A. DEFINICJA

Rozproszona baza danych

Rozproszona baza danych (ang. distributed database) to baza danych, która jest rozłożona na kilku hostach lub maszynach i umożliwia dostęp do danych z dowolnego miejsca w sieci. Rozproszona baza danych ma na celu zwiększenie niezawodności i wydajności oraz umożliwienie dostępu do danych z różnych miejsc w sieci. W przeciwieństwie do centralnej bazy danych, w której dane są przechowywane i przetwarzane na jednym serwerze, rozproszona baza danych rozprowadza dane i obciążenie procesora pomiędzy wieloma serwerami, co pozwala na lepsze wykorzystanie zasobów i lepszą skalowalność.

TCP

Protokół TCP (Transmission Control Protocol) jest to protokół warstwy transportowej sieci komputerowej, który umożliwia niezawodne przesyłanie danych pomiędzy komputerami w sieci. TCP działa poprzez nadawanie danych w pakietach (ang. packets) i kontrolowanie przepływu danych pomiędzy urządzeniami w sieci, aby upewnić się, że dane są poprawnie przesłane i odebrane. Protokół TCP zapewnia niezawodność poprzez wysyłanie potwierdzeń odbioru dla każdego pakietu danych oraz możliwość ponownego wysłania pakietów, jeśli są one utracone lub uszkodzone w trakcie transmisji. Protokół TCP jest często używany do przesyłania danych przez Internet i jest jednym z najważniejszych protokołów sieciowych.

(źródło dla definicji TCP I Rozproszona baza danych https://chat.openai.com/chat)

Sąsiadowanie węzłów

Węzły uznajemy za węzły sąsiadujące, jeżeli węzły są połączone i możliwa jest komunikacja między nimi w celu wykonania operacji.

B. WSTĘP

Głównym celem projektu jest stworzenie rozproszonej bazy danych, którą będzie można skalować. Oznacza to, że wraz ze wzrostem ilości użytkowników lub ilości przechowywanych danych, można w prosty sposób zarządzać (dodawać lub odejmować) ilością węzłów w sieci, co wpływa na wydajność rozwiązania.

Podstawowe zaimplementowane funkcjonalności w ramach projektu:

1. Dołączenie nowego węzła do sieci
2. Odłączenie węzła od sieci i zakończenie jego działania
3. Zmiana klucza i wartości w węźle, do którego podłączony jest klient
4. Zmiana wartości dla podanego przez klienta klucza
5. Wyświetlenie wartości dla podanego przez klienta klucza
6. Wyświetlenie adresu ip i portu węzła, który przechowuje klucz podany przez użytkownika
7. Znalezienie najwyższej wartości i wyświetlenie jej razem z przypisanej do wartości kluczem
8. Znalezienie najmniejszej wartości i wyświetlenie jej razem z przypisanej do wartości kluczem

9. Wyłączenie wszystkich węzłów

Dodatkowe zaimplementowane funkcjonalności:

1.Regeneracja bazy danych

Komunikacja między węzłami oraz między węzłem a klientem, odbywa się za pomocą protokołu TCP.

W poniższej dokumentacji przedstawiono szczegółowo:

1. Opis protokołu

2. Kompilacja i instalacja

3. Funkcjonalności

C. OPIS PROTOKOŁU

Protokół wykorzystywany w komunikacji między klientem a węzłem oraz między węzłem a węzłem, to protokół TCP/IP. Definicja protokołu TCP w sekcji Definicje powyżej. Komunikaty opisane są w sekcji Dokumentacja Techniczna (poniżej).

D. INTERFEJSY SYSTEMU

Interfejsy użytkownika:

1. Klienci rozproszonej bazy danych mogą wysłać zapytania i otrzymywać odpowiedzi poprzez interfejs TCP, który umożliwia im połączenie się z dowolnym węzłem w sieci i wysłanie zapytania za pomocą odpowiednich komend.

Interfejsy systemowe:

1. Węzły komunikują się ze sobą za pomocą protokołu TCP z innymi węzłami w sieci, wysyłając oraz

otrzymując odpowiednie komunikaty.

