International Journal of Emerging Trend in Engineering and Basic Sciences (IJEEBS)
ISSN (Online) 2349-6967 *Volume 2 , Issue 1(Jan-Feb 2015), PP1-5*

Finding Nearest Neighbors For Route Sharing On Android Platform

Rupal Newaskar[1], Utkarsh Bagade[2], Karishma Ramachandran[3], Abhay Ahire[4]

(Computer department, Rajarshi Shahu college of engg /Pune university ,India)

Abstract: In the modern age of technology, smart phones are an integral part of everyone's life.There is an exponential growth in the number of mobile computing applications centered on our daily life. In such applications, location dependent systems have been detected as an important application which accepts join results as long as they are near. For a given object, missing some or even all of the objects that are nearest to it, is often tolerable if the objects being retuned are almost as near to it as those that are nearest. Such an application which uses the architecture and implementation of a location depended system is commonly known as Travel Scout. With the help Near Neighbor Join concept we are implementing Travel Scout, which gathers information of users based on their inputted travelling preferences and making a guide system for travelling by vehicle sharing
Keywords: Android, API, mashup, mobile phones, neighbour join, tracking.

I. INTRODUCTION

We propose a system of gathering preferences of individuals and making a travel guide system based on vehicle sharing for mobile phones, which is able to provide information to the mobile users conveniently.

By installing this application to an android device, it quietly records its location via GPS, Wi-Fi, or cellular data periodically and uploads to a secured server

Our system takes advantage of light-weighted mash up technology that can combine more than one data sources to provide services and overcoming the limitations of mobile devices. Two objects are more likely to be similar if they are both similar to some common object. We have to hold for many real world similarities even those complex ones provided by the users like (GPS co-ordinates of same route travelling persons). This observation leads to our overall strategy: increase the probability of comparing object pairs that are more likely to yield smaller distances by putting objects that are similar to common objects on the same machine.

The near neighbor join algorithm is used for selection of subset by reduction in database. To develop and find a technique that can join extremely large number of objects with complex join functions.

1.1. Scope

The near neighbor join algorithm is used for selection of subset by reduction in database. To develop and find a technique that can join extremely large number of objects with complex joins functions. The developing application has various scopes. The application can be used as smart user guide while travelling and also can be used as pooling app i.e. a companion can be found out while travelling from one place to another.

