# Scalable Spatial Crowdsourcing: A study of distributed algorithms

Abdullah Alfarrarjeh [#1], Tobias Emrich [*2], Cyrus Shahabi [#3]

[#] *Dept. of Computer Science, University of Southern California, USA*

[1] alfarrar@usc.edu, [3] shahabi@usc.edu

[*] *Dept. of Computer Science, Ludwig Maximilian Univ. of Munich, Germany*

[2] emrich@dbs.ifi.lmu.de

***Abstract*— Recently spatial crowdsourcing was introduced as a natural extension to traditional crowdsourcing allowing for tasks to have a geospatial component, i.e., a task can only be performed if a worker is physically present at the location of the task. The problem of assigning spatial tasks to workers in a spatial crowdsourcing system can be formulated as a weighted bipartite *b*-matching graph problem that can be solved optimally by existing methods for the minimum cost maximum flow problem. However, these methods are still too complex to run repeatedly for an online system, especially when the number of incoming workers and tasks increases. Hence, we propose a class of approaches that utilizes an online partitioning method to reduce the problem space across a set of cloud servers to construct independent bipartite graphs and solve the assignment problem in parallel. Our approaches solve the spatial task assignment approximately but competitive to the exact solution. We experimentally verify that our approximate approaches outperform the centralized and MapReduce version of the exact approach with acceptable accuracy and thus suitable for online spatial crowdsourcing at scale.**

***Keywords—distributed spatial task assignment; spatial task assignment; online partitioning; spatial crowdsouring***

## I. INTRODUCTION

In recent years, crowdsourcing has become popular in many research communities (e.g., databases [11], image processing [12][13] and NLP [14]) and is commercially used in the industry (e.g. Amazon Mechanical Turk [7], CrowdFlower [8], CrowdCloud [9] and MicroWorkers [10]). The concept of crowdsourcing refers to outsourcing tasks to a set of people, known as the workers. Tasks in a crowdsourcing system are usually relatively small and require only a moderate effort. A typical task involves translating an article from one language to another, verifying the information written on a business card, or labelling the content of an image.

Due to the ubiquity of smartphones, the workers nowadays are no longer bound to their personal computers to perform tasks but can carry a device with an enormous sensing capability enabling them to perform their tasks at any location. This technical achievement allows for the natural extension of traditional crowdsourcing to what is known as *spatial crowdsourcing* [1] where tasks are additionally specified by a location. A worker is hence required to be physically present at a task location in order to perform the corresponding task. An example of a spatial task is to take a picture of a particular building or capture a video of a concert. In these scenarios, smartphones are used to perform the spatial tasks (e.g., capturing a picture, recording a video or audio) and simultaneously sense additional information (e.g., location, time, direction, speed and acceleration).

According to the taxonomy of spatial crowdsourcing introduced in [1], there are two modes for assigning tasks to workers: *worker-assigned mode* and *server-assigned mode*. In the worker-assigned mode, a worker chooses the best spatial tasks autonomously in his/her vicinity with no coordination with the server or other workers— thus each worker tries to maximize his assignments while minimizing the cost. In the server-assigned mode, the assignment is performed by the spatial crowdsourcing server (SC-server). In this mode, the SC-Server coordinates between all of the workers and tasks in order to globally maximize the number of assigned tasks while minimizing the total cost for the workers. In this paper, we focus on the server-assigned mode.

From the viewpoint of the requester of a set of tasks, the main goal is to have all of the tasks be performed in a short amount of time. This consideration brings the problem of the assignment of current tasks to available workers in a way that maximizes the throughput. Although recent studies [1][2][17] have presented techniques to solve this problem, they focused on the maximization of the assigned tasks rather than the runtime of the assignment in a large-scale setting. Thus, these studies only considered a rather small number of tasks and available workers at a time so that the assignment can usually be performed on a single machine. Generally, the spatial task assignment problem is represented as a weighted bipartite graph and solved as a weighted *b*-matching problem. The latter problem can be converted to a minimum-cost maximum flow problem, which is solved by finding the cheapest augmenting paths successively in a weighted network flow graph (i.e., a generalization of the Ford Fulkerson Algorithm) [1]. In our case, weights (corresponding to the costs associated with the tasks) are non-negative and the graph is acyclic, so the assignment can be performed in $O((m + n \log n) n C)$ time where $n$ is the number of nodes in the graph, $m$ is the number of edges, and $C$ denotes the maximum edge capacity [30]. In this work we aim at a high throughput environment (large number of tasks and workers arriving in a short amount of time). However, considering the above runtime complexity, when the number of incoming tasks and workers grows, the assignment of workers and tasks takes a long time on a single machine.

