PORTFOLIO:

Przygotowanie koncepcji mechanizmów synchronizacji danych w projekcie XTRF

Autorzy: Grzegorz Rogus, Michał Turek

Centrum Inteligentnych Systemów Informatycznych Akademia Górniczo-Hutnicza im. Stanisława Staszica al. Mickiewicza 30, 30-059 Kraków
budynek C-2 pokój 426 tel.: 12 617 44 53 www.isi.agh.edu.pl **isi@agh.edu.pl**

Projekt współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Kraków 21 września 2014

SPIN - raport z prac nad projektem:

Przygotowanie koncepcji mechanizmów synchronizacji danych w projekcie XTRF.

Grzegorz Rogus, Michał Turek

1. Wstęp

Celem projektu XTRF jest opracowanie optymalnej architektury rozwiązanie oraz dopasowanie odpowiednich procesów komunikacji i synchronizacji danych w ramach tej architektury, umożliwiające dostosowanie systemu XTRF™ do funkcjonowania w modelu wysoce rozproszonym o wysokim poziomie dostępności.

Obecnie system XTRF funkcjonuje w zarówno w infrastrukturze chmury obliczeniowej (Cloud/SaaS) jak i jako rozwiązanie lokalne instalowane w środowisku docelowym klienta.

Podstawowym problemem aktualnie rozwijanej architektury jest zapewnienia wysokiej dostępności zasobów i funkcjonalności. Zdarzają się sytuacje, że w wyniku awarii jednego z węzłów systemu użytkownicy końcowi pozbawieni są dostępu do całości lub części funkcjonalność systemu. Docelowy model działania przewiduje, w sytuacji wystąpienia awarii systemu zastąpienie brakujących fragmentów infrastruktury nowymi instancjami.

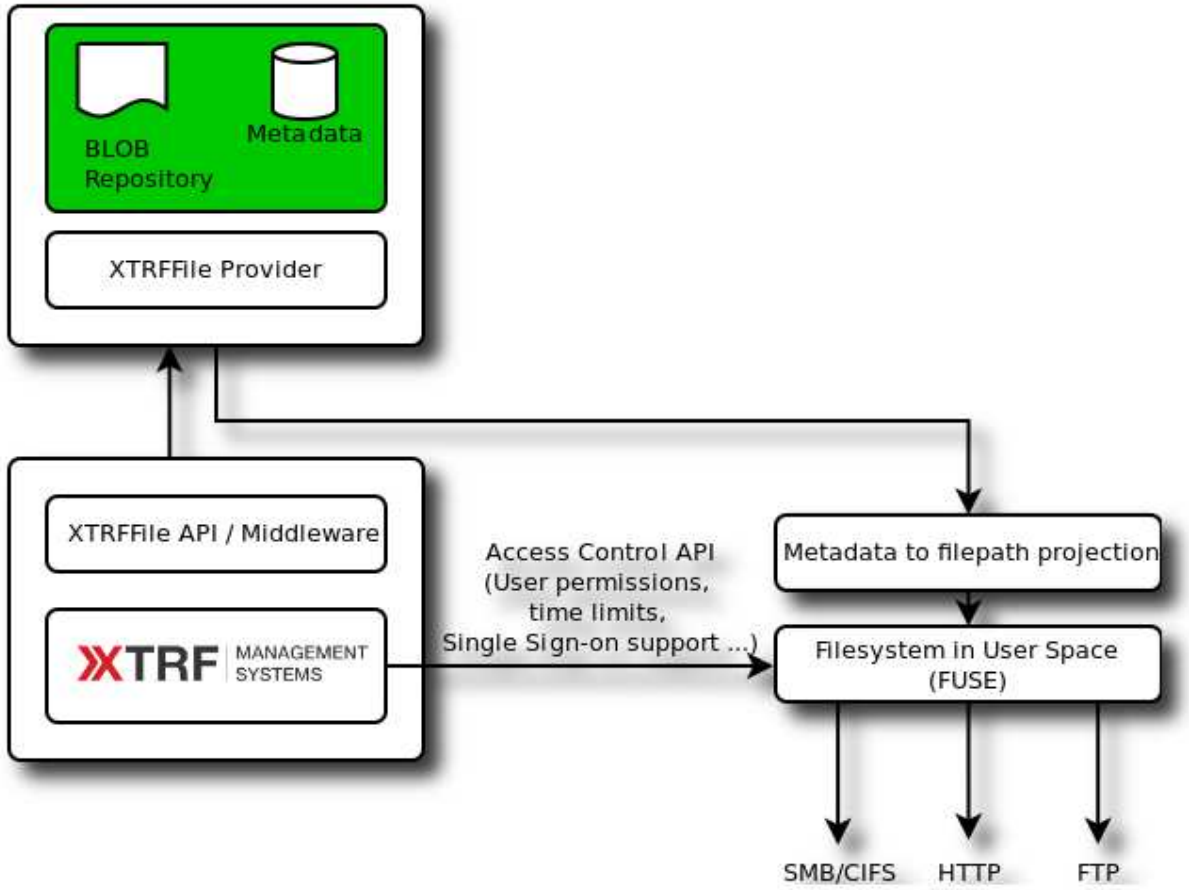
Zasadniczym problemem, który należało rozwiązać jest opracowanie optymalnej organizacji infrastruktury, która umożliwi faktyczną skalowalność i bezpieczeństwo działania systemu poprzez heterogeniczność czyli możliwość uruchomienia równoległego u różnych dostawców. Organizacja infrastruktury systemu obejmuje optymalizację ilości i rozmieszczenia zasobów obejmujących klastry złożone z odpowiednio rozmieszczonych węzłów połączonych siecią, rozproszoną bazę danych i system monitoringu umożliwiający szybkie wykrywanie awarii i udostępnienie nowej instancji aplikacji.

