Factorer MPI - dokumentacja projektowa

Rafał Duraj, Piotr Dowgiałło, Bartosz Janusz, Maciej Kolański

2016-06-07

Spis treści

1 Temat projektu

W dzisiejszych czasach, gdy właściwie wszystko co robimy w jakiś sposób połączone jest z Internetem bezpieczeństwo jest bardzo ważnym tematem.

Faktoryzacja jest to proces podczas którego dla zadanego obiektu odnajduje się inne obiekty, które spełniają to, że ich iloczyn równy jest oryginalnemu obiektowi, w związku z czym te znalezione czynniki są w pewnym sensie od niego prostsze.

Podstawowy algorytm faktoryzacji bazuje na próbowaniu podziału liczby do faktoryzacji n przez wszystkie liczby pierwsze od 2 do $\sqrt{n}$. Tego typu algorytm bardzo dobrze radzi sobie z początkiem faktoryzacji liczby, bo dowolna liczba ma czynnik zarówno małe jak i duże. Jak wiadomo połowa wszystkich liczb dzieli się przez dwa, jedna trzecia licz przez trzy i tak dalej, a więc z dużym prawdopodobieństwem można pozbyć się w prosty sposób niskich czynników.

RSA jest to jeden z pierwszych i też obecnie najpopularniejszych asymetrycznych algorytmów kryptograficznych gdzie klucz jest publiczny. Bezpieczeństwo szyfrowania przy pomocy tego algorytmu jest związane z trudnością faktoryzacji dużych liczb.

2 Zakres projektu

2.1 Cel

Celem projektu jest umożliwienie zlecenia zadania faktoryzacji dużej liczby (większych od $2^{64} - 1$). Jednym z głównych założeń jest udostępnienie prostego w obsłudze interfejsu użytkownika i zmaksymalizowanie elastyczności – system powinień być zdolny do współpracy z zadanymi komputerami, a instalacja wymaganego oprogramowania musi być prosta.

2.2 Funkcjonalność

Podstawowa

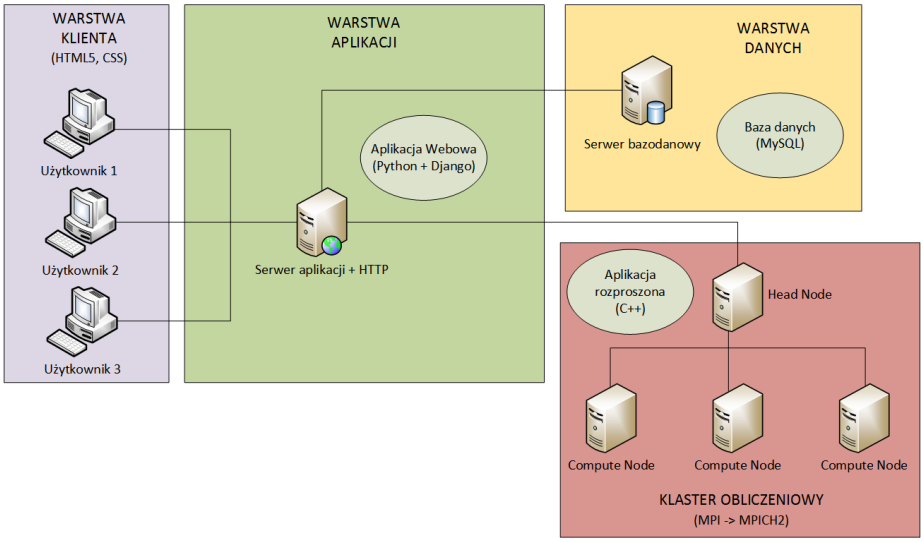
1. Udostępnienie mechanizmów rejestracji i logowania (konta użytkowników)
2. Zarządzanie zadaniami z poziomu interfejsu webowego
   (a) zlecanie zadań
   (b) przegląd historii
3. Możliwość rozbudowy klastra bez ingerencji w kod
4. Faktoryzacja metodą „brutalnej siły”
5. Faktoryzacja metodą CFRAC
6. Zachowywanie wyników pomyślanie wykonanych faktoryzacji

Rozszerzona

1. Łamanie szyfru RSA
2. Faktoryzacja metodą sita kwadratowego
3. Forma graficznej prezentacji wyników pomiarów

3 Narzędzia i technologie zastosowane w projekcie

3.1 Zastosowane technologie



Rysunek 1: Schemat blokowy systemu

Strona klienta - HTML5, CSS Interfejs z którego bedzie korzystal klient projektowanego systemu został napisany w jezyku skryptowym HTML5 z użyciem stylów CSS. HTML5 jest obecnie standardem przy tworzeniu stron internetowych i w wiekszosci wyparl HTML4, w którego specyfikacji bylo duzo niescislosci. Uzycie CSS z kolei pozwoli ujednolicic prezentacje zawartosci w różnych przegladarkach, oraz uprosci organizacje samego kodu.