Let us consider an exemplary environment executing the assignment periodically (e.g., every 5 minutes). During this time suppose 4,000 workers and 16,000 spatial tasks arrive with the average amount of tasks a worker can choose from (candidate tasks) to be 65. The created graph thus contains 20,000 vertices (i.e., total number of tasks and workers) and 280,000 (16,000+4,000+4000*65) edges. Using a machine running CentOS 6.3 with 7.5GB memory and 2 CPUs, the optimal assignment takes 28.18 minutes, meaning that the assignment cannot even finish within the duration of one time period. In this setting, the graph requires roughly 1.14 GB memory space and the testing machine has enough memory capacity. The bottleneck in such scenarios is thus the computation rather than the memory overhead. Another example, when there are 5,000 workers and 10,000 tasks and the average number of candidate tasks per worker is 1844, the assignment takes around 12 hours and 45 minutes on our simulation system. These examples show the inefficiency of a single server environment in which the assignment is performed periodically. One approach to address this is that instead of running the optimal assignment periodically, we could perform an online assignment algorithm [18], where the assignment is executed immediately once a task or a worker arrives. The online assignment yields a faster response time but the total number of assigned tasks would drop dramatically [30][31]. Hence, the need for an adaptive and scalable period-based spatial crowdsourcing solution arises.

An analysis of usage data of popular crowdsourcing platforms shows that the above numbers actually underestimate the immense need for large scale and high throughput environments. Based on Mechanical Turk Tracker [26], which captures the market of Amazon Mechanical Turk every few minutes, the average number of arrived tasks per day is around 300K from January-August 2014. On the other hand, the total number of available crowdsourcing workers per year in a sample of 26 crowdsourcing service providers is 1.34M, 3.1M, and 6.29M in 2009, 2010, and 2011, respectively, [27] which are approximately 3.67K, 8.49K, and 17.23K per day, respectively. These statistics show the adaptation of crowdsourcing systems in terms of the number of engaged workers and received tasks and can be foreseen that these numbers will increase dramatically in the future.

One obvious approach to cope with this large-scale requirement is to solve the assignment problem, represented as graph-matching, in a MapReduce (MR) environment. In [4] the authors proposed a MapReduce version of the maximum flow algorithm that can be used here to solve the assignment problem. The MR solution constructs the bipartite graph first and then partitions it across a cluster of servers and intercommunicates to generate the final assignment output. Thus, the MR solution uses what we call a "*construct first-partition later*" strategy. Our approaches, however, follow an opposite strategy, namely "*partition first – construct later*". That is, we perform online partitioning on the incoming spatial tasks and workers across a cluster of cloud servers, and then each server constructs a partial bipartite graph between its workers and tasks and finally the servers communicate later to generate the final output. Though the proposed approaches return an approximate assignment, we show that the output is competitive to the exact solution yielding a significant runtime improvement and is in contrast to the basic approaches applicable in large-scale real-time scenarios.

The online partitioning can be performed randomly or spatially. We can either partition tasks, workers or both. When one of them is not partitioned, it should be replicated. These permutations generate various algorithms that we evaluate and compare experimentally based on the optimality of the assignment output and the execution time. We also compare our approaches with the centralized and the MR-based solutions. The results show that as compared to the exact approach, in a cluster of four servers, our approaches can achieve a task assignment percentage up to 0.99 and 20 times faster while MR-Based approach with the same system setting achieved a speedup by a factor 2. To the best of our knowledge, we are the first to study spatial crowdsourcing in a distributed setting in order to scale up the task assignment process.

The remainder of this paper is organized as follows. Section II introduces a set of preliminaries in spatial crowdsourcing and a task assignment framework suitable for executing periodically. In Section III, we discuss our proposed distributed approaches for the task assignment problem in spatial crowdsourcing. Section IV presents the experimental results. In Section V, we review the related work. Finally, in Section VI, we conclude and discuss future work.

