AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

dHTTP – Rozproszony system wsparcia serwerów sieci web

dHTTP – Distributed companion for central-server based web

Autor: *Dominik Adamiak*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Piotr Matyasik*

Kraków, 2018

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję promotorowi za wolność działania, a rodzinie i przyjaciołom za wiarę w to, że się uda.

Spis treści

1. Wprowadzenie 7

1.1. Cele pracy 7

1.2. Zawartość 8

2. Droga do rozproszenia 9

2.1. Zarys historyczny 9

2.2. Narzędzia w rozproszeniu danych 10

2.3. Podstawa projektu 12

3. Projektowanie i implementacja 13

3.1. Wymagania funkcjonalne 13

3.2. Definicje, architektura i technologie 14

3.2.1. Wykorzystane technologie i narzędzia 14

3.2.2. Problemy warstwy sieciowej 14

3.2.3. Decentralizacja czy rozproszenie? 14

3.2.4. Bezpieczeństwo 14

3.2.5. Propagacja i przechowywanie danych 14

3.3. Aplikacje 14

3.3.1. Węzeł 14

3.3.2. Klient 14

4. Testy i optymalizacja 15

4.1. Zarys użytkowości i działania systemu 15

4.2. Wydajność systemu 15

4.2.1. Wydajność klienta 15

4.2.2. Wydajność serwera 15

4.3. Podsumowanie rozwiązań optymalizacyjnych 15

5. Podsumowanie 17

5.1. Potencjalne kierunki rozwoju 17

1. Wprowadzenie

Dzisiejsza informatyka jak nigdy stoi przed wyzwaniami związanymi z wydajnością, wymuszanymi poprzez miliony klientów z szerokopasmowym dostępem do sieci. Ich rozwiązywanie nie jest już możliwe przez wzmacnianie podzespołów pojedynczych komputerów – wymaga odpowiedniego sposobu projektowania, który skupia się na możliwościach maksymalnego rozłożenia i łatwego skalowania obciążenia. Ta praca proponuje rozwiązanie, które wpisuje się w ten motyw i w przystępny klientom sposób może znacznie odciążyć rozwijające się części sieci web.

1.1. Cele pracy

Celem tej pracy jest stworzenie protokołu i oprogramowania pozwalającego na rozproszony i bezpieczny dostęp do współdzielonych zasobów, w warstwie użytkowej odpowiadającego obecnemu Hypertext Transfer Protocol (HTTP). W toku pracy, system określany będzie skrótem **dHTTP** – distributed HTTP.

Głównym założeniem projektu jest rozłożenie obciążenia stron stron, które nagle zyskują popularność – często opcją dla właścicieli takich stron jest inwestycja w szybsze łącza czy więcej sprzętu, wiążąca się ze sporymi kosztami. Kosztami które mogą zresztą nie znaleźć uzasadnienia – dodatkowy sprzęt pomoże w chwilach szczytowego ruchu, będzie jednak przez większość czasu leżeć odłogiem, wciąż jednak generując koszty łącz i prądu.

Częściowym rozwiązaniem tego problemu są chmury i udostępniany przez nie *autoscaling* – użytkownicy, zamiast utrzymywać własną infrastrukturę, mogą nie tylko zdać się na serwery zarządzane przez zewnętrzny podmiot, ale także wykorzystać mechanizmy automatycznego rozszerzania i zmniejszania ilości urządzeń czy wykorzystywanego łącza, w zależności od obciążenia całego klastra. To pozwala na drastyczną minimalizację kosztów i jest rozwiązaniem coraz chętniej stosowanym także przez podmioty o gigantycznym ruchu sieciowym, jak serwis streamingowy Netflix ([1]).

Wciąż jednak są to rozwiązania obciążające właściciela serwera, co może stanowić problem w przypadku projektów hobbystycznych czy krajów rozwijających się – koszt autoscalingu w czasach szczytu może okazać się nieproporcjonalnie wysoki w stosunku do czasów spokojnego ruchu.

