ISSN 2350-1022

International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 1, Issue 2, pp: (73-78), Month: October 2014 – March 2015, Available at: www.paperpublications.org

Implementing Map Reduce Based Edmonds-Karp Algorithm to Determine Maximum Flow in Large Network Graph

[1]Dhananjaya Kumar K, [2]Mr. Manjunatha A.S

[2]Senior Assistant Professor
[1,2]Dept. Of Computer Science & Engg., Mangalore Intitute of technology & Engineering, Mangalore, Karnataka, India

Abstract: **Maximum-flow problem are used to find Google spam sites, discover Face book communities, etc., on graphs from the Internet. Such graphs are now so large that they have outgrown conventional memory-resident algorithms. In this paper, we show how to effectively parallelize a maximum flow problem based on the Edmonds-Karp Algorithm (EKA) method on a cluster using the MapReduce framework. Our algorithm exploits the property that such graphs are small-world networks with low diameter and employs optimizations to improve the effectiveness of MapReduce and increase parallelism. We are able to compute maximum flow on a subset of the a large network graph with approximately more number of vertices and more number of edges using a cluster of 4 or 5 machines in reasonable time.**

Keywords: **Algorithm, MapReduce, Hadoop.**

I. INTRODUCTION

The classical maximum flow problem sometimes occurs in settings in which the arc capacities are not fixed but are functions of a single parameter, and the goal is to find the value of the parameter such that the corresponding maximum flow or minimum cut satisfies some side condition. Finding the desired parameter value requires solving a sequence of related maximum flow problems. In this paper it is shown that the recent maximum flow algorithm of Edmonds-Karp can be extended to solve an important class of such parametric maximum flow problems, at the cost of only a constant factor in its best case time.

Hadoop, open source software and the most prevalent implementation of this framework, has been used extensively by many companies on a very large scale. A distributed data processing process the large data in a large cluster and commodity hardware and in make use the programming model and function in the MapReduce model. Many of the data being generated at a fast rate take the form of massive graphs containing millions of nodes and billions of edges. One graph application, which was one of the original motivations for the MapReduce framework, is page-rank the computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce library expresses the computation as two functions: Map and Reduce.

Map, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key I and passes them to the Reduce function.

The Reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per

ISSN 2350-1022

International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 1, Issue 2, pp: (73-78), Month: October 2014 – March 2015, Available at: www.paperpublications.org

Reduce invocation. The intermediate values are supplied to the user's reduce function an iterater. This allows us to handle lists of values that are too large to fit in memory.

The execution time of a MapReduce job depends on the computation times of the map and reduces tasks, the disk I/O time, and the communication time for shuffling intermediate data between the Mapper and reducers. The communication time dominates the computation time and hence, decreasing it will greatly improve the efficiency of a MapReduce job. Previous work required the whole graph to be shuffled to and sorted by the reducers, leading to the inefficient graph analysis. This problem becomes even worse given that the most of these algorithms are iterative in nature, where the computation each iteration depends on the results of the previous iteration.

The MR framework manages the nodes in a cluster. In Hadoop, one node is designated as the master node and the rest are the slave nodes. An MR Job consists of the input records and the user's specified MAP and REDUCE function.

II. LITERATURE SURVEY

Maximum flow algorithm is used to find the spam site, build content voting system, discover communities, etc, on graphs from the internet. Such graphs are showing how to effectively parallelize a max-flow algorithm based on the Ford-Fulkerson method on a cluster using the MapReduce framework. This algorithm increases the MapReduce optimization and also improves the effectiveness of MapReduce and increase parallel run time access [1].

The MapReduce framework has become the de-facto framework for large scale data analysis and data mining. One important area of data analysis is graph analysis. Many graph of interest, such as the web graph and social networks, are very large in size with millions of vertices and billions of edges. These results are correlated with the local graph partition using a merge-join and new improved analysis result associated with only the nodes in the graph partition are generated and dumped to the DFS [2].

An efficient implementation of the push-relabel method for the maximum flow problem the resulting codes are faster than the previous codes, and much faster on same problem families. The speedup is due to the combination of heuristics used in this implementation; it show that the highest-level selection strategy gives better results when combined with both global and gap relabeling heuristics [3].

All previously known efficient maximum-flow algorithms work by finding augmenting paths, either one path at a time (as in the original Ford and Fulkerson algorithm) or all shortest-length augmenting paths at once (using the layered network approach of Dinic). An alternative method based on the preflow concept of Korsakov is introduced. A preflow is like a flow, except that the total amount flowing into a vertex is allowed to exceed the total amount flowing out. The method maintains a preflow in the original network and pushes local flow excess toward the sink along what are estimated to be shortest paths. The algorithm and its analysis are simple and intuitive, yet the algorithm runs as fast as any other known method on dense graphs, achieving an O (n3) time bound on an n-vertex graph [4].