2. Węzły mogą odbierać zapytania od klientów i odpowiadać na nie poprzez interfejs TCP.

3. Administrator węzła ma możliwość śledzenia:

3.1. Dodania nowego sąsiada (Komunikat "ADDNEIGHBOUR")

3.2. Komunikatów od innych węzłów (Komunikat "Przyjęto komende od innego węzła")

3.3. Zapytań od klienta (Komunikat "Przyjęto komende od klienta")

E. WYMAGANIA SYSTEMOWE I KONFIGURACJA SIECI

E.1. Wymagania systemowe

1. Java Development Kit (JDK) w wersji 1.8.

E.2. Konfiguracja systemu

E.2.1. Kompilacja kodu

W celu skompilowania kodu należy włączyć:

-kompilacja.bat na systemie operacyjnym Windows

-kompilacja.sh na systemie operacyjnym Linux

Wskazane pliki znajdują się w katalogu kompilacja.

E.2.2. W celu utworzenia pierwszego węzła w sieci należy uruchomić go podając odpowiednie argumenty w konsoli.

Przykład uruchomienia pierwszego węzła w sieci:

java DatabaseNode -tcpport 9990 -record 5:5

java DatabaseNode -tcpport <numer portu TCP> -record <klucz>:<wartość>

Gdzie:

-tcpport <numer portu TCP> określa numer portu TCP, na którym tworzymy nowy węzeł.

-record <klucz>:<wartość> oznacza parę liczb całkowitych początkowo przechowywanych w bazie na danym węźle, gdzie pierwsza to klucz a druga to wartość związana z tym kluczem. Nie ma wymogu unikalności zarówno klucza, jak i wartości.

E.2.3. W celu utworzenia kolejnych węzłów w sieci należy uruchomić go z dodatkowym parametrem:

Przykład uruchomienia kolejnego węzła w sieci z dwoma węzłami sąsiadującymi (węzły te muszą być już uruchomione):

java DatabaseNode -tcpport 9990 -record 5:5 -connect localhost:9989 -connect localhost:9988

Gdzie:

-connect <adres ip>:<port> oznacza węzeł już uruchomiony węzeł, z którym dany węzeł ma sąsiadować (Definicja sąsiadowania węzłów w sekcji Definicje).

Przykładowy skrypt tworzący sieć:

Utworzenie przykładowej sieci.bat lub .sh

Wskazany skrypt tworzy sieć, którą zilustrowano w pliku Przykładowa sieć.pdf.

E.3. Włączenie klienta

Klienci mogą nawiązać połączenie z dowolnym węzłem sieci za pomocą polecenia w konsoli

Przykładowe wywołanie klienta:

java DatabaseClient -gateway localhost:1 -operation get-min

java DatabaseClient -gateway <adres ip>:<port> -operation <operacja z parametrami>

Gdzie:

-gateway <adres IP>:<port> wskazuje na węzeł, z którym komunikuje się klient.

-operation <operacja z parametrami> dostępne operacje są opisane w Dokumentacji Technicznej w podpunkcie F.3.

F. DOKUMENTACJA TECHNICZNA

F.1. Uruchomienie węzła

Opis działania:

F.1.1. Pobranie parametrów wywołania (DatabaseNode.java: main ()):

-port TCP

-klucz i wartość recordu w węźle

-adres IP i port węzła, do którego nowy węzeł ma się podłączyć

F.1.2. Stworzenie nowej instancji klasy DatabaseNode i przypisanie do niej wcześniej zdobytych wartości (main ())

F.1.3. Włączenie wątku węzła sieci, czyli (DatabaseNode.java: main (), run ()):

-wysłanie do innych węzłów wiadomości "ADDNEIGHBOUR", która ma na celu poinformowanie o dołączeniu nowego

sąsiadującego węzła do sieci. Jeśli węzeł nie ma podanych węzłów, do których ma się podłączyć,

to rozpoczyna działanie jako "bootstrap node", czyli węzeł, który inicjuje sieć.