System zaprezentowany w poniższej pracy – dHTTP – ma posłużyć jako samoskalująca się, rozproszona alternatywa do tego podejścia. Musi wziąć on pod uwagę optymalne rozłożenie

danych, zabezpieczenie przed zmienianiem zawartości zasobów przez niepowołane podmioty (hashe i podpisy kryptograficzne), szybki i możliwie najmniej scentralizowany sposób współdzielenia informacji o stanie systemu (*kto jest online i może dać mi plik* `logo.png` *z hosta* `example.com`*?*) i aktualizacjach zawartości oraz stronę kliencką, będącą w tym przypadku wtyczką do przeglądarki Google Chrome, nakładającą warstwę dHTTP na polecenia pobierania zasobów. Z punktu widzenia użytkownika system jest *przezroczysty*: zostanie węzłem i klientem sieci dHTTP polega wyłącznie na instalacji wtyczki, działającej autonomicznie w tle. Oprócz aplikacji klienckiej, dHTTP udostępniony jest także w trybie *headless* – węzeł uruchamiający może zostać częścią klastra bez użycia przeglądarki.

1.2. Zawartość

Rozdział 1. stanowi krótkie wprowadzenie i definicję celu pracy.

Rozdział 2. zawiera rozważania na temat historycznych i teoretycznych podstaw systemów rozproszonych, oraz przykłady najpopularniejszych narzędzi je implementujących. Wprowadza także do konceptualnych i narzędziowych podstaw implementacji systemu dHTTP.

Rozdział 3. jest szczegółowym opisem wymagań, projektu, modułów, a także implementacji samego systemu, w szczegółach omawiającym funkcjonalności, biblioteki i narzędzia pozwalające na działanie aplikacji, napotkane w toku projektowania problemy i ich rozwiązania, rys architektoniczny sposobu przechowywania danych czy ich propagacji, z uwzględnieniem implementacji niezależnego węzła (*headless mode*) i rozwiązania dla klienta końcowego.

Rozdział 4. to próba walidacji systemu dHTTP – analiza użytkowości systemu pod kątem wrażeń użytkowych, zbiór badań nad wydajnością systemu (z uwzględnieniem różnych mechanizmów propagacji danych) i propozycji na jej dalsze optymalizacje.

Rozdział 5. zawiera krótkie podsumowanie, podkreślające osiągnięte cele i potencjalne kierunki dalszego rozwoju projektu.

2. Droga do rozproszenia

Prawo Moore'a ([2]), wspominające o podwajaniu ilości tranzystorów w procesorach, często parafrazowane jako podwajanie ich mocy obliczeniowej, przestaje działać, podczas gdy złożoność problemów i ilość użytkowników szerokopasmowego internetu rośnie. Próba skalowania wertykalnego – polegającego na wzmacnianiu pojedynczych węzłów, przestaje zdawać próbę czasu.

Problem posiada także drugą stronę – nie wszyscy użytkownicy internetu mogą pozwolić sobie na łącze szerokopasmowe. Szacuje się, że w roku 2017 dostęp do internetu posiada ponad połowa populacji świata; sam wzrost ilości użytkowników sieci Web z Afryki od roku 2000 do 2017 wynosi aż 8503.1% ([3]) – znaczną część tych połączeń stanowią jednak połączenia starych generacji sieci komórkowej, o znacznie ograniczonej przepustowości i ogromnych latencjach. Każdy kolejny węzeł niezbędny do połączenia użytkownika z serwerem końcowym może dokładać cennych milisekund czasu odpowiedzi.

Autorzy IPFS zauważają w swojej pracy ([4]) także inne słabości obecnego internetu – sieć oparta o HTTP jest w prawdzie zdecentralizowana, jako iż treści rozdzielane są pomiędzy miliony węzłów, od gigantycznej sieci Amazon Web Services aż po mikroserwery stojące w domach pasjonatów; brakuje jej jednak faktycznego rozproszenia: ta infrastruktura nie jest gotowa *by design* na przyjmowanie gigantycznego ruchu, nie jest w stanie efektywnie przechowywać i udostępniać wielkich zestawów danych; jest również podatna na znikanie danych, jako iż awaria pojedynczego dysku twardego może zatrzymać udostępnianie całej witryny.

2.1. Zarys historyczny

W świecie informatyki szybko wyewoluowały **rozwiązania współbieżne**. Istnienie wielu wątków – niezależnych od siebie logicznie toków wywołań, operujących na wspólnej pamięci – i ich przeplot rozwiązywał takie problemy, jak nierówny rozmiar żądań – działający sekwencyjnie serwer blokowałby się przy poleceniach zajmujących więcej czasu. Dzięki przeplataniu wątków można uniknąć tej sytuacji, a także pozwolić na iluzję symultaniczności – mimo korzystania z jednego procesora, polecenia wykonywane są *pseudorównolegle*, co pozwala między innymi na responsywność interfejsów użytkownika.