Kolejnym zadaniem jest opracowanie odpowiednich schematów i mechanizmów komunikacji między elementami infrastruktury, które będą zsynchronizowane w sposób umożliwiający płynne przełączanie się między odpowiednimi instancjami w razie wystąpienia problemów z funkcjonowaniem systemu. Inherentne ograniczenia współczesnej infrastruktury chmury publicznej (IaaS i PaaS) sprawiają, że konieczne jest przezwyciężenie tych ograniczeń na poziomie aplikacji i zaszycie odpowiednich mechanizmów jako dodatkowej warstwy w samym systemie.

2. Propozycja Architektury dla systemu synchronizacji danych w XTRF

Zasobami, które w systemie XTRF muszą podlegać synchronizacji są pliki. Typowy użytkownik systemu XTRF posiada dużą liczbę (do 1 TB) stosunkowo małych plików.

Ze względu na charakter wymagań stawianych dla filesystemu – część zarówno mocno techniczna (replikowalność, rozproszenie, wydajność itp.) jak i biznesowa (zarządzanie złożone dostępem, struktura plików i katalogów odpowiadająca strukturze projektów itd.) zaproponowano warstwową architekturę rozwiązania.



Rys.1 Architektura systemu synchronizacji plików w systemie XTRF

Architektura rozwiązania składa się z dwóch warstw wydzielonych ze względu na kompetencje:

1. warstwa fizyczna (na diagramie, zielone pole)

Jej celem jest fizyczna replikacja plików pomiędzy instancjami.

- bazuje na rozproszonym, wydajnym, replikowanym systemie pojedynczych plików

- zawiera repozytorium metadanych do w/w plików

Przykład realizacji warstwy: NFS dla plików leżących plasko w 1 katalogu; nazwy plików to UUID, RDBMS dla metadanych mapujący UUID na te metadane.

Tu wydajność dla zadań batchowych (replikacja, backup itp.) jest KLUCZOWA.

2 - warstwa biznesowa

- warstwa umożliwiająca dostęp do zapisu do 1szej warstwy w sposób abstrakcyjny - poprzez metadane

- pliki z 1szej warstwy są uwidaczniane na zewnątrz poprzez warstwę dokonującą zamiany metadanych na określoną ścieżkę dostępu.

Implementacja mapowania realizowana jest przez mechanizm korzystający z FUSE. Pozwoli to w sposób trywialny dostarczyć dowolnie skomplikowane mechanizmy praw dostępu - (per user, na jaki okres czasu, ile odczytów itp. itd.). Protokoły dostępowe - HTTP, FTP, SMB itd. będą wykorzystywać ten wirtualny filesystem bez żadnych dodatkowych elementów.

Tu wydajność - wyznaczana rytmem działań użytkownika NIE jest KLUCZOWA.

Takie podejście, łatwo da się też łatwo wpasować w Command Query Responsibility Segregation, gdzie poszczególne projekcje metadanych na wirtualny filesystem w przestrzeni użytkownika są realizacją Query service.

Realizacja warstwy pierwszej możliwa jest na kilka sposobów:

A) Dane i metadane trzymane RAZEM w 1 replikowanej bazie danych:
   Np. blob w postgresql, MongoDB ew. inne rozwiązania NoSQL
B) Dane trzymane na filesystemie niesieciowym, metadane w postgresql,
   zastosowana replikacja typu lazy albo na żądanie (zdarzenia w modelu CQRS opartym o Event Sourcing)

C) Dane trzymane na filesystemie sieciowym, metadane w replikowanej bazie danych

Ze względu na fakt, że realizacja warstwy fizycznej może zostać zrealizowana na różne sposoby oraz aby zaproponowane rozwiązanie było maksymalnie nieinwazyjne z punktu widzenia istniejącego systemu XTRF - wydzielono *warstwę XTRFFile API* pełniąca rolę pośrednika pomiędzy warstwą fizyczną a XTRF. Dzięki wydzielonemu komponentowi typu middleware możliwe jest dynamiczne dodawanie nowych technologii realizujących wymagania warstwy fizycznej. W tym celu wystarczy tylko komponent XTRF_API rozszerzyć o funkcje API dla nowo dodanej technologii realizujące wymagania warstwy fizycznej. Pozwoli to na wdrażanie systemu XTRF również w infrastrukturze z narzuconymi ograniczeniami technologicznymi .

