Systemy Rozproszone

Opis zadania domowego - technologie middleware

Zostało zdefiniowanych 8 zadań - 3 aplikacyjne (An) i 5 infrastrukturalnych (Im). Należy zrealizować jedno wybrane zadanie aplikacyjne i jedno (lub więcej) zadanie infrastrukturalne; dodatkowym wymogiem jest by w wybranym zestawie znalazły się zadania (lub ich części) dotyczące (gRPC oraz (Ice lub Thrift)). Każde zadanie ma określoną maksymalną punktację (nominalnie maksimum za całe laboratorium wynosi 15 pkt.).

Zadanie A1 - Subskrypcja na zdarzenia

Wynikiem prac ma być aplikacja klient-serwer w technologii gRPC. Klient powinien móc dokonywać subskrypcji na pewnego rodzaju zdarzenia. To, o czym mają one informować, jest w gestii Wykonawcy, np. o nadchodzącym wydarzeniu lub spotkaniu, którym jesteśmy zainteresowani, o osiągnięciu określonych w żądaniu warunków pogodowych w danym miejscu, itp. Na dane zdarzenie może się naraz zasubskrybować wielu odbiorców i może istnieć wiele niezależnych subskrypcji (tj. np. na wiele różnych instancji spotkań). Należy odpowiednio wykorzystać mechanizm strumieniowania (*stream*). Wiadomości mogą przychodzić z dowolnymi odstępami czasowymi (nawet bardzo długimi), jednak na potrzeby demonstracji rozwiązania należy przyjąć interwał rzędu pojedynczych sekund.

W definicji wiadomości przesyłanych do klienta należy wykorzystać pola *enum*, *string*, ew. *message* - wraz z co najmniej jednym modyfikatorem *repeated*. Etap subskrypcji powinien w jakiś sposób parametryzować otrzymywane wiadomości (np. obejmować wskazanie miasta, którego warunki pogodowe nas interesują.

Dla uproszczenia realizacji zadania można (nie trzeba) pominąć funkcjonalność samego tworzenia instancji wydarzeń lub miejsc, których dotyczy subskrypcja i notyfikacja - może to być zawarte w pliku konfiguracyjnym, a nawet kodzie źródłowym strony serwerowej. Treść wysyłanych zdarzeń może być wynikiem działania bardzo prostego generatora.

W realizacji należy zadbać o odporność komunikacji na błędy sieciowe (które można symulować czasowym wyłączeniem klienta lub serwera lub włączeniem firewalla). Ustanie przerwy w łączności sieciowej musi pozwolić na ponowne ustanowienie komunikacji bez konieczności restartu żadnego z procesów. Wiadomości przeznaczone do dostarczenia powinny być buforowane przez serwer do czasu ponownego ustanowienia łączności. Rozwiązanie musi być także "PAT-friendly" (tj. uwzględniać rozważane na laboratorium sytuacje związane z translacją adresów).

Technologia middleware: gRPC
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 10

Zadanie A2 - Telemetria

Celem zadania jest stworzenie aplikacji klient-serwer na potrzeby telemetrii. Klientem jest aplikacja mająca dostęp do danych pomiarowych z jednego lub więcej czujników np. przepływu wody, zużycia prądu, temperatury itp. (na potrzeby realizacji zadania wystarczy obsługa trzech typów czujników). przypisanych do jednego lub wielu różnych abonentów i udostępniająca je serwerowi. Serwerem jest aplikacja zbierająca te dane (w realizacji zadania może je pomijać logując jedynie na konsolę informując jednak jakich i czyich czujników dotyczą). Może działać wiele Instancji aplikacji klienckich naraz. Dla prostoty rozwiązania można (nie trzeba) założyć, że czujniki różnych typów - a w konsekwencji być może należące do różnych operatorów - mogą być obsługiwane ~~działać na jednym, wspólnym serwerze~~ przez jeden wspólny serwer.
Każdy czujnik ma określony typ i swojego właściciela. Zestaw przesyłanych danych zależy od typu czujnika (choć powinny występować też dane wspólne (rozszerzanie bazowego interfejsu?)). Należy uwzględnić, że po wdrożeniu systemu mogą pojawić się nowe typy czujników, dopuszcza się wówczas wprowadzanie koniecznych modyfikacji systemu (tj. nie należy np. przesyłać wszelakich danych jako ciągu znaków jako sposobu uniknięcia zmian interfejsów).
Trzeba założyć, że ilość generowanych danych może być spora (w symulacji: jedna wiadomość na kilka sekund). Ze względu na docelowe warunki wdrożeniowe, kluczowa dla realizacji zadania jest efektywność sieciowa komunikacji (na poziomie L3). Dla zaoszczędzenia pasma sieciowego jest w szczególności dopuszczalne (a nawet pożądane), by wiadomość z wynikiem pomiaru była wysłana natychmiast, jednak powinna być ona dostarczona do serwera najpóźniej w ciągu zdefiniowanego czasu (w symulacji: kilkanaście sekund) – stąd pożądane jest rozsądne agregowanie wywołań, najlepiej w sposób niewidoczny dla rdzenia aplikacji klienckiej. Bardzo istotna jest elegancja rozwiązania.

