Spark — poszukiwanie spójnej składowej

Jakub Sawicki

24 listopada 2015

1 Analiza PCAM

Analiza rozbita została na bloki: partition, communication, agglomeration oraz mapping. [1]

1.1 Podział

W problemie CC podstawowym elementem jest wierzchołek grafu. Przechowuje on informację o swoim aktualnym numerze porządkowym. Odbiera informacje o numerach porządkowych sąsiadów i uaktualnia swój, na numer sąsiada jeżeli ten jest mniejszy niż jego.

1.2 Komunikacja

Komunikaty wymieniane są pomiędzy wierzchołkami połączonymi krawędziami. Rozważany graf jest nieskierowany, w skierowanym algorytm może dać różne rezultaty w zależności od ułożenia kolejności wierzchołków.

W jednym kroku algorytmu każdy wierzchołek v wysyła $\deg(v)$ komunikatów do swoich sąsiadów i tyle też komunikatów odbiera. W sumie wymienianych jest więc $2\overline{\deg} * n$ komunikatów, gdzie n jest ilością wierzchołków.

1.3 Aglomeracja

Optymalnie było by podzielić graf na pewną liczbę składowych, które są ze sobą dość gęsto połączone. Dzięki temu większość komunikacji odbywała by się wewnątrz danej składowej. Pomiędzy nimi powinno być relatywnie niewiele połączeń. Nie jest jednak łatwo przeprowadzić takiej analizy na dużych grafach.

Nieoptymalny podział może nastąpić poprzez podział wierzchołków na m grup na podstawie funkcji hashującej. Wtedy jeśli średni stopień wierzchołka w grafie wynosi $\overline{\deg}$ to mamy $s_{\text{part}} = \frac{n}{m}\overline{\deg}$ komunikatów wymienianych przez każdą część. Z tego $\frac{m-1}{m}s_{\text{part}}$ wymieniane jest z innymi składowymi.

1.4 Mapowanie

Posiadając informacje o ilości połączeń między poszczególnymi składowymi możliwe jest takie rozmieszczenie ich na węzłach obliczeniowych aby zminimalizować potrzebną komunikację.

2 Implementacja

Algorytm poszukiwania spójnych składowych [2] opiera się na algorytmie PageRank zaimplementowanym w https://github.com/malawski/ar-pagerank. Zrobiłem fork tego repozytorium i dokonałem implementacji na jego podstawie. Jest on dostępny pod adresem https://github.com/jswk/ar-pagerank.

Wprowadzona została dodatkowa komunikacja poprzez akumulator w celu określenia, czy ostateczny wynik został już osiągnięty (brak zmian numerów porządkowych).

3 Wydajność

Pierwsze próby uruchamiania algorytmu dla klastrów różnej wielkości bez ustawionego partitionera dały czasy wykonania, które były stałe ze względu na wielkość klastra.

3.1 Testy dla różnych grafów

Następnie dla obiektów RDD ustawiony został HashPartitioner dzielący wierzchołki na tyle grup, ile zadanie otrzymało przydzielonych procesorów. Testy zostały przeprowadzone dla grafów web-Google, web-NotreDame oraz web-Stanford z [3]. (Graf web-BerkStan miał dużą średnicę i pełne obliczenie zajęło by zbyt dużo czasu.)

Ilość potrzebnych kroków algorytmu jest ograniczona od góry przez średnicę grafu.

- *web-Google* średnica: 21, ilość kroków: 16,
- *web-NotreDame* średnica: 46, ilość kroków: 25,
- *web-Stanford* średnica: 674, ilość kroków: 662.

Wykresy czasu obliczeń oraz efektywności dla poszczególnych grafów pokazane są na Rys. 1.

