Rozporszone Systemy Operacyjne

Rozproszony system plików - Etap III

Skład zespołu:

Marcin Dzieżyc
Piotr Kalinowski
Adam Papros
Mateusz Statkiewicz
Marcin Swend
Jacek Witkowski

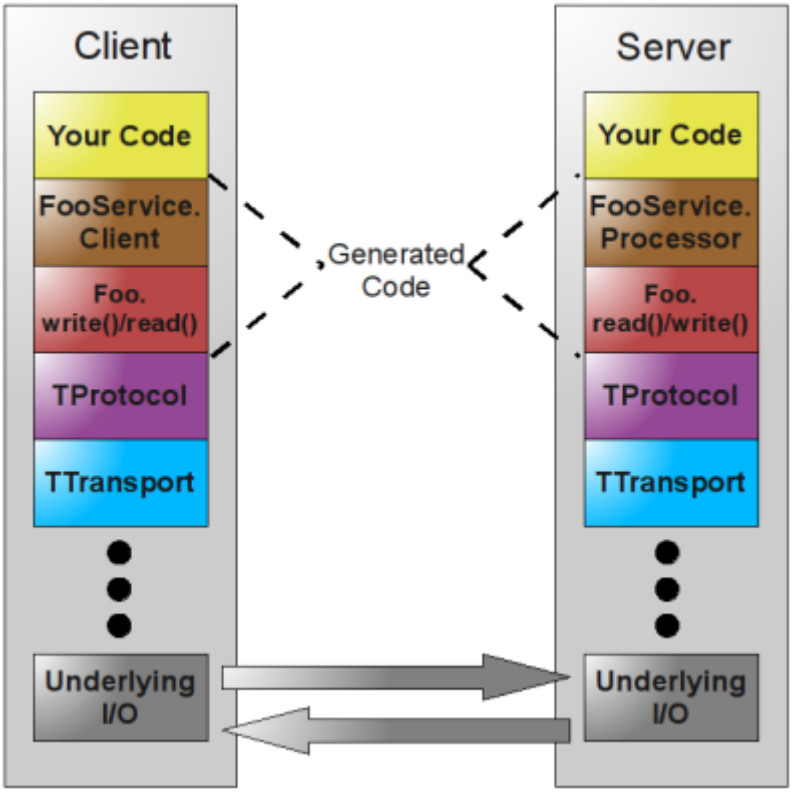
Spis treści

1. Opis rozwiązania Apache Thrift

Thrift to narzędzie, którego głównym celem jest umożliwienie tworzenia skalowanych oraz interoperacyjnych usług. Biblioteka ta pierwotnie została na otwartej licencji Apache 2.0 zaimplementowana i była używana przez Facebooka, ale aktualnie jest dostępna (http://www.apache.org/licenses/LICENSE-2.0.html).

Thrift dzięki prostemu językowi IDL (język opisu interfejsu ang. *Interface Definition Language*) pozwala definiować oraz tworzyć usługi dla wielu popularnych i rozwijanych języków programowania. W oparciu o plik w języku IDL niezależny od języka programowania Thrift generuje biblioteki służące do transportu danych pomiędzy klientem a serwerem, wystawiając do nich odpowiednie interfejsy. Programista zajmuje się jedynie implementacją faktycznego działania aplikacji w oderwaniu od sposobów przekazywania danych, co sprawia, że tworzenie rozwiązań w oparciu o RPC (zdalne wywołanie procedury, ang. *Remote Procedure Call*) jest bardzo proste.

1.1. Architektura Apache Thrift



Rysunek 1.1. Źródło: http://jnb.ociweb.com/jnb/jnbJun2009.html

Diagram przedstawia stos warstw implementacyjnych, które umożliwiają komunikację RPC zgodnie z filozofią Apache Thrift. Pierwsze trzy górne warstwy stanowią kod napisany w tym samym języku, dla którego kompilator IDL Apache Thrift wygenerował kod. Pierwsza warstwa to kod użytkownika-programisty, który wykorzystuje dostarczoną abstrakcję RPC. Dwie następne warstwy to wygenerowany kod realizujący pożądane przez użytkownika usługi. Pozostałe, niższe warstwy są niezależne od pliku opisu interfejsu - wykorzystywany jest kod bibliotek odpowiednich do realizacji komunikacji.

1.2. Apache Thrift IDL

1.2.1. Wspierane podstawowe typy danych

- **bool** - wartość logiczna
- **byte** - bajt ze znakiem
- **i16** - 16 bitowa liczba całkowita ze znakiem
- **i32** - 32 bitowa liczba całkowita ze znakiem
- **i64** - 64 bitowa liczba całkowita ze znakiem
- **double** - 64 bitowa liczba zmienno pozycyjna
- **string** - tekstowy typ danych

1.2.2. Kontenery

Thrift umożliwia definiowanie zbiorów danych przy pomocy trzech rodzajów kontenerów:

- **list**<typ> - uporządkowana kolekcja
- **set**<typ> - nieuporządkowana kolekcja
- **map**<typ_klucza, typ_wartości> - mapa klucz-wartość

1.2.3. Struktury

Thrift umożliwa definiowanie złożonych struktur danych przypominających struktury z języka C. Definiuje się je w następujący sposób:

```
1 struct <nazwa_struktury>
2 {
3   1:<typ_danych_1> <nazwa_skladowej_1>,
4   2:<typ_danych_2> <nazwa_skladowej_2>,
5   3:<typ_danych_3> <nazwa_skladowej_3>,
6 }
```

1.2.4. Wyjątki

Deklarowanie wyjątków jest podobne do definiowania struktur.

```
1 exception <nazwa_wyjatku>
2 {
3   1:<typ_danych_1> <nazwa_skladowej_1>,
4   2:<typ_danych_2> <nazwa_skladowej_2>,
5   3:<typ_danych_3> <nazwa_skladowej_3>,
6 }
```

1.3. Zalety korzystania z rozwiązania Apache Thrift

Wykorzystanie Apache Thrift posiada wiele zalet z perspektywy programisty:

- posiada czytelny dla człowieka, niezależny od języka plik opisu interfejsu,
- umożliwia zmianę protokołu i sposobu transportu danych bez poważnych zmian w kodzie - wykorzystywane są możliwości narzędzia Apache Thrift,
- zapewnia wysoką wydajność serializacji pomiędzy różnymi językami (w porównaniu do popularnych protokołów wykorzystujących XML lub JSON) przez możliwość wykorzystania protokołu binarnego,
- serwer wielowątkowy (lub jednowątkowy), wykorzystujący funkcje blokujące lub nieblokujące dostępny „prosto z pudełka”.

2. Uruchomienie dostępnych w sieci przykładów demonstrujących wykorzystanie Apache Thrift

Jednym z etapów dobrego poznania Apache Thrift, było uruchomienie dostępnych w sieci przykładów wykorzystujących to narzędzie. Kolejnym krokiem, była implementacja prostej usługi przypominającej bardzo uproszczone zadanie projektowe. Uruchomionym przykładem demonstrującym wykorzystanie Apache Thrift był „Thrift by Example" ze strony: *http://thrift-tutorial.readthedocs.org/en/latest/usage-example.html*.

Opisano sposób realizacji prostej usługi zapewniającej operacje matematyczne. Przykład wprowadza do:

- składni oraz możliwości języka opisu interfejsu dla narzędzia - Thrift IDL,
- sposobu użycia kompilatora Thrift IDL,
- sposobu implementacji aplikacji klienta w języku Java,
- sposobu implementacji aplikacji serwera wraz z odpowiednimi klasami realizującymi usługę w języku Java,
- przykład dodatkowo opisuje sposób nawiązywania bezpiecznych połaczeń (przy pomocy SSL).

Przykładowa usługa umożliwia klientowi zapytanie serwera o wynik operacji - w ramach zdalnego wywołania procedury klient przekazywał argumenty oraz typ operacji. Dodatkowo wykorzystano mechanizmy Apache Thrift umożliwiające rzucanie wyjątków (na przykład w przypadku próby dzielenia przez 0).

