Google – jak znaleźć igłę w stogu siana

Marcin Sochacki
wanted@linux.gda.pl

Pingwinaria, Krynica, 2006

Streszczenie

W artykule przedstawiono skróconą historię firmy Google oraz opis filozofii będącej podstawą sukcesu w branży internetowej. Zasadniczą część poświęcono systemowi plików GFS, który jest wykorzystywany do przechowywania nieprzeciętnej ilości danych. Praca jest inspirowana wcześniejszymi artykułami następujących osób: Urs Hölzle, Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. Autor jest od kwietnia 2005 roku pracownikiem Google Ireland Ltd. na stanowisku Cluster System Administrator.

1 Krótka historia firmy

Choć w branży komputerowej 8 lat jest stosunkowo długim okresem, większość specjalistów ocenia rozwój Google jako niezwykle dynamiczny. Od typowej niewielkiej firmy garażowej z Doliny Krzemowej do potentata rynku internetowego konkurującego z tak znanymi wcześniej firmami jak Yahoo! czy Microsoft – taki plan rozwoju z pewnością mógł być tylko w sferze marzeń dwóch ambitnych studentów Uniwersytetu Stanforda w Kalifornii. Historia ich pomysłu jest szerzej opisana w licznych miejscach w Sieci, np. w Wikipedii[1] czy na stronie firmowej[2].

Obecnie główną siedzibą firmy są budynki przejęte od SGI w Mountain View, USA gdzie pracuje około 3000 osób. Ponadto powstało wiele oddziałów międzynarodowych, aktualnie jest ich ponad 20 w różnych krajach, od niedawna również wliczając w to niewielkie biuro w Polsce. Głównym oddziałem na Europę jest Google Ireland Ltd. w Dublinie, zatrudniające obecnie 700 osób.

[1] Artykuł *Google* w Wikipedii PL, http://pl.wikipedia.org/wiki/Google
[2] *Google Milestones*, http://www.google.com/intl/en/corporate/history.html

2 Filozofia działania

2.1 Misja

Misją Google jest „zorganizować wszelkie informacje dostępne na świecie i uczynić je uniwersalnie dostępnymi i użytecznymi". Realizację tej misji widać w kolejnych przedsięwzięciach wykraczających poza ramy tradycyjnej wyszukiwarki internetowej. Na szczególną uwagę zasługuje serwis Google Book Search (wcześniej znany pod nazwą Google Print), dzięki któremu można przeszukiwać zawartość tysięcy książek udostępnionych przez biblioteki w USA i samych wydawców. Założyciele firmy jasno zdefiniowali cel tego serwisu – mnóstwo cennej wiedzy zgromadzono na półkach bibliotek czy księgarni, ale wiedza ta jest słabo dostępna ze względu na brak możliwości przeszukiwania ich zawartości, niewygodne katalogi itp. Jest to dobry przykład informacji dotychczas żyjącej głównie w świecie *offline* przeniesionej na grunt Internetu.

2.2 Skala działania

Biorąc pod uwagę ilość danych, które trzeba przetworzyć w ramach serwisów Google, jest to z pewnością jeden z trudniejszych problemów współczesnej informatyki. Samo WWW chociażby ocenia się na co najmniej kilkanaście miliardów stron, każda o rozmiarze średnio 10 KB. W sumie kilkaset terabajtów samego tekstu. Do tego należy doliczyć wyszukiwarkę grafik, Usenetu, zawartość książek w Book Search, filmy z Google Video itd. Internet rośnie w postępie geometrycznym, zatem i sama firma musi nadążać za tym rozwojem.

Oczywistą sprawą jest konieczność posiadania bardzo rozbudowanej infrastruktury aby utrzymać te serwisy – składają się nań tysiące komputerów, petabajty dysków i szybka sieć. Jednak nie ilość sprzętu, ale oprogramowanie działające na nim decydują o ostatecznym powodzeniu projektu. Kluczowymi wymaganiami takiego systemu są: stabilność pracy, skalowalność, szybkość odpowiedzi i oczywiście rozsądny koszt wdrożenia.