Dla grafu web-Google zachodzi zjawisko superskalarności. W przypadku, gdy obliczenia wykonywane są na pojedynczym węźle, graf nie mieści się w pamięci lokalnej powodując najprawdopodobniej przeniesienie części pamięci do swapa. Ustawienie w konfiguracji Sparka wyższych limitów pamięci nie zmienia sytuacji. Wyjaśnienie zobacz w 3.3.

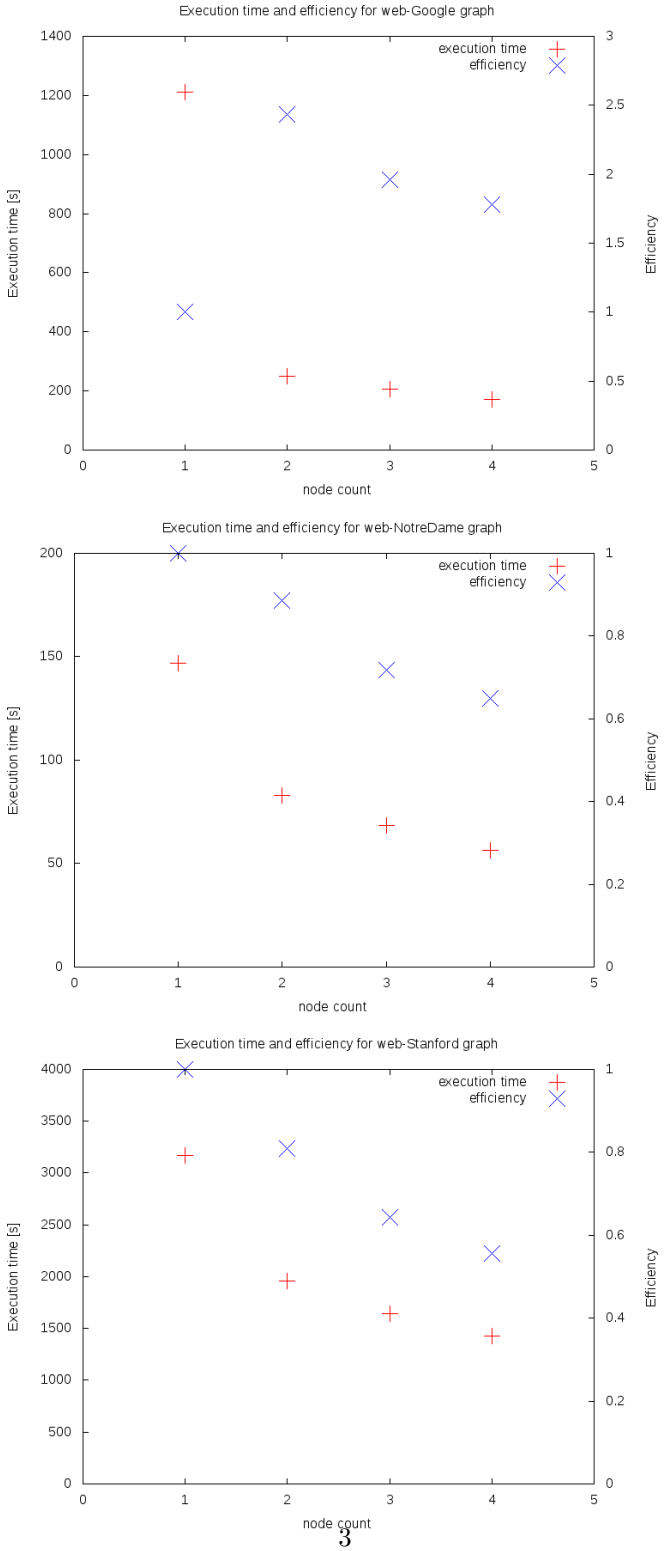
3.2 Multiplikatywność funkcji haszującej

Zbadany został również wpływ multiplikatywności HashPartitioner na czas obliczeń. Na dwóch węzłach z 12 rdzeniami w każdym uruchomione zostało zadanie dla grafu web-NotreDame. Zmieniana była ilość generowanych przez funkcję haszującą wartości. Przetestowane zostały wartości od 2 do 24 z krokiem co 2. Wyniki zaprezentowane są na Rys. 2.