Demonstracja Thrift by Example wprowadza w Apache Thrift, pokazuje przykładowe zastosowanie i proponuje przykładową implementację wykorzystującą to narzędzie.

Drugim elementem wprowadzenia do Apache Thrifta była samodzielna implementacja bardzo uproszczonej wersji usługi, o której mowa w zadaniu projektowym - rozproszonego systemu plików. W ramach tych prac powstały dwie grupy usług: NamingService oraz StorageService, które odpowiadają kolejno warstwie usług lokalizacyjno-nazewniczej oraz warstwie właściwego przechowywania.

```
namespace java NamingService
typedef i32 int

service NamingService
{
  int put(1:string fileName),
  int get(1:string fileName)
}
```

Wydruk 2.1. Interfejs serwera nazewniczego

```
namespace java StorageService
typedef i32 int

service StorageService
{
        void putFile(1:int fileId, 2:list<byte> body),
        list<byte> getFile(1:int fileId)
}
```

Wydruk 2.2. Interfejs serwera przechowującego dane

Aplikacja kliencka, gdy chce wysłać plik do systemu plików, najpierw łączy się z serwerem nazewniczym - w tym momencie zostaje zrealizowane zarejestrowanie nazwy oraz przypisanie identyfikatora pliku. W drugim kroku aplikacja kliencka łączy się z serwerem przechowującym dane, przesyła identyfikator pliku (nadany przez serwer nazewniczy) oraz kolekcje bajtów (ciało pliku).

3. Przegląd istniejących rozwiązań

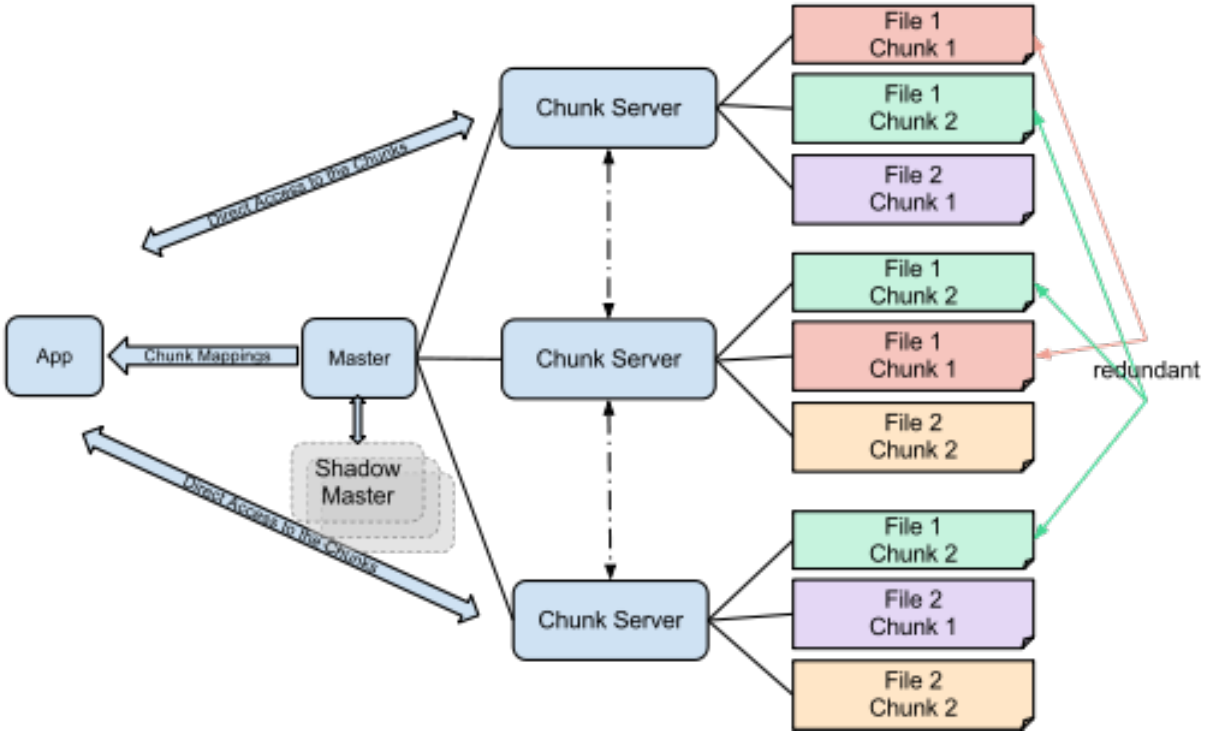
Obecnie na rynku istnieje wiele dostępnych rozproszonych systemów plików, zarówno komercyjnych, jak i darmowych. Swoje implementacje takich systemów oferuje większość znaczących firm informatycznych, takich jak Microsoft, Google, Apache, Oracle, IBM czy RedHat. Poniżej zaprezentowane zostały dwa wybrane rozproszone systemy plików.

3.1. Google File System (GFS)

GFS został zaprojektowany na wewnętrzne potrzeby firmy Google, aby przechowywać duże ilości danych związanych z pracą wyszukiwarki internetowej. W systemie wyróżnić można dwa rodzaje serwerów: Master oraz Chunkserver. Zadaniem tych drugich jest przechowywanie wszystkich plików w systemie. Pliki są gromadzone w blokach (ang. *chunk*), każdy o wielkości 64 megabajtów, które replikowane są pomiędzy kilkoma Chunkserverami (minimum trzy). Podczas operacji czytania lub przesyłania plików klient komunikuje się bezpośrednio z tymi serwerami.

Serwer Master jest z kolei odpowiedzialny za trzymanie wszystkich metadanych o plikach, czyli takich informacji jak odwzorowanie 64-bitowych etykiet plików na ich lokalizację na Chunk Serverach, lokalizację kopii bloków danych oraz informacje o aktualnych operacjach wejścia i wyjścia w systemie. Wszystkie te dane są aktualizowane poprzez periodycznie wysyłane komunikaty od każdego z serwerów.

Poniżej znajduje się schemat rozwiązania GFS:



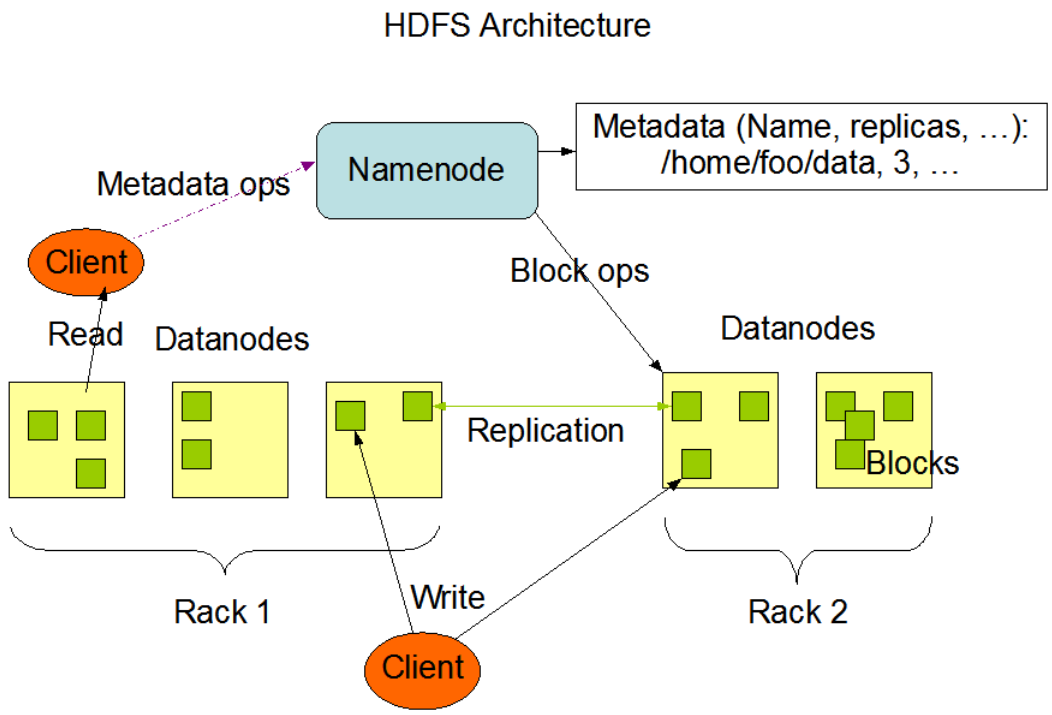
Rysunek 3.1. Źródło: http://en.wikipedia.org/wiki/Google_File_System

3.2. Hadoop File System (HDFS)

Głównymi cechami, które wyróżniają ten rozproszony system plików od innych jest wysoka odporność na błędy oraz skuteczność działania na mniej wydajnym sprzęcie. System ten wchodzi w skład rozwiązania open-source zwanego Apache Hadoop.

