Noname manuscript No.
(will be inserted by the editor)

MapReduce Based Location Selection Algorithm for Utility Maximization with Capacity Constraints

Yu Sun · Jianzhong Qi · Rui Zhang · Yueguo Chen · Xiaoyong Du

Received: date / Accepted: date

Abstract Given a set of facility objects and a set of client objects, where each client is served by her nearest facility and each facility is constrained by a service capacity, we study how to find all the locations on which if a new facility with a given capacity is established, the number of served clients is maximized (in other words, the utility of the facilities is maximized). This problem is intrinsically difficult. An existing algorithm with an exponential complexity is not scalable and cannot handle this problem on large data sets. Therefore, we propose to solve the problem through parallel computing, in particular using MapReduce. We propose an arc-based method to divide the search space into disjoint partitions. For load balancing, we propose a dynamic strategy to assign partitions to reducers so that the estimated load difference is within a threshold. We conduct extensive experiments using both real and synthetic data sets of large sizes. The results demonstrate the efficiency and scalability of the algorithm.

Keywords Location Selection · Capacity Constraints · MapReduce

Y. Sun, J. Qi, and R. Zhang
Department of Computing and Information Systems
University of Melbourne, Australia
Tel.: +61-3-83441332
Fax: +61-3-83441345
E-mail: yusun.aldrich@gmail.com, jianzhong.qi@unimelb.edu.au, rui.zhang@unimelb.edu.au

Y. Chen and X. Du
Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), MOE, China
E-mail: {chenyueguo, duyong}@ruc.edu.cn

1 Introduction

Location selection is a classic problem in operational research and has wide applications in decision support systems. For example, a urban planner may need to decide where to build a public car park or a hospital, a company executive may need to decide where to open a new branch office. In recent years, with the widespread use of Global Positioning Systems (GPS) and smartphones, location based social networks have become popular and location selection has found a new area of application.

In this paper, we study a new location selection problem that has traditional as well as modern applications. Fig. 1(a) illustrates the problem. Let $c_1, c_2, ..., c_{13}$ be a set of office buildings and f_1, f_2, f_3 be a set of car parks. People work in the office buildings want to park their cars in their respective nearest vacant car parks. Since the number of parking lots in a car park is limited, some people may have to park faraway. We study where to build a new car park, so that after the new car park is built, the largest number of people can park in their respective nearest car parks.

Similarly, let us assume a location based social network scenario. Fig. 1(a) denotes a board game group where $c_1, c_2, ..., c_{13}$ are group members while f_1, f_2, f_3 are the activity centers provided by the group organizers. The group members want to play games in their nearest activity centers, but an activity center has a capacity and cannot hold all group members. We study where to set up a new activity center, so that the largest number of group members can play board games in their nearest activity centers. On social networks there are many other kinds of interest groups such as reading groups. Meanwhile, group members can come and leave, while activity centers may be deprecated. Thus, this problem may be asked frequently and the need for an efficient solution to the problem is compelling.

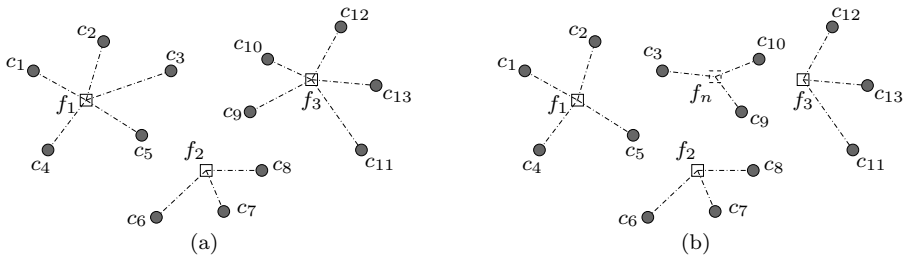


Fig. 1 Problem examples

The above motivating examples are modeled as the problem of **l**ocation **s**election for **u**tility **m**aximization (LSUM): given a set of points C as the clients and a set of points F as the facilities, where each client c is served by her nearest facility and each facility f is constrained by a service capacity $v(f)$, find all the locations in the space on which if a new facility with a given capacity is established, the number of served clients by all the facilities is

maximized (in other words, the utility of the facilities is maximized). Here, every client c is associated with a weight, denoted by $w(c)$, which can be thought of as the number of clients that reside at the same site.

In the example shown in Fig. 1(b), let the weight of each client be 1, and the capabilities of f_1, f_2, f_3 be $4, 3, 3$, respectively. Then the current total service capacity, 10, is less than the number of clients (a weighted sum), 13, and not all clients can be served by their respective nearest facilities. If a new facility with a capacity of 3 is set up on f_n, then it will be the nearest facility of c_3, c_9 and c_{10}. Now the total capacity becomes 15 and every client gets served by her nearest facility. As a result, f_n is one of the locations in the problem answer and we want to find all such locations.

In a recent study [25], the LSUM problem is formalized as the LSUM query and a branch-and-bound based algorithm is proposed to process the query. In the average case, the algorithm has a time complexity of $O(2^\theta|C|)$, where θ increases with $\frac{|C|}{|F|}$. In real applications, $|C|$ is usually very large and $|F|$ is relatively small. In this case, θ can get very large, which leads to prohibitively long query processing time. To overcome the inefficiency, we propose to leverage the power of parallel processing, in particular, the MapReduce framework, to achieve higher query processing efficiency.

There are two main challenges in applying the MapReduce framework for our problem: (i) How to disjointly divide the search space so that no area needs to be searched for more than once? (ii) How to assign balanced loads among different reducers? In this paper, we address these challenges and propose an efficient MapReduce based algorithm for the LUSM query. In summary, we make the following contributions:

1. We study the properties of the LSUM problem and propose a MapReduce based algorithm to solve the problem.
2. We propose a search space dividing strategy that divides the search space based on arcs of the nearest facility circles. This strategy will assign every region to be searched to only one partition. As a result, we achieve disjoint partitions on the search space.
3. We propose a load balancing strategy that further divides the large partitions into sub-partitions dynamically and then forms partitions of similar sizes to be searched. Hence, we achieve partition searching tasks whose estimated workload difference is within a threshold.
4. We conduct extensive experiments on the proposed algorithms using both real and synthetic data sets of large sizes. The results demonstrate the efficiency and scalability of our proposed algorithm.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes preliminaries for the LUSM query. Section 4 gives an overview of our MapReduce based algorithm. Section 5 discusses how to partition the search space and Section 6 discusses load balancing. Experimental results are reported in Section 7 and the paper concludes in Section 8.

2 Related Work

2.1 Location Optimization

Location selection belongs to the category of location optimization problem, which is a classic problem in operational research. Various models [9,10,15, 16] have been proposed to solve location optimization problems of different settings. Some [15,16] have taken the capacity constraints into consideration. However, in general, these models focus more on the demand and/or cost of setting up the facilities rather than maximizing the facility utility. Thus, we will not discuss these models further. Interested readers are refered to some recent reviews [5,17,23].