Upowszechnienie się komputerów wieloprocesorowych i procesorów wielordzeniowych zaowocowało rozwojem **obliczeń równoległych**. Istotne jest rozgraniczenie tych dwóch pojęć – współbieżność jest raczej paradygmatem, sposobem strukturyzacji oprogramowania w sposób,

który pozwala na wykonywanie wielu poleceń niezależnie i jednocześnie; równoległość z kolei to możliwość uruchamiania tego typu oprogramowania w tym samym czasie, dzięki mnogiej liczbie procesorów ([5]).

Współbieżność i równoległość rozwiązują problemy obciążenia, ułatwiając operacje na współdzielonej pamięci. Istnieją całe klasy algorytmów naturalnie podatnych zrównoleglaniu: mapowanie, grupowanie, przeszukiwanie czy sortowanie są jednymi z przykładów. Choć w niektórych przypadkach adaptacja algorytmu wymaga jego konkretnej modyfikacji, możliwość rozwiązywania składowych problemu jako niezależnych wątków znacznie przyspiesza wykonanie w środowiskach wieloprocesorowych ([6]).

Skalowanie rozwiązań tego typu może jednak trafić na ścianę kosztów – niezbędne są duże ilości pamięci RAM i szybkie dyski; cierpi też niezawodność systemu, gdyż jego działanie pozostaje kompletnie uzależnione od utrzymania przy życiu konkretnego komputera. Jeśli komputer musi też propagować dane w sieci, cała komunikacja spoczywa na nim – i wydajność jego łącza może zawieść w przypadku szczytów obciążenia.

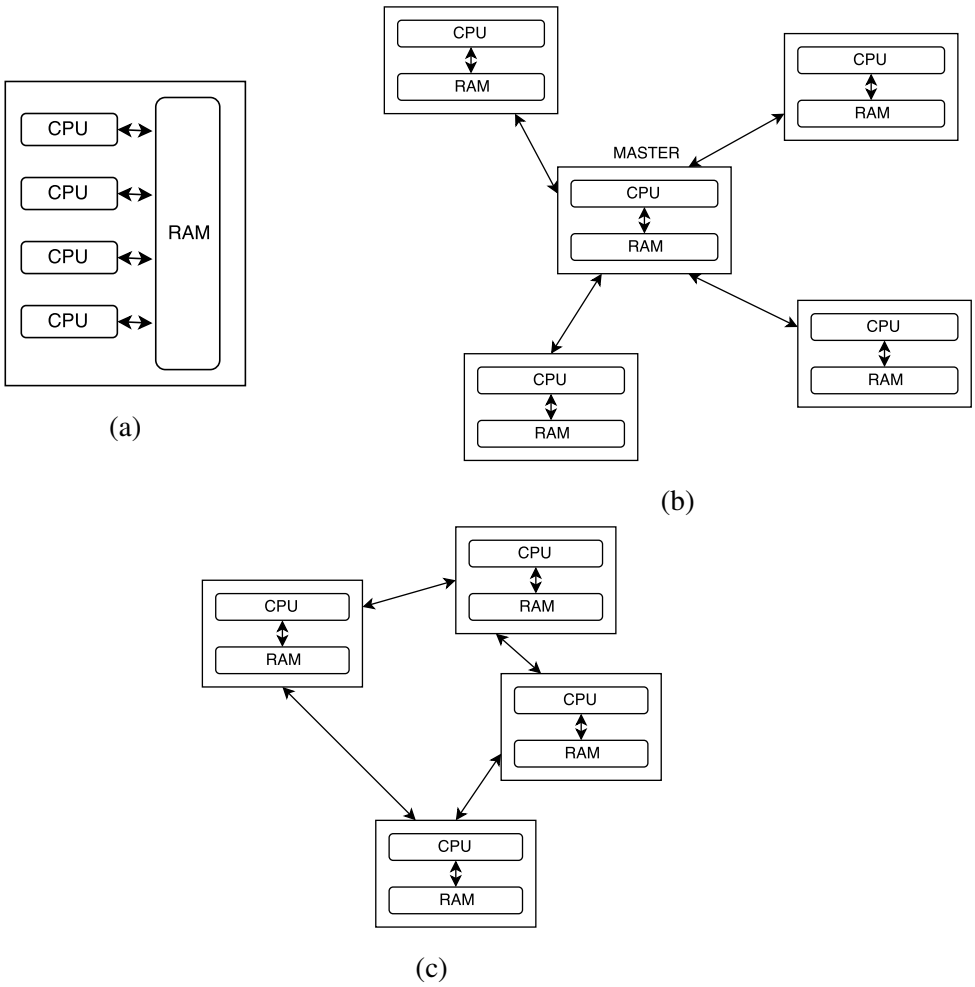
Remedium dla powyższych problemów jest próba rozdzielenia pracy pomiędzy wiele komputerów – tu pojawia się **decentralizacja**, podział problemu na podproblemy, którymi zarządzać mogą mniejsze węzły główne, mające dostęp do węzłów roboczych. To rozwiązanie stawia też nowe wyzwania przed projektantami oprogramowania, którzy nie mogą już polegać na współdzielonej pamięci – dane muszą być przekazywane jakąś sieciową metodą komunikacji.