3 Architektura systemu

3.1 Sprzęt

Pamiętając o skromnych początkach firmy, nie dziwi fakt, że twórcy postanowili wykorzystać najtańszą z dostępnych platform sprzętowych (PC) i jeden z otwartych, darmowych systemów operacyjnych – Linux. Ta filozofia – poczatkowo wymuszona niskim budżetem – okazała się słuszną nawet dla bogatej firmy jaką jest dzisiaj.

Architektura systemu musiała być oczywiście rozproszona – nie istnieją tak silne komputery, które podołałyby podobnemu zadaniu w pojedynkę. Wiele komputerów łączy się w klastry, które realizują określone zadania szybko i niezawodnie, mimo że pojedynczy element klastra jest niezbyt szybki i zawodny.

Decyzja o korzystaniu z taniej architektury PC (x86) miała oczywiste konsekwencje. Sprzęt jest awaryjny, np. średnio jakiś element PC psuje się raz na trzy lata. Mając 1000 komputerów można statystycznie spodziewać się jednej awarii dziennie. Dzięki odpornemu na awarie oprogramowaniu i ciągłemu monitorowaniu sprzęt można naprawiać bez pośpiechu i obawy o bezpieczeństwo systemu – technicy pracujący w serwerowni co jakiś czas generują listę zepsutych komponentów, wymieniają je na nowe hurtem i włączają naprawione maszyny do obiegu. Od skromnych początków przetestowano wiele konfiguracji sprzętowych. Jedną z pierwszych była tzw. *corkboard*, czyli płyta korkowa.[3]

Wówczas (koniec lat 90. XX w.) trzeba było polegać na dostępnych w sprzedaży komponentach sprzętowych. W ramach minimalizacji kosztów i problemów z chłodzeniem, w pierwszym rzędzie zrezygnowano z tradycyjnych (i drogich) obudów rackowych. Płyty główne leżały na metalowych tackach zamontowanych w racku, oddzielone dla izolacji cienką warstwą korka (stąd nazwa kodowa). Ponieważ standardowa płyta główna miała 10 cali szerokości, a rack – 19 cali w celu zaoszczędzenia miejsca upychano dwie płyty na jednej tacy. Oczywiście płyty zachodziły na siebie z marginesem jednego cala, jednak po zaizolowaniu nie stanowiło to większego problemu. Płyty główne były zupełnie standardowe za wyjątkiem faktu, że od początku budowano wyłącznie systemy dwuprocesorowe (SMP). W owym czasie były to procesory Intel Pentium II. Na płytach, dość swobodnie leżały dyski twarde. W kolejnym kroku optymalizacyjnym zdecydowano, że obie płyty będą korzystały z jednego za-

[3] Kompletny rack trafił nawet do Muzeum Historii Komputerów w Mountain View, CA, USA, http://computerhistory.org/.

silacza. Niewątpliwie udało się osiągnąć cel, tj. gęste upakowanie mocy – w racku mieściło się 40 komputerów. Większość komercyjnych serwerowni pobierała opłaty za zajętą powierzchnię podłogi, a nie np. pobór prądu. Stąd pod względem kosztów platforma *corkboard* była bardzo tania. Nietrudno jednak wyobrazić sobie koszmar związany z serwisowaniem tych komputerów. Np. w celu naprawy jednej płyty trzeba było też wyłączyć jej zdrowego sąsiada z uwagi na wspólne zasilanie.