## II. SPATIAL CROWD SOURCING

### A. Background

In this section, we review the concept of spatial crowdsourcing and its formal definitions and its goals [1].

DEFINITION 1 (SPATIAL TASK): A spatial task is a query ($q$) received at time $s$ to be performed at a location ($l$) where $l$ is a 2D point specified by longitude and latitude values. So, a spatial task ($t$) is denoted by the triplet $<q, l, s>$.

DEFINITION 2 (SPATIAL WORKER): A spatial worker is a person who is located at a location ($l$) and is willing to perform spatial tasks. Each worker is specified by two constraints: the work region ($R$), which characterizes the area in which the worker is willing to perform tasks, and the maximum number of tasks ($maxT$) that he is able to perform. Thus, a spatial worker ($w$) is denoted by the triplet $<l, R, maxT>$.

Since spatial tasks and spatial workers arrive at a SC-Server in a continuous manner, at any given time there is a set of workers that are willing to perform tasks and a set of tasks that have not been performed. Based on these two sets, the goal of the SC-server is to perform efficient and effective spatial task assignment.

DEFINITION 3 (SPATIAL TASK ASSIGNMENT): Given a set of spatial tasks ($t_1, t_2 \ldots, t_n$) and a set of workers ($w_1, w_2 \ldots, w_m$), the SC-Server generates a set of assignment pairs $<w_i, t_j>$, assigning each spatial task ($t_j$) to at most one worker ($w_i$) while satisfying worker constraints. Consequently, if $t_j$ is

assigned to $w_i$, $t_j$ must be in the spatial region $R$ of the worker $w_i$. In addition, if $w_i$ is assigned to $k$ tasks, then $k \leq w_i.maxT$.

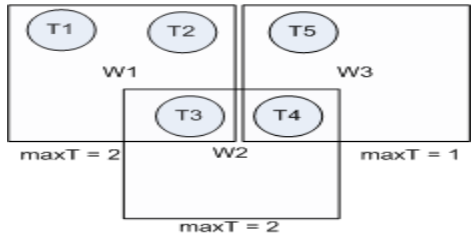
When running the task assignment process, the SC-Server aims to maximize the total number of assignments at a certain time instance ($S_i$). The priority of assigning a task to a worker includes several factors such as the travel cost, the remaining time of the task expiration, worker reputation, or the compatibility of worker skills with the requested task. Choosing one of these factors or all of them to compute the priority is based on the application goal. With spatial crowdsourcing, the travel cost is a critical factor since workers should physically go to the location of spatial task in order to perform the task. Hence, the SC-Server targets to maximize the overall task assignment at every time instance while minimizing the travel cost of the workers. We define the travel cost between a worker $w_i$ to a spatial task $t_j$ in terms of the Euclidean distance.

A weighted bipartite graph $G$ ($U$, $V$, $E$) is a graph whose vertices are divided into two disjoint sets ($U$ and $V$) such that each edge ($u_i$, $v_j$) $\in E$ connects a vertex $u_i \in U$ with a vertex $v_j \in V$ and is associated with a weight $f(u_i, vj)$. A $b$-matching of $G$ is a set ($M$) of $E$ where the degree of each vertex in $M$ is at most $b$. Kazemi and Shahabi [1] showed that the spatial task assignment problem can be formulated as the minimum-weight maximum $b$-matching problem based on a weighted bipartite graph. Therefore each worker ($w_i$) is represented by a vertex $w_i \in U$ and each task ($t_j$) is represented by a vertex $t_j \in V$. Regarding $b$ constraint of each vertex, $b(w_i)$ is $w_i.maxT$ and $b(t_j)$ is 1. An edge ($w_i$, $t_j$) $\in E$ represents a possible assignment between the task $t_j$ and the worker $w_i$. Thus the edge ($w_i$, $t_j$) is existent if the spatial task $t_i$ is in the region of the worker $w_j$ ($t_j.l \in w_i.R$). In addition, each edge is associated with a weight, computed via a function $f(w_i, t_j)$. Although there are several factors to define the weight function $f(w_i, t_j)$ for the edges among tasks and workers, we employ only the Euclidean distance between the worker location ($w_i.l$) and the task location ($t_j.l$). The goal is to maximize |M| where the total weight of these edges is minimized.