Technologia middleware: dowolna pozwalająca na osiągnięcie eleganckiego i efektywnego rozwiązania.
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 8

Zadanie A3 - Sprawy urzędowe
Celem zadania jest skonstruowanie aplikacji klient-serwer służącej do zlecania spraw urzędowych i uzyskiwania informacji o ich wyniku. Czas rozpatrzenia sprawy przez urząd jest zgrubnie określany już w momencie jej zgłaszania i może wynosić około godziny-dwóch (na potrzeby realizacji i demonstracji zadania należy oczywiście przyjąć, że jest krótszy). Zlecający (klient) jest zainteresowany jak najszybszym dowiedzeniem się o wyniku zgłoszonej przez siebie sprawy, jednak spóźnienie rzędu minuty czy dwóch nie będzie stanowić problemu. Nie można założyć, że każdy klient będzie chciał mieć uruchomioną aplikację przez cały czas realizacji sprawy.
Urząd obsługuje wiele różnych typów spraw (różniących się zestawem przesyłanych lub zwracanych informacji), można przyjąć, że wszystkie typy spraw są znane w czasie tworzenia systemu. (na potrzeby realizacji zadania wystarczy przewidzieć trzy przykładowe).
Priorytetem w realizacji zadania jest dobór, zaprojektowanie i realizacja właściwego sposobu komunikacji – minimalizującego liczbę niepotrzebnych wywołań. Dopuszczalne są wszystkie eleganckie opcje realizacji komunikacji.

Technologia middleware: dowolna
Demonstracja zadania: aplikacja kliencka musi pozwalać na efektywne przetestowanie różnych scenariuszy
Języki programowania: dwa (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 7

Zadanie I1 - Porównanie wydajności i efektywności komunikacyjnej technologii middleware
Celem zadania jest porównanie wydajności (czas wykonania) i efektywności komunikacji (liczba bajtów na poziomie L3) omawianych technologii middleware dla porównywalnych danych (podobieństwo interfejsów IDL) i sposobu implementacji strony serwerowej. Testy należy wykonać dla kilku (trzech) jakościowo różnych struktur danych i różnych wielkości samych danych (tj. np. sekwencja o długości 1 i 100 elementów) oraz dla różnych języków programowania. Dla uproszczenia realizacji wszystkie operacje/procedury powinny zwracać wartość pustą (void). Eksperymenty należy wykonać w taki sposób, by wyeliminować/uwzględnić wartości odstające (outlier).

Demonstracja zadania: Omówienie środowiska testowego, zamieszczonego zwięzłego raportu z warunkami eksperymentów i stosownymi wykresami oraz przedstawienie najważniejszych wniosków.
Technologia middleware: Ice, Thrift, gRPC (wszystkie)
Języki programowania: dwa różne (ale najlepiej konsekwentnie te same w każdej z technologii)
Maksymalna punktacja: 9

Zadanie I2. Porównanie sposobów serializacji Thrift
Celem zadania jest stworzenie prostej aplikacji klient-serwer i przeprowadzenie eksperymentów porównujących wydajność (czas wykonania) i efektywność komunikacji (liczba bajtów na poziomie L3) sposobów serializacji oferowanych przez Thrift dla kilku (trzech) jakościowo różnych struktur danych i wielkości samych danych (tj. np. sekwencja o długości 1 i 100 elementów) oraz dla różnych języków programowania. Warto odpowiedzieć na pytanie: jakie negatywne strony ma użycie serializacji oferującej większą efektywność komunikacyjna (jeśli ma)?