Współczesna platforma serwerowa jest znacznie bardziej przemyślana pod względem łatwości naprawy. Sprawni operatorzy są w stanie wymienić dysk czy pamięć w pół minuty, całą płytę w ciągu dwóch minut. Przede wszystkim stosowane są płyty główne budowane na zamówienie zgodne z projektem. Np. zupełnie zbędny chip karty graficznej czy dźwiękowej można usunąć i zaoszczędzić nieco energii. Problemy z zasilaniem i chłodzeniem są w tej chwili największą barierą rozwoju. Duża serwerownia pobiera kilka megawatów mocy, z czego większość jest przetwarzana na ciepło, które następnie trzeba wypompować na zewnątrz. Optymalizacja pod względem oszczędności energii jest obecnie priorytetem przy projektowaniu serwerowni. Poważnym problemem są np. straty energii przy konwertowaniu jej w tradycyjnym zasilaczu z prądu przemiennego wysokiego napięcia na stały o niskim napięciu. Można np. zastosować bardziej efektywne zasilacze bądź konwertować energię na poziomie serwerowni i do racków dostarczać już niskie napięcie.

Ostatecznym kryterium jest koszt sprzętu na jedno zapytanie użytkownika, wyrażony jako suma początkowej inwestycji i kosztów utrzymania podzielona przez wydajność. Realistycznie patrząc, czas życia sprzętu wynosi około trzech lat z uwagi na szybki rozwój architektury x86. Maszyny starsze niż trzy lata są tak wolne w porównaniu z współczesnymi, że stanowią poważny problem dla poprawnego dystrybuowania obciążenia i konfiguracji oprogramowania.

3.2 Oprogramowanie

Rozwiązaniem problemu awaryjnego sprzętu jest sprytne oprogramowanie, które działając na klastrze komputerów jest odporne na awarie. Ponadto z oczywistych względów każda część infrastruktury jest wielokrotnie replikowana. Nie tylko zwiększa to bezpieczeństwo danych w razie awarii (kopia zapasowa), ale zwiększa też szybkość odpowiedzi. Chcąc np. wyeliminować wąskie gardło wystarczy zwykle dodać więcej serwerów realizujących określone zadanie i system automatycznie zacznie z nich korzystać balansując obciążenie w ramach danej grupy. Serwe-

ry pierwszoplanowe (widoczne dla użytkownika z zewnątrz, *frontend*) utrzymują połączenia TCP z serwerami wewnętrznymi (*backend*), dzięki temu kolejne zapytania nie generują nowych połączeń, a są przesyłane istniejącym kanałem.[4] To samo zapytanie jest przesyłane do wielu maszyn; pierwsza która udzieli odpowiedzi „wygrywa", awaria czy wolniejsze działanie jednej z nich nie jest widoczne dla użytkownika.

Istnieje kilka zasad pisania odpornego oprogramowania obowiązujących w Google. Zasada numer jeden to „najprostsze jak się da".[5] Program powinien wykonywać jakąś niedużą część zadania, za to wykonywać ją dobrze i niezawodnie. Łącząc wiele takich programów w łańcuch otrzymujemy system, który daje się łatwo skalować (możliwość dodania większej liczby serwerów danego typu) jak i jest łatwy w diagnostyce (problem można ograniczyć do jednej „cegiełki" zamiast ogromnego systemu). Opracowano w tym celu cały szereg oprogramowania wspomagającego, zestaw protokołów, specyfikacji, narzędzi itp. – dzięki czemu niezależne grupy programistów mogą tworzyć fragmenty systemu współpracujące ze sobą bez większych problemów. Ponadto firma realizuje wewnętrzną politykę otwartych źródeł (*open source*) – każdy inżynier może przeglądać i wykorzystywać kod źródłowy innych.

Częścią optymalizacji systemu jest również interfejs użytkownika – opracowany tak, aby wyświetlał się poprawnie i nie był przeładowany elementami dekoracyjnymi, co nawet przy słabym łączu przejawia się w wygodnym korzystaniu ze strony.

4 Google File System (GFS)

Jednym z kluczowych elementów infrastruktury Google jest naturalnie przechowywanie ogromnych ilości danych. Powstała zatem potrzeba opracowania systemu plików odpowiedniego do potrzeb firmy – jest nim GFS (Google File System).