Podobnie jak GFS, Hadoop File System składa się z głównego serwera zwanego Namenode, który odpowiada za usługi nazewnicze i lokalizacyjne plików oraz udostępnia interfejs dla klienta. Dane są przechowywane na serwerach zwanych Datanodes. Są również pomiędzy nimi replikowane. Pliki mogą być przechowywane w całości lub we fragmentach.

Poniżej przedstawiony został schemat architektury HDFS:



Rysunek 3.2. Źródło: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

Lista innych przykładowych rozproszonych systemów plików:

- Ceph (Inktank),
- FhGFS (Fraunhofer),
- GlusterFS (RedHat),
- Lustre,
- Windows Distributed File System (Microsoft).

Na przykładzie GFS oraz HDFS można zauważyć, że ogólny zamysł architektoniczny rozproszonych systemów plików jest dość podobny. Większym różnicom podlegają szczegóły implementacyjne.

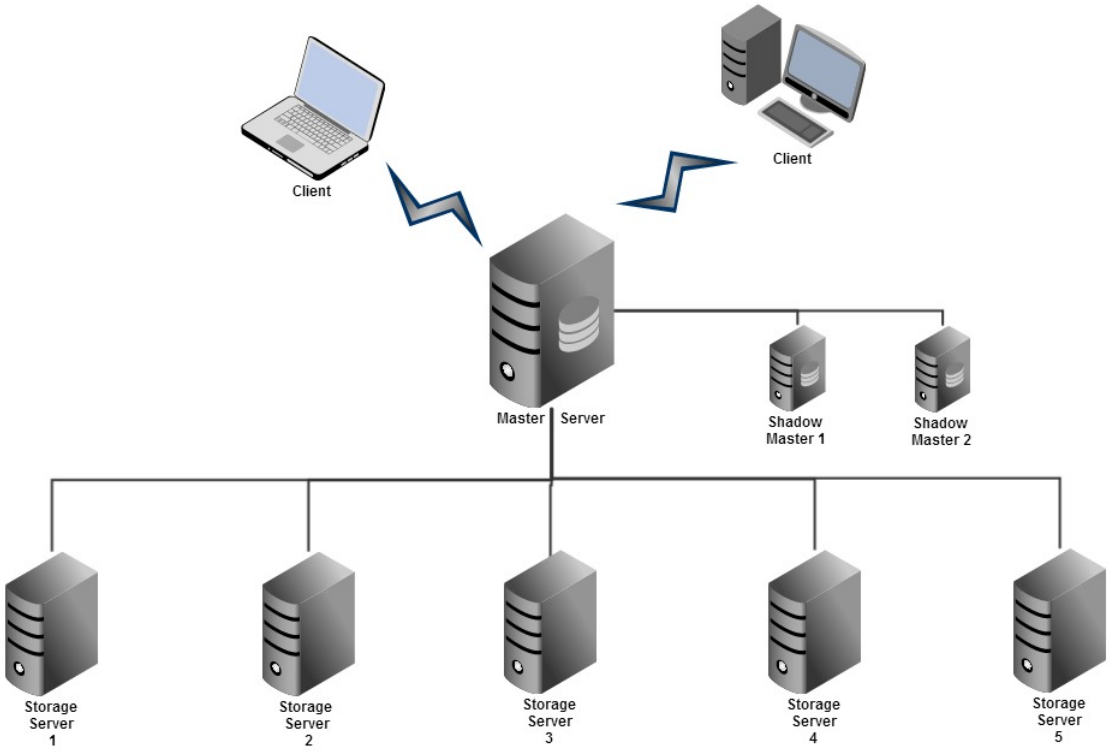
4. Koncepcja rozwiązania własnego

Zadaniem projektowym jest wytworzenie rozproszonego systemu plików przechowującego pliki w pamięci operacyjnej bądź na dyskach lokalnych. System implementowany przez nasz zespół będzie przechowywał pliki na dyskach lokalnych.

Usługa podzielona jest na 2 warstwy:

- warstwa usług lokalizacyjno-nazewniczych (Master Server),
- warstwa właściwego przechowywania (Storage Server).

System opiera się na następującej architekturze:

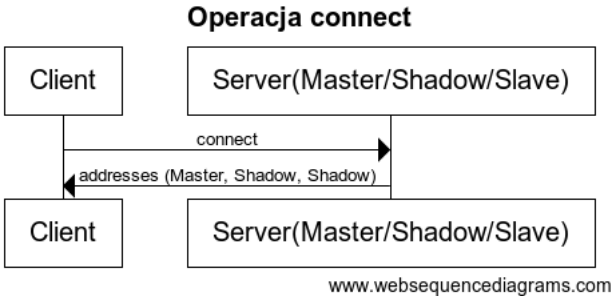


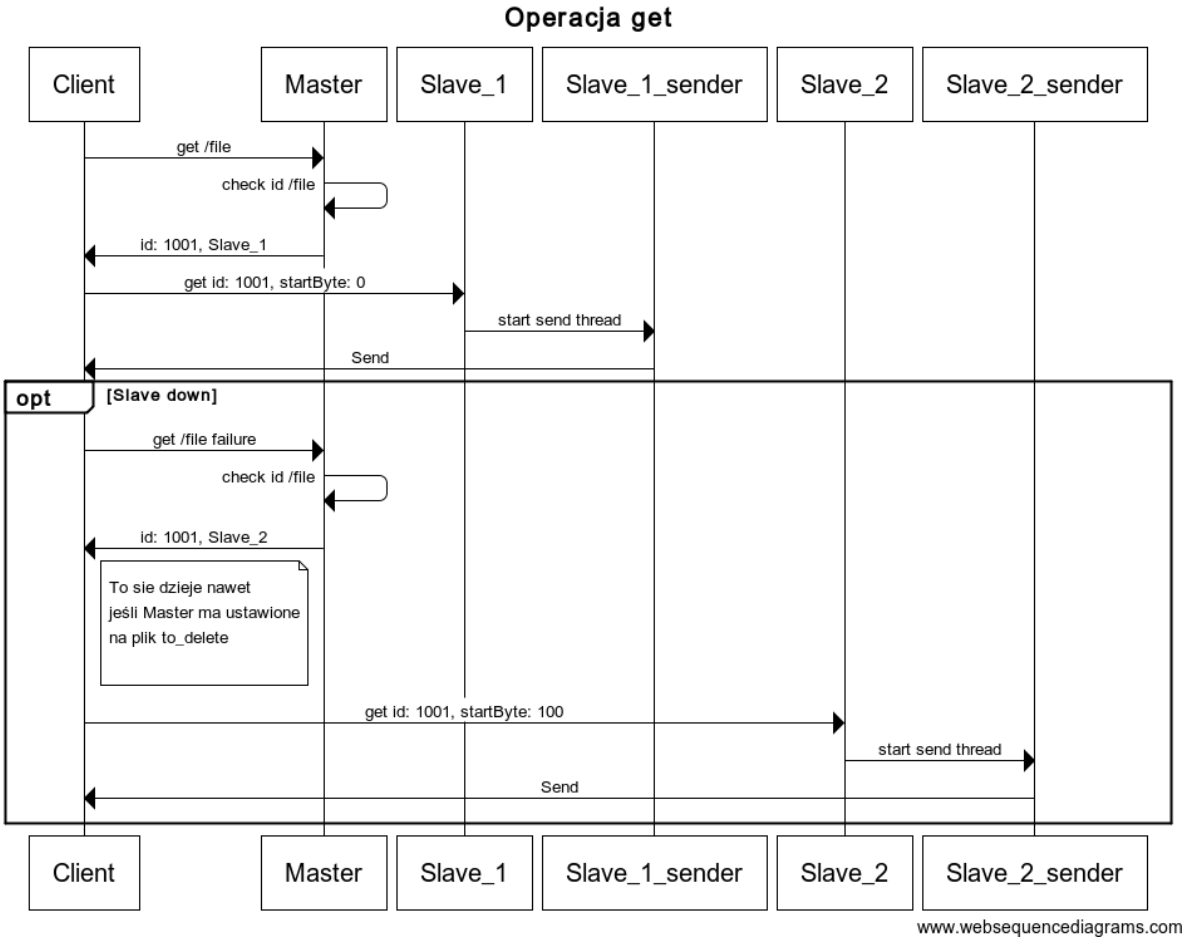
Master Server odpowiada za kontakt z klientem oraz za przydzielanie plików do odpowiednich Storage Serverów. Dodatkowo zarządza bazami danych na serwerach lustrzanych (Shadow Server).