In the database community, some studies solve the location optimization problem by modeling the locations as facility clusters [22] or functional regions [33], while other studies, including ours, model the locations as points. The latter group of studies are mostly based on the **bi**chromatic **r**everse **n**earest **n**eighbor (BRNN) query introduced by Korn and Muthukrishnan [12]. Like many others [18,19,32,36], the BRNN query is a variant of the nearest neighbor (NN) query. Given two object sets C and F and a query object from F, the BRNN query returns objects in C who perceive the query object as their nearest neighbor in F. The BRNN set of an object is also called the influence set of the object. Based on the influence set, Xia et al. [30], Wong et al. [28,31], Zhou et al. [38] and Huang et al. [6,7] have studied how to find the maximum or top-t most influential spatial sites. In addition, Zhan et al. [34] and Zheng et al. [37] considered the uncertainty in the problem. Zhang et al. [35] and Qi et al. [20] investigated the min-dist problem, which minimizes the average distance between the clients and the facilities. These studies did not consider capacity constraints and their algorithms do not apply.

Wong et al. [29] studied the spatial matching problem with capacity constraints. The study tries to assign each client to her nearest facility whose capacity has not exhausted. Due to the capacity constraints of the facilities, a client may be assigned to a facility very far away. U et al. [27] studied the problem further and proposed algorithms that assigned each client to a facility with a capacity constraint while the sum of the distance between each client and its assigned facility is minimized. Sun et al. [24] studied finding the top-k locations from a candidate set that maximize the total number of clients served by the facilities set up on these locations. In another work [25], they proposed the problem studied in this paper and a centralized algorithm for the problem. As we will use the centralized algorithm as our baseline in the experiments, we will detail it in Section 3.2.

2.2 MapReduce for Computation Intensive Problems

Since proposed, MapReduce has gained much popularity in studies to achieve efficiency and scalability. For example, Lu et al. [14] investigated processing

k nearest neighbor joins using MapReduce. Huang et al. [8] proposed efficient MapReduce based algorithms for the earth mover's distance similarity join. Tao et al. [26] studied the minimal MapReduce algorithms that minimize the storage cost, CPU and I/O cost as well as communication cost simultaneously and proposed minimal algorithms for database problems like ranking and spatial skyline queries.

Like in earlier parallel processing techniques such P2P computing [21] and Grid computing [1], many efforts have been made in MapReduce for load balancing on skewness input and complex, non-linear algorithms. For example, Kolb et al. [11] designed methods to handle skewed data for entity resolution. Gufler et al. [3] addressed the load balancing problem in processing MapReduce jobs with complex reducer tasks. They proposed two load balancing approaches that can evenly distribute the workloads on the reducers based on a cost model. They [4] achieved further performance gains by improving the cost estimation through gathering statistics from the mappers. Kwon et al. [13] presented a system that automatically mitigates skewness for user defined MapReduce programs by redistributing the unprocessed input data of the task with the largest expected remaining processing time.

3 Preliminaries

In this section, we first provide a formal definition of the studied problem, and then briefly describe an existing centralized solution to the problem and the MapReduce framework. Table 1 provides a summary of the frequently used symbols and their meanings.

Table 1 Frequently Used Symbols

Symbol	Meaning
S	the data space
C	the set of clients
F	the set of facilities
c	a client in C
f	a facility in F
$w(c)$	the weight of client c
$v(f)$	the capacity of facility f
$\|c, f\|$	the Euclidean distance between client c and facility f
$b(f)$	the bichromatic reverse nearest neighbor set (influence set) of facility f
$d(f)$	the summed weight of the clients in $b(f)$ (demand on facility f)
$u(f)$	the utility of facility f which equals the smaller value between $v(f)$ and $d(f)$
$u(F)$	the summed utility of all facilities in F
M	the set of optimal locations (optimal regions)
$n(c)$	the nearest facility circle of client c
$\bar{n}(c)$	the area outside $n(c)$
R	a consistent region in the data space
p	a point in the data space
$b(p)$	the bichromatic reverse nearest neighbor set (influence set) of point p
$b(R)$	the influence set of a consistent region R

3.1 Problem Definition

The LSUM problem involves a set of clients C and a set of facilitates F, where a client c is associated with an integer as its weight, denoted by $w(c)$, and a facility f is associated with an integer as its capacity, denoted by $v(f)$. The objects in the two sets are points in 2-dimensional Euclidean space. Given two objects c and f, $|c, f|$ denotes their Euclidean distance.

The LSUM problem relies on the bichromatic reverse nearest neighbor (BRNN) query, which helps determine the clients that can be served by a facility.

Definition 1 (bichromatic reverse nearest neighbor (BRNN) query) Given two object sets C and F and a query object $f \in F$, the BRNN query returns a subset of C, denoted by $b(f)$. The objects in $b(f)$ all view f as their nearest object in F, i.e., $\forall c \in b(f)$ and $f_i \in F$, $|c, f| \leq |c, f_i|$.

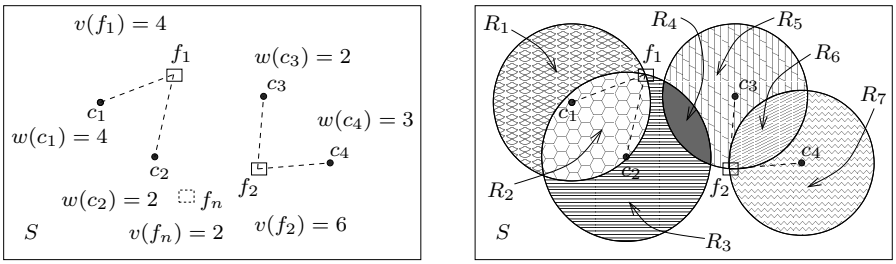


Fig. 2 Basic concepts

We call the BRNN set $b(f)$ the influence set of f. All clients in $b(f)$ will be served by f as long as the total of their weights is within the capacity of f. For example, in Fig. 2(a), c_1, c_2, c_3 and c_4 represent the clients; f_1 and f_2 represent the facilities. The BRNN sets (influence sets) of f_1 and f_2 are $\{c_1, c_2\}$ and $\{c_3, c_4\}$, receptively.

We sum up the weight of the clients in $b(f)$ and call it the demand on f, denoted by $d(f)$, $d(f) = \sum_{c \in b(f)} w(c)$. In Fig. 2(a), the demand on f_1 and f_2 are $w(c_1) + w(c_2) = 6$ and $w(c_3) + w(c_4) = 5$, respectively. A facility's capacity may be smaller or larger than the demand on the facility. For example, $v(f_1) = 4 < 6$ while $v(f_2) = 6 > 5$. We take the smaller value between $v(f)$ and $d(f)$ and define it as the utility of f, denoted by $u(f)$, $u(f) = \min\{v(f), d(f)\}$. In Fig. 2(a), $u(f_1) = \min\{4, 6\} = 4$ and $u(f_2) = \min\{6, 5\} = 5$.

We call the sum of the utility values of a set of facilities F the utility of F, denoted by $u(F)$, $u(F) = \sum_{f \in F} u(f)$. In Fig. 2(a), $u(F) = 9$.