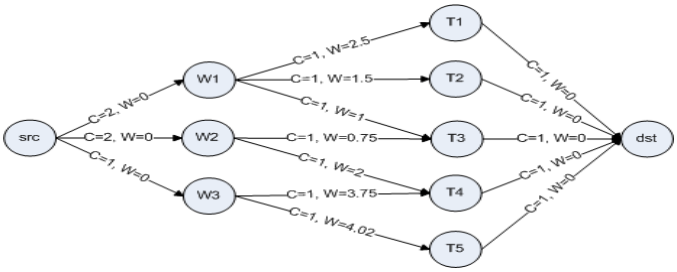
The minimum-weight maximum $b$-matching problem can be solved in different techniques such as the augmenting path algorithm or linear programming. By re-formulating the weighted bipartite graph as weighed flow network graph, the augmenting path algorithm is used to find the cheapest augmenting paths successively. It is a generalized version of the Ford Fulkerson algorithm [6] to compute the maximum flow with minimum cost. Alternatively, linear programming [28] can be used for representing all constraints and objective function (i.e., minimizing the total weight of the matching pairs) in a linear program. The simplex algorithm is the classical method for solving linear programs.

In [1], the spatial task assignment is solved using minimum-cost maximum flow [25]. For this purpose, the weighted bipartite graph $G(U, V, E)$ is reformulated in another weighted flow network graph $G'(V', E')$. The set $V'$ contains |U| + |V| + 2 vertices including two virtual vertices: the source (*src*) and the destination (*dst*). The set $E'$ contains all edges in $E$ and additional edges (*src*, $w_i$), which connect the source vertex with each worker vertex, and edges ($t_j$, *dst*), which connect each task vertex with the destination vertex. The weight of these new edges (i.e., $f$ (*src*, $w_i$) and $f$ ($t_j$, *dst*)) is 0. In flow network graphs, each edge is associated with a capacity value. The capacity of each edge connecting the source vertex with a worker vertex ($w_i$) is the value $w_i.maxT$ while the capacity of other edges is 1. The graph $G'$ ($V'$, $E'$) is termed as the *spatial crowdsourcing graph* (*SC-G*).

*Example: Figure 1 illustrates an example of creating a spatial crowdsourcing graph (SC-G) for a set of spatial workers and tasks. Figure 1.a depicts three workers and five tasks in some geographical area and each worker ($w_i$) is associated with his work region ($w_i.R$) and the maximum number of tasks ($w_i.maxT$) that he is able to perform. The corresponding SC-G is illustrated in Figure 1.b. As shown in the figure, {$t_1.l$, $t_2.l$, $t_3.l$} $\in w_1.R$ so there are three edges connecting $w_1$ with the tasks {$t_1$, $t_2$, $t_3$}. Because $w_1.maxT = 2$, the capacity value of the edge (src, $w_1$) is 2 and the weight of this edge is 0. The weight of edges ($w_i$, $t_j$) is computed via $f(w_i, t_j)$ and the capacity of these edges is 1. The weight of edges ($t_j$, dst) is 0 and the capacity of these edges is 1, as each task may be assigned to at most one worker.*




a) Example of spatial tasks and workers




b) Example of Spatial Crowdsourcing Graph

Fig. 1. An example of spatial crowdsourcing

### *B. Period-based Task Assignment Framework*

In reality the tasks and workers arrive continuously at a SC-Server. This raises the problem of when to run the spatial task assignment. The two extreme approaches are known as *offline* and *online* based assignment. The offline assignment process is executed after receiving all of the tasks and workers while the online assignment [18] is executed once a task or a worker arrives to the system. The offline assignment is optimal in terms of assignment cost minimization as it is performed based on a global knowledge of all tasks and workers [19] [20]; however it is not feasible in a real-time environment as it poses considerable delay for assigning tasks. On the contrary, online assignment supports immediate responses; however the assignment optimality is lowered. Since both of these

approaches have their characteristic drawbacks we envision a hybrid framework which balances between offline and online assignment. In our framework, the system waits for the arrival of tasks and workers for a certain period of time then executes the assignments. This process is repeated periodically. The aim is to support quick responses to tasks and local optimality for the assignment executed in each time period.