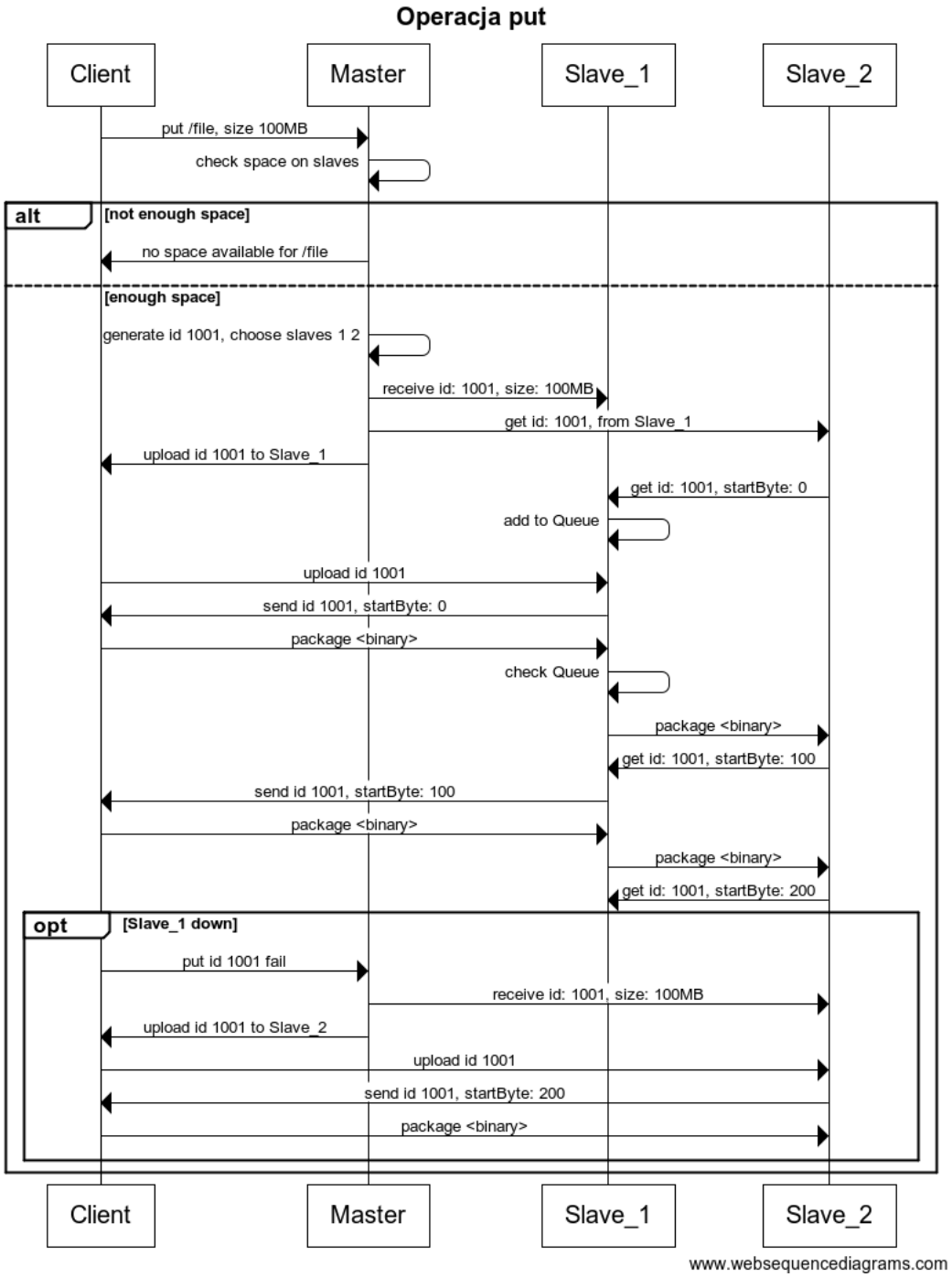
W bazach przechowywane jest drzewo plików załadowanych do systemu, wraz z ich fizycznym miejscem położenia - identyfikatorami Storage Serverów.

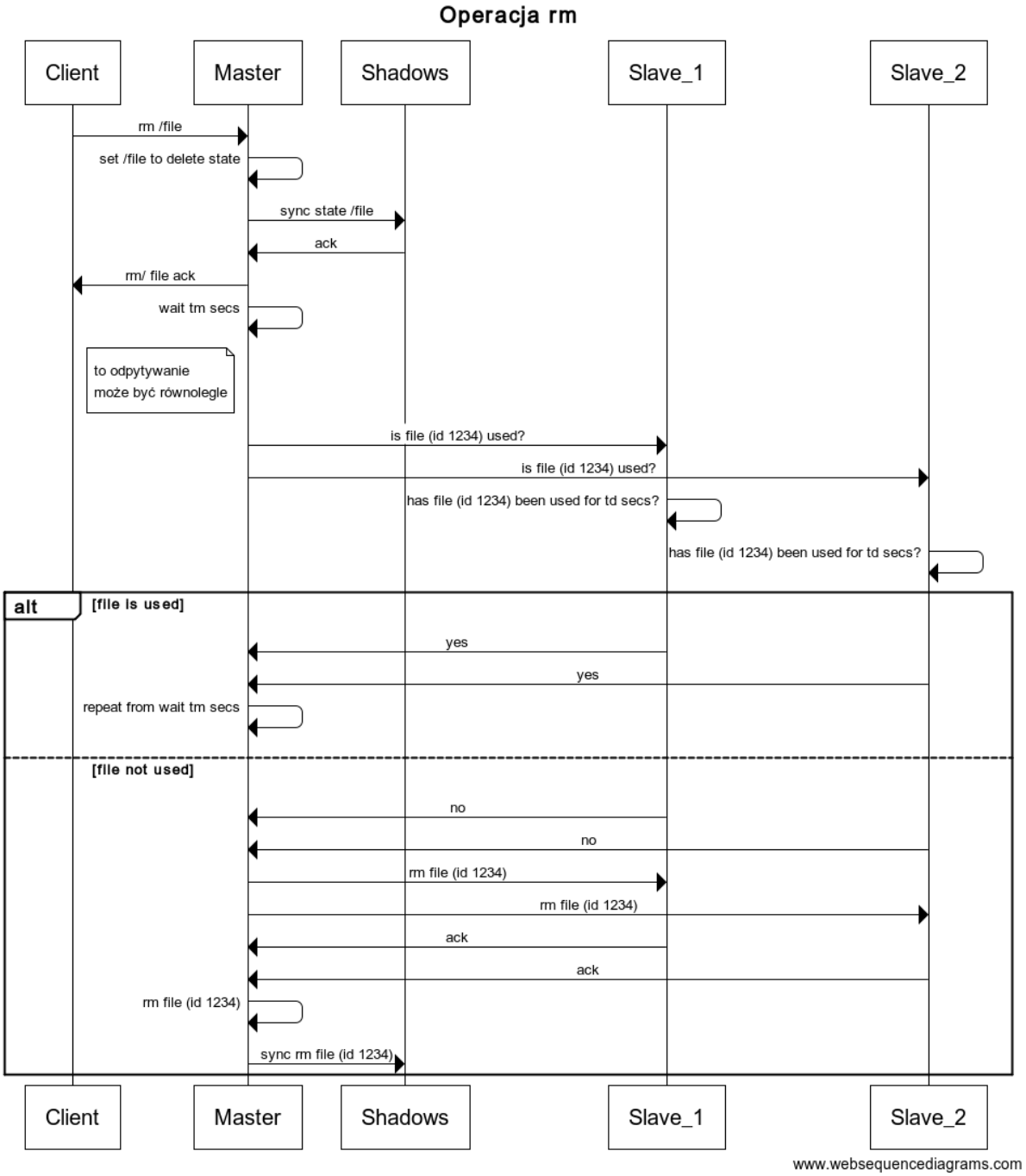
System zapewnia niezawodność poprzez replikację plików, oraz kopie Master Servera. Skalowalność jest zapewniona przez łatwe dołączanie kolejnych Storage Serverów bez strat w wydajności systemu. Rozważano koncepcję odciążenia Master Servera w przypadku operacji nie zmieniających stanu plików (get, status), które mogły by być wykonywane przez Shadow Servery, jednak dla zachowania prostoty zdecydowano by nie realizować tej koncepcji.