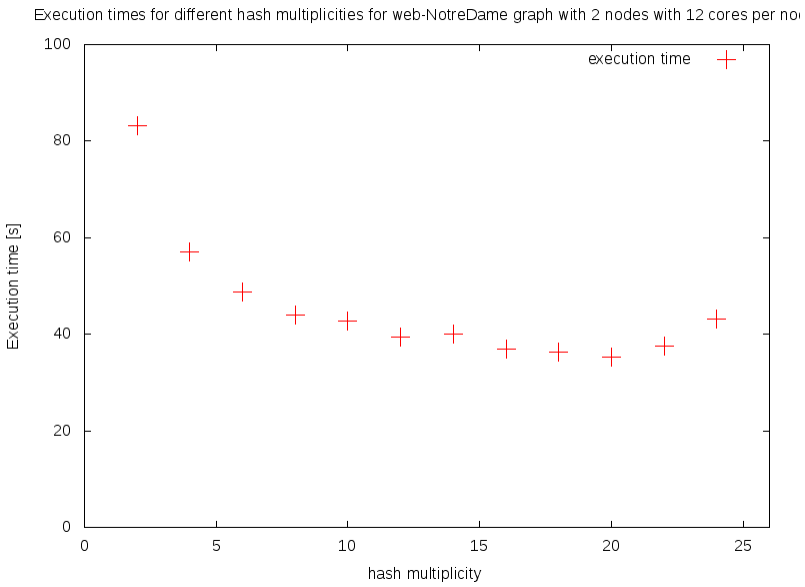
3.3 Graf web-Google a wykorzystanie swap

Sprawdzone zostało w trybie interaktywnym zużycie zasobów przez wykonanie algorytmu dla grafu web-Google. Zarezerwowany został jeden węzeł z 12 rdzeniami.

Obserwacja logów oraz zajętości systemu wykazała, że nie używana była pamięć swap tylko bezpośrednio przestrzeń dyskowa. Widać to na podstawie logów:



Rys. 1: Wykresy pokazują czas wykonania algorytmu poszukiwania spójnych składowych oraz ich efektywność dla testowanych grafów.



Rys. 2: Wykres pokazuje czas obliczeń dla różnej ilości wartości zwracanych przez funkcję haszującą. Obliczenia wykonywane były na dwóch węzłach po 12 rdzeni.

collection.ExternalSorter: Thread 72 spilling in-memory map of 162.1 MB to disk (1 tir

Powodowało to znaczący spadek wydajności dla wykonania się algorytmu na jednym klastrze.

3.4 Wpływ poziomu równoległości na wykonanie

W ramach sesji interaktywnej ustawiane były różne wartości zmiennej konfiguracyjnej spark.default.parallelism.