Aplikacja Webowa - Python 3.4 + Django 1.8.7[1] Python jest jezykiem wysokiego poziomu ogólnego przeznaczenia, z kolei Django to framework webowy dla tego jezyka. Wybralismy ten zestaw z powodu wielu ulatwień przy

tworzeniu aplikacji webowych, które sa przezeń oferowane, np. dynamiczny interfejs bazy danych, automatyczny interfejst administracyjny. Dodatkowo czesc naszej grupy jest zaznajomiona z tymi technologiami, wiec nie ma potrzeby poznawania ich od zera. Istotne sa tutaj wersje srodowisk - najnowsze dystrybucje nie obsluguja MySQL, w zwiazku z tym wybralismy poprzednie.

Baza danych - MySQL SZBD rozwijany przez firme Oracle. Charakteryzuje sie wszystkimi najwazniejszymi funkcjonalnosciami bazy danych oraz prostota tworzenia takiej bazy. Rozważalismy zastosowanie systemu Oracle Database, jednakże jest on zbyt rozbudowany jak na nasze potrzeby, a co za tym idzie trudny w obsudze. Mamy również doswiadczenie w integracji bazy MySQL z aplikacjami napisanymi w Pythonie (Django).

Klaster obliczeniowy - standard MPI[2] MPICH2 to darmowa implementacja standardu MPI dla systemów Linux. Umożliwa proste tworzenie klastrów obliczeniowych, rozdzielania zadań miedzy poszczególne wezly i zbierania wyników. Oferuje interfejs dla jezyka C++. Jest wykorzystywany w wiekszosci topowych urzadzen wieloprocesorowych i ta popularnosc znaczaco wplynelo na jego wybór.

Aplikacaja rozproszona - C++11 Wybór padl na ten jezyk ze wzgledu na jego znajomosc przez czlonków grupy oraz interfejs udostepniany przez srodowisko MPICH2.

3.2 Narzedzia wykorzystane w projekcie

- Aplikacja webowa PyCharm 5.0.4 - IDE do Pythona, obsuguje Django
- Aplikacja rozproszona - CodeLite 9.1 IDE do C++, wersja na system Linux
- Baza danych - developer do MySQL
- Organizacja pracy - Trello (`https://trello.com/`)
- Hosting plików - GitHub (`https://github.com/`)

4 Aktualny stan rynku

GGNFS (GPL'd implementation of General Number Field Sieve) Aktywny rozwój. faktoryzacja liczb do 180 znaków, srednio do 140. Kilka wiekszych liczb tez. W wiekszosci przypadkow program GGNFS jest stabilny dla liczb skladajacych sie do 150-160 znakow. Posiada bugi. Nie jest czarna skrzynka, trzeba miec odpowiednia wiedze zeby go uzywać.

Cunningham Project Projekt faktoryzujacy liczby b^n+-1 dla b=2,3,5,6,7,10,11,12 i duze n.[3]

RSA Factoring Challenge Zawody zorganizowane przez RSA Security. Otwarte zawody dla wszystkich mające na celu zwiększyć zainteresowanie faktoryzacją liczb. Opublikowana została lista pseudopierwszych liczb (rozkładających się na dokładnie dwa czynniki), nazwanych liczbami RSA.Za rozłożenie niektórych z nich wyznaczono pieniężną nagrodę. Najmniejsza z nich, 100-cyfrowa liczba RSA-100 została rozłożona w ciągu kilku dni, ale większość do dziś pozostaje niezłamana.Zawody miały na celu śledzenie rozwoju możliwości komputerów w faktoryzacji. Jest to niezwykle istotne przy wyborze długości klucza w szyfrowaniu asymetrycznym metodą RSA. Postęp w łamaniu kolejnych liczb powinien zdradzać jakie długości klucza można jeszcze uznawać za bezpieczne.[4]

5 Projekt techniczny

5.1 Klaster obliczeniowy - Komunikacja

5.2 Klaster obliczeniowy - Algorytm Brute Force

5.3 Klaster obliczeniowy - Algorytm CFRAC

5.3.1 Opis algorytmu[5][6]

Metoda CFRAC jest algorytmem faktoryzacji liczb całkowitych. Jest to uniwersalny algorytm będący w stanie rozłożyć na czynniki każdą liczbę, nie polegając na żadnych ograniczeniach czy warunkach. Został on opisany przez D.H. Lehmer'a oraz R. E. Powers'a w 1931 roku, oraz wdrożony na komputery pierwszy raz przez Michael'a A. Morisson'a oraz John'a Brillhart'a w 1975 roku.