System pozwala klientowi ładować pliki do systemu, pobierać je, usuwać oraz sprawdzać ich stan. Podstawowe operacje przedstawiono na poniższych diagramach:

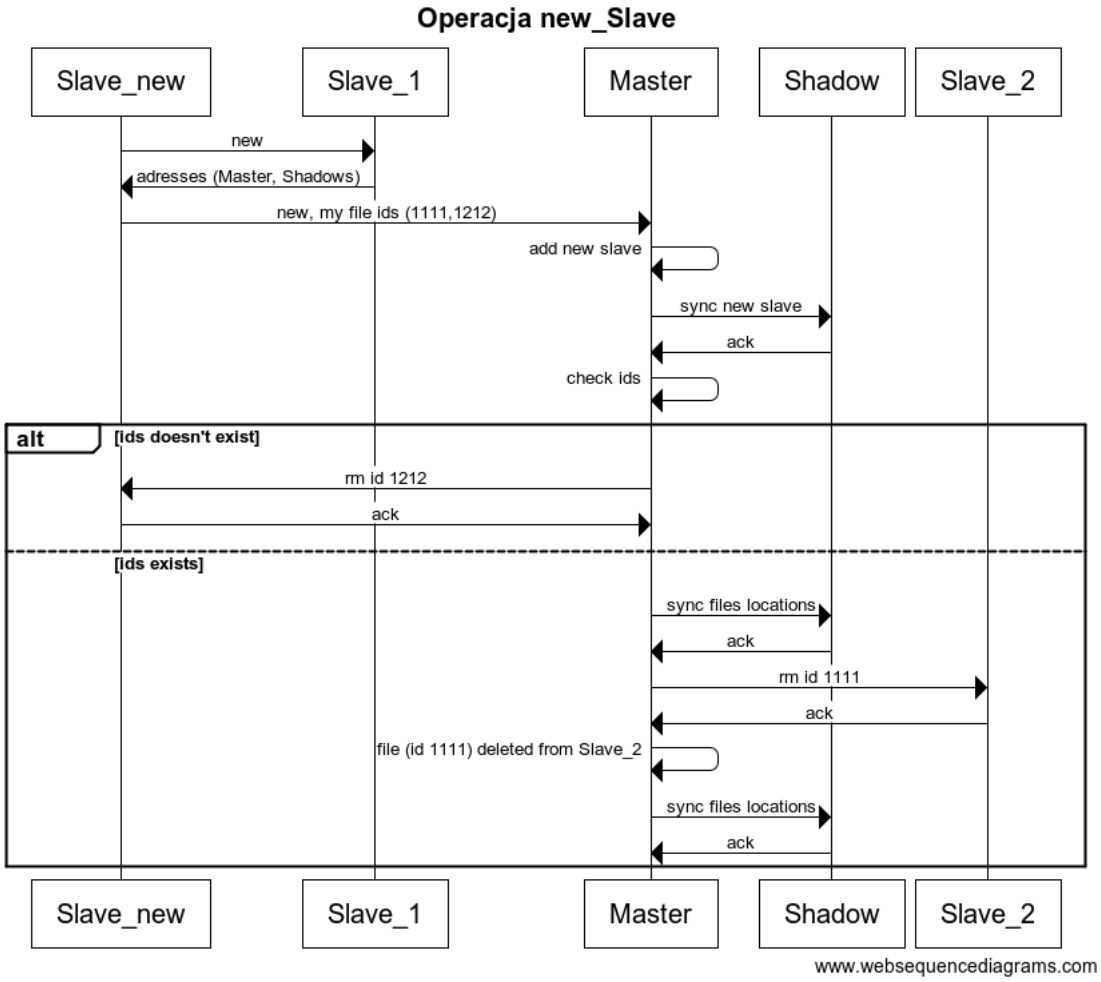



Operacja get
Client
Master
Slave_1
Slave_1_sender
Slave_2
Slave_2_sender
get /file
check id /file
id: 1001, Slave_1
get id: 1001, startByte: 0
start send thread
Send
opt
[Slave down]
get /file failure
check id /file
id: 1001, Slave_2
To sie dzieje nawet
jeśli Master ma ustawione
na plik to_delete
get id: 1001, startByte: 100
start send thread
Send
Client
Master
Slave_1
Slave_1_sender
Slave_2
Slave_2_sender
www.websequencediagrams.com


Operacja put
Client
Master
Slave_1
Slave_2
put /file, size 100MB
check space on slaves
alt
[not enough space]
no space available for /file
[enough space]
generate id 1001, choose slaves 1 2
receive id: 1001, size: 100MB
get id: 1001, from Slave_1
upload id 1001 to Slave_1
get id: 1001, startByte: 0
add to Queue
upload id 1001
send id 1001, startByte: 0
package <binary>
check Queue
package <binary>
get id: 1001, startByte: 100
send id 1001, startByte: 100
package <binary>
package <binary>
get id: 1001, startByte: 200
opt
[Slave_1 down]
put id 1001 fail
receive id: 1001, size: 100MB
upload id 1001 to Slave_2
upload id 1001
send id 1001, startByte: 200
package <binary>
Client
Master
Slave_1
Slave_2
www.websequencediagrams.com



Od strony administratorskiej - początkowe serwery są włączane za pomocą skryptu uruchamiającego usługę. Operacja przyłączenia nowego serwera, który przez określony czas nie odpowiadał, przebiega w następujący sposób:


Operacja new_Slave
Slave_new
Slave_1
Master
Shadow
Slave_2
new
adresses (Master, Shadows)
new, my file ids (1111,1212)
add new slave
sync new slave
ack
check ids
alt
[ids doesn't exist]
rm id 1212
ack
[ids exists]
sync files locations
ack
rm id 1111
ack
file (id 1111) deleted from Slave_2
sync files locations
ack
Slave_new
Slave_1
Master
Shadow
Slave_2
www.websequencediagrams.com

5. Implementacja

Podczas prac nad systemem powstały takie artefakty jak:

- skrypt uruchamiający usługę,
- aplikacja kliencka,
- aplikacja serwerowa,
- skrypt demostracyjny.