When adding a new facility at f_n with a capacity $v(f_n)$ to the set of facilities, the utility value of all facilities changes to $u(F \cup \{f_n\}) = \sum_{f \in F \cup \{f_n\}} u(f)$.

The goal of this study is to identify all locations in the whole data space S that, after establishing the facility on any of these locations, $u(F \cup \{f_n\})$ is maximized.

Definition 2 (location selection for utility maximization (LSUM)) Given the capacity of a new facility, the location selection for utility maximization problem finds a set of locations M from the data space S. If the new facility is set up on any location $f_n \in M$, the utilization of $F \cup \{f_n\}$, $u(F \cup \{f_n\})$, is maximized. Here, $u(F \cup \{f_n\}) = \sum_{f \in F \cup \{f_n\}} u(f)$.

Here, the capacity of the new facility is a parameter given by the query user, which reflects the scale of the new facility that the user plans to set up. In this study, to simplify our discussion, we assume that this parameter is not affected by the chosen facility locations, i.e., we assume that we can set up a new facility with the given capacity at any location. Since we are finding a set of locations for the new facility, in real world scenarios, we can further process the found locations and identify those that allow setting up a new facility with the given capacity.

In Fig. 2(a), if a new facility of a capacity of 2 is added at f_n to be the new nearest neighbor of c_2 (but not any other clients), then the utility of all facilities becomes $4 + 2 + 5 = 11$, which is maximized since the total demand is 11. As a result, f_n is a point in the solution set of the LSUM problem.

To solve the problem we need the following concepts. We use the **n**earest **f**acility **c**ircle (NFC) [12] to identify the influence sets.

Definition 3 (nearest facility circle (NFC)) Given a client c, the nearest facility circle of c, denoted by $n(c)$, is a circle that centers at c and has a radius of $|c, f_c|$, where f_c is the nearest facility of c.

The influence set of f is formed by the clients whose NFCs enclose f. As shown in Fig. 2(b), the circles represent the NFCs of four clients. The NFCs of c_1 and c_2 enclose f_1 and hence, $b(f_1) = \{c_1, c_2\}$. Similarly, the NFCs of c_3 and c_4 enclose f_2 and hence, $b(f_2) = \{c_3, c_4\}$.

In addition to $n(c)$, we use $\bar{n}(c)$ to denote the area outside $n(c)$, i.e., $\bar{n}(c) = S \setminus n(c)$, where S denotes the whole data space. Further, we use $n(c)$ and $\bar{n}(c)$ to define the consistent region to identify facilities with the same influence sets.

Definition 4 (consistent region) A consistent region R is formed by a set of points that are either all in $n(c)$ or all in $\bar{n}(c)$ for any client $c \in C$, i.e., $\forall c \in C$, $p_i, p_j \in R$, $p_i \neq p_j$, either $p_i, p_j \in n(c)$ or $p_i, p_j \in \bar{n}(c)$.

Straightforwardly it is proved [25] that all points in a consistent region R have the same influence set, i.e., $\forall p_i, p_j \in R$, $b(p_i) = b(p_j)$. We call this same influence set the influence set of R, denoted by $b(R)$. We further define the maximal region, which is formed by all points in S that have the same influence set.

Definition 5 (maximal region) A consistent region R is a maximal region if $\forall R' \cap R \neq \emptyset$ and $b(R') = b(R)$, $R' \subseteq R$.

As shown in Fig. 2(b), each of the shaded region denotes a maximal region. For example, the horizontal-lined region contains all points enclosed by $n(c_2)$ whose influence set is $\{c_2\}$. Similarly, the gray region contains all points enclosed by $n(c_2)$ and $n(c_3)$ whose influence set is $\{c_2, c_3\}$.

Adding a new facility to a different maximal region may result in different utility values of all the facilities. In Fig. 2(b), suppose the new facility has a capacity of 2. Then adding it to the horizontal-lined region will result in the maximum utility value 11 as analyzed above, while adding it to the gray region will result in a smaller utility, 9 ($u(\{f_1, f_2, f_n\}) = u(f_1)+u(f_2)+u(f_n) = 4 + 3 + 2 = 9$). To find the optimal locations, we need to search each of the maximal regions.

3.2 Existing Centralized Algorithm

An existing centralized algorithm [25] solve the problem by iterating all maximal regions to find the optimal one, during the process some pruning techniques are applied to avoid searching the unpromising maximal regions. We focus on the maximal region generation process as the pruning techniques are irrelevant in our MapReduce based algorithm design.

We illustrate the generation process using the example shown in Fig. 2. To solve the problem, the algorithm needs to generate the maximal regions as shown by the shaded regions. It scans all NFCs according to the ascending order of the client IDs and uses the intersection among NFCs to generate the maximal regions. First it generates all maximal regions within $n(c_1)$. Since $n(c_1)$ only intersects $n(c_2)$, it obtains maximal regions R_1 and R_2. Next, the algorithm processes $n(c_2)$, which intersects $n(c_1)$ and $n(c_3)$. To avoid repetitive search, it only consider intersections from NFCs whose corresponding client ID is larger than that of the current NFC. Thus, the generated maximal regions are R_3 and R_4. This process continues until all NFCs have been scanned and all maximal regions are produced during the process.

3.3 MapReduce Framework

MapReduce [2] is a popular parallel computing framework. The execution of a MapReduce algorithm is called a job (or round) while the data is organized in the form of key-value pairs. A MapReduce job is processed in three stages.

1. **Map** Each map task takes the input and generates a list of key-value pairs $\langle k, v \rangle$.

2. **Shuffle** In the shuffle phase, the key-value pairs are distributed to reducers (i.e., machines that performs reduce tasks) based on the keys.

3. **Reduce** In the reduce phase, key-value pairs that are of the same key are processed together and the results are output for different keys.

4 Overview of the MapReduce Algorithm for the LSUM Problem

To take advantage of parallelism, we divide the data space into partitions and search the maximal regions in each partition synchronously. We then merge the local results and identify the final result. The whole process is achieved by three MapReduce jobs:

1. **Search space partitioning**, where the search space is divided into partitions;
2. **Local optimal region searching**, where the maximal regions in each partition is searched in parallel to identify local optimal regions, and
3. **Result merging**, where the local optimal regions are merged to generate the final result.

4.1 Search Space Partitioning

The first job reads in all the two datasets C and F and computes the NFCs and the influence sets to set up the search space. Then it divides the search space into partitions based on the NFCs (detailed in Section 5). For load balancing, it estimates the workload of each partition and subdivide the partitions to obtain sub-partitions that are with similar workloads (detailed in Section 6). These sub-partitions will then be searched in parallel in the next job.

The output of this job is partitions in the form of key-value pair $\langle t_i, m_i \rangle$, where the key t_i denotes the partition ID and the value m_i denotes the data that represent the partition. Here m_i typically involves the facilities and clients in the partition, the relevant influence sets and NFCs.

We take advantage of spatial index to compute the NFCs and influence sets efficiently. In particular we use two R-trees to index the clients and the facilities separately, and then use a popular R-tree based algorithm described by Korn and Muthukrishnan [12] to obtain the NFCs and influence sets. Since the R-tree based algorithm does not scale well in the MapReduce framework, while is very efficient on a single machine, we perform the first job on a single machine.

