Distributed Lock Manager i jego implementacje

Krzysztof Kotuła

10 grudnia 2009

Spis treści

1. Distributed Lock Manager w teorii
2. ARCS
3. HP OpenVMS
4. Googly Chubby

W teorii...

Distributed Lock Manager - narzędzie służące do zarządzania dostępem do współdzielonych zasobów w systemie rozproszonym.

...i to wszystko

W praktyce...

- Serwer bądź sieć serwerów
- Zarządza uprawnieniami (np. do plików)
- Granularność dostępu zależna od implementacji
- Sam w sobie nie musi udostępniać zasobów
- ...ani nawet pilnować poprawnego ich wykorzystywania

Zyski

- Uproszczona architektura
- Klarowniejsza semantyka użytkowania
- **Brak problemu spójności danych**
- **Prosta synchronizacja**
- **Możliwość cache'owania**
- Odporność na awarię klienta
- Łatwe monitorowanie zasobów
- ...

Problemy

- Centralizacja
- Zakleszczenie klientów
- Zagłodzenie
- ...

Przyjrzyjmy się kilku dostępnym implementacjom...

ARCS

Application Resource Control Service - komercyjny produkt firmy Littlearth, Inc.

- Chyba najprostszy serwer DLM
- Udostępnia łatwy interfejs blokowania zasobów
- Zasobem może być cokolwiek
- Wysoce stabilny
- Komunikuje się przez protokół HTTP (web service)
- Jest w zasadzie samoistnym bytem
- Udostępnia narzędzie monitorujące (przez przeglądarkę)

API

Dostępne metody:

- Begin - rozpoczęcie sesji
- End - zakończenie sesji
- CheckIn - utrzymanie sesji (KeepAlive)
- Bind - zablokowanie zasobu
- Release - zwolnienie zasobu
- Info - dodatkowe informacje o zadanym użytkowniku/zasobie

Funkcjonalność

- Blokowanie zasobów dla użytkownika(ów)
- Podczas blokowania możliwe jest zadeklarowanie maks. liczby współużytkowników
- Brak kontroli uprawnień
- Brak powiadomienia o udostępnieniu zasobu
- Brak bazy zasobów
- ...

Wydajność

(2007) *ARCS radzi sobie z kilkoma tysiącami użytkowników, każdym wysyłającym zapytanie średnio co 5 minut.*

Podsumowanie

Czy taka aplikacja warta jest 800 USD?

OpenVMS

Open Virtual Memory System - system operacyjny przeznaczony dla serwerów (klastrów), obecnie własność Hewlett-Packard.

- Dostępny na architektury VAX, Alpha oraz Itanium.
- Darmowy (GPL) projekt FreeVMS obecnie w wersji 0.3.x.
- **Wykorzystuje mechanizm DLM do zarządzania między innymi rozproszonym systemem plików**

Files-11

Cechy charakterystyczne:

- Hierarchiczny system plików
- Wsparcie dla ACL (listy kontroli dostępu)
- Operacje I/O zorientowane rekordowo
- **Dostęp zdalny poprzez sieć**
- Wersjonuje pliki (zrealizowane w najprostszy sposób)
- Poprzednik NTFS

Struktura katalogów

Mocno zbliżona do UNIXa, czyli:

- Jest jeden root
- Każdy plik/katalog (oprócz roota) znajduje się w jakimś katalogu
- Istnieją twarde dowiązania do plików

Nazewnictwo

Pełna nazwa pliku jest raczej skomplikowana:
NODE "account password" :: device : [dir.subdir] filename.type;ver

- Większość na szczęście jest pomijalna
- Człon *device* wskazuje konkretne urządzenie w klastrze.
- Brak zatem transparentności położenia.

