Betweenness Centrality

Max Kießling, Wolfgang Otto, Sören Reichardt

Unitversität Leipzig

March 3, 2016

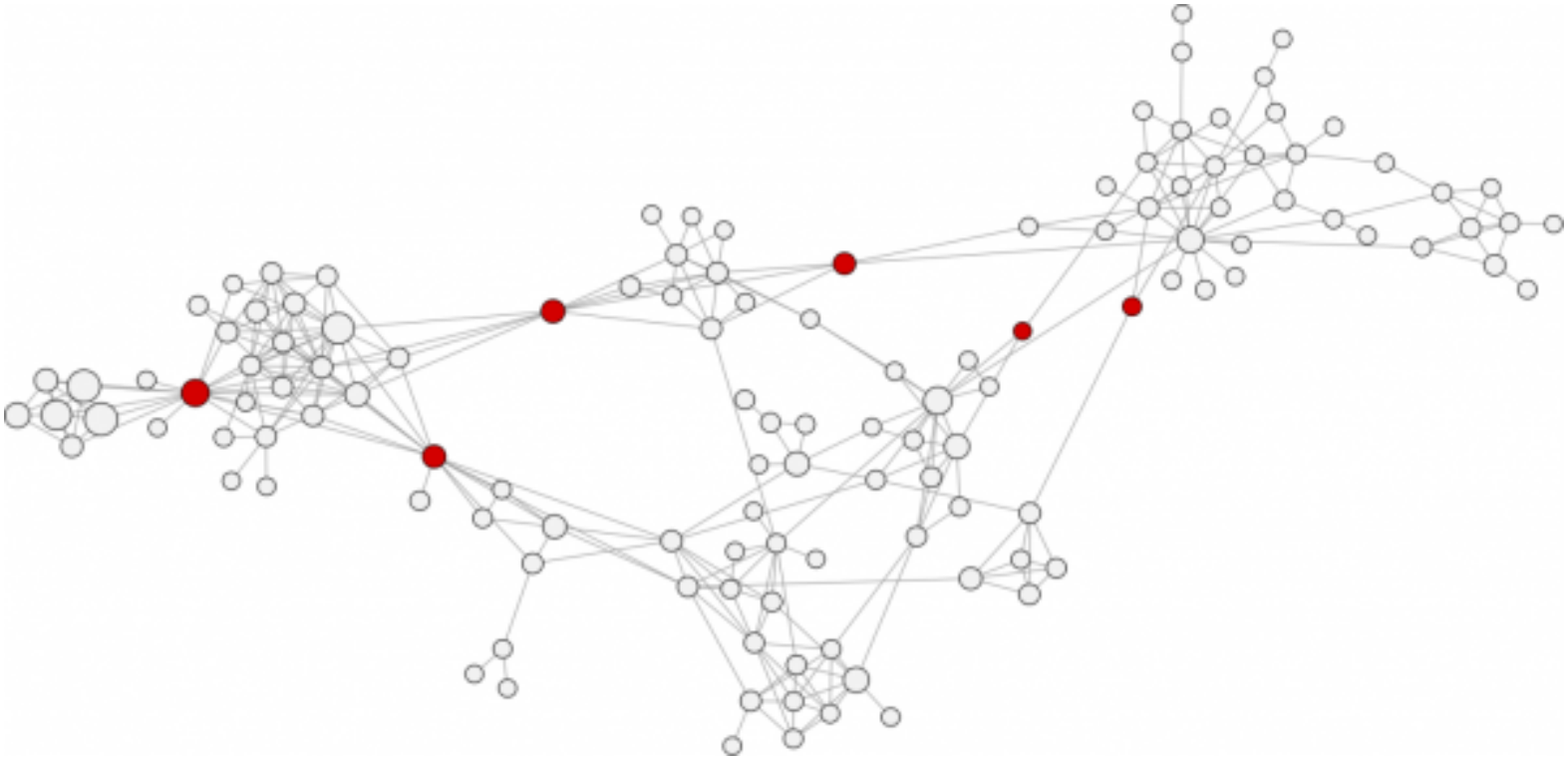
Thema

Umsetzung der Berechnung des Betweenness Centrality Maßes im Big Data Umfeld

Was ist Betweenness Centrality?