-oczekiwanie na połączenie od klienta lub innego węzła

F.2. Algorytmy komunikacji (MessageHandler.java; DatabaseNode.java: run () i node service ())

F.2.1. Algorytm komunikacji klient-węzeł

Opis działania:

F.2.1.1. Węzeł otrzymuje komunikat o chęci połączenia od klienta.

F.2.1.2. Tworzy nową instancję klasy MessageHandler.java przypisując jej socket klienta i węzeł

(do którego jest podłączony klient)

F.2.1.3. Uruchamiany jest nowy wątek (MessageHandler.java: run ())

Opis działania MessageHandler.java znajduje się w punkcie F.2.3.

F.2.2. Algorytm komunikacji węzeł-węzeł

Zgodnie z opisem komunikacji klient-węzeł. Punkt powyżej.

F.2.3. Odbiór i interpretacja zapytania (MessageHandler.java: run (), check ())

F.2.3.1. Odebranie wiadomości od klienta / wątku

F.2.3.2. Sprawdzenie jaki typ zapytania wysłał nam klient / wątek

F.2.3.3. Jeżeli dane zapytanie jest obsługiwane

(obsługiwane zapytania wraz z funkcjami, które je obsługują znajdują się w punkcie F.3.)

to włącza odpowiadająca mu metodę. (w przypadku komunikatu od innego węzła wiadomość jest przetwarzana ponownie w funkcji DatabaseNode.java: processMessage (). Działanie tej funkcji jest opisane w punkcie F.4.2.)

F.2.3.4. W przeciwnym przypadku wysyła odpowiedź "Nie ma takiej komendy"

F.2.3.5. Zamknięcie socketu i wyłączenie wątku

F.3. Komunikacja klient-węzeł

Operacje, które może użyć klient:

F.3.1. set-value <klucz>:<wartość> (DatabaseNode.java: handleRequest ())

Opis ogólny

Zapytanie wysłane bezpośrednio do węzła w celu zmiany wartości przypisanej do klucza dla każdego węzła w sieci.

Opis działania

F.3.1.1. Węzeł po otrzymaniu wskazanego zapytania sprawdza, czy klucz, który chce zmienić klient, to klucz przypisany do niego.

F.3.1.2. Jeżeli wskazany klucz jest przypisany do danego węzła, zmieniamy wartość i wysyłamy odpowiedź "OK" i kończy działanie.

F.3.1.3. W przeciwnym przypadku węzeł rozsyła komunikat "NODE tcpport id SET klucz wartość"

F.3.1.4. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.3.1.5. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.3.1.6. Jeżeli co najmniej jeden węzeł w sieci zmienił wartość, to wysłana jest do klienta odpowiedź "OK"

F.3.1.7. W przeciwnym przypadku wysłana odpowiedź to "ERROR" oznacza, że nie ma podanego klucza w sieci.

F.3.2. get-value <klucz> (DatabaseNode.java: handleRequest ())

Opis ogólny

Zapytanie, które ma na celu zwrot komunikatu <klucz>:<wartość>, klucz jest podawany przez klienta,

natomiast zwracana jest wartość przypisana do klucza wraz z kluczem.

Opis działania

F.3.2.1. Węzeł po otrzymaniu wskazanego zapytania sprawdza, czy klucz, który wskazał klient, to klucz przypisany do niego.

F.3.2.2. Jeżeli wskazany klucz jest przypisany do danego węzła, to wysyłamy klucz i wartość przypisana do niego i kończy działanie.

F.3.2.3. W przeciwnym przypadku węzeł rozsyła komunikat "NODE tcpport id GET klucz"

F.3.2.4. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.3.2.5. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.3.2.6. Jeżeli co najmniej jeden węzeł w sieci wskazał wartość dla danego klucza, to wysłana jest do klienta odpowiedź

klucz i przypisana do niego wartość

F.3.2.7. W przeciwnym przypadku wysłana odpowiedź to "ERROR" oznacza, że nie ma podanego klucza w sieci.

F.3.3. find-key <klucz> (DatabaseNode.java: handleRequest ())

Opis ogólny

Zapytanie, które ma na celu zwrot adresu ip i portu węzła, w którym znajduje się record o podanym

kluczu. Odpowiedź jest w postaci <adres ip>:<port>.