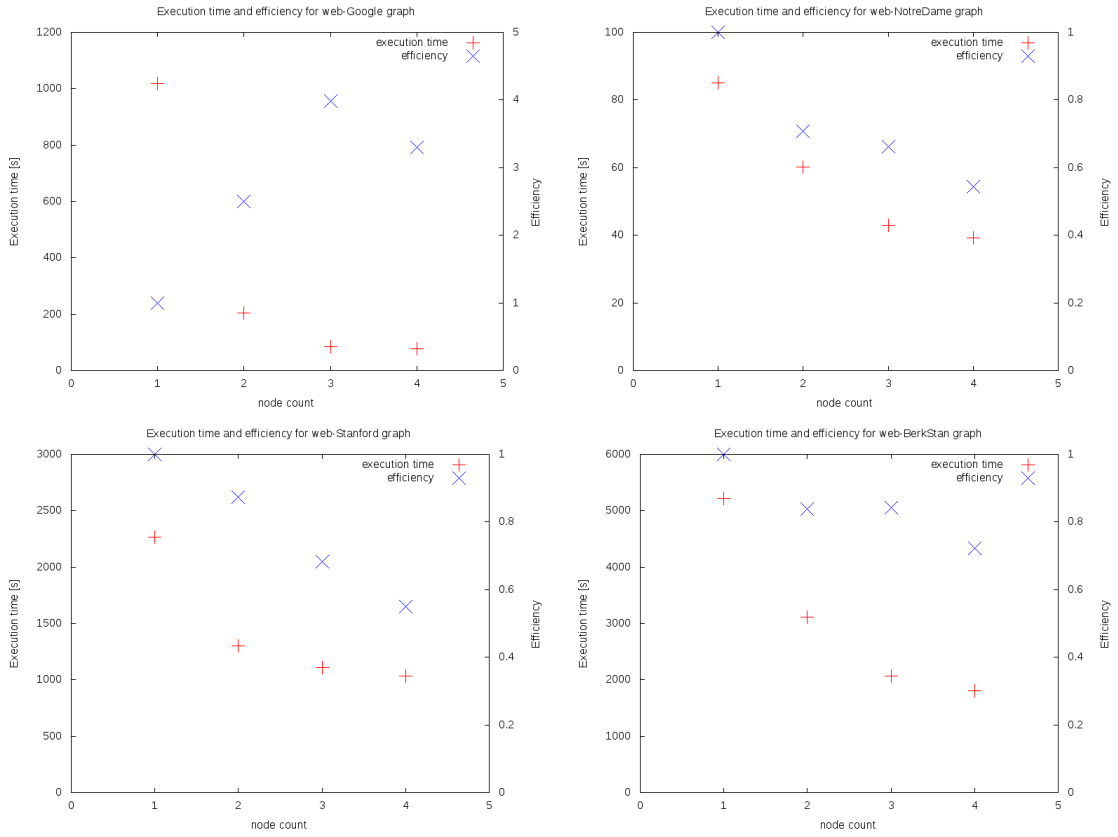
Wykonanie algorytmu dla wartości 1 potwierdziło, że wykorzystywany wtedy jest tylko jeden rdzeń z 12. Ustawienie wartości 12 pozwoliło lepiej wykorzystać możliwości węzła, zajęte wtedy były wszystkie rdzenie. Podniesienie tej wartości do 36 dodatkowo pozwoliło kilka dodatkowych procent wydajności.

Czasy wykonania algorytmu dla grafu web-NotreDame na jednym 12-rdzeniowym węźle prezentowały się następująco.

- lvl 1: 120s,
- lvl 12: 86s,
- lvl 36: 78s.

3.5 Ponowienie testów dla grafów

Testy z 3.1 zostały powtórzone dla poziomu równoległości 36. Przed każdym uruchomieniem klastra w pliku konfiguracyjnym ustawione zostały też wartości ilości pamięci dla workerów oraz drivera (na 8GB). Gdyby nie to, wartości te nie zostały by uwzględnione.



Rys. 3: Wykresy pokazują czas wykonania algorytmu poszukiwania spójnych składowych oraz ich efektywność dla testowanych grafów.

Wyniki testów przedstawione zostały na Rys. 3. Tym razem z powodzeniem udało się uruchomić algorytm również dla grafu web-BerkStan. W porównaniu do Rys. 1 prawie dwukrotnie zmniejszyły się czasy wykonania. Zależności efektywności pozostały natomiast podobne.

Literatura

[1] I. Foster: *Designing and Building Parallel Programs* http://www.mcs.anl.gov/dbpp/ dostęp 2015.10.14

[2] Reza Zadeh: *Lecture 8* http://stanford.edu/~rezab/dao/notes/lec8.pdf dostęp 2015.11.17

[3] https://snap.stanford.edu/data/index.html#web