3. Stworzenia warstwy unifikującej wykorzystanie rozproszonych systemów do składowania danych

W związku z zapotrzebowaniem firmy XTRF na uniwersalny komponent umożliwiający wspieranie funkcjonalności systemu ewidencjonowania i składowania dokumentów (o tej samej nazwie: XTRF) podjęto działania mające na celu opracowanie mechanizmów, które umożliwią wykorzystanie popularnych systemów DFS (Dsitributed File System) jako nośnika danych wspierającego XTRF. Komponent ten będzie agregował "wtyczki", udostępniające poszczególne DSF dla XTRF.

W ramach prac nad projektem przeprowadzono analizy w następujących kierunkach:

- Analiza wysoko poziomowych wymagań dotyczących obróbki dokumentów, stawianych systemowi XTRF w jego wersjach obecnych i planowanych przyszłych. Wymagania te mają znaczenie przy projektowaniu funkcjonalności dalszych rozwiązań, będących przedmiotem rozważań - na ich podstawie ustabilizowana zostanie funkcjonalność.
- Określenie standardowej funkcjonalności DFS, oraz koniecznych jej rozszerzeń, aby można było nadbudować funkcjonalność wymaganą przez XTRF i utworzyć wtyczkę.
- Wstępny przegląd i analiza systemów oraz technologii mogących brać udział w projekcie jako warstwa fizyczna proponowanego systemu synchronizacji plików miedzy rozproszonymi lokalizacjami.
- Walidacja wytypowanych przez XTFR systemów DFS pod kątem:
    - Posiadania API umożliwiającego utworzenie wtyczki udostępniającej funkcjonalność
    - Poprawności funkcjonowania API w praktyce (uruchomienie modułu na bazie API, sprawdzenie puli języków/technologii w których API jest dostępne - w tym integracja i testowe uruchomienie, sprawdzenie poprawności działania wszystkich koniecznych funkcji API)

4. Analiza wysoko poziomowych wymagań dotyczących obróbki dokumentów, stawianych systemowi XTRF oraz projekt interfejsów komunikacyjnych pomiędzy middleware i wtyczkami

Przedmiotem rozważań jest określenie interfejsów do tworzonej warstwy systemu XTRF umożliwiających użytkowanie wtyczek. Wtyczki te mają zapewnić proces unifikowania funkcjonalności udostępnianej systemowi XTRF przez rozmaite dostępne obecnie na rynku rozwiązania implementujące DFS (Distributed File System). W ramach wtyczek tworzone będą konwertery adaptujące kolejny DFS do wymagań XTRF - z emulacją odpowiedniej funkcjonalności na podstawie otrzymanej od DFS lub (przynajmniej) rozliczeniem jej braków.

Wyjściem do rozważań nad kształtem interfejsów dla wspomnianych wtyczek są dwa źródła informacji:

- Postawione wymagania specjalne dedykowane dla systemu XTRF, a w tym:
    - Niskopoziomowe interpretowanie (w celach optymalizacji wydajnościowej) rozszerzonych atrybutów czasowych dotyczących plików: pliki, przynależąc do różnych

projektów klienta, będą mogły zostać przez niego zakwalifikowane jako należące do projektu już zamkniętego - co sugeruje archiwizację pliku z utrzymaniem opcji dalszego dostępu do niego (po dekompresji)
  - Otwarcie możliwości strumieniowania z zasobów plikowych z buforowaniem strumienia w systemie XTRF. Dodatkowo umożliwienie utrzymania strumieniowania ciągłego (np. wprowadzanie przesunięcia czasowego dla radia internetowego czy transmisji telekonferencji)
  - Wprowadzenie hierarchicznego systemu zarządzania uprawnieniami do plików - administrator globalny systemu oraz zarządca uprawnieniami użytkowników po stronie klienta powinni dysponować własnymi strukturami danych do konfiguracji (dotyczącymi kontroli dostępu do innej funkcjonalności)
  - Możliwość przenoszenia dużych struktur katalogów klienta pomiędzy różnymi jego "projektami".
- Standardowa funkcjonalność konieczna do zapewniania usług typowego DSF

4.1. Funkcjonalność wymagana przez XTRF

4.1.1 Archiwizacja projektów już zakończonych (oraz plików do nich przypisanych)

Cel: optymalizacja systemu zakładająca archiwizację projektów zakończonych - z udostępnieniem użytkownikowi interfejsu umożliwiającego oznakowanie tych projektów.

Konieczne będzie przekazanie z XTRF informacji o przynależności pliku do projektu. W przypadku gdy plik zostanie zakwalifikowany do projektu archiwalnego - nastąpi dodanie go do archiwum. Odczyt z takiego pliku będzie wymagał dekompresji w tle.