Thus, we discretise the time into intervals ($I_0$, $I_1$, $I_2$…, $I_n$). The framework performs the task assignment process at the beginning of each interval (i.e., $I_k$) and considers spatial tasks and workers received during $I_{k-1}$ in addition to the tasks that have not yet been assigned.

At each time interval $I_k$ the system maintains a spatial crowdsourcing graph *SC-G*$_k$ which will be used in the next interval and hence includes all spatial tasks and workers received during $I_k$. At the arrival of any spatial task or worker, the *SC-G*$_k$ is updated immediately. For example, at the arrival of a spatial task ($t_j$), the system searches for all workers whose work regions include the task location (i.e., $\forall w_i : t_j.l \in w_i.R$) and adds edges between this new task vertex and the vertices of the selected workers. Similarly, the graph is updated at the arrival of a new worker.

At the beginning of each time interval $I_k$, the framework creates a new spatial crowdsourcing graph (*SC-G*$_k$) in order to receive new spatial tasks and workers. It also executes the task assignment process on the graph of the previous interval *SC-G*$_{k-1}$. Based on the task assignment output, the unassigned tasks and workers are added *SC-G*$_k$ to be considered in the next task assignment process. Thus, the task assignment process initiated at the beginning of the time interval $I_{k+1}$ works on *SC-G*$_k$, which includes the tasks and workers received in $I_k$ and all of the unassigned tasks and workers during $I_{k-1}$. *Figure 2* depicts the system timeline in the intervals $I_{k-1}$, $I_k$, and $I_{k+1}$.

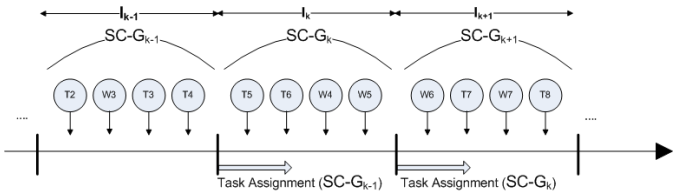



Fig. 2. Spatial Crowdsourcing System Time Line

### C. MapReduce-based Solution

MapReduce (MR) is a programming model which runs a distributed computation on a cluster of servers and it is a framework for processing a large-scale data [21]. In [4], the maximum flow problem in a network flow graph is studied using MapReduce. This approach is based on the Ford Fulkerson algorithm [6] and it uses a multiple MapReduce rounds of bi-directional search technique from the source and the sink vertices to find multiple augmenting paths concurrently. We adopted this approach to solve the weighted maximum flow problem and hence our spatial task assignment problem. We name this method the *MapReduce-based Approach* (*MR-A*).

A MapReduce program is mainly composed of two functions: *map* and *reduce*. A *map* function performs filtering and sorting and a *reduce* function performs a summary operation. The input of MapReduce should be independent set of records consisting of <*key*, *value*> pairs and this allows partitioning records across servers. The MapReduce framework has its own file system which is known as *Distributed File System* (*DFS*). Thus, before executing a MapReduce job the input should be transferred to *DFS* and the output should also be collected from *DFS*.

Although a MR-based approach is able to scale the problem and provides an exact assignment output compared with the centralized system, it has some drawbacks. First, it requires collecting spatial tasks, workers and finding out the candidacy relations between them (i.e., constructing the bipartite graph), then transferring the constructed graph to the MR distributed file system to be partitioned across servers (i.e., *construct first – partition later*). In contrast, our approaches partition graph information across servers in an online-fashion. Second, it poses a high volume of intercommunication when partitioning data across several parallel Map functions running on multiple servers and re-arranging data before running several parallel reduce functions to generate the final result. Third, the input records (*i.e., bipartite graph*) must be transferred to *DFS* and the output should be retrieved from *DFS*. Thus, these constraints slow down the execution of this approach.

In this approach, we have a control server (C-Server) which stores all incoming tasks and workers at each time interval and updates *SC-G*. At the beginning of a new interval ($I_k$), the C-Server transfers the data of *SC-G*$_{k-1}$ to *DFS* and then a MapReduce job is executed. This job partitions the data of *SC-G*$_{k-1}$ across a cluster of servers and performs the assignment process. The C-Server retrieves the assignment output from *DFS* to recognize the unassigned tasks and workers in *SC-G*$_{k-1}$ and add them into *SC-G*$_k$ to be considered in the next assignment process.