5.1. Interfejs usługi

W ramach prac nad systemem opracowano API zdefiniowane w języku Apache Thrift IDL, które przedstawiono na poniższym wydruku:

```
1  namespace java rso.dfs.generated
2
3  typedef i32 int
4  typedef i64 long
5  typedef string IPType
6
7  enum ServerType {
8    Slave,
9    Master,
10   Shadow
11 }
12
13 struct ServerStatus
14 {
15   1: required ServerType type;
16   2: required i32 filesNumber;
17   3: required i64 freeSpace;
18   4: required i64 usedSpace;
19   5: required IPType serverIP;
20 }
21
22 struct SystemStatus {
23   1:required i32 filesNumber;
24   2:required list<ServerStatus> serversStatuses;
25 }
26
```

```
27 struct CoreStatus {
28   1:IPType masterAddress;
29   2:list<IPType> shadowsAddresses;
30 }
31
32 // describes part of file
33 struct FilePartDescription {
34   1:int fileId;
35   2:long offset;
36 }
37
38 //represents part of a file
39 struct FilePart {
40   1:int fileId;
41   2:long offset;
42   3:binary data;
43 }
44
45 // communication initiated by new node
46 // new node sends to master file list
47 struct NewSlaveRequest {
48   1:required IPType slaveIP;
49   2:list<int> fileIds;
50   3:list<long> fileSizes;
51 }
52 struct GetFileParams
53 {
54   1:required i32 fileId;
55   2:required IPType slaveIp;
56   3:required i64 size;
57 }
58
59 struct PutFileParams
60 {
61   1:required bool canPut;
62   2:required i32 fileId;
63   3:required IPType slaveIp;
64 }
65
66 service Service
67 {
68   //returns administrative system status
69   SystemStatus getStatus(),
70
71   // returns list of file names
72   list<string> listFileNames(),
```

```
73
74  //infrastucure building
75  //slave sends request to master to register to serve
76  CoreStatus registerSlave(1:NewSlaveRequest req),
77
78  //with this request master makes slave register again.
79  void forceRegister(1:CoreStatus status),
80
81  //master updates slaves status
82  void updateCoreStatus(1:CoreStatus status),
83
84  //master sends request to slave
85  void becomeShadow(1: CoreStatus status),
86
87  //client - anyone
88  CoreStatus getCoreStatus(), // returns master/shadows list
89
90  //ping server for checking whether it's alive
91  void pingServer(),
92
93  // file id, file size; master slave
94  // force slave to be ready for file (fileId)
95  // which will be sent from client
96  void prepareForReceiving(1: int fileID, 2:long size),
97  //file id, slave ip
98  void replicate(1:int fileID, 2:IPType slaveIP, 3:long size),
99
100 //master - slave
101 bool isFileUsed(1:int fileID),
102 void removeFileSlave(1:int fileID),
103
104 GetFileParams getFile(1:string filepath),
105 GetFileParams getFileFailure(1:string filepath),
106 PutFileParams putFile(1:string filepath, 2:long size),
107 PutFileParams putFileFailure(1:string filepath, 2:long size),
108 bool removeFile(1:string filepath),
109
110 // returns: which part of file should be sent next
111 FilePartDescription sendFileToSlaveRequest(1: int fileId),
112
113 // Input: part of file which has to be sent
114 // returns: which part of file should be sent next,
115 //          special value in case of finish
116 FilePartDescription sendFilePartToSlave(1: FilePart filePart),
117 FilePart getFileFromSlave(1: FilePartDescription filePartDescription),
118
```

```
119   void fileUploadSuccess(1:int fileID, 2: IPType slaveIP)
120 }
```

5.2. Zarządzanie usługą

Podczas prac nad systemem wytworzono skrypt powłoki (*shell script*) ułatwiający zarządzanie usługą. Umożliwa on m.in. uruchamianie oraz zatrzymywanie usługi, a także pozwala wyświetlić informacje o bieżącym stanie usługi. W celu korzystania ze skryptu konieczne jest posiadanie systemu z rodziny Unix z zainstalowanymi narzędziami: *awk, timeout oraz nmap*. Możliwe wywołania skryptu zamieszczono na poniższym zestawieniu.

start [timeout]	próbuje uruchomić *naming service* na każdym z IP, po pierwszym sukcesie uruchamia slave'y. Implementacja daje [timeout w milisekundach] milisekund (domyślnie 5000) na uruchomienie się aplikacji serwera od chwili wykonania komendy ssh.
stop	zatrzymuje usługę
haltvm	zatrzymuje każdy z serwerów (wykonuje na nim polecenie powłoki *halt*)
status	uruchamia *system status* dla pierwszego servera, z którym uda się zkomunikować
client	nawiązuje połączenie z pierwszym serwerem, z którym uda się zkomunikować

6. Scenariusze testowe

Testowanie oprogramowania będzie polegało na wywoływaniu szeregu scenariuszy powiązanych z możliwymi usterkami występującymi podczas działania systemu zarówno z powodu wystąpienia sytuacji nadzwyczajnych, jak i z powodów zewnętrznych, nie związanych z błędami w samym oprogramowaniu. Scenariusze będą polegały na wywoływaniu operacji po stronie klienta, a następnie symulacji objawów usterek związanych z zadanym scenariuszem testowym; weryfikacja testu odbywać się będzie na podstawie porównania reakcji systemu z oczekiwanymi poprawnymi reakcjami na wysłanie żądania. Potencjalne błędy, które testy będą musiały sprawdzić, obejmują w szczególności problemy synchronizacyjne: opóźnienia oraz desynchronizację, ale również najczęściej spotykane usterki nie powiązane z zagadnieniami rozproszonymi.

Scenariusz	**Możliwe powody**	**Rozwiązanie**
Timeout	Podany serwer jest niedostępny	Ponowne przeszukanie puli dostępnych serwerów
Odpowiedź: IP serwerów typu Master i Shadow	Klient podłączył się do innego serwera niż master	Nawiązanie połączenia z masterem na podstawie otrzymanych danych
Klient otrzymuje niewłaściwe adresy - próbuje połączyć się do serwera o niepoprawnym typie	Opóźnienie propagacji, zmiana serwera master.	Ponowne odpytanie serwerów ze znanej puli shadow.

Tabela 6.1. Scenariusze dla operacji *connect (Client -> Server)*

Scenariusz	**Możliwe powody**	**Rozwiązanie**
File not found	Master nie może znaleźć pliku w bazie lub plik ma ustawiony tryb do usunięcia	(Klient) Sprawdzić poprawność ścieżki oraz sprawdzić listing plików
Slave down (timeout)	Podany serwer jest niedostępny	Wykonanie zwrotnego zapytania do mastera w celu wybrania nowego slave'a
Not enough space (u klienta)	Brak miejsca u klienta	Zwrócenie błędu

Tabela 6.2. Scenariusze dla operacji *get*

Scenariusz	**Możliwe powody**	**Rozwiązanie**
Not enough space	Zapełniona przestrzeń dyskowa slave'ów	Usunąć pliki z systemu
(M-S) Timeout	Wybrany serwer jest niedostępny	Master wybiera nowego slave'a do puli replikacyjnej i deaktywuje niedostępnego
(C-S) Timeout	Wybrany slave jest niedostępny	Klient ponawia próbę wysłania pliku za pośrednictwem mastera. Dodatkowo: master wybiera dodatkowy slave do replikacji i informuje całą resztę o zmianie lokalizacji repliki głównej.
(S-S) Timeout	Połączenie replikacyjne nie może zostać ustabilizowane.	Master zostaje poinformowany o zaburzeniu procesu replikacyjnego i wybiera nowe brakujące slave'y do replikacji.

Tabela 6.3. Scenariusze dla operacji *put*

Scenariusz	**Możliwe powody**	**Rozwiązanie**
File not found.	Plik nie istnieje w systemie.	Zwrócenie błędu
(M-SM) Timeout	Serwer poboczny jest niedostępny.	Master wywołuje proces elekcji serwera pobocznego.
(M-S) Timeout	Serwer magazynujący jest niedostępny.	Master zamyka połączenie; plik zostanie usunięty z serwera magazynującego podczas kolejnego połączenia.
Master ulega awarii w trakcie oczekiwania na slave'y		Serwer poboczny wykonuje zapytanie o pliki do usunięcia z bazy i wznawia proces.