1.2. Proposed System

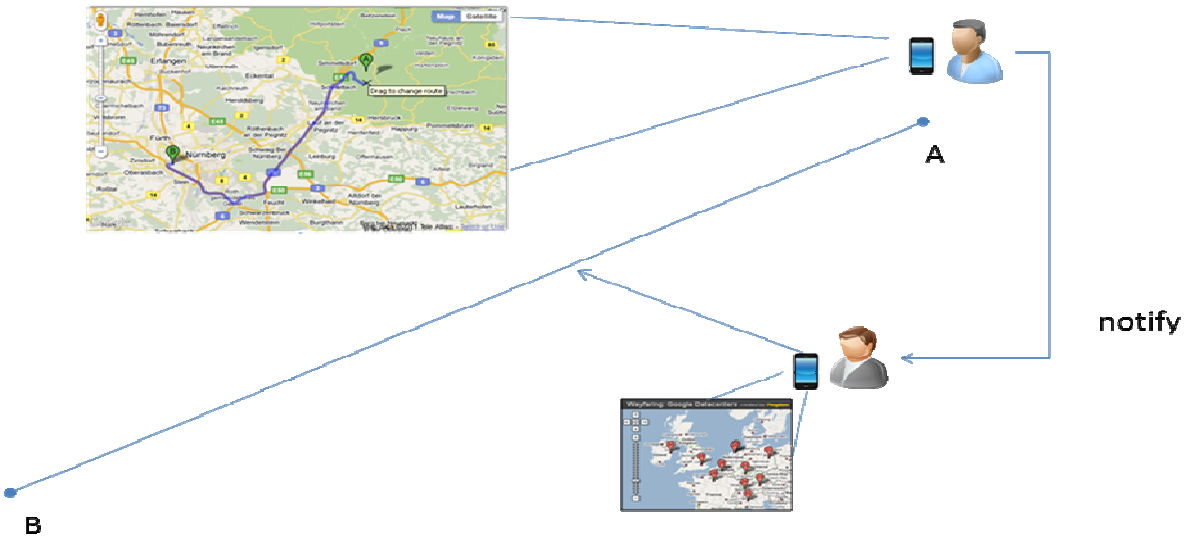

Fig: 1.1

An increasing number of Web applications depend on the ability to join objects at scale. The traditional approach taken is nearest neighbor join (also called similarity join), whose goal is to find, based on a given join function, the closest set of objects or all the objects within a distance threshold to each object in the input. The scalability of techniques utilizing this approach often depends on the characteristics of the objects and the join function.

Join has become one of the most important operations for Web applications. For example, social networking sites routinely compare the behaviors of hundreds of millions of users to identify and provide them with recommendations. Further, in many search applications it is often desirable to showcase additional results among billions of candidates that are related to those already returned. The join functions at the core of these applications are often very complex: they go beyond the database style - joins or the set-similarity style joins.

We used a Scalable Approximate Join system that performs near neighbor join of billions of objects of any type with a broader set of complex join functions, where the only expectation on the join function is that it satisfies the triangle inequality. More specifically, algorithm aims to solve the following problem: Given (1) a set I of N objects of type T, where N can be billions; (2) a complex Join function FJ : T_T ! R that takes two objects in I and returns their similarity; and (3) resource constraints (specified as machine per-task computation capacity and number of machines). For all objects, find k objects in I that are similar and according to FJ without violating the machine constraints. Based on the top-down and bottom-up approach, we build an end-to-end join system capable of processing near neighbor joins over billions of objects without requiring detailed knowledge about the join function.

1.2.1 System Block Diagram

Our system follows the three tier model-view-controller architecture. The interaction between the android application and the database is handled by the webserver. The application sends packets based on JSON (JavaScript Object Notation) which is in the form of XML. Webserver acknowledges the request and responses with the co-ordinates of the user. The block diagram explains the working of the system in brief.