The paper states the maximum flow problem gives the Ford-Fulkerson labeling method for it solution, and points out that an improper choice of flow augmenting paths can lead to severe computational difficulties. Then rules of choice that avoid these difficulties are given. We show that, if each flow augmentation is made along an augmenting path having a minimum number of arcs, then a maximum flow in an n-node network will be obtained after no more than $\frac{1}{2}(n^2-n)$ augmentations new algorithm is given for the minimum-cost flow problem, in which all shortest-path computations are performed on networks with all weights nonnegative. In particular, this algorithm solves the n * n assignment problem in $O(n^3)$ [8].

III. PROBLEM DEFENITION

The problem of finding a Maximum flow in a directed graph with Edge capacities arise in many setting in operation research and other fields, and efficient algorithms for the problem as received a great deal of attention, Extension. Problems which require processing large graphs have become popular recently due to the rapid growth of online communities and social networks. In the MR framework, some large graph algorithms have been developed such as s-t graph connectivity.

ISSN 2350-1022

International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 1, Issue 2, pp: (73-78), Month: October 2014 – March 2015, Available at: www.paperpublications.org

IV. DEFINATION OF MAXIMUM FLOW PROBLEM

The Maximum Flow is a flow network G = (V, E) is a directed graph where each edge (u, v) Є E has a non-negative capacity C(u, v) ≥ 0. There are two special vertices in a flow network: the source vertex s and the sink vertex t. Without loss of generality, we can assume there is only one source and sink vertex, which we call s and t respectively. A flow is a function F: V * V → R satisfying the following three constraints: (a) capacity constraint: F(u, v) ≤ C(u; v) for all u, v belongs V , (b) skew symmetry: F(u; v) = -F(v; u) for all u; v Є V , and (c) flow conservation: ∑ F(u; v) = 0 for u Є V –{s, t} and v Є V . The flow value of the network is P F(s, v) for all v Є V. In the max-flow problem, we want to find a flow F* such that |F*| has maximum value over all such flows. Two important concepts used in flow networks are residual network (or graph) and augmenting path. For a given flow network G = (V, E) with a flow f associated to it, the residual network G_f = (V, E_f) is the set of edges E_f that have positive residual capacity cf. that is, E_f = {(u; v) Є E: Cf (u; v) = C (u; v) - F (u; v) > 0}. An augmenting path is a simple path from s to t in the residual network.

The Edmonds-Karp method is a well known algorithm schema to solve the max-flow problem. The idea is to repeatedly find shortest augmenting paths in the current residual network until no augmenting paths can be found.

1. While true do
2. P = find an shortest augmenting path in Gf
3. If (P does not exist) break
4. Augment the flow f along the shortest path P

Procedure1: Edmonds-Karp Algorithm.

The above algorithm defined by the Maximum flow in a flow network using the method of Edmonds-Karp Algorithm (EKA), in first round of the algorithm if all vertices and edges including capacities are true, now finding the Residual graph Gf in the Flow network graph. In the residual graph find the augmenting path P which is the shortest capacity to flow through the source to sink in the second step. If once does finding the path which can flow through the source to sink in the residual capacity which is minimum in to the shortest path. Finally all path can be exist we calculate the Maximum flow in the flow network graph.

V. METHODOLOGY

We start the flow from the main program EKA method in figure 2, initially round will be zero while the network graph is true now we create the Job in MapReduce and set the number of path including vertices, edges and Capacities to the flow network graph. Now assign the job to the master node up to complete job set when master assign the job the slave node, each node work in equally and processing the job up to completion. From the network graph contain only one single source and sink to assign the values to find the maximum flow in a network graph. As well iteration goes up to completion of job work.

1. Round = 0
2. While true do
3. Job = new Job () // create a new MapReduce job
4. Set the job's MAP and REDUCE class, input

   And output path, the number of reducers, etc.
5. Job.waitForCompletion () // submit the job and wait
6. c = job.getCounters () // event counters
7. Sm = c.getValue ("source_move");
8. Si = c.getValue ("sink_move");
9. If (Round > 0 ∧ (Sm = 0 ∨ Si = 0)) break

ISSN 2350-1022

International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 1, Issue 2, pp: (73-78), Month: October 2014 – March 2015, Available at: www.paperpublications.org

10. Round = Round + 1

Procedure2. The pseudo code of the main program of EKA

The Map for EKA is given in figure 3; its job is updating all edges in the current residual graph for the main flow network graph. First we need to update the residual graph which is finding the shortest augmenting path in the shortest edge capacity and update to the edge flow in the residual graph. Now filter the local job send to the accumulator and for each source and sink will do as accept the short path in the residual graph, and emit to the key/value pair set and concatenated to the source and sink values. If source is a excess path if it exists edge become low capacity and pick one way source path to sink and added to the forward and backward capacity, if exist added to the backward edge and if does not exist flow through the sink. In map function always emit the intermediate key/value pair set.

