Sprawozdanie z projektu

Lista uczestników:  
Osoba 1: Grzegorz Adamczyk

Osoba 2: Piotr Adryan

Osoba 3: Dawid Bydłosz

Osoba 4: Patryk Bogusz

Osoba 5: Rafał Bąk

Osoba 6: Paweł Bobołowicz

Technologia: RPC

Język programowania: Python

# Analiza dostępności

W systemach rozproszonych RPC (Remote Procedure Call) czyli zdalne wywołanie procedury jest to system w którym możliwe jest aby komputer wywołał procedurę w innej przestrzeni adresowej (na innym komputerze w tej samej dzielonej sieci). Procedura ta jest wywoływana taka jak by była wywoływana na maszynie lokalnej. To znaczy że programista pisze ten sam kod nie zależnie czy jest on uruchomiany lokalnie czy na zdalnej maszynie. Jest to model klient-serwer.

RPC jest swego rodzaju protokołem zapytanie-odpowiedź. Jest inicjowane przez klienta który wysyła zapytanie to znanego mu serwera zdalnego aby wywołał daną procedurę z podanymi parametrami. Zdalny serwer wysyła odpowiedź do klienta i aplikacja kontynuuje działanie. Kiedy serwer obsługuje zapytanie, klient jest zablokowany (czeka dopóki serwer nie skończy przetwarzania zadania), chyba że klient wyśle asynchroniczne zapytanie do serwera np. XHTTP.

Znaczna różnica w zdalnym wywołaniu procedury, a w lokalnym jest taka że zdalne zapytanie może się nie udać ze względu na nieprzewidziane problemy z siecią. Dodatkowo, klienci muszą radzić sobie z tego rodzaju problemami bez wiedzy czy zdalna procedura została odebrana.

Na typowy scenariusz użytkowania takiego protokołu składają się:

1. [Serwer](https://pl.wikipedia.org/wiki/Serwer) (czyli program oferujący usługi, np. drukowania) przez cały czas nasłuchuje na wybranym porcie, czy ktoś się z nim nie łączy.
2. [Klient](https://pl.wikipedia.org/wiki/Klient_(informatyka)) (czyli program, który potrzebuje jakiejś usługi od serwera na innym komputerze) nawiązuje z nim łączność poprzez [sieć komputerową](https://pl.wikipedia.org/wiki/Sie%C4%87_komputerowa).
3. Klient wysyła swoje dane we wcześniej ustalonym przez programistów klienta i serwera formacie.
4. Serwer realizuje usługę i odsyła potwierdzenie lub kod błędu.

RPC lub Remote Python Call jest przejrzystą i symetryczną biblioteką do Pythona zapewniająca zdalne wywołanie procedur, gromadzenie i działanie rozproszone. RPC używa obiektów proxy, jest to technika która wykorzystuje dynamiczną naturę Pythona, aby pokonać fizyczną granicę między procesami, a komputerami. Dzięki temu zdalne obiekty mogą być manipulowane tak jak obiekty lokalne.

Ważną cecha RPC jest jego przejrzystość, dzięki temu możesz używać RPC w istniejącym już kodzie praktycznie bez ponoszenia dodatkowych kosztów. Nie ma potrzeby pisania skomplikowanych plików definicji, konfiguracji serwera, konfiguracji transportu (http), ani używania specjalnych komend wywołujących. W konsekwencji tego RPC jest także symetryczny, czyli nie ma tu ścisłego rozróżnienia na serwer lub klient. Jedna maszyna może raz służyć jako klient po czym zmienić swoją rolę. Dzięki symetryczności dostępna jest funkcja powrotnego zapytania (callback).