4.2 Local Optimal Region Searching

The input of the second job is partitions to be searched, and the expected output is the optimal regions in each partition which are called local optimal regions. The map phase of this job simply distributes the partitions based on their IDs to the reducers. In reduce phase, the reducers search the partitions in parallel. A reducer r searches the partitions assigned to it one after another using a local version of the algorithm described in Section 3.2. After the search, each reduce task outputs the local optimal regions in the form of a key-value pair $\langle u, R \rangle$. Here, the key u is a utility value, and the value R is the region that achieves the utility. The utility value u in each output key-value pair is

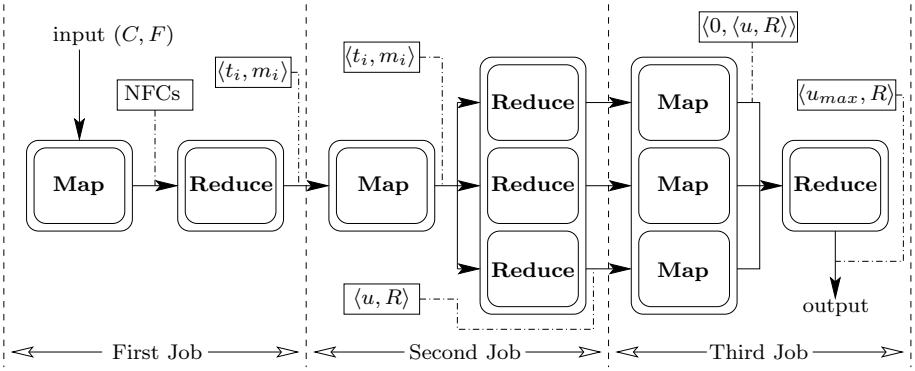


Fig. 3 Algorithm overview

represented as an integer, which will be used by the third job to pick out the global optimal regions.

Note that our partitioning strategy (detailed in Section 5) does not change the number of NFCs a region is enclosed by, and hence dose not change the "optimality" of a region. Thus, it does not change the region "optimality" comparison result whether the comparison is done in parallel or in a centralized fashion. Therefore, in the third job, we just need to identify the global optimal regions from the local optimal regions computed in parallel in the second job.

4.3 Result Merging

The third job collects the results output by the second job, identifies regions with the overall maximum utility and outputs the final result. Here, the map phase collects the output of every reducer of the second job, and shuffles them all to a reducer, where the local optimal regions are scanned to identify the global optimal regions. If the number of local optimal regions is too large to be handled by one reducer, we use multiple reducers where each identifies the optimal regions from the subset of local optimal regions assigned to it. We then repeat the third job until the number of local optimal regions is small enough to be handled by one reducer and the global optimal regions are identified.

Fig. 3 summarizes the three jobs and the data flow.

5 Search Space Partitioning

A straightforward way to divide the search space is to use a grid, where the maximal regions intersecting each cell form a partition. The drawback of this strategy is that a maximal region can intersect multiple cells and hence will be assigned to multiple partitions. For example, in Fig. 4, the search space is divided into four cells. The gray region intersects all four cells. Thus, it will be assigned to four partitions. This drawback will result in multiple searches

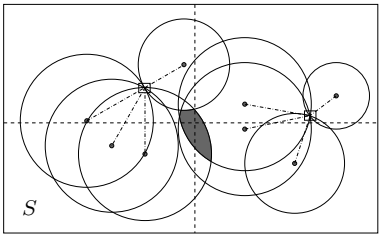


Fig. 4 Grid partitioning

on one maximal region, which is a waste of the limited computation resources. As the grid granularity gets finer, the problem gets worse.

Arc-based Partitioning. To avoid the overlaps between partitions, we propose to use the arcs from the NFCs to divide the space and form partitions. Since maximal regions are formed by NFCs, arc-based partitioning forms disjoint partitions naturally. For example, in Fig. 5, the space is divided into four partitions that do not share any maximal regions.

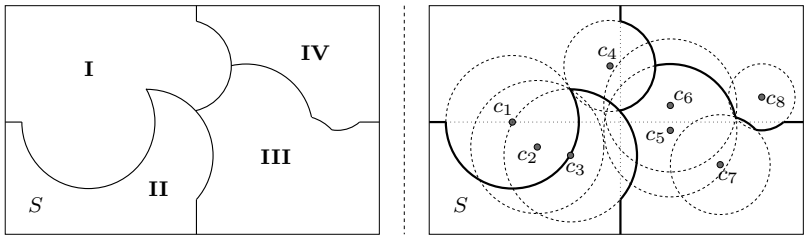


Fig. 5 Arc-based partitioning

Arc-based partitions are obtained from the grid-based partitions. Specifically, the partitions and their boundaries are computed as follows. For every cell g in the grid, if g contains the center of an NFC $n(c_i)$, then we assign a maximal region inside NFC $n(c_i)$ to g provided this maximal region is outside of any other NFC $n(c_j)$ with an ID smaller than that of $n(c_i)$ (i.e. $1 \leq j < i$). In this way, every maximal region will only be assigned to one partition and hence be searched once in the search job. The maximal regions assigned to g form a partition. Their bounding arcs, together with the edges of g, form the boundary of the arc-based partition. As an example, cell I in Fig. 5 contains the center of NFC $n(c_1)$ and $n(c_4)$. All the maximal regions in $n(c_1)$, plus those in $n(c_4)$ but not in $n(c_2)$ or $n(c_3)$, are assigned to this cell. Their bounding arcs (i.e., arcs of $n(c_1)$, $n(c_3)$ and $n(c_4)$) and the boundary of the cell form the boundary of the partition.

As shown in Fig. 5, the arc-based partitioning may lead to partitions of quite different sizes and hence different workloads for the reducers in the search job. In what follows, we discuss how to sub-divide the partitions to obtain approximately balanced workloads.

6 Load Balancing

For simplicity we assume the nodes in the MapReduce cluster have the same computation capability.

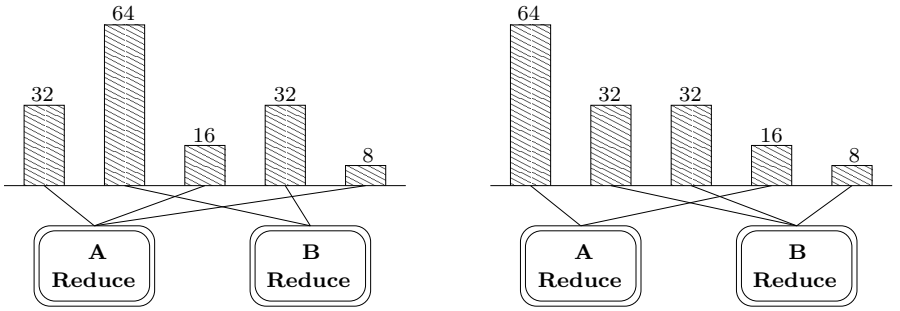
We first model the workload of processing a partition. The search cost of a partition is determined by the number of NFCs in the partition. We assume the finest level partitioning, i.e., each NFC forms a partition. Then the workload of a partition is determined by the number of maximal regions its NFC contains. Let the cost of searching a maximal region be the unit workload. In the worst case, an NFC that intersects n NFCs can form 2^n maximal regions to be searched (as described in Section 3.2). Thus, a partition that intersects n NFCs will have 2^n unit workloads in the worst case. We use this worst case workload as an estimation to balance the workload among the reduce tasks.