Opis działania

F.3.3.1. Węzeł po otrzymaniu wskazanego zapytania sprawdza, czy klucz, który wskazał klient, to klucz przypisany do niego.

F.3.3.2. Jeżeli wskazany klucz jest przypisany do danego węzła, to wysyłamy adres ip i port węzła i kończy działanie.

F.3.3.3. W przeciwnym przypadku węzeł rozsyła komunikat "NODE tcpport id FIND klucz"

F.3.3.4. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.3.3.5. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.3.3.6. Jeżeli co najmniej jeden węzeł w sieci wskazał adres ip i port, to wysłana jest do klienta odpowiedź adres ip i port węzła, do którego przypisany jest dany klucz.

F.3.3.7. W przeciwnym przypadku wysłana odpowiedź to "ERROR" oznacza, że nie ma podanego klucza w sieci.

F.3.4. get-max (DatabaseNode.java: getExtremum ())

Opis ogólny

Zapytanie mające na celu zwrot największej wartości w sieci, odpowiedź jest w postaci <klucz>:<wartość>.

Opis działania

F.3.4.1. Węzeł po otrzymaniu wskazanego zapytania przyjmuje, że maksymalna wartość jest to jego lokalna wartość, czyli

przyjmuje jako maksymalna wartość swoją wartość, a jako klucz swój klucz

F.3.4.2. Węzeł rozsyła komunikat "NODE tcpport id MAX"

F.3.4.3. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów

F.3.4.4. Każdą odpowiedź porównuje z przypisanym maxem

F.3.4.5. Jeżeli dana odpowiedź jest większa od wskazanego powyżej maxa, to przypisuje ją jako wartość maksymalną, oraz

wskazany klucz jest kluczem maksymalnej wartości

F.3.4.6. W przeciwnym przypadku przechodzi do następnej odpowiedzi

F.3.4.7. Po sprawdzeniu wszystkich odpowiedzi wysyła do klienta maksymalną wartość i klucz maksymalnej wartości

F.3.5. get-min (DatabaseNode.java: getExtremum ())

Opis ogólny

Zapytanie, które ma na celu zwrot najmniejszej wartości w sieci. Odpowiedź ma postać <klucz>:<wartość>.

Opis działania

F.3.5.1. Węzeł po otrzymaniu wskazanego zapytania przyjmuje, że minimalna wartość jest to jego lokalna wartość, czyli

przyjmuje jako minimalna wartość swoją wartość, a jako klucz swój klucz

F.3.5.2. Węzeł rozsyła komunikat "NODE tcpport id MIN"

F.3.5.3. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów

F.3.5.4. Każdą odpowiedź porównuje z przypisanym minem

F.3.5.5. Jeżeli dana odpowiedź jest większa od wskazanego powyżej mina, to przypisuje ją jako wartość minimalna, oraz

wskazany klucz jest kluczem minimalnej wartości

F.3.5.6. W przeciwnym przypadku przechodzi do następnej odpowiedzi

F.3.5.7. Po sprawdzeniu wszystkich odpowiedzi wysyła do klienta minimalna wartość i klucz minimalnej wartości

F.3.6. new-record <klucz>:<wartość> (DatabaseNode.java: changeRecord ())

Opis ogólny

Zapytanie mające na celu zmianę klucza, jak i wartości dla węzła, do którego podłączony jest klient.

Opis działania

Węzeł otrzymując zapytanie, od klienta zmienia klucz i wartość w swoim rekordzie, po czym zwraca odpowiedź "OK",

po zakończonej operacji, co oznacza, że zmiana wartości i klucza przebiegła pomyślnie.

F.3.7. terminate (DatabaseNode.java: terminate ())

Opis ogólny

Zapytanie, które ma na celu zakończenie działania węzła, do którego jest podłączony klient i odłączenie go od sieci.

Opis działania

F.3.7.1. Węzeł otrzymując taką wiadomość wysyła, do wszystkich sąsiadujących węzłów "NODE tcpport id TERMINATE".

F.3.7.2. Kończy swoje działanie i zwalnia socket.