Projektując GFS programiści musieli wziąć pod uwagę kilka ograniczeń i wymagań:

- dyski IDE są tanie, ale zawodne,
- z drugiej strony jednak IDE oferują dużą pojemność i są całkiem szybkie,

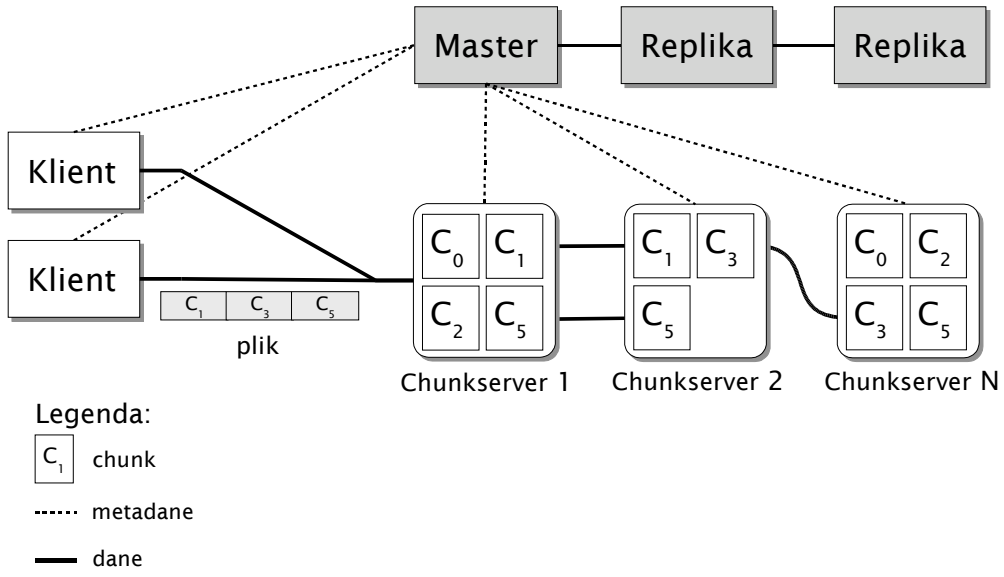
[4] TCP wymaga przesłania trzech pakietów wyłącznie do nawiązania połączenia, które z punktu widzenia optymalizacji są poważną stratą czasu.

[5] . . . zgodnie z regułą KISS (*Keep It Simple, Stupid*) – czasem tłumaczoną na polski jako reguła BUZI (*Bez Udziwnień Zapisu, Idioto*).

- największe macierze RAID czy inne komercyjne rozwiązania w branży przechowywania danych to wciąż za mało,
- system plików musi być wydajny i odporny na awarie.

Rozwiązanie jakie wdrożono bazuje oczywiście na Linuksie i standardowym systemie plików ext2. W klastrze GFS może pracować do kilku tysięcy komputerów widocznych dla użytkownika jako jeden system plików o pojemności rzędu petabajtów i średnim transferze od kilku gigabajtów na sekundę wzwyż.

GFS nie jest typowym systemem plików – nie operuje bezpośrednio na sektorach dysku twardego. Jest strukturą logiczną, która może działać na bazie dowolnego natywnego systemu plików w Linuksie (tutaj akurat ext2, ale nie ma przeszkód aby to był ext3, reiserfs czy XFS). Architekturę systemu przedstawiono na Rysunku 1.



Rysunek 1: Architektura GFS

Podstawową koncepcją GFS jest podział plików na „kawałki" – chunki, które jednak w przeciwieństwie do tradycyjnych systemów plików mogą być znacznie większe – do 64 MB. GFS jest zoptymalizowany do przechowywania dużych plików, jednak do większości zastosowań jest to wystarczające (np. małe pliki są „sklejane" w większe archiwa). W celu zapewnienia bezpieczeństwa danych, każdy chunk jest replikowany –

domyślnie trzykrotnie (poziom replikacji jest konfigurowany przez użytkownika). Integralność danych zapewniają sumy kontrolne na poziomie chunków.