6.1 Partition Assignment

Let s be the number of reducers the search job. We now discuss how to assign the $|C|$ single NFC partitions to the reducers to obtain balanced workloads.

6.1.1 Round-robin Assignment

Straightforwardly we can assign the partitions in a round-robin fashion. We arrange the s reducers in a row and the ith partition is assigned to the $(i \bmod s)^{th}$ reducer. For example, in Fig. 6(a), we have five partitions whose costs are 32, 64, 16, 32 and 8, respectively, and two reducers A and B. Applying the round-robin strategy, the first, third and fifth partitions are assigned to reducer A and the others are assigned to B, which results in workloads on A and B being 56 and 96, respectively. We use this strategy as one of the baseline strategies. However, since it does not guarantee balanced workloads, we still need better strategies.



(a) Round-robin strategy (b) Least-load strategy

Fig. 6 Basic assignment strategies

6.1.2 Least-load Assignment

We adopt a greedy assignment approach proposed by Gufler et al. [3] and call it the least-load assignment strategy. This strategy always assign the next most expensive partition to the reducer that have been assigned the least workload. The assignment process continuous until all partitions are assigned.

Applying this strategy, as shown in Fig. 6(b), partitions with cost 64 and 16 are assigned to reducer A, while the rest of the partitions are assigned to reducer B. After the assignment, A and B have workloads of 80 and 72, respectively. Algorithm 1 summarizes the least-load assignment process.

Algorithm 1: LeastloadAssignment

Input: Q_p /∗partition queue∗/
PQ_r /∗reduce task priority queue∗/
Output: PQ_r /∗reduce task queue with assigned partitions∗/

```
1 while Q_p is not empty do
2     p ← head(Q_p)                          /*remove the head of Q_p to p*/
3     r ← head(PQ_r)
4     r.addPartition(p)                      /*assign partition p to reduce task r*/
5     r.load ← r.load + p.cost
6     PQ_r.insert(r)                         /*reinsert r*/
7 return PQ_r
```

The least-load assignment strategy produces more balanced workloads than the round-robin strategy does on average. However, when there are a few partitions with workloads that are much larger than most of the other partitions, this strategy fails. Two examples are shown in Fig. 7. As shown by Fig. 7-I, assuming that we have two reducers to process these partitions and the least-load assignment strategy is used, the workloads of these two reducers will be 1024 and 120, respectively. Similarly, if we have three reducers to process the partitions as shown by Fig. 7-II, then their workloads will be 1024, 512, and 56, respectively. In either case, the workloads are very skew and the advantage of parallelism will not be fully exploited.

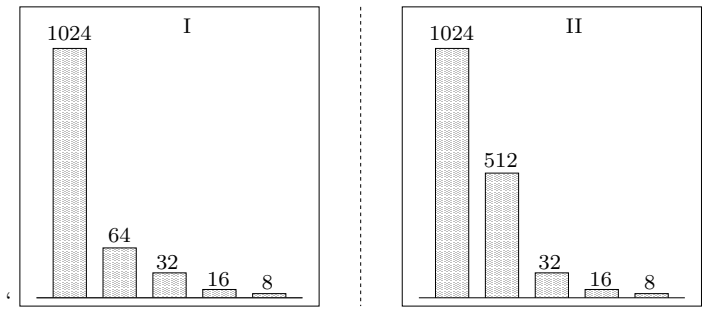


Fig. 7 Examples when the least-load assignment strategy fails

6.1.3 Dynamic Assignment

We further propose a dynamic assignment strategy that achieves balanced workloads by iteratively refining the workload on each reducer through subdividing the partitions. In each iteration, we get the average workload per partition for all partitions to be assigned, and subdivide the partitions whose workloads are larger than the average by a predefined threshold ε into sub-partitions. After that, we assign the sub-partitions as well as the partitions that have not been subdivided with the least-load assignment strategy. After the assignment, we get the average workload per reducer. If the workload of a reducer is larger than the average by ε, then on this reducer, the partition with the highest workload is removed from the reducer and subdivided. The above procedure repeats until the workload of each reducer is within the threshold.

Algorithm 2 summarizes the dynamic assignment strategy.

Algorithm 2: DynamicAssignment

Input: L_p /*list of partitions*/
PQ_r /*reduce task priority queue*/
ε /*threshold $\varepsilon \geq 0$*/
Output: PQ_r /*reduce task queue with assigned partitions*/

```
1  even ← false
2  t ← first integer that makes 2^t ≥ PQ_r.size
3  while true do
4      ave ← average(L_p)                                  /*get the average cost*/
5      for each p ∈ L_p do
6          if p.cost > (1 + ε) × ave then
7              /*subdivide partition p into 2^t subparts and return the existing ones*/
8              subparts[2^t] ← subdivide(p, 2^t)
9              L_p.insertAll(subparts)          /*insert all sub-parts into partition list*/
10     LeastloadAssignment⟨L_p, PQ_r⟩  /*use same strategy as least-load assignment*/
11     ave ← average(L_p)                                  /*get the new average cost*/
12     even ← true
13     for each r ∈ PQ_r do
14         if r.load > (1 + ε) × ave then
15             even ← false                          /*load of reduce task r is too large*/
16             ep ← expPartition(r)                  /*return the most expensive partition*/
17             subparts[2^t] ← subdivide(ep, 2^t)
18             L_p.insertAll(subparts)
19         L_p.insertAll(r.partitions)               /*reinsert all partitions in r to repeat*/
20     if even then
21         break
22 return PQ_r
```

Subdividing a partition. The subdividing of a partition is performed in a binary fashion using the NFCs intersecting the partition. As shown in Fig. 8(I), a partition is formed by an NFC where the shaded regions are excluded from the partition. Besides the shaded regions, this NFC intersects 3 NFCs as shown by the dashed arcs a_1, a_2 and a_3. These 3 arcs form $2^3 = 8$ maximal regions in

the NFC, as denoted by $000, 001, 010, 011, 100, 101, 110$, and 111, respectively. Thus, the workload of processing this partition is 8 workload units. We first use a_1 to divide the partition. Then 4 maximal regions $010, 110, 100$, and 000 fall in one partition P_1 while the rest of the maximal regions fall in the other partition P_2. We repeat this process by using a_2 and a_3 to further divide P_1 and P_2 until the partitions cannot be divided any more (i.e., each partition contains only one maximal region) or the number of resultant partitions is larger than the number of reducers s, since at this point the resultant partitions can be assigned to the reducers approximately evenly already.