F.3.7.3. Klientowi jest wysłana odpowiedź "OK", co oznacza, że całość przebiegła pomyślnie.

F.3.8. terminate-all (DatabaseNode.java: terminateAll ())

Opis ogólny

Zapytanie, które ma na celu zakończenie działania wszystkich węzłów należących do jednej sieci.

Opis działania

F.3.7.1. Węzeł otrzymując taką wiadomość wysyła, do wszystkich sąsiadujących węzłów "NODE tcpport id TERMINATEALL".

F.3.7.2. Kończy swoje działanie i zwalnia socket.

F.3.7.3. Klientowi jest wysłana odpowiedź "OK", co oznacza, że całość przebiegła pomyślnie.

F.4.Komunikacja węzeł-węzeł:

F.4.1. Konstrukcja komunikatów węzłów wygląda następująco:

NODE <tcpport> <id> <operacja z parametrami>

Gdzie:

-tcpport to port tcp węzła, z którego rozsyłany jest komunikat do innych węzłów.

-id to indywidualne dla każdego węzła id pytania, które kazał rozesłać po sieci.

-operacja to operacja, której wymaga dany węzeł. Ta część może, ale nie musi zawierać parametry w zależności, jaką operacje rozsyła węzeł.

F.4.2. Metoda przetwarzająca komunikaty od węzła (DatabaseNode.java: processMessage ())

F.4.2.1. Sprawdzenie jaki typ komunikatu wysłał nam wątek

F.4.2.2. Uruchamiana jest przypisana do typu komunikatu metoda

F.4.2.3. Zwrócenie odpowiedzi do (MessageHandler.java: run ())

(obsługiwane zapytania wraz z funkcjami, które je obsługują znajdują się w punkcie F.4.3)

F.4.3. Operacje, których może użyć węzeł:

F.4.3.1. GET klucz (DatabaseNode.java: processMessage (), getResponse (), handleRequest ())

Opis ogólny

Komunikat, który ma na celu zwrot <klucz>:<wartość>, klucz jest podawany przez inny węzeł,

natomiast zwracana jest wartość przypisana do klucza wraz z kluczem.

Opis działania

F.4.3.1.1. Węzeł weryfikuje czy pobrany komunikat był już obsłużony

F.4.3.1.2. Jeżeli tak, to wysyła odpowiedź "ERROR"

F.4.3.1.3. W przeciwnym przypadku węzeł sprawdza, czy klucz, który wskazał klient, to klucz przypisany do niego.

F.4.3.1.4. Jeżeli wskazany klucz jest przypisany do danego węzła, to wysyłamy klucz i wartość przypisana do niego i kończy działanie.

F.4.3.1.5. W przeciwnym przypadku węzeł rozsyła pobrany komunikat do sąsiadujących węzłów.

F.4.3.1.6. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.4.3.1.7. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.4.3.1.8. Jeżeli co najmniej jeden węzeł w sieci wskazał wartość dla danego klucza, to wysłana jest do węzła,

od którego przyjęto komunikat odpowiedź klucz i przypisana do niego wartość

F.4.3.1.9. W przeciwnym przypadku wysłana odpowiedź to "ERROR".

F.4.3.2. SET klucz wartość (DatabaseNode.java: processMessage (), setResponse (), handleRequest ())

Opis ogólny

Komunikat wysłany do węzła w celu zmiany wartości przypisanej do klucza.

Opis działania

F.4.3.2.1. Węzeł weryfikuje czy pobrany komunikat był już obsłużony

F.4.3.2.2. Jeżeli tak, to wysyła odpowiedź "ERROR"

F.4.3.2.3. W przeciwnym przypadku węzeł sprawdza, czy klucz, który chce zmienić klient, to klucz przypisany do niego.

F.4.3.2.4. Jeżeli wskazany klucz jest przypisany do danego węzła, zmieniamy wartość i wysyłamy odpowiedź "OK" i kończy działanie.

F.4.3.2.5. W przeciwnym przypadku węzeł rozsyła pobrany komunikat do sąsiadujących węzłów.