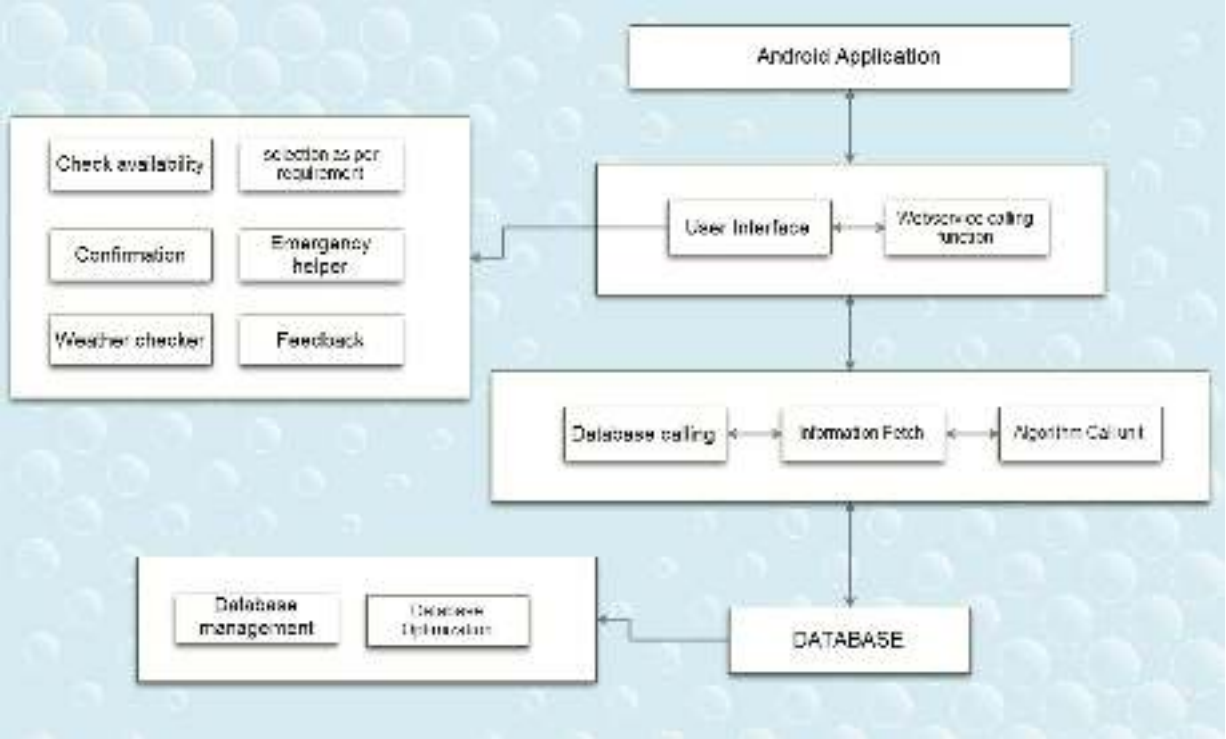

Fig: 1.2

II. RELATED WORK

The idea behind dynamic programming is to solve different parts of the problem called as sub problems and then combining the individual solutions to reach a final solution. Often when used the native method these sub problems are generated many times. To reduce computation complexity, dynamic programming tries to solve these sub problems only once. The solutions to sub problems are kept in memory and when needed they are simply looked up. This method is useful especially when the number of repeating sub problems grows exponentially as a function of the size of the input.

Dynamic programming algorithms are used for optimization like finding the shortest path between two locations, or the fastest way to multiply matrices. A dynamic programming algorithm will analyze the solved sub problems and combine their solutions to give the best possible solution for a given problem.

We are using the near neighbor join algorithm and map-reduce algorithm for enhancing functionality of the application. Map-reduce forms up small regions of datasets from whole lot of dataset while near neighbor computes the most probabilistic dataset from map reduced dataset.

2.1 Algorithms

2.1.1. MapReduceTo perform kNN join using MapReduce which is a well-accepted framework for data-intensive applications over clusters of computers. An effective mapping mechanism that exploits pruning rules for distance filtering, and hence reduces both the shuffling and computational costs is designed.

Input: current state

Output: final result set of specified users_set.

for each parameter k in S: Set T operation_MapReduce able

do

input=Deploy(User_parameters)

Partition_data=PARTITION(User_parameters,Database_parameters)

resultset=dataset_comparison(User_parameters,Partition_data)

DS=CreatefinalresultState()

Return DS

2.1.2. Near Neighbor Join

The algorithm operates on the data set (DS) and performs the iterations of the scalable approximate join. It works in three phases (1) Bottom Up (2) Top Down and (3) Merge

Input: Current user data

Output: Information of most parameter matching dataset

for each symbol k in DS: StateTransitionTable

do

addr=Deploy(parameters);

resultset=dataset_comparison(Parameters, Database_parameters)

DS=CreatefinalresultState()

