FACULTY OF ENGINEERING

Incremental calculation of shortest path in dynamic graphs.

Distributed Systems

Presented by

1- **Khaled A. Elaish (19)**
2- **Sherif Mohi Mahmoud Hafez (26)**
3- **Omar Elfarouk A. Farahat (40)**

3/26/2016

Incremental calculation of shortest paths in dynamic graphs is an important topic in graph theory. Our approach is using a threaded BFS in 3 different ways. We try to avoid any unnecessary recalculations, and if it's inevitable, we minimize the needed recalculations. Most of our results were consistent but there were some things that we couldn't justify. But finally, the results were consistent with our first assumption that a threaded implementation of the problem is better than a sequential one.

Table of Contents

1 Problem Definition

Incremental calculation of shortest paths in dynamic graphs is an interesting topic in graph theory which is still open to research. It is used in many applications like GPS navigation, routing schemes in computer networks, search engines, and social networks.

The shortest path problem is defined as getting the least cost path from a node to another. This first part of the problem is constructing the graph, then answering queries in batches, whether adding, deleting, or getting the cost of a path from a certain node to another, and as stated our program is expected to answer all of those queries correctly as if they were in the same order of the batch. The graph in our problem is directed but all edges are constant unit weights.

We are required to divide the computational load of all-pair shortest paths in a way to solve the problem efficiently.

2 Algorithms

The common approach that we took was to divide the all-pair shortest computations in threaded fashion on a multi-core machine.

We used the classic breadth-first search algorithm, with some modifications which will be stated.

We made three programs to test three separate performances to the same problem.

The first approach was a naive sequential one, the program calls the BFS methods in a sequential way for each node and calculates the shortest paths from this source to all other destinations. There's nothing distributed about this approach but it was used in the performance evaluation.

The second approach was threaded one, the program asks for the number of available cores -dynamically for scalability- and opens multiple concurrent threads. Each thread computes the shortest path from the source that called it all other destinations.

The third and last approach was a dynamic programming threaded one, it does all the features of the second approach, but it takes the costs of the already computed nodes and uses it so that it doesn't need to traverse that path or anything further than that point.

The graph is represented as a map of maps, we didn't use a array list of maps, because there might be gaps in the number of nodes.

4 Implementation

Our three programs were implemented in Java, and were tested on a 8-core single machine.

We tested our three implementations with three test cases that our TA provided, with multiple runs -5- for each test case to avoid any anomalies.

After we finish constructing, and before we start taking in the batch of queries, we run the BFS algorithm.

Then we start accepting batches and applying add and delete operations to them, and then answer queries.

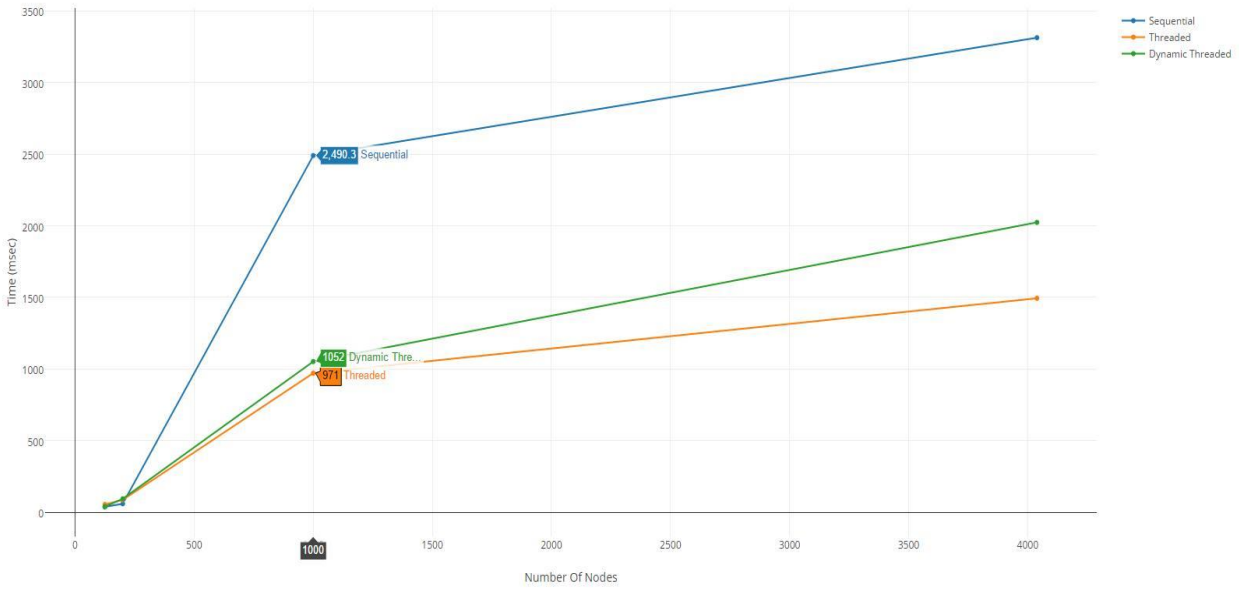
We made two optimizations regarding queries; The first one, between any two queries, we process the additions and deletions and scratch out any two queries that contradict, for example if the batch was A 1 4, then D 1 4, then Q X X, we don't do any recomputations because this means that our graph didn't change.

The second optimization, is that for any edge we store the list of nodes that uses this edge in one of its shortest paths, then if any deletions occur in a batch file we don't recompute the shortest paths for all nodes, but only for the nodes that were affected by this node.

The three test cases had 125, 200, and 1000 nodes respectively.

5 Results

Number of Nodes/Implementation	Sequential	Threaded	Dynamic Threaded
125	37.6	54.25	38.75
200	58.75	86	93.5
1000	2490.3	971	1052
4039	3313.3	1493.6667	2023.75



The sequential approach yielded better results than the normal threaded approach in a graph with small numbers, which was illogical, but can be explained by the argument that with a small graph, dividing the load is an overhead. However for larger graphs, the threaded and the dynamic threaded approach were faster.

6 Conclusion

This assignment is a prototype to a distributed approach to the Incremental calculation of the shortest path on dynamic graphs, using some heuristics to minimize unnecessary recomputations so that queries can be answered in the fastest time possible.

Our approach is to divide the computation of the BFS algorithm of the nodes as sources in the graph to enhance the overall performance; The problem can be divided into independent tasks, and the division must be in a good way to exploit this parallelism to our advantage.

An approach to optimize the efficiency of the dynamic threading approach, is to choose the nodes that ran their BFS algorithm first in a smart way, maybe using topological sort and then starting our threads with the nodes on the middle, level.

Another approach was graph partitioning in which the main topology is to be divided in crowded clusters, and these clusters are separated with a very small number of edges. All-pair shortest paths costs are stored for each cluster, but costs between any two nodes in different clusters are calculated on-Demand.