Opis przypadków użycia (diagramowanie pominięto z uwagi na znikomą ilość relacji wiążących przypadku użycia /include, extend/ na tym etapie analizy wymagań):

-> zarejestruj plik w projekcie (plik, projekt)

-> wyrejestruj plik z projektu (plik)

<- status zlecenia

-> zarejestruj katalog w projekcie (katalog, projekt)

-> wyrejestruj katalog z projektu (katalog)

<- status zlecenia

-> sprawdź przynależność pliku do projektu (plik, projekt)

<- informacja o przynależności

-> oznacz projekt jako archiwalny (projekt)

-> oznacz projekt bieżący /wycofaj z archiwum/ (projekt)

<- status zlecenia

4.1.2 Strumieniowanie z zasobów plikowych (z buforowaniem)

Cel: umożliwienie użytkownikom składowania nieskończenie długich strumieni danych, pozwalających na strumieniowanie transmisji audio/video lub inne podobne zastosowania

Najbardziej skutecznym sposobem implementacji usługi strumieniowania jest wprowadzenie dodatkowego atrybutu będącego numerem sekwencji dla danych w strumieniu i przetwarzanie bufora zawierającego fragment strumienia w oparciu o ten numer. Funkcja czytająca z pliku będzie korzystała z rosnącego numeru sekwencji. Wartość numeru sekwencji będzie zwiększana przez operację usuwania przeterminowanych danych, numer sekwencji powiększony o wielkość przechowywanego bufora będzie wskazywał na dane najnowsze. Przypadek szczególny, który należy obsłużyć: przelanie wartości numeru sekwencji.

Numer sekwencji może zostać uzupełniony o datownik czasu rzeczywistego (gdy szerokość bitowa strumienia multimedialnego nie jest stała w czasie).

Dodatkowo cenne będzie uwzględnienie konieczności szybkiego wysłania zawartości nowego bufora danych multimedialnych po wykonaniu funkcji seek() przez użytkownika (co powinno być obsługiwane niskopoziomowo)

Możliwe tryby odczytu będą się sprowadzały do następujących:

- Czytanie z użyciem numeru sekwencji - przy otwieraniu pliku zostanie przekazana aktualna wartość numeru sekwencji odpowiadająca początkowi bufora (wcześniejsze dane już zostały skasowane). Moduł klienta będzie czytał dane posługując się numerem sekwencji (zamiast jedynie standardowym przesunięciem bajtowym w pliku). Wtyczka będzie dokonywała mapowania tych wartości .

- Czytanie zwykłe - ten tryb będzie nadal kompatybilny z systemem XTRF (pomimo przetworzenia pliku na strumień bufor strumieniowania to nadal zwykły plik). Będzie zatem można np. wielokrotnie otwierać plik i czytając z jego początku pobierać bieżącą wartość danych z początku buforowanego strumienia - np. klatkę animacji materiału przekazywanego przez strumień.

Opis przypadków użycia (jak poprzednio, diagramowanie pominięto z uwagi na znikomą ilość relacji wiążących przypadki użycia):

Zarządzanie:

-> przetwórz plik do postaci strumienia

<- status zlecenia

-> przetwórz strumień na zwykły plik

<- status zlecenia

-> załóż plik z danymi do strumieniowania /strumień/

<- status zlecenia

-> usuń plik z danymi do strumieniowania

<- status zlecenia

-> wyszukaj plik z danymi do strumieniowania

<- status zlecenia

Użytkowanie:

-> otwórz plik z pobraniem numeru sekwencji (plik)

<- status zlecenia

-> czytaj dane z bufora (numer sekwencji, ilość)

<- status zlecenia i dane

-> czytaj w trybie wysokiego priorytetu po seek (numer sekwencji, ilość)

<- status zlecenia i dane odczytane z wysokim priorytetem\

-> usuń dane na początku i przesuń bazowy numer sekwencji (ilość)

<- status zlecenia

-> dopisz dane na końcu (dane)

<- status zlecenia

4.1.3. Zarządzanie (prowadzone przez klientów) własnymi bazami użytkowników w celu kontrolowania dostępu plików i projektów

Cel: Dodanie funkcjonalności umożliwiającej klientom XTRF (administratorom) kontrolowanie dostępu do posiadanych plików - z utrzymywaniem autoryzacji i niezaprzeczalności komunikacji dla definiowanych przez siebie użytkowników.

W chwili obecnej brak jest informacji o środkach technicznych (baza danych itp.) mogących wspierać taką funkcjonalność po stronie XTRF

4.1.4. Modyfikowanie drzew katalogów przypisanych do projektów definiowanych przez użytkowników systemu XTRF

Cel: Umożliwienie użytkownikom przenoszenia kompletnych drzew katalogów pomiędzy projektami.