Return Final_prioritised_resultSet

III. THEORETICAL ANALYSIS

We are given input $I = \{O_i\}_{i=0 \text{ to } N}$ where N is very large (eg. Billions of objects); a user-constructed join function F J: I*I→R that computes a distance between any two objects in I, and that we assume to knowledge except that Fj satisfies the triangle inequality property (for SAJ to provide quality expectation); a parameter k indicating the desired number of neighbors per object; and machine resource constraints. The two key resource constraints are :1)the number of objects each machine can be expected to perform an all-pairs comparison on; and 2) the maximum number of records each Shuffle phase in the program is able to handle, which can be derived from the number of machines available in the cluster. We produce output $O = \{(o_i \rightarrow R_i)_{i=1\text{to } N}$ where Ri is the set of k

objects in I we discover to be near O_i according to Fj. For each O_i, let $R_{i\ nearest}$ be the top –k nearest neighbors of O_i. i.e. $\forall$ (j,k), $O_j \notin R_{i\ nearest}$, O_k , $\forall$(j,k),$o_j \notin R_{i\ nearest,}$ $o_k \in R_{i\ nearest,}$ $F_j(o_i, o_j) \geq F_j(o_i, o_j)$.

Let $E_i = AVG_o \in R_i(F_j(o_i,o)) - AVG_o \in {}_{Ri}^{nearest}(F_j(o_i,o))$ be average distance error of Ri. System attempts to reduce AVGi(Ei) in a best-effort fashion while respecting the resource constraints. Here we show the number of MapReduce iterations is $O(\log(N/n)\log(kN+n^2P)/P)/ \log(n/m)\log(p/P))$ and we bound the work done by the tasks in each of the BU,TU and Merge phases.MR Iterations. Let L be the number of MapReduce iterations in the BU phase. Suppose at each iteration i we partition the input from the previous iteration into Li partitions. Each partition R has size $[mL_{i-1}/L_i]$ since m representatives from each of the Li-1 partitions are sent to the next level. The number Li is chosen such that n/2<=|R|<+n ensuring that the partitions in that level are the large but fit in one task (i.e do not exceed n).This implies that at each iteration we reduce the size of the input data by a

factor $c=L_{i-1}/L_i \geq n/(2m)$. The BU phase stops at the first iteration whose input size is <=n (and w.l.o.g. assume it is at least n/2). Hence, the number of iterations is at most:

$L=\log_c(N)-\log_c(n/2)=\log_c(2N/n) \geq \log(2N/n)/\log(n/(2m))$

IV. CONCLUSION

The algorithms for near neighbor joins can be used to increase the efficiency in computation in the top-down and bottom-up approach. This results in better pairs for comparisons to make groups of similar objects. Our proposed system helps in finding nearest neighbors travelling on a similar path or in the specified area and reduces the steps required to do so by using complex join functions.

REFERENCES

[1] *Kllapi,H., Harb, B.; Cong Yu, Dept. of Inf. & Telecommunication., Univ. of Athens, Athens, Greece, "Near neighbor join,"*

[2] *J. DEAN AND S. GHEMAWAT, "MAPREDUCE: SIMPLIFIED DATA PROCESSING ON LARGE CLUSTERS," COMMUN. ACM, VOL. 51, PP. 107-113, JANUARY 2008.*

[3] *A. Okcan and M. Riedewald, "Processing theta-joins using MapReduce," in SIGMOD, 2011.*

[4] *HONGRAE LEE, RAYMOND T. NG, KYUSEOK SHIM "SIMILARITY JOIN SIZE ESTIMATION USING LOCALITY SENSITIVE HASHING"*

[5] *Processing of k Nearest Neighbor Joins using MapReduce Wei Lu Yanyan Shen Su Chen Beng Chin Ooi National University of Singapore*

[6] *Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph Kiana Hajebi and Yasin Abbasi-Yadkori and Hossein Shahbazi and Hong Zhang Department of Computing Science University of Alberta*

[7] *The k-Nearest Neighbor Join: Turbo Charging the KDD Process Christian Böhm Florian Krebs University for Health Informatics and TechnologyUniversity for Health Informatics and Technology*

[8] *Processing In-Route Nearest Neighbor Queries: A Comparison of Alternative Approaches Shashi Shekhar Department of Computer Science and Engineering, University of Minnesota, Jin Soung Yoo Department of Computer Science and Engineering, University of Minnesota*