F.4.3.2.6. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.4.3.2.7. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.4.3.2.8. Jeżeli co najmniej jeden węzeł w sieci zmienił wartość, to wysłana jest do węzła, od którego przyjęto

komunikat, odpowiedź "OK"

F.4.3.2.9. W przeciwnym przypadku wysłana odpowiedź to "ERROR" oznacza, że nie ma podanego klucza w sieci.

F.4.3.3. FIND klucz (DatabaseNode.java: processMessage (), findResponse (), handleRequest ())

Opis ogólny

Komunikat, który ma na celu zwrot adresu ip i portu węzła, w którym znajduje się record o podanym

kluczu. Odpowiedź jest w postaci <adres ip>:<port>.

Opis działania

F.4.3.3.1. Węzeł weryfikuje czy pobrany komunikat był już obsłużony

F.4.3.3.2. Jeżeli tak, to wysyła odpowiedź "ERROR"

F.4.3.3.3. W przeciwnym przypadku sprawdza, czy klucz, który wskazał klient, to klucz przypisany do niego.

F.4.3.3.4. Jeżeli wskazany klucz jest przypisany do danego węzła, to wysyłamy adres ip i port węzła i kończy działanie.

F.4.3.3.5. W przeciwnym przypadku węzeł rozsyła pobrany komunikat do sąsiadujących węzłów.

F.4.3.3.6. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów.

F.4.3.3.7. Węzeł przyjmuje odpowiedzi od sąsiadujących węzłów i weryfikuje je. (DatabaseNode.java: getCheckResponse ())

F.4.3.3.8. Jeżeli co najmniej jeden węzeł w sieci wskazał adres ip i port, to wysłana jest do węzła, od którego

przyjęto komunikat, odpowiedź adres ip i port węzła, do którego przypisany jest dany klucz.

F.4.3.3.9. W przeciwnym przypadku wysłana odpowiedź to "ERROR" oznacza, że nie ma podanego klucza w sieci.

F.4.3.4. MAX (DatabaseNode.java: processMessage (), maxResponse (), getExtremum ())

Opis ogólny

Komunikat mający na celu zwrot największej wartości w sieci, odpowiedź jest w postaci <klucz>:<wartość>.

Opis działania

F.4.3.4.1. Węzeł weryfikuje czy pobrany komunikat był już obsłużony

F.4.3.4.2. Jeżeli tak, to wysyła odpowiedź w postaci swojego klucza i przypisanej do niej wartości

F.4.3.4.3. W przeciwnym przypadku węzeł przyjmuje, że maksymalna wartość jest to jego lokalna wartość, czyli

przyjmuje jako maksymalna wartość swoją wartość, a jako klucz swój klucz

F.4.3.4.4. Węzeł rozsyła pobrany komunikat do sąsiadujących węzłów.

F.4.3.4.5. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów

F.4.3.4.6. Każdą odpowiedź porównuje z przypisanym maxem

F.4.3.4.7. Jeżeli dana odpowiedź jest większa od wskazanego powyżej maxa, to przypisuje ją jako wartość maksymalną, oraz

wskazany klucz jest kluczem maksymalnej wartości

F.4.3.4.8. W przeciwnym przypadku przechodzi do następnej odpowiedzi

F.4.3.4.9. Po sprawdzeniu wszystkich odpowiedzi wysyła do węzła, od którego otrzymał komunikat, odpowiedź w postaci

maksymalnej wartości i klucza maksymalnej wartości

F.4.3.5. MIN (DatabaseNode.java: processMessage (), minResponse (), getExtremum ())

Opis ogólny

Komunikat, który ma na celu zwrot najmniejszej wartości w sieci. Odpowiedź ma postać <klucz>:<wartość>.

Opis działania

F.4.3.5.1. Węzeł weryfikuje czy pobrany komunikat był już obsłużony

F.4.3.5.2. Jeżeli tak, to wysyła odpowiedź w postaci swojego klucza i przypisanej do niej wartości

F.4.3.5.3. W przeciwnym przypadku węzeł przyjmuje, że minimalna wartość jest to jego lokalna wartość, czyli przyjmuje jako minimalna wartość swoją wartość, a jako klucz swój klucz

F.4.3.5.4. Węzeł rozsyła pobrany komunikat do sąsiadujących węzłów.