Tabela 6.4. Scenariusze dla operacji *rm*

7. Harmonogram realizacji projektu. Przydział ról w projekcie

Podczas pierwszego spotkania projektowego ustalono podział ról oraz terminy przyszłych spotkań. Podział obowiązków ulegał modyfikacjom. Ostatecznie ustalono następujący przydział ról:

- kierownik projektu: Jacek Witkowski,
- architekt: Marcin Swend,
- zarządca repozytorium: Jacek Witkowski,
- dokumentalista: Jacek Witkowski,
- tester: Marcin Dzieżyc,
- handlowiec: Mateusz Statkiewicz.

Oprócz powyżej przydzielonych ról specjalnych, każdy z członków zespołu pełnił również rolę programisty (za wyjątkiem kierownika projektu). Po analizie wymagań zawartych w instrukcji do projektu dostępnej pod adresem: *http://www.ia.pw.edu.pl/ tkruk/edu/rsob2014/lab/rso_projekt2014.txt* ustalono następujący harmonogram prac:

- **4 marca 2014**: ustalenie podziału ról oraz organizacji środowiska programistycznego projektu,
- **11 marca 2014**: ustalenie funkcjonalności oferowanych przez system, ustalenie zawartości pliku konfiguracyjnego, ustalenie zarysu architektury systemu,
- **14 marca 2014**: ustalenie scenariuszy wysyłania i pobierania pliku,
- **18 marca 2014**: ustalenie scenariuszy uruchomienia i zamnięcia systemu,
- **25 marca 2014**: ustalenie funkcjonalności realizowanych w fazie drugiej,
- **1 kwietnia 2014**: przygotowanie wstępnej wersji dokumentacji,
- **4 kwietnia 2014**: opracowanie interfejsów komunikacyjnych udostępnianych przez serwer nazewniczy (master), serwer przechowujący (slave) oraz klienta; opracowanie komunikatów wymienianych między masterem i slave'em, masterem i klientem oraz slave'em i klientem,
- **6 kwietnia 2014**: przygotowanie dokumentacji spełniającej wszystkie wymogi dotyczące I etapu projektu,
- **22 kwietnia 2014**: implementacja operacji wysyłania i pobierania danych z systemu przy założeniu bezawaryjności urządzeń oraz komunikacji między nimi; nie będzie zaimplementowana replikacja; ustalenie pełnego planu testów; opracowanie końcowej demonstracji projektu,

- **29 kwietnia 2014**: implementacja operacji usuwania danych,
- **4 maja 2014**: opracowanie wersji dokumentacji spełniającej wszystkie wymogi dotyczące II etapu projektu,
- **16 maja 2014**: zapewnienie odpowiedniej niezawodności przechowywania danych (m.in. implementacja replikacji plików, uwzględnienie możliwych do wystąpienia sytuacji awaryjnych),
- **27 maja 2014**: usunięcie wykrytych błędów istniejących w systemie,
- **2 czerwca 2014**: przygotowanie pełnej dokumentacji projektu spełniającej wymogi dotyczące III etapu projektu.

8. Organizacja środowiska programistycznego projektu

Podczas wykonywania projektu zostały wykorzystane następujące narzędzia:

- **Git** - jako system kontroli wersji,
- **GitHub** - jako zdalne repozytorium dla narzędzia git,
- **Google Docs** - do wspólnego tworzenia wstępnej dokumentacji projektowej,
- **Latex** - do przygotowania dokumentacji projektowej,
- **Redmine** - do zarządzania projektem.

Wybór gita jako systemu kontroli wersji był podyktowany prostotą jego użytkowania oraz wysoką funkcjonalnością. Dodatkowo, wszyscy członkowie zespołu znają to narzędzie.

Kolejny wybór dotyczył zdalnego repozytorium, na którym miały by być przechowywane zasoby z gita. Wybór padł na serwis GitHub, gdyż oferuje on darmowe usługi dla projektów niekomercyjnych oraz udostępnia wiele danych statystycznych pomocnych przy zarządzaniu projektem (np. wykres aktywności).

Jako narzędzie do zarządzania projektem wybrano system Redmine. Jego głównymi zaletami są:

- wysoka funkcjonalność,
- możliwość integracji z innymi systemami (np. z systemem kontroli wersji git),
- łatwość użytkowania.

9. Raporty ze spotkań

9.1. Spotkanie nr 1

data: 2014-03-04
czas: 18:00-18:30

uczestnicy:
- Jacek Witkowski
- Mateusz Statkiewicz
- Marcin Swend
- Piotr Kalinowski

cele:
- Organizacja:
  — wymiana danych kontaktowych,
  — określenie swoich preferencji co do ról istniejących w projekcie,
  — ustalenie możliwych terminów i formy spotkań.
- Wstępna analiza wymagań dotyczących realizacji projektu.
- Przegląd i ewentualny wybór narzędzi używanych podczas realizacji projektu.

ustalenia:
Każdy z członków zespołu określił role projektowe, które chciałby wykonywać. Wstępnie przyjęto następujący podział ról:
- architekt: Marcin Swend,
- zarządca repozytorium: Jacek Witkowski,
- dokumentalista: Jacek Witkowski,
- tester: Mateusz Statkiewicz,
- handlowiec: Jacek Witkowski.

Ponadto, każdy z członków zespołu (poza kierownikiem projektu) przyjmuje rolę programisty.

Ustalono, że możliwe terminy spotkań to wtorki o godzinie 18.00 oraz piątki o godzinie 18.00. Istnieje również możliwość odbywania telekonferencji w weekendy.

Dokumentacja będzie prowadzona przy użyciu Google Docs, ponieważ każdy z członków projektu już teraz posiada konto w serwisie Google. Stąd, korzystanie z narzędzia nie niesie ze sobą konieczności dodatkowego rejestrowania się. Do zarządzania projektem zostanie użyty system Redmine, który zostanie uruchomiony na serwerze Michała Statkiewicza. Jako narzędzie kontroli wersji zostanie wykorzystany git. Repozytorium zostanie utworzone w serwisie GitHub.

9.2. Spotkanie nr 2

data: 2014-03-11
czas: 18:00-18:30

uczestnicy:
- Jacek Witkowski
- Mateusz Statkiewicz
- Marcin Swend
- Adam Papros
- Marcin Dzieżyc

cele:
- ustalenie wymagań funkcjonalnych projektu,
- ustalenie założeń dotyczących środowiska uruchomieniowego,
- ustalenie zawartości pliku konfiguracyjnego,
- ustalenie wstępnej architektury systemu.

ustalenia:
Założenia dotyczące projektu:
- aplikacja kliencka będzie programem konsolowym,
- będą możliwe do użycia następujące komendy:
  - ls - wylistowanie zawartości folderu,
  - cd - zmiana bieżącego katalogu,
  - rm - usunięcie wskazanego pliku lub folderu,
  - cp - kopiowanie pliku/folderu z miejsca źródłowego do docelowego,
  - mv - przemieszczenie pliku/folderu z miejsca źródłowego do docelowego.

Środowisko uruchomieniowe: 4 laptopy działające w jednej podsieci. Laptopy będą połączone za pomocą routera oferującego możliwość bezprzewodowego połączenia. Na laptopach uruchomione będą maszyny wirtualne, na których uruchomione będą aplikacje klienckie lub serwerowe.

Spis rzeczy, które muszą znaleźć się w pliku konfiguracyjnym:

- lista reprezentująca adresy ip dozwolone dla urządzeń stanowiących część serwerową w systemie,
- stopień redundancji danych.

Wstępna architektura systemu:
- urządzenia stanowiące część usługową będą podzielone na dwie klasy: część lokalizacyjno-nazewniczą oraz cześć przechowawczą,
- wśród urządzeń klasy lokalizacyjno-nazewniczej wyróżnione będzie jedno urządzenie typu master, które będzie przechowywać informacje o rozmieszczeniu plików w katalogach oraz o ich lokalizacji na poszczególnych urządzeniach przechowawczych.
- urządzenia z klasy lokalizacyjno-nazewniczej nie będące masterem, będą tzw. *shadow masterami* stanowiącymi dokładną kopię mastera i będącymi w stanie go zastąpić w razie jego awarii.