Algorytm ten bazuje na metodzie faktoryzacji Diaxona. Metoda Diaxona polegała na losowaniu kolejnych liczb 'a' takie, że: $sqrt[n] < a < n$ (gdzie n to liczba, którą chcemy sfaktoryzować) i sprawdzamy(używając algorytmu naiwnego), czy $b^2 = a^2 mod(n)$ jest liczbą B-gładką, dla ustalonego B. Jeżeli tak to dodajemy znalezioną parę do zbioru (liczba B-gładka to taka liczba, której wszystkie dzielniki pierwsze są mniejsze bądź równe dla ustalonego B).

W algorytmie CFRAC idea pozostaje bez zmian, definiowany jest natomiast sposób wybierania par. Zamiast losowania ich wykorzystywany jest ciąg rozwinięcia sqrt(n) w ułamek łańcuchowy. Złożoność obliczeniowa algorytmu CFRAC jest rzędu $O(e^s qrt(2 * logn * loglogn))$ ¡- na wikipedi lepiej przedstawiony wzor

5.3.2 Implementacja[7]

Program generuję rekurencyjnie elementy tablicy, której elementy są ze sobą ściśle powiązane, następującymi wzorami:

$Q[n] = Q[n-2] + q[n-1] * (r[n-1] - r[n-2]) G[n] = 2g - r[n-1] q[n] = G[n]/Q[n] r[n] = G[n] - q[n]Q[n] A[n] = q[n] * A[n-1] + A[n-2] mod N$

gdzie

$Q[-1] = NQ[0] = 1q[0] = gr[-1] = gr[0] = 0A[-1] = 1A[0] = gg = sqrt(N)$

Dla powyższych reguł generowane są rekurencyjnie kolejne rekordy. Przy wygenerowaniu każdego kolejnego "zestawu" wyników badany był element Q[i]. Jeżeli był on możliwy do spierwiastkowania(reszta z pierwiastka kwadratowego jego wartości była równa zeru), to następnym krokiem w celu uzyskania szukanego faktora było obliczenie następującej wartości:

$temp = A[i-1] - sqrt(Q[i])$

Ostatnim krokiem było obliczenie NWD (największego wspólnego dzielnika)pomiędzy uzyskaną wartością (temp), a liczbą dla której szukamy faktora(N). Wykorzystanym algorytmem do obliczania NWD był algorytm euklidesa, który jest aktualnie najefektywniejszym algorytmem wykorzystywanym do tej operacji.

5.3.3 Podział zadań (Master Slave)

Z powodu rekurencyjnego generowania wyników, efektywny podział pracy pomiędzy różne maszyny(slave'y) był bardzo ciężki do zrealizowania. Ostatecznie udało nam się wymyślić sposób podziału prac, który może nie jest najlepszy ale zapewnia w pewnym stopniu poprawe wydajności przy wykorzystaniu wielu maszyn.

Polega on na wprowadzeniu do algorytmu parametru K. Dla każdej maszyny oprócz danej liczby do faktoryzacji wysyłamy również indywidualny dla niej parametr, przez który mnoży on otrzymaną liczbę do faktoryzacji. Dla nowej uzyskanej liczby k*N, algorytm znajduję wyniki w innym czasie. Wadą tego rozwiązania jest nie zawsze poprawny wynik, dlatego po jego odebraniu trzeba sprawdzić jego poprawność (np odrzucić wyniki, które dzielą się przez k)

Należy też wspomnieć, że CFRAC jest efektywnym algorytmem tylko dla dużych liczb, dlatego w celu znacznego skrócenia czasu operacji mniejsze liczby faktoryzowane są prostym algorytmem naiwnym przez maszyne Master.