Przypadku użycia:

-> przenieś katalog do projektu (katalog, projekt)

<- status zlecenia

Zbiór pozostałych przypadków użycia koniecznych wsparcia tej funkcjonalności zawarty jest w zbiorze opisanym w pkt. 4.1.1: Archiwizacja projektów.

5. Standardowa funkcjonalność konieczna do zapewniania usług DFS

Definicja interfejsu opisującego funkcjonalność uniwersalną dla DFS powinna mieć oczywiste oparcie w dostępnych standardach - istnieje spore prawdopodobieństwo, że DFS udostępniający funkcjonalność dla przyszłych wtyczek będzie korzystał ze standardowych protokołów udostępniania plików więc treść wtyczki będzie musiała zostać do nich zaadaptowana.

W większości przypadków wtyczka, aby dostarczyć funkcjonalność wykraczającą poza standardowy DSF (czyli tu dedykowaną dla XTRF), będzie musiała wtórnie zwrócić sie do XTRF o udostępnienie usługi. Interfejs pomiędzy projektowaną warstwą, a wtyczką można więc podzielić:

- na służący do dostarczania faktycznych usług związanych obsługą plików,
- na zapewniający środowisko XRTF dla samej wtyczki.

Niektóre przypadki, w których interakcja XTRF z wtyczką będzie dodatkowo potrzebna:

- Zarządzanie przypisaniami plików do projektów oraz zlecenia archiwizacji projektów
- Zarządzanie numerami sekwencji w przypadku operowania na plikach ze strumieniowaniem - numery te trzeba przechowywać w bazie danych poza DFS
- Zarządzanie danymi własnych użytkowników przez administratorów poszczególnych klientów systemu XTRF

5.1. Dobór zestawu operacji wymaganych dla standardowych usług DFS

Zestaw czynności (przypadków użycia) wymagających unifikowania w ramach wtyczki.
Diagramowanie (jak poprzednio) pominięto z uwagi na znikomą ilość relacji wiążących przypadki użycia:

Operacje na katalogach:

-> aktywuj katalog /chdir i lock/ (katalog)

<- status zlecenia

-> deaktywuj katalog /unlock/ (katalog)

<- status zlecenia

-> podaj zawartość katalogu (katalog)

<- status zlecenia

-> stwórz katalog (katalog)

<- status zlecenia

-> usuń katalog (katalog)

<- status zlecenia

-> czytaj atrybuty katalogu

<- status zlecenia

-> zapisz atrybuty katalogu

<- status zlecenia

Operacje na plikach:

-> otwórz plik (uchwyt, przesunięcie)

<- status zlecenia

-> zamknij plik (uchwyt)

<- status zlecenia

-> trwale blokuj plik (plik, tryb blokady)

<- status zlecenia

-> odblokuj plik (plik, tryb blokady)

<- status zlecenia

-> sprawdź blokadę pliku (plik)

<- stan blokady i status zlecenia

-> czytaj dane z pliku (uchwyt, przesunięcie)

<- dane

-> zapisz dane do pliku (uchwyt, dane)

<- status zlecenia

-> czytaj uprawnienia (plik)

<- status zlecenia

-> zapisz uprawnienia (plik)

<- status zlecenia

-> zmień właściciela (plik, użytkownik)

<- status zlecenia

-> pobierz właściciela (plik)

<- dane właściciela i status zlecenia

-> czytaj atrybuty pliku (plik)

<- status zlecenia i flagi atrybutów

-> zapisz atrybuty pliku (plik, flagi atrybutów)

<- status zlecenia

-> twórz plik (nazwa)

<- status zlecenia

-> usuń plik (plik)

<- status zlecenia

5.2 Możliwe wsparcie protokołami komunikacyjnymi dla DFS

NFS (Network File System)

Protokół wykorzystywany do zdalnego udostępniania plików znajdujących się na innych maszynach bez konieczności ich kopiowania. Liczne adaptacje w systemach operacyjnych i w rozmaitych produktach korzystających z DFS lub emulujących DFS. Interfejs VFS (Virtual File System)

implementowany w jądrze systemu-klienta mapuje standardowe komendy lokalnego systemu na komendy dla serwera NFS na zdalnej maszynie

V-węzły (v-nodes) systemu NFS - analogiczne do i-węzłów (i-nodes) systemu plików UNIX.