## Cechy RPC:

* Przejrzysty dostęp do obiektów zdalnych, programowanie zdalne takie jak lokalne
* Symetryczny protokół, gdzie klient i serwer mogą obsługiwać zapytania, zezwolenie na callback
* Synchroniczna i asynchroniczna praca
* Wieloplatformowość
* Integracja z TLS/SSL

## Sposoby użytkowania RPC:

* Usługi zdalne:

Zaczynając od wersji RPC 3.00 biblioteka ta jest zorientowana na usługi. Dzięki temu zapewnienie bezpieczeństwa usług jest trywialne. Usług jest klasą która reprezentuje dobrze zdefiniowane funkcje i obiekty zdalne. Dzięki temu te funkcje mogę być odebrana przez klientów usługi aby otrzymać wyniki.

* Administracja i Centrum Kontroli:

Wydajny system administracyjny jest dość trudny do utrzymania: duża liczba platform która musi być pod kontrolą, inne narzędzia administracyjne, inny język administracyjny (sh,tcsh, batch files itd.). Dodatkowo trzeba pracować na różnych protokołach transportu (telnet, ftp, ssh itd.).

Z pomocą przychodzi python jako wieloplatformowy potężny język programowania z wieloma bibliotekami i dużym suportem. Dzięki serwerowi RPC możesz kontrolować każdą maszynę z jednego miejsca.

* Zarządzanie hardware’em:

Jest wiele przypadków kiedy musisz użyć zewnętrznego fizycznego hardware do przeprowadzenia operacji które są niemożliwe do zrobienia na twojej stacji roboczej. Zakładając że twój komputer posiada C bindings i wiersz poleceń możesz postawić serwer RPC na swoim komputerze i podpiąć się do wybranej stacji roboczej.

* Uruchomienia równoległe:

W Pythonie GIL zapobiega wykonaniu pythonowego bytecodu przez wiele wątków w jednym czasie. To upraszcza budowę interpretera ale konsekwencją tego Python nie może wykorzystywać wielu CPU. Jedyną możliwością jest używanie wielu procesów, zamiast wielu wątków. Używając RPC, wieloprocesowość staję się bardzo łatwa ponieważ w RPC możemy połączyć procesy w jeden duży proces.

* System Rozproszony:

Dzięki swojej architekturze i wsparciu synchronicznego i asynchronicznego odbioru, serwer i klient są symetryczni. Dzięki czemu możliwe jest stworzenie systemu rozproszonego.

* Testowanie:

RPC jest świetną platformą aby przeprowadzać centralne testy na wielu maszynach i platformach. Nie ma potrzeby kopiowania i synchronizowania plików na różnych maszynach. Praca ta najlepiej sprawdza się na bezpiecznych i szybkich sieciach lokalnych z uwagi na to że RPC wymaga dużo sieciowego „ping-ponga”.

## Zastosowania RPC:

* Kontrolowanie wielu komputerów z centralnego punktu zarządzania.
* Zdalny dostęp do zasobów komputera.
* Rozdzielania zadania (pracy) na wielu maszynach (system rozporoszony).
* Implementacja usług zdalnych takich jak SOAP lub RMI.
* Testowanie.

# Diagramy UML

## Przepływu danych

## Sekwencjihttps://scontent-waw1-1.xx.fbcdn.net/v/t34.0-12/19190821_1064010340402512_286070262_n.png?oh=abaaaee3945284629ed6ab8924c4a569&oe=5945B1D6

## Diagram sekwencji https://scontent-waw1-1.xx.fbcdn.net/v/t34.0-12/19190862_1064010760402470_679897035_n.png?oh=e3998dc29ab30d4b27ba2fb64b1717dd&oe=5945748D

# Lista zależności konieczna do uruchomienia

* System Unix – testowano na Debianie, Ubuntu oraz Centosie
* Python w wersji 3.x
* Biblioteka RestrickedPython
* Prawa administratora na systemie

# Narzędzia komunikacyjne

* Slack: <https://srirhardpartyy.slack.com/messages/C5UDKSW77/>
* Trello: <https://trello.com/b/9MgRyFP2/srir-project>
* Github: <https://github.com/DawidBydlosz/SRiR>

