GHS Algo Implementation

Sahil Jain - 2016BSY7510,Vamsi - 2015MCS2358

February 2017

GHS Implementation Design

1 Introduction to Algorithm

The GHS algorithm of Gallager, Humblet and Spira is one of the best-known algorithms in distributed computing theory. This algorithm can construct the MST (Minimum Spanning Tree) in asynchronous Message-passing model.It assumes that there exists an undirected connected graph with all unique edge weights.
A processor exists at each node that only knows about its neighbours and the weights of the edges with neighbour nodes.All the nodes follow the same algorithm and exchange messages only with immediate neighbours until the tree is completed.

2 Code Implementation

2.1 File Structure

Code is implemented using 5 sets of java files : Main.java, Node.java, Neighbour.java, Message.java, MessageQueue.java. Each node has its own thread and follows the rules of GHS Algorithm.

2.1.1 Main.java

– It loads the graph from the input text file and creates an adjacency matrix of vertices's and edges.Also, it computes the maximum no. of nodes in the graph.
– Using the adjacency matrix it computes the neighbours for each Node(Vertex) and sort them in the order of min to max edge weights.
– Nodes are then created and given the node IDs and their neighbours.
– Once all the nodes are created their threads starts running. – Initially just the first node calls the wake up. Node0 is decided as the first node that wakes up.

2.1.2 Node.java

–Each node has the required parameters and implements a runnable thread.
– Every node thread runs in the infinite loop looking for new messages and processes them .
– Each node is initialised with its NodeID, set of neighbours (in sorted form) and a SLEEP state.
– Every node except the first node remains in the sleep state and do not send any message until they receive a first message. As soon as a node receives a message, it wakes up , initialises itself and sends a connect message to the neighbour with least edge weight.
– Once the node wakes up it is in receive and process mode till the algorithm terminates.

Node Thread Implementation –Thread starts in an infinite loop , trying to read messages from its queue. Reading is non-blocking. – Once a message is read, it is processed using various functions(processConnect, processInitiate, processReject, processTest, processReport, processAccept....) according to message type.
–Handling of Wait Case in case of Connect and Test Message Processing :
. Thread is made to wait for some time using a for loop (introducing delay) in message processing. . The wait message(message to be processed later) is then written back to the own queue. . This introduces the possibility of getting another message in the meantime and the level of node getting changed to be able to process the message later.

2.1.3 Edge.java

Edge represents the edge of a node.
– It has 3 fields :
1. ID1 /* ID of the first node */
2. ID2 /* ID of the second node*/
3. weight /* weight of the edge */

2.1.4 Message.java

– It represents the structure of the message passed between different nodes.
– Structure message is common for all types of messages.
– To identify the relevant fields to be processed , there are message types , each to uniquely identify the kind of message.

Message Structure
1.protected String mType; /*Message type to identify message*/
2.private String message; /* Actual message containing the data */
3.protected Node src; /* Sender nodeID */

Message Functions
1.setMessage /* To create the message using input string */

2.getMessage /* To read the message from the structure */

2.1.5 MessageQueue.java

– Message Queue for each node is kept public so that any nod can write a message to any other node.
– To allow for thread safety message queue is implemented as vector queue allowing for queue access synchronisation.
– Message Queue is implemented as non-blocking queue so that a node doesn't hang up waiting for messages.

MessageQueue Functions 1.addMessage
2.viewMessage
3.getMessage
4.isQueueFull
5.queueSize

2.2 Message Passing

– Message is identified using 3 fields : senderID, destID and msgType
– A node can directly write into the message queue of other node using the nodeID
– Since nodes are created as structure of arrays, every node can be accessed using nodeID as the index.
– Once the node is accessed, its queue can also be accessed and written into.

2.3 Code Termination

– A global stop variable is defined. Every thread runs till the value of the stop is FALSE.
–Once the MST tree is complete and created, the stop variable is set to TRUE and main thread and all threads goes out of infinite loop.
–Once the Main thread stops, the MST edges are written onto the output text file as per the specified format.
– All threads are then terminated and resources are freed.