Obliczenia decentralizowane uznać można za podklasę **obliczeń rozproszonych**. Dla potrzeb tej pracy, obliczenia rozproszone zdefiniowane zostaną jako prowadzone przez w pełni autonomiczne węzły, które dzielą między sobą pracę i radzą sobie z pojawianiem się i znikaniem kolejnych elementów sieci; innymi słowy, w warstwie logicznej system równoległy powinien zachowywać się identycznie zarówno przy istnieniu zaledwie jednego węzła sieci, jak i przy milionie – różnicą powinien być wyłącznie spadek wydajności systemu.

2.2. Narzędzia w rozproszeniu danych

Najbardziej istotnym problemem z dziedziny rozproszenia dla systemu dHTTP jest rozproszenie danych – próba propagowania informacji pomiędzy wiele węzłów sieci, w celach takich jak podniesienie wydajności, obniżenie kosztów, zwiększenie dostępności czy zapewnienie trwałości informacji.

Dobrym przykładem rozwiązania rozproszonego w operacjach na danych jest system kontroli wersji `git`. Został stworzony przez Linusa Torvaldsa, autora jądra Linux, jako otwarta alternatywa dla systemu BitKeeper, który wycofał z dystrybucji darmową wersję swojej aplikacji projektom open source. `git` jest projektem czysto rozproszonym – każdy użytkownik posiada kompletną kopię repozytorium, z informacjami o wszystkich zmianach i ich autorach, na której może wykonywać operacje w trybie offline; nie jest uzależniony od istnienia centralnego serwera (choć jego stosowanie jest powszechne; każdy użytkownik o aktualnej wersji



Rys. 2.1. Wizualizacja różnicy pomiędzy systemem równoległym (a), jednym z podgrafów systemu zdecentralizowanego (b) i rozproszonym (c).

repozytorium mógłby jednak pełnić jego rolę), a zmiany mogą być propagowane na różne sposoby, takie jak email czy przekazywanie łat w formie tekstowej. Dzięki swojej wydajności i silne zdecentralizowanemu designowi, `git` jest chętnie wybieranym rozwiązaniem nie tylko w środowiskach programistów, a w środowiskach profesjonalnych wypiera konkurencyjne rozwiązania, takie jak Apache Subversion czy CVS ([7]).

Nie zawsze jednak motywacje stojące za wyciąganiem danych z centralnych serwerów były szczytne. Duży wpływ na rozwój technik stosowanych obecnie – w tym wykorzystywanych przez dHTTP – mają rozwiązania rozpowszechnione głównie przez piractwo komputerowe. Rozwiązania peer-to-peer – polegające na współpracy równorzędnych węzłów – zostały spopularyzowane dzięki aplikacji Napster, która w roku 2001 zgromadziła około 80 milionów użytkowników. Zaprojektowana jako system współdzielenia plików, wyspecjalizowała się w łatwym udostępnianiu plików MP3, z reguły nieleganie. Napster wzburzył wiele kontrowersji – w systemie pojawiały się niewydane jeszcze utwory znanych artystów, prowadząc do milionowych strat i procesu, który pogrążył działanie systemu. Problemem Napstera była architektura oparta o centralny serwer indeksujący – każdy podpięty węzeł informował o posiadanych przez siebie

plikach, a punkt centralny był niezbędny do przeszukiwania bazy plików i przekazywania poleceń pobrania innym węzłom. To rozwiązanie było niebezpieczne, i pozwoliło obciążyć twórców programu odpowiedzialnością za szerzone w nim treści.