Note that a byproduct of the partition subdividing process is the refinement of partition workload estimation. Given a partition that intersects n NFCs, we estimate the workload of processing the partition as 2^n workload units. This is assuming that the n NFCs all intersect with each other and hence form 2^n maximal regions, which may not always be the case. As shown in Fig. 8(II), the 3 arcs now only form 5 maximal regions, as denoted by $000, 001, 010, 100$, and 110. The original workload estimation has an error of $8-5=3$. After we divide the partition by a_1, we will have two sub-partitions P_1 and P_2, containing sets of maximal regions $\{110, 100\}$ and $\{010, 000, 001\}$, respectively. The estimated workloads of P_1 and P_2 will be 2 and 4, respectively, and the total estimation error now is reduced to 1.

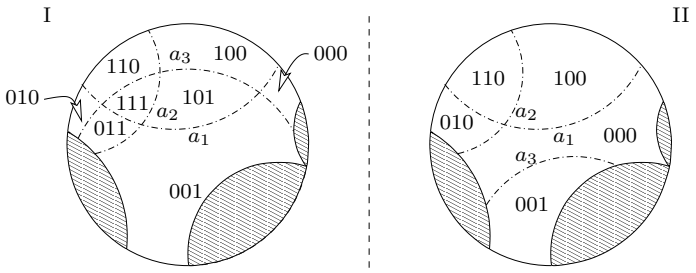


Fig. 8 Subdividing a partition

7 Experiments

We evaluate the performance of our MapReduce algorithm on an in-house cluster. The cluster consists of 4 computing nodes. Among them, two are equipped with Intel i7-3770 3.4GHz quadcore processors, and the other two are with Intel i7-2600 3.4GHz quadcore processors. One node has 32GB main memory, the other three each has 16GB of main memory. Each node has a 3TB SATA hard disk and is connented to a gigabit Ethernet. The software used CentOS 6.0 operating system, Java 1.7.0 with a 64-bit server VM and Hadoop 2.0.0-cdh4.2.1. In Hadoop, the replication factor is set to 2 and the size of virtual memory for each map and reduce task is set to 4GB.

We evaluate the following approaches in the experiments.

1. **Centralized** is the centralized algorithm [25] describe in Section 3.2.
2. **Round-robin** is our MapReduce algorithm with the round-robin partition assignment strategy described in Section 6.1.1.
3. **Least-load** is our MapReduce algorithm with the least-load partition assignment strategy described in Section 6.1.2.
4. **Dynamic** is our MapReduce algorithm with the dynamic partition assignment strategy described in Section 6.1.3.

Both real data sets and synthetic data sets are used. Three real data sets **NA**, **NE** and **US** are retrieved from the R-tree Portal[1]. Table 2 lists the details of the real data sets. When using a real dataset, we uniformly sample from it to generate C and F. We generate synthetic data sets denoted by **SN**. To simulate real-world scenarios, we generate C and F with the Zipfian distribution and the skew coefficient is set at 0.2 in a space domain of $10^4 \times 10^4$. We set the weight of each client at 1 and generate the capacity of the facilities following the Gaussian distribution while the mean and the standard deviation are set at $2 \times \frac{|C|}{|F|}$ and $0.4 \times \frac{|C|}{|F|}$, respectively.

Table 2 Real Datasets

Dataset	Cardinality	Description
NA	24,360	locations in north America apart
NE	119,898	addresses in northeast of the U.S.
US	14,478	locations in the U.S. apart from Hawaii and Alaska

We measure the response time (denoted by "Response time") and the running time variance (denoted by "Time variance") of different reducers in the cluster.

7.1 Fine Tuning the Dynamic Assignment Strategy

We first measure the effect of the partition subdividing threshold (denoted by ε) to optimize the dynamic assignment strategy. We vary the threshold from 10^{-3} to 10^{-7} and Fig. 9 shows the performance of our MapReduce algorithm on different data sets.

As can be seen from the figure, the algorithm performance varies in different patterns on different data sets. However, in general, when $\varepsilon = 10^{-6}$, our algorithm performs the best in response time (cf. Fig. 9(a)). As a result, we will use this value in the following experiments.

Meanwhile, we observe in Fig. 9(b) that the running time variance of different nodes decreases with the decrease of ε. This is expected since the threshold ε controls the workload difference between the reducers. However, a smaller workload difference itself does not guarantee overall minimum response time.

[1] http://www.chorochronos.org/

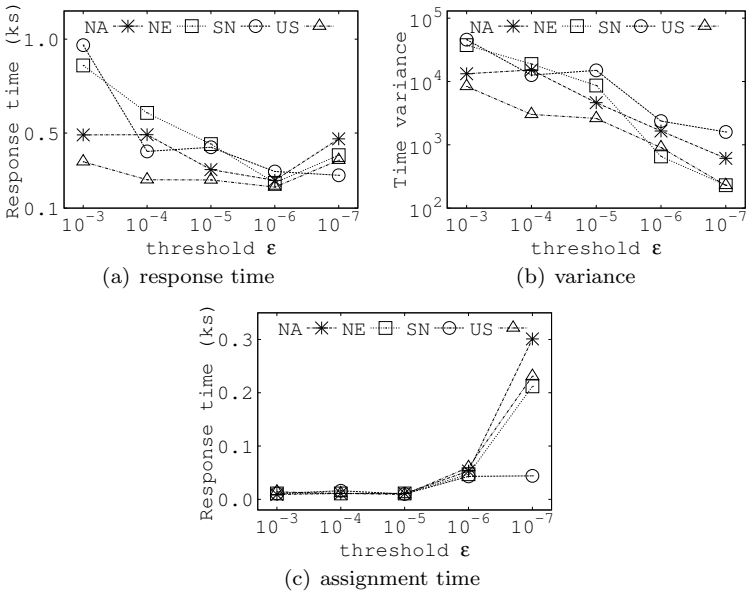


Fig. 9 Effect of ε on the dynamic assignment strategy

This is because to achieve a smaller workload difference requires a larger number of iterations in the dynamic assignment strategy. As shown in Fig. 9(c), $\varepsilon = 10^{-7}$ leads to high assignment time, which results in higher total response time even when the workload difference is smaller.

7.2 Comparative Study

Next we compare the performance of different algorithms varying different parameters.

7.2.1 Effect of $\frac{|C|}{|F|}$

First we evaluate the effect of the ratio $\frac{|C|}{|F|}$. In this set of experiments, the value of $\frac{|C|}{|F|}$ is varied from 10 to 50 and the results on different data sets are presented in Fig. 10.

As $\frac{|C|}{|F|}$ increases, the response time of all methods increase. This is because when $\frac{|C|}{|F|}$ increases, either $|C|$ increases or $|F|$ decreases. In the former case, the number of NFCs increases as well as the number of maximal regions to be searched. In the latter case, the sizes of the NFCs increase and hence the number of maximal regions also increases. As a result, it takes more time for all methods to solve the problem.