Nazewnictwo logiczne

- Brak odpowiednika w standardzie POSIX
- Podobne do zmiennych środowiskowych
- Deklarowane na poziomie systemu plików, a nie aplikacji (np. konsoli)
- Pozwala po części realizować transparentność położenia

RMS

Record Management Services - warstwa I/O systemu, rozszerza pojęcie pliku

Przykład relacji dla pliku tekstowego:

- system plików <-> baza danych
- plik <-> tabela
- linia <-> rekord
- słowo <-> pole

Dostęp do pliku

Podczas otwierania pliku wymagane są dwie informacje:

- Co aplikacja chce robić (read/write/update)
- Jak aplikacja chce się dzielić plikiem

Hierarchia zasobów

- Baza danych
- Tabela
- Rekord
- Pole

Blokady

- Blokady nie są obowiązkowe
- Blokadę można założyć na dowolnym poziomie
- Wymaga to blokady na wyższym
- Potrzebny jest więc rozbudowany zbiór możliwych blokad

Rodzaje blokad

- Null
- Concurrent Read
- Concurrent Write
- Protected Read
- Protected Write
- Exclusive

Macierz zgodności

Mode	NL	CR	CW	PR	PW	EX
NL	Yes	Yes	Yes	Yes	Yes	Yes
CR	Yes	Yes	Yes	Yes	Yes	No
CW	Yes	Yes	Yes	No	No	No
PR	Yes	Yes	No	Yes	No	No
PW	Yes	Yes	No	No	No	No
EX	Yes	No	No	No	No	No

DLM

- Proces może zgłosić żądanie blokady
- Żądania są kolejkowane
- Realizacja odbywa się na zasadzie FIFO
- Blokady można konwertować z jednego poziomu na inny
- Żądanie może być wykonane synchronicznie lub asynchronicznie
- Istnieje możliwość sprawdzania stanu blokad
- DLM potrafi również notyfikować o tym, że posiadana blokada jest pożądana przez inny proces

Null lock

Poziom *pustej* blokady istnieje z dwóch powodów:

- Operacja konwersji blokady jest relatywnie szybsza do usuwania
- Operacja usunięcia nie potwierdza wykonania - nie ma gwarancji zdjęcia blokady

Lock value block

- Z blokadą można związać pewną wartość
- Odczyt/zmiana tej wartości wymaga stosownej blokady na zasobie
- W ten sposób można np. trzymać numer wersji zasobu
- Pozwala to na zrealizowanie cache'owania danych

Wykrywanie zakleszczeń

- System próbuje periodycznie wykryć zakleszczenia
- W przypadku znalezienia, informuje o tym jeden z uczestniczących procesów

Google Chubby

Chubby - DLM stworzony przez Google, de facto rozproszony system plików.

Charakterystyka przeznaczenia:

- Wsparcie szeroko pojętej synchronizacji
- Z założenia - gruboziarnistość czasów operacji
- W szczególności - rozwiązanie problemu elekcji

Paxos

- Rodzina protokołów służąca do rozwiązywania problemu uzgadniania (consensus)
- Opublikowana w 1998 roku
- Zakłada zawodność maszyn
- Wykorzystana w Chubby'm
- Google twierdzi, że wszystkie znane im protokoły opierają się na Paxosie

Inne wspierane mechanizmy

- Powiadamianie o zdarzeniach (events)
- Wsparcie dla cache'owania danych i metadanych
- Kontrola uprawnień - Access Control Lists (ACL)
- Blokady na plikach (locks)
- Odporność na awarię
- Replikowanie danych
- ...

Zalety

- Skalowalność aplikacji
- Łatwe wykorzystanie dzięki bibliotece po stronie klienta
- Rozwiązanie typowych problemów synchronizacji
- Możliwość wykorzystania jednej instancji przez różne aplikacje
- Intuicyjny interfejs blokad
- Brak problemu kworum

Zastosowanie

GFS:

- Ustalenie serwera master
- Przechowywanie metadanych

Bigtable:

- Ustalenie serwera master
- Przechowywanie metadanych
- Listowanie serwerów podporządkowanych
- Odnajdowanie serwera master przez klientów

MapReduce:

- Synchronizacja

Architektura

- 4 serwery replik + 1 ustalony master
- Klienci komunikują się za pośrednictwem biblioteki
- Serwer master wybierany drogą elekcji (typowo kilka sekund)
- Master wykonuje wszystkie akcje, repliki tylko do niego przekierowują
- Klienci znają wszystkie serwery i zapamiętują, kto jest serwerem master
- Dodatkowo system ma dostęp puli zapasowych serwerów

Baza danych

- Dane podręczne (jak np. informacje o sesji) trafiają do bazy danych
- Kopie bazy danych utrzymywane są wśród replik
- W razie awarii pomaga ona przywrócić stan systemu bez rozłączania klientów
- W drodze optymalizacji część danych przestała być utrzymywana w bazie
- Na początku Berkeley, potem własna, uproszczona

Udostępniany system plików

- Ogólnie katalogi i pliki to węzły
- Z założenia pliki przechowują tylko małe porcje danych
- Przykładowa ścieżka:
  /ls/chubby-cell/katalog1/katalog2/.../plik
- chubby-cell identyfikuje instancje Chubby'ego
- Specjalna nazwa local wskazuje na lokalną dla klienta
- Podobieństwo do UNIXa tylko z pozoru
- Wyszukiwanie odbywa się z pomocą serwerów DNS

Węzły

- Uprawnienia opisywane poprzez ACL
- Można zakładać na nich blokady (do czytanie lub pisania)
- Węzły można traktować z zewnątrz jako blokady i kanały komunikacyjne

ACL

- Domyślnie dziedziczone z węzła-rodzica
- Same w sobie są również plikami
- Umieszczone są w ustalonej lokacji w systemie plików

Metadane

Pięć liczb 64-bitowych:

- instance no.: zawsze większy niż wartość poprzednich węzłów o tej samej nazwie
- content generation no. (pliki): zwiększany przy zapisie
- lock genereation no.: zwiększany przy zdobyciu blokady na węzeł
- ACL genereation no.: zwiększany przy zapisie reguły dla węzła
- checksum, dostępny dla klientów

Zapis danych

- Klient wysyła nowe dane do zapisu
- Master rozsyła informacje o zmianach do replik poprzez protokół uzgadniania
- Wykonuje i potwierdza zapis do klienta, gdy otrzyma potwierdzenie od większości replik

Blokady

- Tryby read (shared) i write (exclusive)
- Blokowanie wymaga uprawnień do zapisu

Korzystanie z blokad nie jest obowiązkowe:

- Chubby często zarządza zasobami zewnętrznymi
- Wolny dostęp do plików ułatwia administrowanie/debuggowanie
- Google zaleca pisanie asercji sprawdzających posiadanie blokady

Wsparcie dla rozproszoności

- Brak operacji mv
- Brak czasu modyfikacji
- Brak czasu ostatniego dostępu
- Brak dowiązań (twardych i miękkich)
- Uprawnienia zależą tylko od samego obiektu, tj. nie od katalogów na ścieżce do niego
- Węzły ulotne, kasowane automatyczne

Sequencers

- W systemie rozproszonym komunikaty mogą dochodzić w złej kolejności
- Chubby używa logicznego numerowania (tylko operacji na blokadach)
- Sequencer - dane związane z konkretną blokadą (i jej plikiem)
- Zawiera: nazwę węzła, tryb blokady oraz lock generation number
- Klient przekazuje sequencer, jeśli operacja zdalna ma być chroniona
- Serwer zdalny ma obowiązek sprawdzić aktualność i tryb sequencera
- W ten sposób eliminowane są komunikaty spoza kolejności

Mechanizm nr 2

- Jeśli Chubby stwierdzi, że klient posiadający blokadę nie działa - nie zwalnia jej przez czas lock-delay
- Czas ten jest deklarowany przez klienta (obecnie max 1 min.)
- Zgubiona wiadomość ma więc czas dotrzeć do Chubby'ego
- W typowych sytuacjach mechanizm ten dobrze się sprawdza

Events

Chubby potrafi notyfikować klienta o następujących zdarzeniach:

- Zmiana treści pliku
- Usunięcie, dodanie, zmiana węzła potomnego (również ulotnego)
- Awaria serwera master
- Zdobycie blokady; w praktyce nieużywany
- Konflikt blokad - analogicznie do OpenVMS

Notyfikacje przesyłane są razem z pakietami KeepAlive.