SMB (Server Message Block)

Protokół wykorzystywany do zdalnego udostępniania zasobów systemowych (pliki, urządzenia). Inna nazwa: CIFS (Common Internet File System). Protokół nadbudowany nad NetBIOS i TCP lub NetBEUI. SMB to popularne narzędzie (Serwer Plików) korzystające z SMB: SAMBA. Ponadto SMB jest szeroko wykorzystywany przez systemy operacyjne firmy Microsoft

LDAP (Lightweight Directory Access Protocol)

Protokół wspierający składowanie zasobów w hierarchicznej bazie danych. Cele: Zcentralizowane przechowywanie i dystrybucja zasobów (niekoniecznie plikowych), eliminacja kolizji nazewnictwa zasobów. LDAP jest tworzone z użyciem nazewnictwa DNS wkomponowanego we własną bazę danych. Porządkowanie treści oparte o relację komponent-kontener: Directory Context (kontener ogólnego stosowania, używany do oznaczenia domen), Organizational Unit (jednostka organizacyjna, kontener), Common Name (encja)

6. Walidacja wytypowanych przez XTFR systemów DFS pod kątem przydatności (i poprawności funkcjonowania API)

Systemy składowania danych, nadające się do integracji z systemem i wybrane jako istotne podzielono na dwie grupy:

A. systemy WWW - głównie Google Drive i Google Clout Storage

B. systemy standalone - instalowane

Głównym kryterium była tu popularność systemu, możliwość użytkowania go jako wsparcia dla realizacji funkcjonalności warstwy fizycznej.

Analizowane systemy - wymieniono tylko te które zostały pozytywne zweryfikowane pod katem API

A. Systemy (technologie) cloudowe dostępne w technologii klienta Web posiadające dedykowane API.

S3, Glacier - dostęp za pomocą REST API - http://aws.amazon.com/documentation/s3/

CloudStore - API jest - rozwiązanie oparte na GFS (Google Files System)
https://developers.google.com/storage/docs/json_api/v1/

Szczególnym przypadkiem zbudowanym na bazie CloudStore jest Google Drive

Google Drive - dostępne API - ale tylko dla podstawowych operacji
https://developers.google.com/drive/v2/reference/

Sparkleshare - API – jest http://www.programmableweb.com/api/sparkleshare

ownCloud - API – jest
http://doc.owncloud.org/server/6.0/developer_manual/app/app/api/index.html

Dodatkowo porównanie 8 systemów typu chmura mogących być wykorzystane do synchronizacji plików

.

								
Cena	$4.49	$4.49	$4.49	$4.95	$4.99	$9.99	$7.99	$9.99
Storage	Unlimited	Unlimited	Unlimited	Unlimited	60GB	50GB	125GB	25GB
Moneyback	Anytime	Yes	Anytime	Anytime	Yes	Yes	No	Yes
Free Trial	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Founded	2010	2011	2011	2011	2008	2007	2005	2005
Location	USA	UK	USA	UK	USA	USA	USA	USA

Interface	Web based / API	Web based / API	Web based / API	Web based / API	Web based / API	Web based / API	Web based / API	Web based / API
Reliablility	★★★★★	★★★★★	★★★★★	★★★★★	★★★★☆	★★★☆☆	★★★★☆	★★★★☆
Security	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★	★★☆☆☆	★★★★★	★★★★☆
Storage Space	★★★★★	★★★★★	★★★★★	★★★★★	★★☆☆☆	★★★☆☆	★★☆☆☆	★☆☆☆☆
Easy-Of-Use	★★★★★	★★★★★	★★★★★	★★★★★	★★★★☆	★★★★☆	★★★☆☆	★★★☆☆
Upload Speed	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆
Customer Care	★★★★★	★★★★★	★★★★★	★★★★★	★★★★★	★★★★☆	★★★☆☆	★★★☆☆
Multiple Computers	✔	✔	✔	✔	✔	✔	✔	✔
Scheduled Backups	✔	✔	✔	✔	✔	✔	✔	✔
File Protection	✔	✔	✔	✔	✔	✔	✔	✔
File Sharing	✔	✔	✔	✔	✔	✔	✖	✔
Anywhere	✔	✔	✔	✔	✔	✔	✔	✔

Access								
Folder Sharing	✔	✔	✔	✔	✔	✔	✔	✔
Sync Multiple Devices	✔	✔	✔	✔	✖	✔	✔	✔
Restore Deleted Files	✖	✖	✖	✖	✖	✔	✔	✔
Sync Folder	✔	✔	✔	✔	✔	✔	✔	✔
100% Automated	✔	✔	✔	✔	✔	✔	✔	✖
Remote Access Manager	✔	✖	✔	✔	✔	✔	✖	✖
Resume After Interruption	✔	✖	✔	✔	✖	✔	✖	✖
File Review	✖	✖	✖	✖	✖	✔	✖	✔
Unlimited Backup Storage	✔	✔	✔	✔	✖	✖	✖	✖

File Versioning	✔	✔	✔	✔	✔	✖	✔	✔
Referral Program	✔	✔	✔	✔	✔	✔	✔	✔
Encrypted	✔	✔	✔	✔	✔	✔	✔	✔
Idle Backups	✖	✔	✖	✖	✔	✔	✔	✔
Selective Backups	✔	✔	✔	✔	✔	✔	✔	✔
Geo-Redundant Storage	✔	✔	✔	✔	✔	✖	✔	✔
24/7 Support	✔	✔	✔	✔	✔	✔	✔	✔
SSL Secure Data Transfer	✔	✔	✔	✔	✔	✔	✔	✔
User Generated Encryption Key	✖	✖	✖	✖	✖	✖	✖	✖
Multilevel File	✔	✔	✔	✔	✔	✔	✔	✔