Demonstracja zadania: Omówienie środowiska testowego, zamieszczonego zwięzłego raportu z warunkami eksperymentów i stosownymi wykresami oraz przedstawienie najważniejszych wniosków.
Technologia middleware: Thrift
Języki programowania: trzy różne
Maksymalna punktacja: 7

Zadanie I3. Wywołanie dynamiczne
Celem zadania jest demonstracja działania wywołania dynamicznego po stronie klienta middleware. Wywołanie dynamiczne to takie, w którym nie jest wymagana znajomość interfejsu zdalnego obiektu lub usługi w czasie kompilacji, lecz jedynie w czasie wykonania. Wywołania powinny być zrealizowane dla kilku (trzech) różnych operacji/procedur używających (przynajmniej w jednym przypadku) nietrywialnych struktur danych. Nie trzeba tworzyć żadnego formatu opisującego żądanie użytkownika ani parsera jego żądań - wystarczy zawrzeć to wywołanie "na sztywno" w kodzie źródłowym, co najwyżej z konsoli parametryzując szczegóły danych. Można jako bazę wykorzystać projekt z zajęć.
Ice: Dynamic Invocation https://doc.zeroc.com/ice/3.7/client-server-features/dynamic-ice/dynamic-invocation-and-dispatch
gRPC: dynamicgrpc (->Google)

Technologia middleware: Ice, gRPC
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 5

Zadanie I4. Efektywne zarządzanie serwantami
Celem zadania jest demonstracja (na bardzo prostym przykładzie) mechanizmu zarządzania serwantami technologii Ice. Zadanie powinno mieć postać bardzo prostej aplikacji klient-serwer, w której strona serwerowa obsługuje wiele obiektów Ice. Obiekty middleware są dwojakiego typu: część powinna być zrealizowana przy pomocy dedykowanego dla każdego z nich serwanta, druga część może korzystać ze współdzielonego dla nich wszystkich serwanta . Zarządzanie serwantami ma być efektywne, np. dla dedykowanych serwantów, taki serwant jest instancjonowany dopiero w momencie pierwszego zapotrzebowania na niego.

Interfejs obiektu może być superprosty, a aplikacja kliencka powinna jedynie umożliwić zademonstrowanie funkcjonalności serwera. Logi na konsoli po stronie serwera powinny pozwolić się zorientować, na którym obiekcie i na którym serwancie zostało wywołane żądanie i kiedy nastąpiło instancjonowanie serwanta.
W zadaniu trzeba korzystać bezpośrednio z mechanizmów zarządzania serwantami oferowanego przez technologię, a nie własnych mechanizmów realizujących np. wzorzec factory.

Technologia middleware: Ice
Języki programowania: dwa różne (jeden dla klienta, drugi dla serwera)
Maksymalna punktacja: 6

Zadanie I5. Kontrola przepływu w komunikacji middleware
Celem zadania jest demonstracja (na bardzo prostym przykładzie - można nawet wykorzystać elementy kodu z laboratorium) mechanizmu kontroli przepływu, w którym odbiorca lub medium transmisyjne może spowolniać nadawcę. Kontrola przepływu ma dotyczyć wywołań realizowanych asynchronicznie lub strumieniowo.
Demonstrowane zadanie nie może być wierną kopią rozwiązań znalezionych w Internecie lub w dokumentacji technologii.

Technologia middleware: gRPC, ew. Ice
Języki programowania: wystarczy jeden, mile widziane dwa
Maksymalna punktacja: 7

Zadanie I6. Reverse-proxy technologii gRPC
Celem zadania jest demonstracja (na bardzo prostym przykładzie - można nawet wykorzystać elementy kodu z laboratorium) mechanizmu reverse-proxy technologii gRPC.
Demonstrowane zadanie nie może być wierną kopią rozwiązań znalezionych w Internecie lub w dokumentacji technologii.