Na klaster GFS składają się trzy typy serwerów: master, replika i chunkserver. **Master** jest kontrolerem danego klastra, zarządza przepływem danych, przechowuje metadane, przydziela blokady do zapisu itp. Na metadane składa się przede wszystkim struktura systemu plików, uprawnienia, typowe atrybuty Uniksowe oraz mapowanie plik↔chunki. Model Uniksowy został jednak dość znacznie uproszczony, zaimplementowano tylko niezbędną funkcjonalność (np. brak obsługi linków symbolicznych). **Replika** jest praktycznie kopią mastera, przechowuje logi wszelkich transakcji, jednak nie ma możliwości przydzielania blokad zapisu – inaczej dochodziłoby do konfliktów. Nie ma przeszkód aby korzystać z replik do operacji odczytu – zmniejsza to obciążenie na masterze. Replika może zostać promowana na mastera, np. w przypadku jego awarii. **Chunkserver** natomiast jest właściwym miejscem przechowywania danych, zwykle jest zatem wyposażony w duże dyski. W typowym klastrze jest kilka kontrolerów i od kilkunastu do kilku tysięcy chunkserverów. Aplikacje klienckie komunikują się w pierwszej kolejności z masterem wymieniając metadane (tj. informacje na temat samych danych – np. lokalizacje chunków). W drugim kroku aplikacja kontaktuje się bezpośrednio z właściwym chunkserverem i pobiera/zapisuje chunk. Takie podejście zapewnia tak wysoką wydajność GFS – wiele chunkserverów przesyła dane w tym samym czasie. Należy zwrócić uwagę, że nie ma obecnie możliwości „zamontowania" tego systemu, jak np. NFS w drzewie katalogów – bo nie było takiej potrzeby. Wszelkie operacje wykonuje się przy pomocy programów narzędziowych bądź za pośrednictwem bibliotek zlinkowanych z aplikacją użytkownika.

Ani klient ani chunkservery nie korzystają z buforowania danych (ang. *cache*). Cache po stronie klienta oferowałby niewielki zysk, ponieważ większość aplikacji odczytuje sekwencyjnie duże pliki, lub też operuje na zbyt dużych zbiorach danych. Buforowane są jednak metadane – informacje o strukturze systemu plików, lokalizacje chunków itp. Aplikacja może np. zbuforować lokalizacje chunków składających się na kilkuterabajtowy zbiór danych kontaktując się jednokrotnie z masterem i następnie już komunikując się wyłącznie z odpowiedniemi chunkserverami. Redukuje to znacząco obciążenie mastera. Z kolei na chunkserverach cache jest zbędny, ponieważ operujemy na bazie istniejącego systemu plików ext2 w Linuksie, który buforuje najczęściej używane dane.

Spójne zarządzanie modyfikacjami plików osiągnięto poprzez nadanie chunkom dodatkowego atrybutu – identyfikatora wersji. Każda modyfi-

kacja chunka zmienia identyfikator wersji, co umożliwia np. aktualnie czytającym programom odwołanie się do starszej kopii danego chunka aby zapewnić spójność operacji. Bez wersjonowania dochodziłoby do sytuacji, że w trakcie odczytu dużego pliku inny proces mógłby go zmodyfikować i w rezultacie odczytany plik byłby uszkodzony.

Master dba również o zapewnienie bezpieczeństwa i wydajności klastra. W pierwszym rzędzie zajmuje się **klonowaniem** niewystarczająco zreplikowanych chunków. Np. bezpośrednio po zapisie pliku, jego chunki mają tylko jedną kopię. Master natychmiast zleca odpowiednim chunkserverom ich replikację. Oczywiście dane są przesyłane bez pośrednictwa mastera, jedynie po zakończeniu operacji otrzymuje jej status. Podobnie dzieje się np. po awarii jednego z chunkserverów. Aby wykrywać awarie, master bez przerwy monitoruje stan serwerów, otrzymując status na temat sprawności poszczególnych dysków, łączności, obciążenia itd. wysyłając pakiety zwane *biciem serca* (ang. *heartbeat*).