Security								
Geo-Redundant Storage	✔	✖	✔	✔	✖	✖	✖	✖
256-bit Encryption	✔	✔	✔	✔	✔	✔	✔	✔
Encrypted Storage	✔	✔	✔	✔	✔	✔	✔	✔

B. Rozproszone systemy plikowe - systemy typu standalone - instalowane lokalnie

Po wstępnej analizie wybrane zostały do bardziej szczegółowych testów następujące systemy:
Cosmos,
General Parallel File System(GPFS)
GlusterFS
Lustre
MogileFS
Chiron FS

Testy będą przeprowadzone dla dwóch różnych zestawów danych: z przewagą dużej ilości małych plików oraz małej ilości dużych plików.

Interesujące nas zagadnienia do przetestowania to:

- dostępne metody synchronizacji plików (jaka replikacja? Całościowa czy przyrostowa? Czy konfigurowalna – w jaki sposób)

- współistnienie z siecią podzieloną firewallami
- wsparcie dla eventual consistency
- wersjonowanie

- metody udostępniania plików (całe katalogi czy tylko pojedyncze pliki)
- backupowanie
- wyszukiwanie

6.1 Sprawdzenie funkcjonowania bibliotek udostępnianych przez Google do obsługi CLOUD STORAGE I GOOGLE DRIVE [1]

W ramach prowadzonych testów sprawdzono większość API oraz gotowych narzędzi pod kątem:

- pokrycia funkcjonalności logowania na podstawie konta google

- przetwarzania API KEY (posługiwanie się API KEY polega na jego wygenerowaniu i podawaniu przy dalszych żądaniach dostępu do danego obiektu)

- zapisu obiektu

- odczytu obiektu

Odrzucono jedynie te, które nadbudowane są nad technologami niszowymi lub nie figurującymi na liście wspieranych przez inne DFS (finalnie - konieczne będzie określenie platformy, nad którą moduł wtyczek będzie działał).

6.1.1 Użyte w pracach i sprawdzone metody dostępu do CLOUD STORAGE

Dostęp przy użyciu XHTML z użyciem przeglądarki.

Sprawdzono także na własnoręcznie napisanym kliencie HTTP (czy przeglądarka nie dorzuca elementów środowiska, które pominięto w przykładach Google API). Rozwiązanie działa - sprowadza się do otwierania strumienia HTTP, wysyłania żądania komenda GET lub POST i podawania URL typu:

storage.googleapis.com/<bucket>/<object>

<bucket>.storage.googleapis.com/<object>

Różnorodne techniki dostępu w technologii JSON (JavaScript Object Access):

Pierwszy test dotyczył użycia XMLHttpRequest i JSON.parse() do obiektu podanego w XML - nie działało

Kolejny: z użyciem Google APIs Client Library for JavaScript. Działa tylko tak:

Pobranie API:

```
<script src="https://apis.google.com/js/client.js?onload=load"></script>
```

następnie:

```
gapi.client.setApiKey('KLUCZ');

gapi.client.load('link', 'v1', Pobranie);
```

i tworzymy callback Pobranie, a w nim

```
var request = gapi.client.urlshortener.url.get({
  'shortUrl': 'http://linkdoobiektu'
});
```

Autoryzacja przy JavaScript: należy pobrać API key do OAuth v1 lub v2

Biblioteki .NET

Działają poprawnie - testowano na .NET 2.0 i .NET 3.5 w projekcie ASP.NET oraz w aplikacji stand-alone.

Istnieją problemy z funkcjonowaniem OAuth v1 i OAuth v2 w .NET - konieczne jest posiadanie indywidualnej wersji platformy .NET dla każdej z wersji OAuth. Użycie OAuth w dwóch wersjach nad jednym .NET blokuje go.

Biblioteki dla JAVA

Działają poprawnie - przykłady Google sprawdzano nad JDK 1.7. Były problemy z kompilacją na starszych wersjach Javy (testowano na J2SDK 1.4)

Biblioteki dla PYTHON

Testowano jedynie narzędzie gsutil funkcjonujące nad PYTHON (opis narzędzia później) z założeniem, że dostęp z API musi działać, skoro działało to narzędzie

Biblioteki dla PHP

Testowano nad Apache 2.2 i PHP5. Moduł nie ładuje się z php.ini, zapewne na skutek złej wersji PHP.

