HPdgraph – Distributed algorithms for graph analytics

User Guide

Contents

User Guide

1. Introduction

The HPdgraph package provides a few distributed algorithms for graph analytics. It is written based on the infrastructure created in HP-Labs for distributed computing in R. The main functions of this package are:

- hpdpagerank: It is a distributed function for computing pagerank vector of a graph.
- hpdwhich.max: It finds and returns the index of the maximum value stored in a darray.

2. hpdpagerank

hpdpagerank function is a distributed implementation of pagerank algorithm for directed graphs. Power Method is used to calculate the pagerank vector of a graph. Self-loops are allowed, but multiple edges are not considered. The input graph should be in the adjacency matrix format and stored in a darray. Moreover, it must be partitioned column-wise.

More detail about the interface of the function and its input arguments can be found in the manual of the package.

Example

As an example, let's use the real-world graph of Live-Journal social network which can be downloaded from http://snap.stanford.edu/data/soc-LiveJournal1.html. This graph has 4,847,571 vertices and 68,993,773 edges. The downloaded file should be uncompressed to its text file (using gunzip) before it can be used in our example. Let's assume that the text file is stored in this path "/graph/soc-LiveJournal1.txt" and we want to run hpdpagerank on two node with 12 cores each. Therefore, in this example we need to split the input graph to 24 parts. The procedure is displayed in Figure 1. Please note that the file "cluster2.xml" is used to configure the cluster specification. More information about this configuration file can be found in the documents of DistributedR. Also, more information about "splitGraphFile" and "file2graph" functions can be found in the manual of HPdata package.

```
> library(HPdata)

> distributedR_start(cluster="cluster2.xml")

> ret <- splitGraphFile(inputFile="/graph/soc-LiveJournal1.txt", outputPath="/graph/temp",
nSplits=24)
Sending files to node1
soc-LiveJournal1.txt0
soc-LiveJournal1.txt1
…
Sending files to node2
soc-LiveJournal1.txt0
soc-LiveJournal1.txt1
…

> dg <- file2graph(ret$pathPrefix, ret$nVertices, ret$verticesInSplit, ret$isWeighted)
Openning files:
/graph/temp/soc-LiveJournal1.txt0
... until ...
/graph/temp/soc-LiveJournal1.txt23
Progress: 100%

> dim(dg$X) # dimensions of the adjacency matrix
[1] 4847571 4847571

> sum(dg$X) # number of edges
[1] 68993773

> library(HPdgraph)

> pr <- hpdpagerank(dg$X)

> dim(pr)     # the dimensions of the pagerank vector
[1] 1 4847571

> hpdwhich.max(pr)  # the vertex with highest pagerank value
[1] 214407
```

Figure 1. An example for running hpdpagerank for Live-Journal social network

3. hpdwhich.max

"hpdwhich.max" function is a distributed version of "which.max" function for a 1D-array which has darray as its input argument. As it is shown in Figure 1, it is a useful function for finding the vertex with highest ranke in the pagerank vector returned by hpdpagerank function.

4. Performance evaluation

The following experiments illustrate the performance of hpdpagerank on a few graphs. The tests are performed on two graphs: the graph of Twitter with about 51 million vertices and a synthesized graph with 1 billion vertices.

Specification of each node for experiments:

- 2 CPU per Node
- Core(s) per socket: 6
- CPU model: Intel® Xeon® X5650 (12MB Cache, 2.67GHz)
- 96GB RAM
- 10Gb/s Ethernet Network
- CentOS release 6.4 (Final)
- R version 3.0.1

Experiment 1 – Twitter Graph

Twitter graph is used for this experiment. This graph has 51,217,936 vertices and 1,963,083,442 edges. It should be noticed that the degree distribution of the vertices follows a *power law*; i.e., it is highly skewed. Therefore, the workload of the R-executors is very imbalanced, as the adjacency matrix of the graph is split column-wise to partitions with respect to the number of vertices. The results are represented in Table 1 and illustrated in Figure 2 and Figure 3. The total running time also includes the time for loading the graph from a Vertica database to a darray. The algorithm for calculating pagerank vector converged in 3 iterations. The size of the adjacency matrix of this graph is about 15GB. The file of the graph, which is in edge list format, takes about 28GB of disk space.

