie wykorzytywała wielordzeniowe procesory, to możemy liczyć na poważne przyspieszenie obliczeń na architekturach klastrowych wykorzytujące te jednostki.

V. Podsumowanie

Java Cluster to bardzo młode rozwiązanie, nie będące jeszcze w pełni ukończone. Jest to pewnego rodzaju *p*roff of concept na bazie kórego może uda się kiedyś wytworzyć coś co będzie rozwiązaniem docelowym.Kilka rzeczy będzie jeszcze wymagało dopracowania, co wiadomo już dziś. Jedną z podstawowych bolączek, zakładajac że

środowisko pracy, w kórym obliczenia będą wykonywane nie jest stabilne, jest to że odłączenie się dowolnego z węzłów, na którym jest wykonywana część obliczeń, powoduje anulowanie pracy reszty węzłów i rozpoczęcie wykonywania algorytmu od początku. Aby tego uniknąć trzeba by zrealizować coś w stylu dzinnika logów, który będzie pozwalał na wrócenie się do tego momentu obliczeń gdzie wszystko jeszcze przebiegało dobrze i rozpoczęcie ich na innej maszynie. Takie podejście jest jednak pracochłonne i warto się zastanowić kiedy ma ono sens. Moim zdaniem w środowiskach w których to rozwiązanie będzie wykorzytywane, czyli sieć LAN, stan sieci i maszyn jest stosunkowo stabilny i potencjalne awarie powinny się zdarzać sporadycznie.

Kolejną optymalizacją nad którą warto pomyśleć jest przesyłanie tylko zmienionych danych. W obecnym rozwiązaniu gdy wchodzimy do sekcji krytycznej sciągamy wszystkie dane, w momencie wyjścia wysyłamy wszystkie. O wiele wydajniej by było gdyby wysyłać tylko to co się zmieniło.Oczywiście przy założeniu że migracja danych powinna zachodzić stosunkowo rzadko, w stosunku do wszystkich obliczeń będących ciałem algorytmu nie jest to wielki problem. Rozwiązanie takie jest możliwe do realizacji ponieważ już w obecnej wersji systemu każdy obiekt musi być jednoznacznie identyfikowany w kontekście klienta, głównie dla celów synchronizacyjnych oraz dlatego aby istaniała możliwość stwierdzenia, który obiekt jest starszy a który nowszy. W związku z tym istnieje możliwość wykrycia jakie dane się zmieniły podczas wykonywania algorytmu i przesłania tylko ich.

Kolejnym krokiem jaki musi zostać podjęty aby jednoznacznie stwierdzić czy rozwiązanie to spełnia wszystkie postawione przed nim założenia, które były postulowane, są testy i praktyczne wykorzystanie tego mechanizmu. Na dzień dziśiejszy przewidywane testy mają tylko sprawdzić poprawność implementacji oraz pewnych założeń techniczych. W następnej kolejności przewidziane jest badanie wydajnosci i dokonywanie optymalizacji.

Literatura

[1] A. S. Tanenbaum and M. van Steen, *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

[2] T. Lindholm and F. Yellin, *Java Virtual Machine Specification*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] W. Zhu, C.-L. Wang, and F. C. M. Lau, "Jessica2: A distributed java virtual machine with transparent thread migration support," in *IEEE Fourth International Conference on Cluster Computing*, Chicago, USA, September 2002. [Online]. Available: citeseer.ist.psu.edu/zhu02jessica.html

[4] T. Sakamoto, T. Sekiguchi, and A. Yonezawa, "Bytecode transformation for portable thread migration in java," in *ASA/MA*, 2000, pp. 16–28. [Online]. Available: citeseer.ist.psu.edu/sakamoto00bytecode.html

[5] S. Bouchenak, "Pickling threads state in the java system," 1999. [Online]. Available: citeseer.ist.psu.edu/bouchenak00pickling.html

[6] E. Truyen, B. Robben, B. Vanhaute, T. Coninx, W. Joosen, and P. Verbaeten, "Portable support for transparent thread migration in java," in *ASA/MA*, 2000, pp. 29–43. [Online]. Available: citeseer.ist.psu.edu/truyen00portable.html