9.3. Spotkanie nr 3

data: 2014-03-14
czas: 16:00-18:30

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Ustalenie scenariuszy dla następujących działań:
- wysyłanie pliku,
- pobieranie pliku.

ustalenia:
Scenariusz pobierania pliku:
1. Klient łączy się do dowolnego serwera.
2. Klient otrzymuje adres mastera.
3. Klient łączy się z masterem.
4. Klient wydaje polecenie pobrania pliku o określonej ścieżce.
5. Master wyszukuje plik w hierarchii plików.

6. Master odsyła do klienta listę slave'ów przechowujących pliki wraz z identyfikatorem pliku.
7. Klient nawiązuje połączenie z pierwszym z listy slave'ów i wysyła żądanie pobrania pliku o wskazanym identyfikatorze od zerowego bajta.
8. Slave rozpoczyna wysyłanie pliku.
9. Klient odbiera plik.

Scenariusz wysyłania pliku:

1. Klient łączy się do dowolnego serwera.
2. Klient otrzymuje adres mastera.
3. Klient łączy się z masterem.
4. Klient informuje mastera o chęci wysłania pliku o określonej ścieżce i rozmiarze.
5. Master sprawdza czy plik już istnieje oraz wysyła do wybranych slave'ów informację o identyfikatorze pliku, który ma zostać odebrany wraz z listą wszystkich slave'ów, do których została wysłana ta lista (każdy z wybranej grupy slave'ów otrzymuje tę samą listę).
6. Master wysyła identyfikator pliku i listę która była rozsyłana w p. 5 do klienta.
7. Klient łączy się z pierwszym slave'em z listy i podaje identyfikator pliku.
8. Klient rozpoczyna wysyłanie pliku do slave'a.
9. Slave odbiera fragmenty pliku i przesyła je do kolejnego slave'a z otrzymanej od mastera listy. Slave'y rozsyłają między sobą odbierane fragmenty.
10. Klient kończy wysyłanie.

9.4. Spotkanie nr 4

data: 2014-03-18
czas: 18:00-21:15

uczestnicy:

- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Ustalenie scenariuszy dla następujących działań:

- uruchomienie systemu,
- usuwanie pliku z systemu,
- kopiowanie pliku do systemu,

- kopiowanie pliku z systemu,
- otrzymanie informacji o stanie systemu (jako całości oraz poszczególnych węzłów).

ustalenia:
W warstwie lokalizacyjno-nazewniczej istnieje tylko jedno aktywne urządzenie obsługujące wszystkie żądania trafiające do tej warstwy. Urządzenie to będziemy nazywać masterem. Pozostałe urządzenia z tej warstwy będą jedynie kopiami mastera. Będziemy je nazywać Shadow Masterami.

Master oraz Shadow Mastery wykorzystują synchronizację on-line. Utrzymują lokalnie tablicę trwających procesów, która jest między nimi synchronizowana w ten sposób, by awaria mastera w dowolnym momencie nie spowodowała awarii całego systemu.

Warunkiem koniecznym uznania dowolnego żądania klienta za zakończone, jest zsynchronizowanie informacji dotyczących tego żądania na masterze oraz shadow masterach.

W pliku konfiguracyjnym będzie zawarta lista urządzeń, które mają być pierwotnymi urządzeniami istniejącymi w systemie. Informacja ta będzie wykorzystywana w skrypcie uruchamiającym cały system. Zakładamy, że podczas działania skryptu urządzenie, które udało się uruchomić nie ulegnie awarii.

Na spotkaniu opracowano również kolejne wersję scenariuszy wysyłania pliku do systemu oraz jego odbierania, a także usuwania. Scenariusze te zostały umieszczone w dokumentacji projektu.

9.5. Spotkanie nr 5

data: 2014-03-21
czas: 16:00-16:30

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Jacek Witkowski

cele:
Weryfikacja scenariuszy opracowanych przez zespół dla następujących czynności:

- wysyłanie pliku do systemu,
- pobieranie pliku z systemu,
- usuwanie pliku z systemu.

Zaplanowanie czynności do wykonania w ciągu najbliższych 2 tygodni.

ustalenia:
W ciągu najbliższych dwóch tygodni należy:
- ustalić metody udostępniane przez aplikację serwerową oraz aplikację kliencką,
- ustalić struktury komunikatów wymienianych między urządzeniami działającymi w systemie (komuniakcja wewnątrzsystemowa oraz komunikacja z klientami).

9.6. Spotkanie nr 6

data: 2014-03-25
czas: 16:00-16:30

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Ustalenie artefaktów jakie mają powstać w drugim etapie projektu. Podział zadań dotyczących implementacji podstawowych funkcjonalności.

ustalenia:
Ustalono, że w ramach drugiego etapu wykonany zostanie system realizujący funkcjonalności: wysyłania/pobierania danych do/z systemu, usuwania plików, pobierania informacji o urządzeniach działających w systemie (operacja status), uruchamianie/zatrzymywanie systemu, przyłączanie nowych urządzeń do systemu.

W pierwszej kolejności należy ustalić interfejsy jakie mają oferować aplikacje klienta, mastera oraz slave'a oraz komunikaty wymieniane między nimi. Podzielono zadania w następujący sposób:
- komunikacja Master-Klient: Piotr Kalinowski, Jacek Witkowski,

- komunikacja Master-Slave: Marcin Dzieżyc, Adam Papros,
- komunikacja Klient-Slave: Mateusz Statkiewicz, Marcin Swend.

Zadania należy wykonać do 4 kwietnia.

9.7. Spotkanie nr 7

data: 2014-04-04
czas: 16:00-16:30

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Marcin Swend
- Jacek Witkowski

cele:
Zebranie i uspójnienie przygotowanych interfejsów i wiadomości koniecznych do komunikacji pomiędzy klientem, masterem i slave'em.

ustalenia:
Po ujednoliceniu stworzonych interfejsów oraz postaci komunikatów przydzielono zadania implementacji poszczególnych funkcjonalności członków zespołu.

9.8. Spotkanie nr 8

data: 2014-03-18
czas: 16.00-17.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Dopracowanie serwisów jakie mają powstać w systemie oraz wstępne ustalenie podziału zadań pomiędzy członkami zespołu.

9.9. Spotkanie nr 9

data: 2014-04-22
czas: 18.00-19.30

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Ustalenie postępu prac. Identyfikacja problemów jakie powstały podczas implementowania serwisów oraz próba ich rozwiązania.

9.10. Spotkanie nr 10

data: 2014-05-05
czas: 19.00-22.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend

cele:
Finalizacja prac nad implementacją funkcjonalności zaplanowanych do wykonania w ramach drugiej fazy projektu.

9.11. Spotkanie nr 11

data: 2014-13-05
czas: 18.00-19.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski

- Adam Papros
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Określenie postępu prac nad implementacją replikacji plików.

9.12. Spotkanie nr 12

data: 2014-05-20
czas: 18.00-19.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Określenie postępu prac nad implementacją replikacji plików. Przydział zadań implementacji obsługi sytuacji awaryjnych występujących w systemie.

9.13. Spotkanie nr 13

data: 2014-05-23
czas: 16.00-17.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Jacek Witkowski

cele:
Ustalenie postępu prac nad projektem. Identyfikacja powstałych problemów oraz próba ich rozwiązania.

9.14. Spotkanie nr 14

data: 2014-05-24
czas: 9.00-13.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Ustalenie postępu prac nad projektem. Wspólna implementacja obsługi sytuacji błędnych. Dopracowywanie replikacji plików oraz masterów.

9.15. Spotkanie nr 15

data: 2014-06-01
czas: 18.30-22.00

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Mateusz Statkiewicz
- Marcin Swend
- Jacek Witkowski

cele:
Przeprowadzenie pełnych testów aplikacji.

9.16. Spotkanie nr 16

data: 2014-06-02
czas: 19.30-23.50

uczestnicy:
- Marcin Dzieżyc
- Piotr Kalinowski
- Adam Papros
- Mateusz Statkiewicz

- Marcin Swend
- Jacek Witkowski

cele:
Finalizacja prac nad projektem.