Technologia middleware: gRPC,
Języki programowania: wystarczy jeden
Maksymalna punktacja: 8

Zadanie I7. gRPC-Web
Celem zadania jest demonstracja (na bardzo prostym przykładzie) aplikacji klient-serwer zrealizowanej w technologii gRPC, gdzie aplikacja kliencka działa w przeglądarce WWW. Ważnym elementem zadania jest dokonanie chociaż pobieżnej ewaluacji aplikacji pod kątem wydajności i ograniczeń komunikacji.
Demonstrowane zadanie nie może być wierną kopią rozwiązań znalezionych w Internecie lub w dokumentacji technologii.

Technologia middleware: gRPC
Języki programowania: wystarczy jeden
Maksymalna punktacja: 8

Uwagi wspólne:

- Interfejsy IDL powinny być proste, ale zaprojektowane w sposób dojrzały (odpowiednie typy proste, właściwe wykorzystanie typów złożonych), w zadaniach aplikacyjnych dodatkowo uwzględniając możliwość wystąpienia różnego rodzaju błędów. Tam gdzie to możliwe i uzasadnione należy wykorzystać dziedziczenie interfejsów IDL.
- Działanie aplikacji może (nie musi) być demonstrowane na jednej maszynie.
- Kod źródłowy zadania powinien być demonstrowany w IDE a dodatkowe elementy (np. raporty z testów) przy pomocy oprogramowania pozwalającego na wygodne i szybkie zapoznanie się z nimi.
- Aktywność poszczególnych elementów aplikacji należy odpowiednio logować (wystarczy na konsolę) by móc sprawnie ocenić poprawność jej działania.
- Aplikacja kliencka powinna mieć postać tekstową i może być minimalistyczna, lecz musi pozwalać na przetestowanie funkcjonalności aplikacji szybko i na różny sposób (musi więc być przynajmniej w części interaktywna).
- Pliki generowane (stub, skeleton, itp.) powinny się znajdować w osobnym katalogu niż kod źródłowy klienta i serwera. Pliki stanowiące wynik kompilacji (.class, .o itp) powinny być w osobnych katalogach niż pliki źródłowe.

Sposób oceniania:

Wykonanie tylko jednego z zadań **nie pozwoli** na uzyskanie zaliczenia zadania.

Sposób wykonania zadania będzie miał zasadniczy wpływ na ocenę. W szczególności:

- niestarannie przygotowany interfejs IDL: -2 pkt.
- niestarannie napisany kod (m.in. zła obsługa wyjątków, błędy działania w czasie demonstracji): -2 pkt.

- brak aplikacji w więcej niż jednym języku programowania: -2 pkt.
- brak wymaganej funkcjonalności lub realizacja funkcjonalności w sposób niezgodny z wytycznymi: -8 pkt.
- nieznajomość zasad działania aplikacji w zakresie zastosowanych mechanizmów: -10 pkt,
- dodatkowa funkcjonalność: +3 pkt.

Punktacja dotyczy sytuacji ekstremalnych - całkowitego braku pewnego mechanizmu albo pełnej i poprawnej implementacji - możliwe jest przyznanie części punktów (lub punktów karnych).

Pozostałe uwagi:

- Zadanie trzeba prezentować sprawnie, będzie na to ok. 13 minut.
- Termin nadesłania zadania **dla wszystkich grup**: 7 czerwca 2021, godz. 8:00.
- Prezentowane muszą być dokładnie te zadania, które zostały zamieszczone na moodle, tj. nie są dopuszczalne żadne późniejsze poprawki.
- Przypominam o konieczności dołączenia do zadania oświadczenia o samodzielnym jego wykonaniu. Konsultowanie się z innymi studentami jak zrealizować poszczególne funkcjonalności czy wzorowanie się na przykładach dostępnych w Internecie (a w szczególności w dokumentacji technologii) nie jest oczywiście traktowane jako niesamodzielność wykonania.

Ostatnia modyfikacja: niedziela, 6 czerwca 2021, 00:14

◄ QUIC jest już standardem IETF (RFC 9000)!

Przejdź do...

Zadanie - technologie middleware ►

Platforma e-Learningowa obsługiwana jest przez:
Centrum e-Learningu AGH oraz Centrum Rozwiązań Informatycznych AGH

Pobierz aplikację mobilną