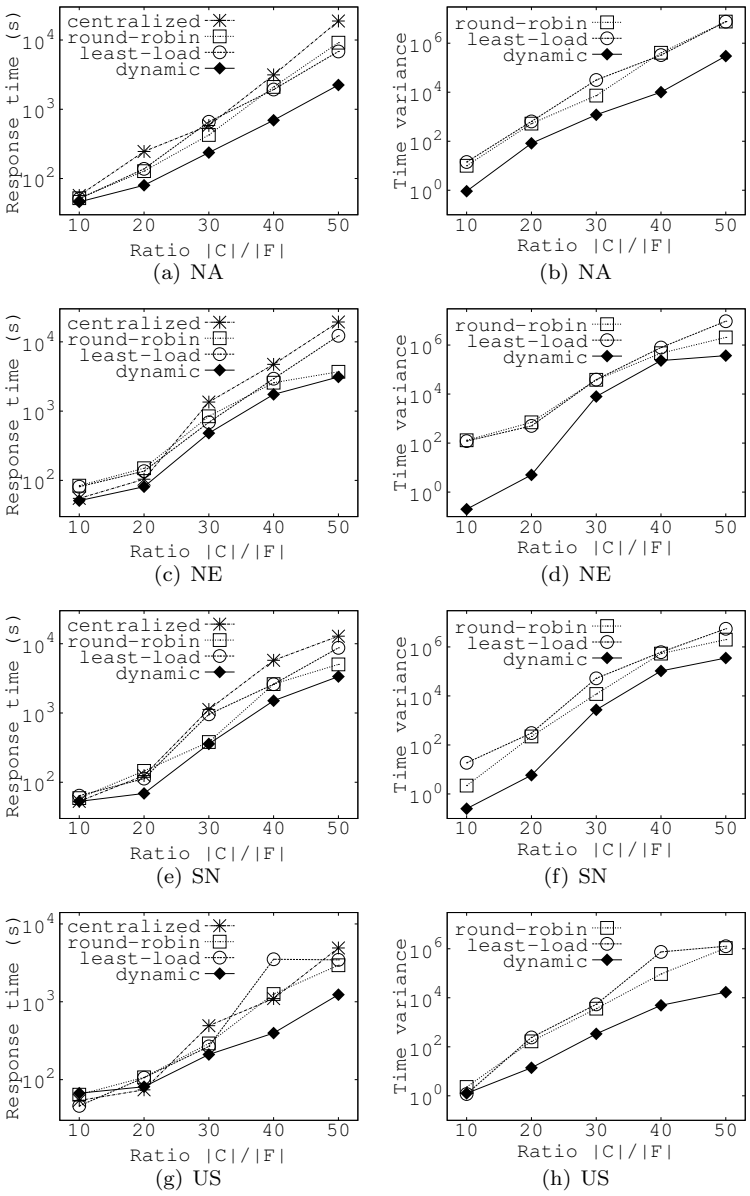


Fig. 10 Effect of the ratio $\frac{|C|}{|F|}$

As the figure shows, the centralized algorithm is the slowest while dynamic is the fastest in most cases. The performance of round-robin and least-load lie in between. On average, dynamic requires only 30% of the response time of the centralized algorithm. Considering that our cluster only has 4 nodes, this speed-up demonstrates the superiority of our proposed MapReduce algorithm.

In terms of the running time variance on the different nodes, dynamic achieves the smallest time variance in all cases. This justify the proposal of the dynamic assignment strategy.

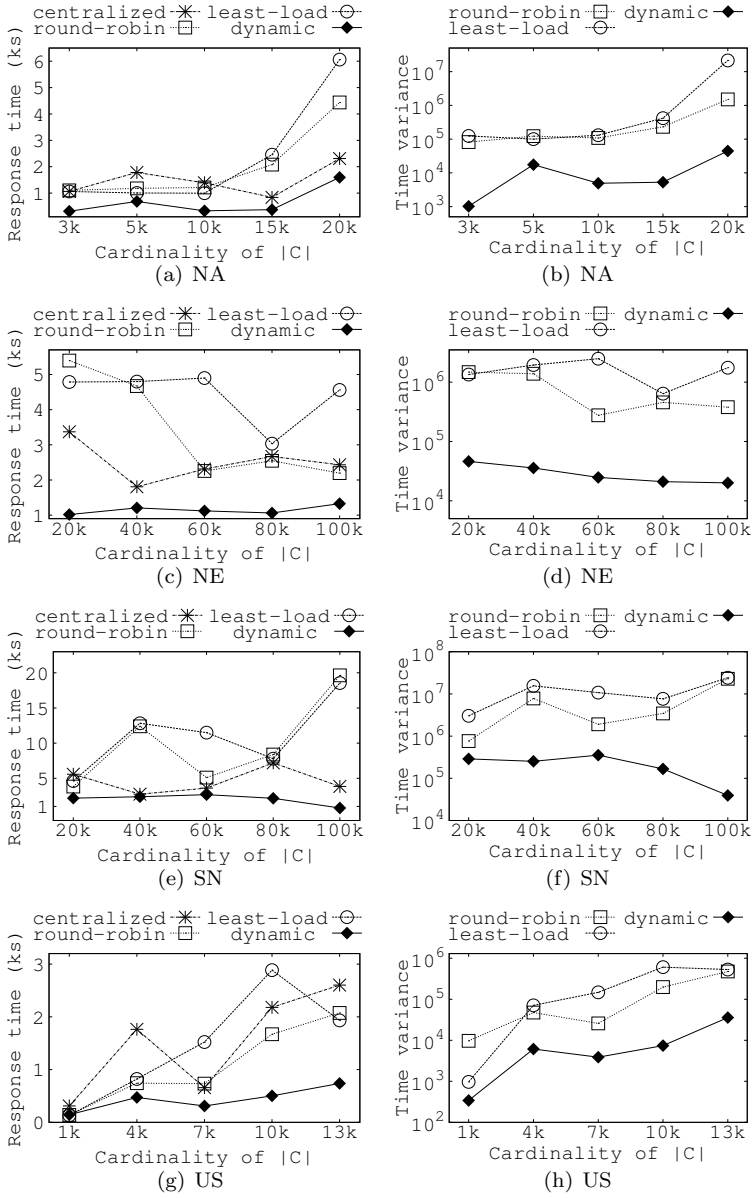


Fig. 11 Effect of data set cardinality

7.2.2 Effect of Data Set Cardinality

The second set of the experiments studies the effects of data set cardinality. The ratio of $\frac{|C|}{|F|}$ is fixed at 25. For NA and US data sets, we vary the cardinality of $|C|$ from 3,000 to 20,000 and 1,000 to 13,000, respectively. For NE and SN, $|C|$ is varied from 20,000 to 100,000.

As shown in Fig. 11, dynamic again outperforms the other methods constantly. Its response time is relatively steady when $|C|$ increases, which demonstrates the scalability of our proposed method. Meanwhile, due to imbalanced workloads, round-robin and least-load cannot outperform the centralized algorithm in all cases, which again validates the necessity of our proposed dynamic assignment strategy.

8 Conclusions

In this paper, we proposed a MapReduce based algorithm to solve the problem of location selection for utility maximization. To fully exploit the capability of parallelism, we proposed an arc-based search space partitioning strategy as well as a dynamic load balancing strategy. By arc-based partitioning, we divide the search space into a number of disjoint partitions. Thus, every region in the space will only be assigned to at most one partition and searched for once. By dynamic assignment strategy, we subdivide the disjoint partitions obtained from arc-based partitioning and assign them to the reducers, where the estimated workload difference among different reducers is controlled by a predefined threshold. We empirically study the effect of the workload difference threshold to optimize our MapReduce based algorithm and compare it with a centralized algorithm on both real and synthetic data sets. The results show that our proposed MapReduce based algorithm outperforms the centralized algorithm significantly and validate the effectiveness of the arc-based partitioning and dynamic assignment strategies.