Błędy Napstera zostały zauważone przy kolejnych projektach tego typu. Gnutella, chcąc uniknąć istnienia *single point of failure*, rozgłaszała polecenia do wszystkich maszyn w sieci, co owocowało jednak drastycznym spadkiem wydajności ([8]).

Alternatywne podejście podjęte zostało przez sieć Freenet, gdzie zastosowano heurystyczny routing oparty o klucze; każdy plik otrzymuje klucz, podobne klucze lądowały na zbliżonych do siebie węzłach, dzięki czemu przeszukiwanie sieci nie wymagało punktu centralnego, a polecenia kierowane były w sposób nie wymagający wielu przeskoków. Niestety, ta metoda nie gwarantowała znalezienia danych ([9]).

Chcąc pogodzić kwestie bezpieczeństwa, wydajności i spójności systemu, rozpoczęte zostały intensywne prace nad rozproszonymi tablicami haszującymi (distributed hash table – **DHT**).

2.3. Podstawa projektu

Wspomniane rozwiązania stanowią kanwę dla projektu IPFS, który stara się tworzyć efektywny, realnie rozproszony system plików, obsługujący gigantyczne zasoby danych z niskimi latencjami, redundancją danych i bez wymogu zaufania uczestnikom sieci. Dane przechowywane w drzewie Merkla, dzięki czemu przeszukiwanie sieci ma niską złożoność obliczeniową. Projekt skupia się na warstwie infrastruktury – po stronie użytkowej przypomina nieco system kontroli wersji `git`, a dane indeksowane są dzięki sumom kontrolnym. Takie podejście doskonale sprawdza się w skryptach i przy pracach prowadzonych przez ludzi obytych z tego typu narzędziami, nie jest to jednak system gotowy dla użytkowników domowych. Celem tej pracy jest adaptacja dokonań projektu IPFS jako „przezroczysty" system wsparcia HTTP. System taki musi spełniać szereg wymagań: po pierwsze, nie może próbować „zastąpić" istniejących rozwiązań. Internet w obecnym kształcie nie jest gotowy na rewolucje – przykładem może być poziom przyjęcia się IPv6 ([10]), następnej generacji protokołu IP, która ma między innymi rozwiązać problem rozmiaru puli adresów. Brak realnej kompatybilności wstecznej sprawia, że konieczne są rozwiązania przejściowe. dHTTP jest bardziej nakładką, rozwinięciem obecnych rozwiązań – co daje perspektywy spokojnego przyjęcia takiego rozwiązania przez rynek.

IPFS posiada wzorcową implementację w języku Go, i jest podstawą stosu sieciowego `libp2p`. Zbiór bibliotek udostępniony w projekcie `js-libp2p` stanowi fundament projektu dHTTP, który, dzięki natywnej implementacji w JavaScripcie, może być w pełni uruchamialny z poziomu przeglądarki, zapewniając użytkową przezroczystość odbiorcy.

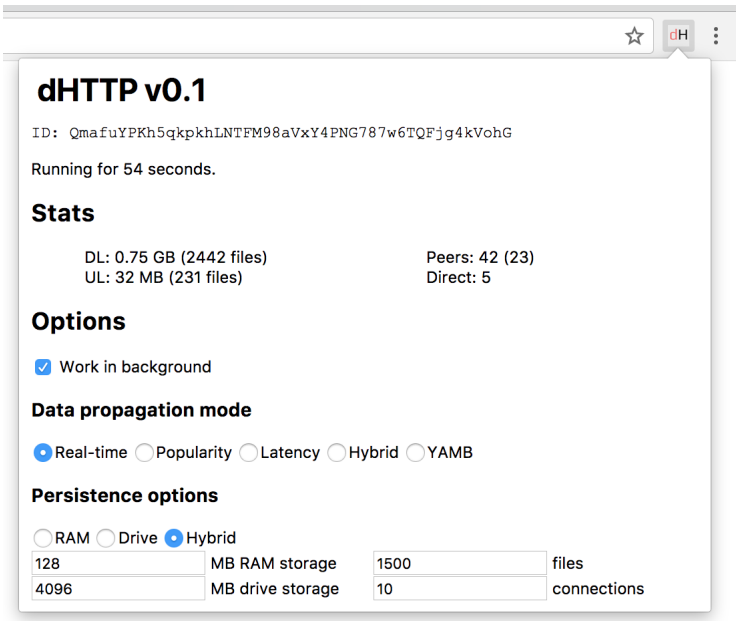
3. Projektowanie i implementacja

Nakreślony problem – rozproszonego systemu, który dzięki propagacji danych między użytkownikami obniży obciążenie serwerów sieci Web – stawia wiele wyzwań projektowych i architektonicznych.