- Mass der Zentralität eines Knotens:
  - entspricht der Anzahl der kürzesten Wege zwischen allen Paaren von Knoten, die durch den betrachteten Knoten führen
- Knoten mit hohen Zentralitätswerten haben großen Einfluss auf das Netzwerk
- wird häufig zur Analyse sozialer Netzwerke verwendet

Was ist Betweenness Centrality?



Berechnung - Idee

- Zwei Schritte:
  - Berechnung aller kürzesten Pfade
  - Für jeden Knoten zählen, auf wieviel kürzesten Pfaden er liegt
- Iterativer Ansatz
  - Berechnung der kürzesten Pfade von jedem Ausgangsknoten
  - Berechnung der Betweenness Centrality relativ zu jedem dieser Ausgangsknoten
  - Aufsummieren der Werte
- Innovation des von Ulrik Brandes vorgestellten Algorithmus:
  - Zur Berechnung eines Wertes für einen Knoten ist nicht das Wissen über komplette Pfade notwendig
  - Denn: Liegt Knoten A auf dem kürzesten Pfad zu Knoten B, so liegt Knoten A auch auf allen kürzesten Pfaden auf denen B liegt
  - Idee: Aufsummieren der Werte der Nachfolger

Algorithmus: Betweenness Centrality

- Algorithmus besteht grob aus zwei Schritten:
  1. modifiziertes Single Source Shortest Paths
     - Knoten speichern zusätzlich alle direkten Vorgänger
  2. "einsammeln der Daten"
     - Ausgehend von den entferntesten Knoten:
     - verschicken von Nachrichten an alle Vorgänger
     - Bsp.: bei 2 Vorgängern verschickt ein Knoten $1/AnzahlVorgaenger$, also $1/2$ an beide Vorgängerknoten
- dieser Prozess wird für alle Knotenpaare wiederholt, sowie die entstehenden Zentralitätswerte für jeden Knoten aufsummiert

Unser Vorgehen bei der Umsetzung des Algorithums in einer Big Data Architektur

1. Ausprobieren von Flink Gelly und Spark GraphX durch implementierung von SSSP in beiden Frameworks
2. Implementierung der Berechungung der Betweenness Centrality in Spark GraphX
3. Implementierung einer Berechnungsalternative, die die Nachrichtenzahl reduziert
4. Testorientiertes Vorgehen

Implementierung

- Implementierung mittels *Spark Graphx* in Scala
- Apache Spark ist ein Open Source Framework zur verteilten Analyse großer Datenmengen
- die Hauptmerkmal von Spark sind *resilient distributed datasets (RDD)*, verteilte Mengen von Elementen und vordefinierten parallelen Operationen

GraphX

- *GraphX* ist eine API von Apache Spark für (parallele) Berechnungen auf Graphen
- Unterstützt ein Google Pregel ähnliches Programmiermodell für iterative Berechnungen auf Graphen
- Pregel ist eine *bulk-synchronous parallel messaging* Abstraktion
  - Knotenbasierte Sichtweise ("think like a vertex")
  - Ausführung erfolgt in Supersteps
  - ein Programm besteht aus 3 Funktionen:
    1. Vertex Program
    2. Send Messages
    3. Merge Messages

Ausblick

- Fehleranalyse sowie Test auf großem Cluster
- Optimierung des Speicherbedarfs / Laufzeit

Lessons Learned

- Einstieg in das Programmieren mit Scala
- Einstieg in zwei Graph Processing Frameworks im Big Data Umfeld
- Tieferes Verständnis für die Adaption von Algorithmen in praktische Umgebungen
- Bewältigung von Problemen die das Spark Datenflussmodell mit sich bringt

References

Ulrik Brandes(2001)

A Faster Algorithm for Betweenness Centrality

URL http://algo.uni-konstanz.de/publications/b-fabc-01.pdf