9 Acknowledgements

This work is supported by the Australian Research Council (ARC) Discovery Project DP130104587. Dr. Rui Zhang is supported by the ARC Future Fellowships Project FT120100832. Dr. Yueguo Chen is partially supported by the National Science Foundation of China under grant No. 61003085. Dr. Xiaoyong Du is partially supported by the National Science Foundation of China under grant No. 61170010.

References

1. A. Al-Khateeb, N. A. Rashid, and R. Abdullah. An enhanced meta-scheduling system for grid computing that considers the job type and priority. Computing, pages 389–410, 2012.

2. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI, pages 137–150, 2004.
3. B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Handling data skew in mapreduce. In The First International Conference on Cloud Computing and Services Science, pages 574–583, 2011.
4. B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Load balancing in mapreduce based on scalable cardinality estimates. In ICDE, pages 522–533, 2012.
5. T. S. Hale and C. R. Moberg. Location science research: a review. Annals of Operations Research, 123(1-4):21–35, 2003.
6. J. Huang, Z. Wen, M. Pathan, K. Taylor, Y. Xue, and R. Zhang. Ranking locations for facility selection based on potential influences. In The 37th Annual Conference on IEEE Industrial Electronics Society, pages 2411–2416, 2011.
7. J. Huang, Z. Wen, J. Qi, R. Zhang, J. Chen, and Z. He. Top-k most influential locations selection. In CIKM, pages 2377–2380, 2011.
8. J. Huang, R. Zhang, R. Buyya, and J. Chen. Melody-join: Efficient earth mover's distance similarity join using mapreduce. In ICDE, 2014.
9. C. Kahraman, D. Ruan, and I. Doan. Fuzzy group decision-making for facility location selection. Information Sciences, 157:135–153, 2003.
10. A. Klose and A. Drexl. Facility location models for distribution system design. European Journal of Operational Research, 162(1):4–29, 2005.
11. L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In ICDE, pages 618–629, 2012.
12. F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In SIGMOD, pages 201–212, 2000.
13. Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: mitigating skew in mapreduce applications. In SIGMOD, pages 25–36, 2012.
14. W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. Proc. VLDB Endow., 5(10):1016–1027, 2012.
15. S. Melkote and M. S. Daskin. Capacitated facility location/network design problems. European Journal of Operational Research, 129(3):481–495, 2001.
16. M. Melo, S. Nickel, and F. Saldanha da Gama. Dynamic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. Computers and Operations Research, 33(1):181–208, 2006.
17. M. T. Melo, S. Nickel, and F. Saldanha-Da-Gama. Facility location and supply chain management–a review. European Journal of Operational Research, 196(2):401–412, 2009.
18. S. Nutanong, E. Tanin, and R. Zhang. Incremental evaluation of visible nearest neighbor queries. TKDE, 22(5):665–681, 2010.
19. S. Nutanong, R. Zhang, E. Tanin, and L. Kulik. Analysis and evaluation of v*-*k*nn: an efficient algorithm for moving *k*nn queries. VLDB J., 19(3):307–332, 2010.
20. J. Qi, R. Zhang, L. Kulik, D. Lin, and Y. Xue. The min-dist location selection query. In ICDE, pages 366–377, 2012.
21. Y. Qiao and G. von Bochmann. Load balancing in peer-to-peer systems using a diffusive approach. Computing, pages 649–678, 2012.
22. X. Quan, L. Wenyin, W. Dou, H. Xiong, and Y. Ge. Link graph analysis for business site selection. Computer, 45(3):64–69, 2012.
23. C. S. Revelle, H. A. Eiselt, and M. S. Daskin. A bibliography for some fundamental problem categories in discrete location science. European Journal of Operational Research, 184(3):817–848, 2008.
24. Y. Sun, J. Huang, Y. Chen, X. Du, and R. Zhang. Top-k most incremental location selection with capacity constraint. In WAIM, pages 165–171, 2012.
25. Y. Sun, J. Huang, Y. Chen, R. Zhang, and X. Du. Location selection for utility maximization with capacity constraints. In CIKM, pages 2154–2158, 2012.
26. Y. Tao, W. Lin, and X. Xiao. Minimal mapreduce algorithms. In SIGMOD, 2013 to appear.
27. L. H. U, K. Mouratidis, M. L. Yiu, and N. Mamoulis. Optimal matching between spatial datasets under capacity constraints. TODS, 35(2):9:1–9:44, 2010.

28. R. C.-W. Wong, M. T. Özsu, A. W.-C. Fu, P. S. Yu, L. Liu, and Y. Liu. Maximizing bichromatic reverse nearest neighbor for l p -norm in two- and three-dimensional spaces. VLDB J., 20(6):893–919, 2011.
29. R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. On efficient spatial matching. In VLDB, pages 579–590, 2007.
30. T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In VLDB, pages 946–957, 2005.
31. D. Yan, R. C.-W. Wong, and W. Ng. Efficient methods for finding influential locations with adaptive grids. In CIKM, pages 1475–1484, 2011.
32. C. Yu, R. Zhang, Y. Huang, and H. Xiong. High-dimensional knn joins with incremental updates. GeoInformatica, 14(1):55–82, 2010.
33. J. Yuan, Y. Zheng, and X. Xie. Discovering regions of different functions in a city using human mobility and pois. In KDD, pages 186–194, 2012.
34. L. Zhan, Y. Zhang, W. Zhang, and X. Lin. Finding top k most influential spatial facilities over uncertain objects. In CIKM, pages 922–931, 2012.
35. D. Zhang, Y. Du, T. Xia, and Y. Tao. Progressive computation of the min-dist optimal-location query. In VLDB, pages 643–654, 2006.
36. R. Zhang, H. V. Jagadish, B. T. Dai, and K. Ramamohanarao. Optimized algorithms for predictive range and knn queries on moving objects. Inf. Syst., 35(8):911–932, 2010.
37. K. Zheng, Z. Huang, A. zhou, and X. Zhou. Discovering the most influential sites over uncertain data: A rank-based approach. TKDE, 24(12):2156–2169, 2012.
38. Z. Zhou, W. Wu, X. Li, M. L. Lee, and W. Hsu. Maxfirst for maxbrknn. In ICDE, pages 828–839, 2011.

Author/s:
Sun, Y;Qi, J;Zhang, R;Chen, Y;Du, X

Title:
MapReduce based location selection algorithm for utility maximization with capacity constraints

Date:
2015-04

Citation:
Sun, Y., Qi, J., Zhang, R., Chen, Y. & Du, X. (2015). MapReduce based location selection algorithm for utility maximization with capacity constraints. COMPUTING, 97 (4), pp.403-423. https://doi.org/10.1007/s00607-013-0382-5.

Persistent Link:
http://hdl.handle.net/11343/282917