Narzędzie gsutil

Narzędzie, umożliwiające dostęp z linii komend, narzędzie funkcjonuje. Testowano w środowisku Windows z Python 3.4.1 i Python 2.7.6 (konieczna wersja Python dla Windows)

Uruchomienie z linii komend: przeszły poprawnie testy logowania, upload, download obiektu

Inne API, lecz jeszcze nie testowane (gdyż istnieją tylko wersje alpha)

- Client Library for Go, http://code.google.com/p/google-api-go-client/
- Client Library for Node.js, https://github.com/google/google-api-nodejs-client/
- Client Library for Ruby, https://developers.google.com/api-client- library/ruby/start/get_started

6.1.2 Użyte w pracach i sprawdzone metody dostępu do GOOGLE DRIVE

HTTP

Przetestowano użycie protokołu HTTP od obróbki plików w Google Drive. W tym celu napisano kolejnego klienta, który jest w stanie złożyć nietypowe zapytanie HTTP (stosując nowe komendy HTTP Request). Testy prowadzono pomyślnie na przykładach:

DELETE /folder/plik - usunięcie pliku

GET /folder/plik - pobranie pliku

POST /folder/plik - dodanie pliku

GET /folder - otrzymanie listy directory browsing

Pełna przykładowa składnia (w Google Drive słowa kluczowe oznaczone kursywą decydują o typie operacji):

DELETE /files/folder/children/plik

Sprawdzono również comments, np.:

DELETE /files/plik/comments/komentarz

oraz properties, np.:

DELETE /files/plik/properties/cecha

Ich działanie może mieć kluczowe znaczenie przy markowaniu plików na potrzeby XTRF.

Biblioteki .NET

Biblioteka działa poprawnie. W niektórych instalacjach .NET trzeba ręcznie rejestrować .NET Assemblies od Google (nie działa instalator). Konieczne jest korzystanie z przestrzeni:

Google;

Google.Apis.Auth.OAuth2;

Google.Apis.Drive.v2;

Google.Apis.Drive.v2.Data;

Google.Apis.Services;

Uwaga: konstruktor obiektu typu DriveService musi dostać jakąkolwiek niepustą wartość dla właściwości ApplicationName. Inaczej nie działa.

javaScript

Skryty działają poprawnie. Należy użyć skryptu apis.google.com/js/client.js i podać swoją metodę do uruchomienia:

```
<script type="text/javascript" src="https://apis.google.com/js/client.js?onload=metoda"> </script>
```

następnie metod:

gapi.auth.authorize(...)

gapi.client.load(...)

biblioteki JAVA

Biblioteka działa poprawnie. Konieczne jest google-api-services-drive-v2-[version].jar

i przestrzeń (należy sprawdzić dostępność):

com.google.api.client.googleapis.auth.oauth2.GoogleAuthorizationCodeFlow;

com.google.api.client.googleapis.auth.oauth2.GoogleCredential;

com.google.api.client.googleapis.auth.oauth2.GoogleTokenResponse;

com.google.api.client.http.FileContent;

com.google.api.client.http.HttpTransport;

com.google.api.client.http.javanet.NetHttpTransport;

com.google.api.client.json.JsonFactory;

com.google.api.client.json.jackson2.JacksonFactory;

com.google.api.services.drive.Drive;

com.google.api.services.drive.DriveScopes;

com.google.api.services.drive.model.File;

6.1.3 Wnioski po testach

Prace miały na celu przeprowadzenie testów dających następujące wyniki:

- wstępne opracowanie wymagań dla warstwy unifikującej usługi udostępniane przez DFS i wspomniane API
- ustalenie puli gotowych API umożliwiających dostęp do popularnych systemów składowania obiektów (głównie plików)
- sprawdzenie technicznych możliwości wykorzystania API
- sprawdzenie udokumentowanej funkcjonalności API w praktyce
- dokonanie wyboru w zakresie najbardziej popularnych technologii wspierających funkcjonalnie przyszłą warstwę

Generalne wnioski z bieżących prac: konieczne jest zawężenie puli API, które należy uważać jako przydatne do implementowania poszczególnych wtyczek i celowe do dalszego prowadzenia testów. Bieżące prace polegały głównie na integrowaniu dostarczonych przez Google bibliotek w kolejnych platformach oraz w kolejnych warstwach nadbudowanych nad HTTP lub podobnymi protokołami - z udowadnianiem, że dana funkcjonalność faktycznie działa lub ma szansę działać, gdy zrezygnuje się z systemu autoryzacji lub szyfrowania. W ramach danej platformy czy technologii sama liczba możliwości uzyskania dostępu do obiektu lub po prostu pliku także często jest duża (szczególnie dotyczy to technologii skryptowych nad WWW, gdyż to jest najłatwiejsze do nadbudowania i oferta jest tu szeroka).

Testów wydajnościowych nie robiono - gdyż w większości przypadków testowana byłaby jedynie jakość łącza lub wydajność serwera, testy są sensowne jedynie przy rozwiązaniach offline (choć i tam mało miarodajne, gdyż dalej w dużym stopniu zależne od przypadku)

Większość rozwiązań działa (udało się uruchomić i wykorzystać podstawowe usługi gwarantujące dostęp do obiektu w repozytorium) - choć do uruchomienia danego rozwiązania często konieczna jest wyszukana wersja danej platformy u klienta czy kompilatora (szczególnie dotyczy to API Javy).