Function MAP of EKA (u, s, t, Eu)

1. for each (e Є s, t, Eu) do // update all edges

2. a = ShortAugmentedEdges[round-1].get(eid)

3. If (a exists) ef = ef + af // update edge flow

4. Remove saturated excess paths in s and t

5. A = new Accumulator () // local filter

6. for each (Se Є s, Te Є t) do

7. If (A. accept (Se ⏐ Te)) // Se | Te is an shortest augmenting path

8. EMIT-INTERMEDIATE (t, <Se ⏐ Te>)

9. If (S ≠ null) // extend source excess path if it exists

10. For each (e Є Eu, ef < ec) do

11. Se = pick one source excess path from Su

12. EMIT-INTERMEDIATE (ev, <Se | e>)

13. If (t ≠ null) // extend sink excess path if it exists

14. For each (e Є Eu, -ef < ec) do

15. Te = pick one sink excess path from t

16. EMIT-INTERMEDIATE (ev, < e | Te>)

17 EMIT-INTERMEDIATE (u, (s, u, Eu))

Figure3. The MAP function in the EKA algorithm

The Reduce for the EKA in figure 4; when assign the map record accumulator collect all the record in reducer and accumulate the path, source and sink vertices it will be a null values (< > empty null set). For each source sink and capacity belongs to map values if edge vertices become empty in the sense null, if all source become sink and again merge and filter the content of map records or job we will filter it. If accept all the shortest augment path then should be doing the further execution step otherwise return the condition. If source and sink is increment collect all the augmenting edges in residual graph and finding the shortest path and also finally calculate the maximum floe in a flow network get back to result in to the master node.

Function REDUCE of EKA (u, values)

1. Ap, As, At = new Accumulator ()

2. Sm = Tm = Su = Tu = Eu = < >

3. For each (Sv, Tv, Ev) Є values) do

4. If (Ev ≠ < >) Sm = Sv, Tm = Tv, Eu = Ev

5. For each (se Є Sv) do // merge / filter Sv

6. If (u = t) Ap. Accept (se) // se = Shortest augmenting path

7. Else if (|Su| < k ^ As. accept (se)) Su = Su U se

8. For each (te Є Tv) do // merge / filter Tv

9. If (|Tu| < k ^ At. accept (te)) Tu = Tu U te

10. If (|Sm| = 0 ^ |Su| > 0) INCR ('source_move')

11. If (|Tm| = 0 ^ |Tu| > 0) INCR ('sink_move')

12. If (u = t) // collect all augmented edges in Ap

13. For each (e 2 Ap) do

14. ShortestAugmentedEdges [round].put (eid, ef)

15. EMIT (u, (s, u, Eu))

Figure4. The REDUCE function in the EKA algorithm

VI. RESULTS AND DISCUSSION

The Map and Reduce correlation between maximum flow value with run time and number of rounds: the experiment test the effect of iteration to increase the maximum flow value in order to run time and increase the number of rounds using the large graph.

MapReduce optimization effectiveness and more Complexity: the experiment goes on to show the effectiveness of the MapReduce job work to increase the accumulator optimization and run time logarithmic scale and more number of round execution.

The Reduction and Scalability of EKA with effective graph size: Basically the experiment goes on increase the graph size and also increases the more number of machines. Each successive algorithm reduces the byte of data and shuffled with the map and reduces job work.

VI. CONCLUSION

The implementation of the Edmonds-Karp algorithm that works when the capacities are integral, and has a much better running time than the Ford-Fulkerson method Edmonds-Karp algorithm is to achieve faster maximum flow problem than the other methods. Edmonds-Karp algorithm achieves more effective and complexity run time in to the beat case, and average cases.

REFERENCES

[1] F. Halim, R H.C. yap, Yougzheng Wu, "A MapReduce Based Maximum flow Algorithm for large small world network graph" National University of Singapore.

[2] U. Gupta, L. Fegars, "Map Based Graph Analysis on MapReduce" University of Texas at Arlington, 2013 IEEE.

[3] B. V. Cherkassy and A. V. Goldberg, "On Implementing the Push–Relabel Method for the Maximum Flow Problem".

[4] A.V. Goldberg, "A new approach to the maximum-flow problem".

[5] en.wikipedia.org/wiki/Ford–Fulkerson-algorithm

[6] http://hadoop.apache.org.

[7] J.Lin and M.Schatz. "Design patterns for efficient graph algorithms in MapReduce", Mining and Learning with Graphs Workshop, 2010.

[8] J.Edmonds and R.M.Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems", J. Assoc. Mach., 1972.

Author's Profile:

Dhananjaya Kumar K completed the bachelor's degree in Computer Science & Engineering from Shirdi Sai Engineering College at Bangalore and presently pursuing Master Technology in Computer Science & Engineering at Mangalore Institute of Technology, Mangalore.

Manjunatha A. S. completed bachelors and masters degree in Computer Science and Engineering. Currently he is working Senior Assistant Professor in Mangalore Institute of Technology and Engineering, Mangalore.