API

- Open(nazwa, tryb, zdarzenia, lock-delay, czy utworzyć [, ACL, początkowe-dane]) - otwiera plik, zwraca jego handle
- Close() - zamyka plik, zawsze kończy się sukcesem
- Poison() - kończy wszystkie operacje na pliku, ale nie zamyka go
- GetContentsAndStat() - zwraca pełne dane/metadane, atomowo
- GetStat() - zwraca metadane, atomowo
- ReadDir() - zwraca metadane katalogu i listę węzłów, atomowo

API cd.

- SetContents() - zapisuje dane, atomowo; możliwe jest podanie content gen. no., wtedy zapis tylko gdy wartość aktualna
- SetACL() - ustawia uprawnienia
- Delete() - usuwa węzeł, jeśli nie zawiera dzieci
- Acquire(), TryAcquire(), Release() - operacje na blokadach
- GetSequencer() - zwraca sequencer pliku, wymaga posiadania blokady
- SetSequencer() - zapisuje sequencer razem z handle, wymaga blokady; wszystkie operacje wymagają podania aktualnej wartości
- CheckSequencer() - weryfikuje podany sequencer

Uwagi

- Metody zawodzą, gdy plik został usunięty, nawet gdy został ponownie utworzony
- Handle wiąże się więc z instancją pliku, a nie jego nazwą
- Możliwe są wywołania asynchroniczne
- Serwer udostępnia na życzenie rozszerzone informacje o błędach
- Razem z wywołaniem wysyłany jest epoch no.

Przykład elekcji

- Wszyscy otwierają znany plik i próbują go zablokować do zapisu
- Ten któremu się uda - zostaje zwycięzcą
- Zwycięzca zapisuje do pliku swoją tożsamość
- Pozostali ją odczytują reagując na zdarzenie o zmianie treści pliku
- Dalsza weryfikacja tożsamości zwycięzcy odbywa się z użyciem sequencera
- Naturalnie po zwolnieniu pliku przez zwycięzcę nikt nie może go zablokować

Cache lokalny

- Master wie, kto potencjalnie może utrzymywać pamięć podręczną
- Tuż przed wykonaniem zapisu informuje klientów, że dane się zmieniły
- Oczekuje na komplet potwierdzeń przed właściwym zapisem
- Cache lokalny może być utrzymywany tylko ustalony czas
- Czas ten jest jednocześnie maksymalnym czasem oczekiwania na potwierdzenie
- Klient, który nie zdąży potwierdzić musi wyczyścić cache

Cache lokalny cd.

- W trakcie notyfikowania klientów plik jest niecache'owalny
- Informacja o zmianie wysyłana jest razem z pakietem KeepAlive
- Taka realizacja zapewnia, że węzeł zawsze da się odczytać od razu
- Biblioteka potrafi również przetrzymywać wskaźniki do plików, ograniczając zdalne wywołania Open()
- Możliwe jest przechowywanie blokad, analogicznie do systemu OpenVMS

KeepAlive

- Sesja klient-serwer utrzymywana jest przez okres lease-time
- Aby przedłużyć sesję master wysyła periodycznie do klienta pakiet KeepAlive
- Klient jest zobowiązany potwierdzić jego odbiór natychmiast
- Typowo KeepAlive wysyłany jest z ustalonym odstępem
- Serwer może jednak wysłać go wcześniej, jeśli chce dołączyć do niego inne informacje
- Razem z pakietem KeepAlive można wysłać informację o zajściu zdarzenia lub zapisie danych

KeepAlive cd.