Table 1. Experiment of hpdpagerank with Twitter Graph

#Workers	#Instances/worker	Average iteration time (sec)	Total running time (sec)
1	1	82.7	2436.2
1	2	57.3	1621.1
1	4	47.1	1586.8
1	8	38.0	1553.8
1	12	37.5	1557.8

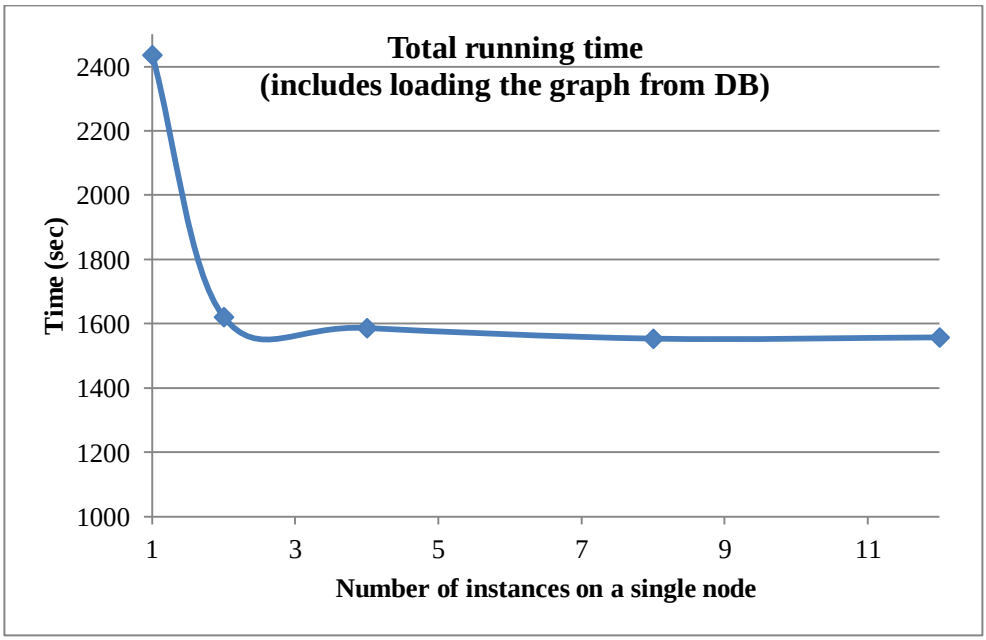

Figure 2. Total running time for calculating pagerank vector of Twitter graph

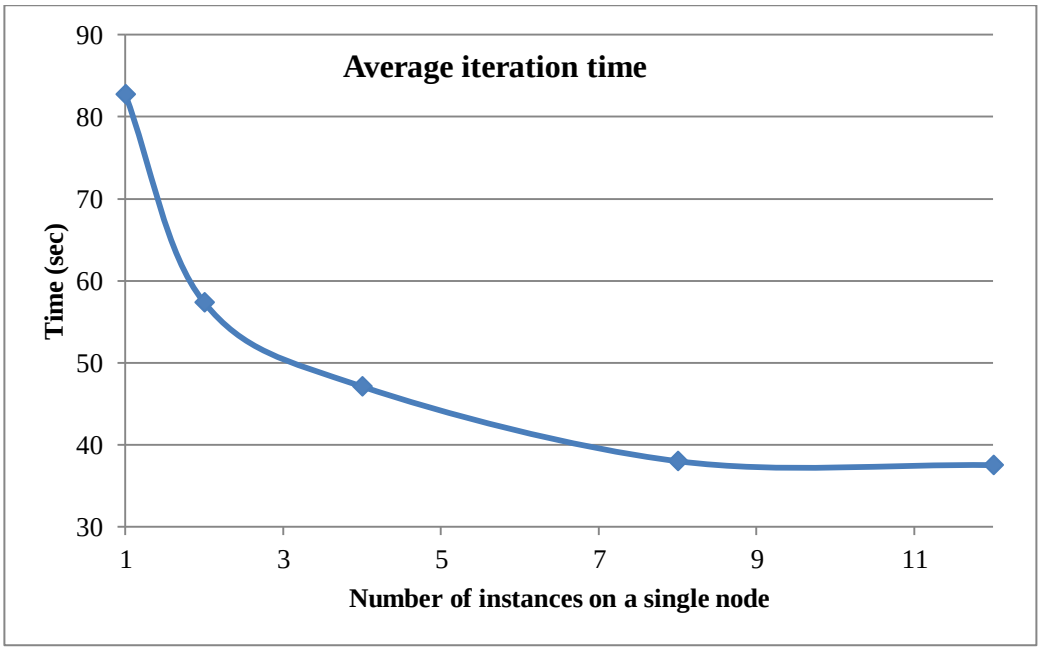

Figure 3. Average iteration time in hpdpagerank for calculating pagerank vector of Twitter graph

Experiment 2 – Synthesized Graph with 1 billion vertices

A synthesized with 1 billion vertices and about 10 billion edges is randomly generated for this experiments. The degree distribution in this graph follows a uniform distribution; therefore the workload is expected to be balanced. The results are represented in Table 1, and also displayed in Figure 4Figure 2 and Figure 5. The total running time includes the time for loading the graph from a Vertica database to a darray. The algorithm for calculating pagerank vector converged in 2 iterations. It should be noticed that this graph cannot be loaded to the memory of a single node because of its large size. The size of the adjacency matrix of this graph is about 80GB. The size of the file which contains the graph in edge list format is about 180GB.