3.1. Wymagania funkcjonalne

System dHTTP, z punktu widzenia użytkownika, ma spełniać jedną funkcjonalność: utrzymać lub poprawić płynność dostępu do interesujących go witryn internetowych, nie wpływając na ich treść i nie naruszając prywatności.

Projekt udostępnia interfejs zapewniający dostęp do statystyk, a także preferencji użytkownika. Udostępnione preferencje dotyczą: stopnia działania aplikacji w tle, trybów propagacji i przechowywania danych.



Rys. 3.1. Wstępna implementacja interfejsu operacji dla systemu dHTTP: pop-up dostarczany przez wtyczkę do przeglądarki Google Chrome pozwala na obserwację statystyk i zmianę preferencji.

Wymogiem dla projektu, koniecznym z racji potrzeby prostej automatyzacji rozrostu sieci, jest tryb niezależny aplikacji – *headless mode* – pozwalający na wystartowanie niezależnego

węzła jednym poleceniem. Niezbędnym jest, aby węzeł tego typu udostępniał statystyki użycia i wstępną konfigurację przy użyciu poleceń interfejsu konsolowego, pozwalając jednak na funkcjonalne uruchomienie z domyślną konfiguracją.

3.2. Definicje, architektura i technologie

W celu uniknięcia niejednoznaczności w dalszym toku pracy, zdefiniowane zostaną następujące pojęcia:

– **węzeł** – pojedynczy, autonomiczny element systemu, który wykorzystuje komunikację sieciową w celu rozgłaszania i pobierania danych.

– **klaster**

– The third etc ...

3.2.1. Wykorzystane technologie i narzędzia

`libp2p` stanowi względnie wysokopoziomową kanwę dla projektu dHTTP.

3.2.2. Problemy warstwy sieciowej

3.2.3. Decentralizacja czy rozproszenie?

3.2.4. Bezpieczeństwo

3.2.4.1. Czy autentykacja to nasza brożka?

3.2.4.2. Gdzie leżą granice zdrowych heurystyk?

3.2.5. Propagacja i przechowywanie danych

3.3. Aplikacje

3.3.1. Węzeł

3.3.2. Klient

4. Testy i optymalizacja

4.1. Zarys użytkowości i działania systemu

4.2. Wydajność systemu

4.2.1. Wydajność klienta

4.2.2. Wydajność serwera

4.2.2.1. Wydajność rozwiązań propagacji danych

4.2.2.2. Wydajność protokołów komunikacji między węzłami

4.3. Podsumowanie rozwiązań optymalizacyjnych

5. Podsumowanie

5.1. Potencjalne kierunki rozwoju

Bibliografia

[1] Amazon Web Services Inc. *Auto Scaling*. 2017. URL: *http://web.archive.org/web/20171127225615/https://aws.amazon.com/autoscaling/* (term. wiz. 2017-12-31).

[2] Gordon E Moore. „Cramming more components onto integrated circuits". W: *Proceedings of the IEEE* 86.1 (1998).

[3] Miniwatts Marketing Group. *World Internet Users and 2017 Population Stats*. 2017. URL: *http://web.archive.org/web/20171226094500/http://www.internetworldstats.com/stats.htm* (term. wiz. 2017-12-31).

[4] Juan Benet. „IPFS - Content Addressed, Versioned, P2P File System". W: *CoRR* (2014).

[5] Rob Pike. „Concurrency is not Parallelism". W: 2012.

[6] Sebastian Wyngaard Simon Perkins James Gain Michelle Kuttel i Jason Brownbridge. „Parallel Algorithms". W: 2011.

[7] Karthik Ram. „Git can facilitate greater reproducibility and increased transparency in science". W: *Source code for biology and medicine* 8.1 (2013), s. 7.

[8] Stefan Saroiu, Krishna P Gummadi i Steven D Gribble. „Measuring and analyzing the characteristics of Napster and Gnutella hosts". W: *Multimedia systems* 9.2 (2003), s. 170–184.

[9] Oskar Sandberg. „Searching in a Small World". Prac. dokt. Göteborg University, 2005.

[10] Google Inc. *IPv6 Statistics*. 2017. URL: *http://web.archive.org/web/20171220144243/https://www.google.com/intl/en/ipv6/statistics.html* (term. wiz. 2017-12-31).