- Klient (biblioteka) też odlicza czas
- Jeśli serwer nie wyśle KeepAlive przez długi czas, sesja przechodzi w stan zagrożenia (jeopardy)
- Licząc czas klient musi brać poprawkę na opóźnienia transmisji
- W stanie zagrożenia klienta opróżnia i wyłącza cache lokalny
- Następnie odczekuje kolejny ustalony czas (grace-period)
- Jeśli w tym czasie wymieni z serwerem KeepAlive - wszystko wraca do normy
- W przeciwnym wypadku - uznaje sesję za utraconą

KeepAlive cd.

- Biblioteka potrafi notyfikować aplikację o zagrożeniu i o przywróceniu łączności
- W przypadku utraty sesji, wszystkie operacje poczynając od pewnej są przerywane
- Gwarantuje to spójny ciąg poprawnych i błędnych wywołań

Awaria repliki

- Jeśli replika zniknie na kilka godzin, wybierana jest nowa z puli zapasowych
- Master periodycznie odpytuje serwer DNS, więc sam zauważy zmianę
- Uaktualnia listę replik w bazie danych, za pomocą zwykłego protokołu powielania
- Nowa replika w międzyczasie kompletuje dane z kopii bazy danych i od pozostałych

Awaria serwera master

- W przypadku awarii wybierany jest nowy serwer master drogą elekcji
- W tym czasie nie jest odliczany lease-time
- Istnieje spora szansa, że nowy master zostanie wybrany, zanim sesja klienta stanie się zagrożona
- Istnieje jeszcze większa szansa, że Chubby dokona wymiany w czasie grace-period

Awaria serwera master cd.

- Wszystkie dane (m. in. o otwartych sesjach) są zabezpieczone replikacją
- Master zwiększa licznik epoch no.; wszystkie wywołania ze starszym numerem są odrzucane
- Notyfikuje także klientów o awarii; klienci muszą m. in. opróżnić cache lokalny
- Klienci muszą potwierdzić odbiór wiadomości lub ich sesja wygasa
- Po dodatkowym ustalonym czasie serwer usuwa nieużywane pliki ulotne

Kopie zapasowe

- Co kilka godzin Chubby zapisuje obraz bazy danych do serwera GFS
- Typowo serwer kopii zapasowej powinien znajdować się w innym budynku
- Zapis w tym samym budynku groziłby użyciem tego samego serwera Chubby
- Kopie zapasowe używane są m. in. aktywowanie nowej repliki bez obciążania już pracujących

Statystyki

Statystyki pracy typowego serwera w losowym momencie:

- Ostatnia awaria serwera master: 18 dni temu
- Czas wymiany serwera master: 14 s
- Klientów: 54k
- Otwartych plików: 12k
- Wpisów *plik w pamięci podręcznej*: 230k
- Blokad do zapisu: 1k
- Blokad do odczytu: 0

Statystyki

- Katalogów: 8k
- Katalogów ulotnych: 0.1
- Plików: 22k
- Plików $<$ 1kB: 90
- Plików $>$ 10kB: 0.2
- Plików ulotnych: 3
- Prędkość RPC (10 minut): 1-2k/s
- Pakietów KeepAlive: 93
- Pakietów GetStat: 2

Wnioski wyciągnięte przez Google

- Programiści za bardzo zakładają stabilność usługi
- Programiści traktują informację o awarii jako błąd krytyczny
- Gruboziarnistość blokad jest wymagana przy optymalizacji aplikacji
- Brakuje operacji Cancel(), która nie niszczyłaby metadanych
- Łączenie zdarzeń z KeepAlive działa sprawnie, ale wymaga użycia UDP
- Być może powstanie więc operacja GetEvent() oparta na TCP, równoległa do KeepAlive

Bibliografia

- Wiki: Distributed Lock Manager, Files-11, OpenVMS, Paxos
- http://www.arcs.us/
- http://h71000.www7.hp.com/index.html
- http://labs.hoffmanlabs.com/node/492
- http://www.pottsoft.com/vms/vms_tutorial.html
- http://labs.google.com/papers/chubby-osdi06.pdf