Table 2. Experiment of hpdpagerank with a synthesized graph of 1 billion vertices

#Workers	#Instances/worker	Average iteration time (sec)	Total running time (sec)
6	1	238.4	2452.1
6	2	147.2	1494.2
6	4	96.2	1273.3
6	8	73.3	1275.3
6	12	76.3	1389.8

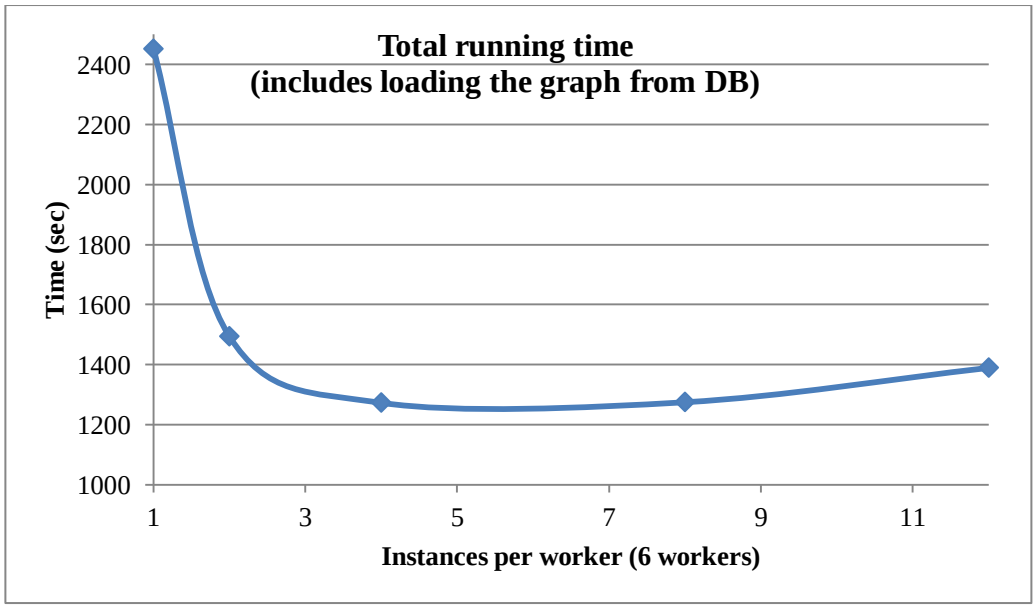

Figure 4. Total running time for calculating pagerank vector of the synthesized graph

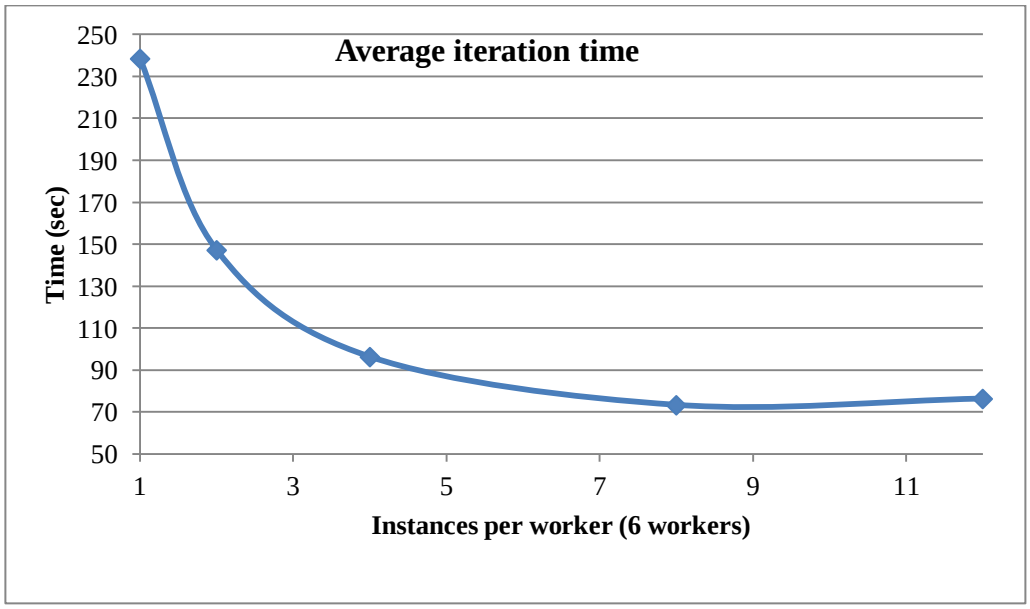

Figure 5. Average iteration time in hpdpagerank for calculating pagerank vector of the synthesized graph