MIS Quarterly

BIG DATA & ANALYTICS IN NETWORKED BUSINESS

LARGE-SCALE NETWORK ANALYSIS FOR ONLINE SOCIAL BRAND ADVERTISING

Kunpeng Zhang
Department of DOIT, Robert H. Smith School of Business, University of Maryland,
College Park, MD 27042 U.S.A. {kzhang@rhsmith.umd.edu}

Siddhartha Bhattacharyya
Department of IDS, College of Business Administration, University of Illinois, Chicago,
Chicago, IL 60607 U.S.A. {sidb@uic.edu}

Sudha Ram
Department of MIS, Eller College of Management, University of Arizona,
Tucson, AZ 85721 U.S.A. {ram@eller.arizona.edu}

Appendix A

Network Generation

Algorithm 1: Chaining Two MapReduce Jobs to the Brand–Brand Network

Input: A text file contains lines of <$brand_{id}$, $user_{id}$, # of activities>
Output: A text file contains lines of <$brand_i$, $brand_j$, # of common users>

1: /* The first job */
2: input: <$brand_{id}$, $user_{id}$> // Each line in the text file

3: **function** MAPPER
4: output < $user_{id}$, $brand_{id}$>
5: **end function**

6: **function** REDUCER
7: **for all** $v \in values$ **do**
8: add $v \rightarrow list$
9: **end for**
10: **for all** <b, b_j>, b_i, $b_j \in list$ **do**
11: add $v \rightarrow list$
12: **end for**
13: **output** <k_2, v_2>
14: **end function**

15: /* The second job */

```
16:  function IDENTITY MAPPER
17:  end function
18:  function REDUCER
19:        for all v ∈ values do
20:              sum += v
21:        end for
22:   output <key, sum>
23:  end function
```

Appendix B

Hierarchical Community Detection

Algorithm 2: Hierarchical Community Detection

```
1:  C* ← {∅}
2:  function DIVIDE(B_n, s)    // s is the threshold and B_n is the network
3:     C: {C_1, C_2, …, C_k} ← Modularity-Based Detection (B_n)
4:     for all C_i ∈ C do     // this can be processed in parallel
5:           if | C_i | ≥ s then
6:                 C ← DIVIDE(C_i, s)
7:           else
8:                 C* ← C* ∪ C_i
9:           end if
10:    end for
11:    return C*
```

Appendix C

Brand Ranking

Algorithm 3: Distributed bRank: Mapper and Reducer Functions to Rank Brands

1: /* The job for Mapper is to invert the input */

2: **function** MAPPER
3: **for all** $brand_j \in (brand_1, brand_2, \ldots, brand_k)$ **do**
4: **output** $brand_j \leftarrow <brand_i, rank_i * \frac{w_{ij}}{\sum w_i}>$ //w_i is weights of all out-links from i
5: **end for**
6: **output** $brand_i \rightarrow brand_1, brand_2, \ldots, brand_k$
7: **end function**

8: /* The job for Reducer is to update the ranking using the in-links */

9: **function** REDUCER
10: Input is in a format of (∗). The key: $brand_k$
11: **for all** in-link $brand_i \in (brand_1, brand_2, \ldots, brand_n)$ **do**
12: $rank_k += rank_k * \frac{w_{ij}}{\sum w_i} \beta$ // is weights of all out-links from i
13: **end for**
14: $rank_k = (1 - \beta + rank_k) * C_n(k)$
15: **output** $<brand_k, rank_k> \rightarrow <brand_1, brand_2, \ldots, brand_n>$
// $brand_1, brand_2, \ldots, brand_n$ are out-links of $brand_k$
16: **end function**

After map function, we have temporary files in the following structure (∗):
$brand_k \rightarrow <brand_1, rank_1>$,
$<brand_2, rank_2>$,
…,
$<brand_n, rankn2>$,
$<brand_{k1}, brand_{k2}, \ldots, brand_{kn}>$

Where $brand_1, brand_2, \ldots, brand_n$ are in-links of $brand_n$ and $brand_{k1}, brand_{k2}, \ldots, brand_{kn}$ are out-links.