W ramach dbania o niezawodność master również zleca inny typ operacji – **balansowanie**. Idea polega na tym, że w miarę używania klastra, poziom zapełnienia chunkserverów się różnicuje. Balansowanie przesyła chunki, tak aby osiągnąć w przybliżeniu jednakową zajętość miejsca na dyskach. Nie dotyczy to jedynie operacji lokalnych; balansowanie również działa w makroskali. GFS znając strukturę fizyczną sieci stara się umieszczać repliki chunka na różnych rackach. Dzięki temu wyłączenie całego racka (np. w wyniku awarii zasilania lub switcha) w normalnych warunkach nie powoduje utraty danych. Ponadto, balansowanie ma nie tylko znaczenie dla niezawodności; dzięki równomiernemu rozdystrybuowaniu danych, ogólna szybkość klastra jest optymalna.

Interesującą funkcją jest sposób kasowania plików – zmieniana jest po prostu ich nazwa i są ukrywane. Dzięki temu można łatwo odzyskać przypadkowo skasowane pliki. Po określonym czasie (konfigurowalnym) master przegląda skasowane pliki i kasuje je ostatecznie z chunkserverów. Czynność tę określa się mianem **garbage collection**. Temu samemu procesowi podlegają nieużywane wersje chunków, zgodnie z opisaną wcześniej ideą wersjonowania.

Do rzadziej używanych, ale użytecznych opcji należą *snapshoty*. Zlecając wykonanie snapshotu danego katalogu, master tworzy wirtualną, kopię stanu tego katalogu dostępną tylko do odczytu. Umożliwia to np. wykonywanie spójnych kopii bezpieczeństwa.

Jeśli chodzi o stronę użytkową, jednym z najważniejszych zastosowań GFS jest przechowywanie logów. Przy tysiącach zapytań na sekundę, logowanie parametrów każdego z nich jest trudnym wyzwaniem, które jednak trzeba było podjąć – logi są bardzo ważną częścią systemu, nie-

zbędną do np. rozliczania reklamodawców. GFS został w tym celu zoptymalizowany pod kątem operacji dołączenia danych (ang. *append*) do istniejącego pliku. Operacja ta jest niepodzielna (ang. *atomic*), dzięki czemu wiele klientów może jednocześnie zapisywać logi do tego samego pliku bez jakiejkolwiek wzajemnej synchronizacji.

Z oczywistych względów niniejszy opis GFS jest bardzo skrótowy. Więcej informacji można znaleźć w publikacji autorów systemu.[6] Jak w każdym dużym systemie jest też dużo drobnych, aczkolwiek ważnych szczegółów niezbędnych do administrowania klastrami.

5 Praca w Google

Google jest wciąż dynamicznie rozwijającą się firmą, a do rozwoju potrzebuje dobrych ludzi. Kandydatom stawiane są dość wysokie wymagania, ale w zamian otrzymuje się pracę w zespole prawdziwych specjalistów i pasjonatów.

To co firma może zaoferować inżynierom (programistom, administratorom) to z pewnością dostęp do nadzwyczajnej technologii, tysięcy serwerów, wielkiej mocy obliczeniowej i oczywiście kopii Internetu na lokalnych dyskach :-) Ponadto obowiązuje reguła 20% – tyle czasu można przeznaczyć na wybrany projekt, niekoniecznie związany z głównym zajęciem w firmie. Prowadzi to czasem to ciekawych statystyk – nie jest niczym dziwnym że np. nad danym projektem pracuje 3,60 osób.

Jednak nie tylko inżynierowie są poszukiwani – główną częścią biura w Dublinie jest obsługa sprzedaży (programy reklamowe), wsparcie użytkowników, jak też stosunkowo młody zespół OTE (Online Testing Evaluator) zajmujący się m.in. problemem spamu w sieci. Każdy z tych zespołów potrzebuje ludzi we wszystkich wspieranych językach, stąd oddział irlandzki jest wyjątkowym, międzynarodowym towarzystwem.

Szczegóły na temat rekrutacji można znaleźć pod adresem http://www.google.com/jobs/.

[6] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, *The Google File System*, 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003. http://labs.google.com/papers/gfs.html