## Nakład pracy w godzinach

# Wnioski

Łącznie przepracowaliśmy około 50 godzin nad projektem. Implementacja projektu była znacznie przyjemniejsza i prostsza w uzupełnieniu o dodatkowe funkcjonalności, niż poprzednio pisane sockety P2P w języku C. „Projekt stanowił niezłą okazję do zmierzenia się z nieco poważniejszym, trudniejszym – ale bardziej rzetelnym podejściem do postawionego zagadnienia[[1]](#endnote-1)”. Wiązało się to z użyciem zestawu dedykowanych rozwiązań RPC w postaci: bibliotek, wsparcia ze strony RestrictedPython oraz składni języka będącej poziom abstrakcji wyżej. Udało nam się zrealizować wszystkie postawione przed nami problemy, program posiada wszystkie zakładane ( a nawet dodatkowe) funkcjonalności, niestety brak czasu, umiejętności i fakt, iż jest to nasze pierwsze spotkanie z tym językiem i technologią nie pozwoliły nam w terminie wykonać testów modułów oraz testów jednostkowych. Dużo czasu poświęciliśmy na przegląd literatury oraz materiałów dostępnych w Internecie, aby zapoznać się z założeniami technologii. Ogromna część systemów, z którymi spotykamy się na co dzień opiera się na architekturach rozproszonych, a w naszym projekcie mamy do czynienia z takową na dwóch płaszczyznach; pierwsza – lokalna, tj. serwer pracuje na kilku wątkach, wykonujących zadania niezależnie od siebie, na różnych portach, druga – zadania projektu mogą być wykonywane są na kilku komputerach z różnymi systemami UNIX. W przypadku zastosowania takiego rozwiązania w rzeczywistym projekcie pozwoliłoby to na rozłożenie obciążeń np. łączy czy też mocy obliczeniowych, realizując zadania szybciej, sprawniej, efektywniej i minimalizując ryzyko awarii krytycznej.

# Bibliografia

[1] J. Palach, „Parallel Programming with Python”, Packt Publishing open source, Birmingham 2014

[2] https://docs.python.org/2/library/xmlrpclib.html (dostęp 3.06.2017r.)

[3] http://spyne.io/#inprot=HttpRpc&outprot=JsonDocument&s=rpc&tpt=WsgiApplication&validator=true ( dostęp 3.06.2017r.)

[4] https://github.com/joshmarshall/jsonrpclib (dostęp 3.06.2017r.)

[5] https://www.rabbitmq.com/tutorials/tutorial-six-python.html (dostęp 3.06.2017r.)

[6] K. Pavlov „json-rpc Documentation Release 1.10.3” dokumentacja: https://media.readthedocs.org/pdf/json-rpc/latest/json-rpc.pdf (dostęp 3.06.2017r.)

[7] M. Cutler „python-wordpress-xmlrpc Documentation Release 2.3” dokumentacja: https://media.readthedocs.org/pdf/python-wordpress-xmlrpc/latest/python-wordpress-xmlrpc.pdf (dostęp 3.06.2017r.)

[8] „Rozszerzony kurs języka Python” strona z materiałami dydaktycznymi dr M. Młotkowskiego http://www.ii.uni.wroc.pl/~marcinm/dyd/python/ (dostęp 3.06.2017r.)

[9] https://pythonhosted.org/Flask-XML-RPC/ (dostęp 3.06.2017r.)

[10] B. Rhodes, J. Goerzen, „Foundations of Python Network Programming”, Apress, 2014

[11] Dave Kuhlman, „Multiprocessing in Python: a guided tour with examples ” http://www.davekuhlman.org/python\_multiprocessing\_01.html (dostęp 4.06.2017r.)

[12] Roland Koebler „RPC / JSON-RPC” https://www.simple-is-better.org/rpc/ (dostęp 4.06.2017r.)

1. <http://home.agh.edu.pl/~kamich/SRiR/SRiR_Projekt_2016_17.pdf> - dostęp z dnia 15.06.2017, autor: mgr inż. Kazimierz Michalik [↑](#endnote-ref-1)