F.4.3.5.5. Węzeł dodaje wskazany w punkcie powyżej komunikat do listy obsłużonych komunikatów

F.4.3.5.6. Każdą odpowiedź porównuje z przypisanym minem

F.4.3.5.7. Jeżeli dana odpowiedź jest mniejsza od wskazanego powyżej mina, to przypisuje ją jako wartość minimalną, oraz

wskazany klucz jest kluczem minimalnej wartości

F.4.3.5.8. W przeciwnym przypadku przechodzi do następnej odpowiedzi

F.4.3.5.9. Po sprawdzeniu wszystkich odpowiedzi wysyła do węzła, od którego otrzymał komunikat, odpowiedź w postaci minimalnej wartości i klucza minimalnej wartości

F.4.3.6. TERMINATE (DatabaseNode.java: processMessage (), terminateResponse ())

Opis ogólny

Komunikat ma na celu poinformowanie sąsiadujących węzłów o tym, że sąsiadujący do nich węzeł odłącza się od sieci i kończy działanie

Opis działania

F.4.3.6.1. Węzeł usuwa z listy sąsiadujących węzłów węzeł, który wysłał komunikat.

F.4.3.6.2. Węzeł usuwa wszystkie obsłużone pytania od tego węzła z listy pytań.

F.4.3.6.3. Węzeł nie wysyła żadnej odpowiedzi

F.4.3.7. TERMINATEALL (DatabaseNode.java: processMessage (), terminateAll ())

Opis ogólny

Komunikat ma na celu wyłączenie działania węzła i wysłanie do sąsiadujących węzłów tego samego komunikatu, aby wszystkie węzły w sieci zostały wyłączone.

Opis działania

F.4.3.7.1. Węzeł usuwa z listy sąsiadujących węzłów węzeł, który wysłał komunikat.

F.4.3.7.2. Węzeł wysyła, do wszystkich sąsiadujących węzłów "NODE tcpport id TERMINATEALL".

F.4.3.7.3. Kończy swoje działanie i zwalnia socket.

F.4.3.7.4. Węzeł nie wysyła żadnej odpowiedzi

F.4.3.8. ADDNEIGHBOUR (DatabaseNode.java: processMessage (), addNeighbour ())

Opis ogólny

Komunikat ma na celu poinformowanie o dołączeniu do sieci nowego węzła i dodanie go do listy sąsiadów przez te węzły, z którymi sąsiaduje

Opis działania

Jedyną czynnością, która wykonuje węzeł po otrzymaniu komunikatu, to dodanie nowego węzła do listy sąsiadów.

F.5. Testy

Przygotowane i wykonane testy jednostkowe znajdują się w podfolderze Rozproszona Baza Danych/tests.

Testy ogólne (znajdują się w Rozproszona Baza Danych/src/testy):

-utworzenie sieci składającej się z 25 węzłów (Rysunek sieci przedstawiony w pliku Przykładowa sieć.pdf)

-uruchomienie kilku klientów

-wysłanie zapytań do sieci od klientów

-wyłączenie wszystkich węzłów (wyłączenie sieci)

W tym samym folderze znajduje się test z grupy wykładowej.

F.6. Regeneracja baz danych

Regeneracja została zaimportowana za pomocą kilku nowych komunikatów:

F.6.1. YOUFIRST (DatabaseNode.java: processMessage (), youFirst())

Komunikat ten zostaje wysłany w momencie zamknięcia bootstrap node, do kolejnego utworzonego węzła, przez to on teraz staje się bootstrap nodem.

F.6.2. RUFIRST (DatabaseNode.java: terminateResponse(), checkConnection(), checkFirst())

Komunikat ma na celu sprawdzenie, czy węzeł po odłączeniu węzła dalej ma połączenie z bootstrap nodem, jeżeli nie to wysyła do wszystkich sąsiadów wyłączonego węzła, aby dodał go sąsiadów, aby nie utracić połączenia z bootstrap nodem.