Algorytm podziału zadań przez maszyna Master wygląda w następujący sposób:

Listing 1: Sekwencja instrukcji algorytmu CFRAC

```
1    Utworz kolejke na wynik
2    Wrzuc pierwsza liczbe (N) do kolejki liczb do faktoryzacji
3    Dopoki kolejka nie jest pusta
4        Dopoki nie znalazles ostatniej liczby pierwszej
```

```
5           Wyciagnij liczbe A z kolejki
6           Jezeli A jest liczba duza //(efektywny CFRAC)
7               Zainicjuj maszyny (Slaves)
8               Rozeslij zadania wraz z indywidualnym parametrem K
9               Wyrzuc aktualna liczbe A z kolejki
10              Wykonuj dopoki nie odbierzesz wynikow od wszystkich maszyn
11                  Czekaj na wynik
12                  Sprawdz jego poprawnosc
13                  Jezeli wynik jest poprawny:
14                      Jezeli jest liczba pierwsza, wrzuc do kolejki
                            wynikow
15                      Jezeli nie jest liczba pierwsza wrzuc go oraz jego
                            iloraz z liczba A do kolejki liczb do
                            faktoryzacji
16                  Jezeli zadna maszyna nie zwrocila poprawnego wyniku,
                        zmien parametr k i wrzuc z powrotem liczbe A do
                        kolejki liczb do faktoryzacji
17          Dla malej liczby uzyj algorytmu naiwnego (bruteforce)
```

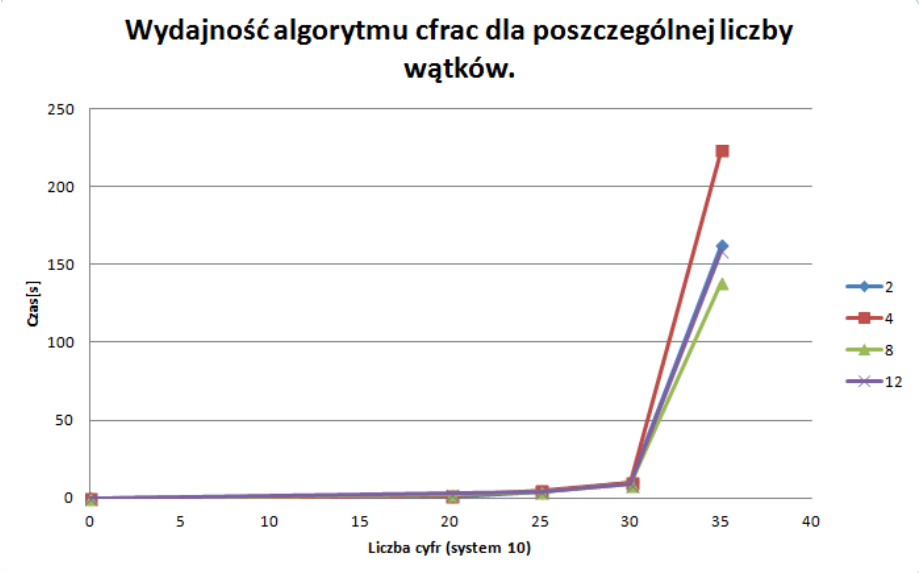
5.4 Strona internetowa

6 Dokumentacja powykonawcza (instalacyjna)

7 Istotne elementy kodu z komentarzem

8 Przykładowe wyniki badań efektywności programu równoległego

8.1 Algorytm CFRAC



Rysunek 2: Wydajność algorytmu CFRAC dla poszczególnej liczby wątków

		Liczba wątków				
liczba bitów	Liczba cyfr(sys. 10)	2	4	8	12	
64-66	20	1,1	1,5	2,5	3,1	Czas[s]
79-82	25	3,9	4,7	4,2	4,2	
96-99	30	10,2	10,1	8,9	8,8	
112 - 116	35	162,2	223,2	138,2	158,7	

Rysunek 3: Dane pomiarów wydajność algorytmu CFRAC

Badania przeprowadzone zostały na pojedyńczym komputerze wykorzystując symulację mpi dla danej ilości wątków.

9 Wnioski

Literatura

[1] *Dokumentacja Django*. https://docs.djangoproject.com/en/1.8/, 2016

[2] *Dokumentacja MPICH2*. http://www.mpich.org/documentation/guides/, 2016.

[3] *Cunningham Project*. http://homes.cerias.purdue.edu/~ssw/cun/

[4] *RSA Factoring Challenge*. http://pl.wikipedia.org/wiki/RSA_Factoring_Challenge

[5] *CFRAC opis*. https://en.wikipedia.org/wiki/Continued_fraction_factorization

[6] *CFRAC zastosowanie*. http://ki.agh.edu.pl/sites/default/files/usefiles/172/theses/mateusz.niezabitowski.algorytmy.faktoryzacji.w.zastosowaniach.kryptograficznych.v1.0-final.pdf

[7] *CFRAC implementacja*. https://math.dartmouth.edu/~carlp/PDF/implementation.pdf