## III. DISTRIBUTED SPATIAL CROWDSOURCING APPROACHES

In this section, we will discuss approaches for period-based task assignment in a distributed environment. In this environment, there are multiple servers dedicated to a spatial crowdsourcing system whose main task is to distribute the task assignment process and scale up the overall system performance.

### A. General Setting

From the set of available servers we select one to be the partitioning server (*P-Server*), which continuously receives and partitions the incoming spatial tasks and workers across the other servers which are named matching servers (*M-Servers*). The P-Server is also responsible for syncing and discretising the system time into discrete intervals. At the beginning of each time interval, the P-Server sends parallel requests to all M-Servers to perform the task assignment based on the available information.

Each M-Server maintains its own *SC-G* and continuously receives task and worker information from the P-Server. At the arrival of a new task or worker, the *SC-G* is updated

immediately. At the beginning of each time interval $I_k$ (when a matching request from the P-Server is received), each M-Server simultaneously

- performs the task assignment process locally based on SC-$G_{k-1}$,and
- creates a new *SC-*$G_k$ to append new tasks or workers received in the interval $I_k$.

Our approaches mainly differ by the distribution of spatial workers and tasks across the servers and are named accordingly. We propose three approaches, which are afterwards compared with a *MapReduce-based* approach.

### *B. Random Task Partitioning with Worker Replication Approach (RTP-WR-A)*

The main idea of the first approach is to send incoming workers to all M-Servers whereas the incoming tasks are partitioned across M-Servers. Since workers are replicated among M-Servers and each M-Server performs the task assignment individually on its own SC-G, across the results of M-Servers there are some workers assigned to a number of tasks greater than their capacity (i.e., $w_i.maxT$). The assignments for these overloaded workers are resolved at a synchronizing server (S-Server) which generates the final assignment output as shown in *Figure 3*. Because each M-Server executes the assignment for a subset of tasks and some tasks might be un-assigned from overloaded workers at the S-Server, the final assignment output might miss some assignments compared to the centralized system.

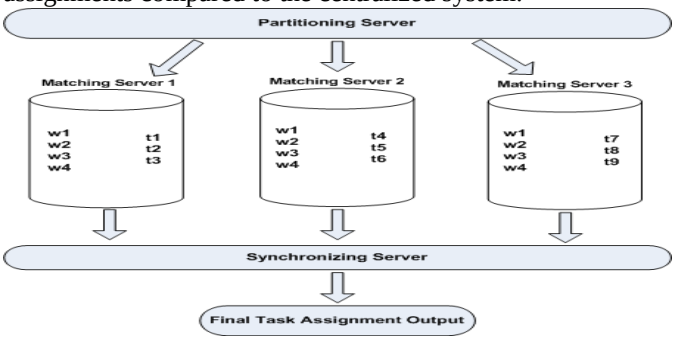



Fig. 3. Random Task Partitioning with Worker Replication Approach

At the P-Server, a round-robin mechanism is used to balance the distribution of tasks among M-Servers. At the beginning of a new time interval $I_k$, the P-Server checks the status of all M-Servers and the S-Server. If all of them are idle (i.e. not involved in a task assignment process), the P-Server sends a request to all M-Servers to execute the assignment process. If one of the servers is busy, the P-Server postpones all assignment requests to the beginning of the next time interval $I_{k+1}$.

At any time instance, all M-Servers "know" the same set of workers but disjoint sets of tasks. Formally, let $W_x$ be the set of workers and $T_x$ be the set of tasks in M-Server $MS_x$. Then any time instance for any two M-Server $MS_i$, $MS_j$, $T_i = T_j$ and $W_i \cap W_j = \emptyset$. When the request of the P-Server to start the assignment arrives at an M-Server, the task assignment is performed on its local *SC-*$G_{k-1}$ and the status of the server is set to busy. After finishing the local assignment the result is sent to the S-Server. The distributed workflow is illustrated in *Figure 4*. Once the final assignment is generated in the S-Server, it is transferred to all M-Servers. Every M-Server then adds the unassigned tasks and workers into *SC-*$G_k$ to be considered in the next assignment process and the server status is set to idle again.

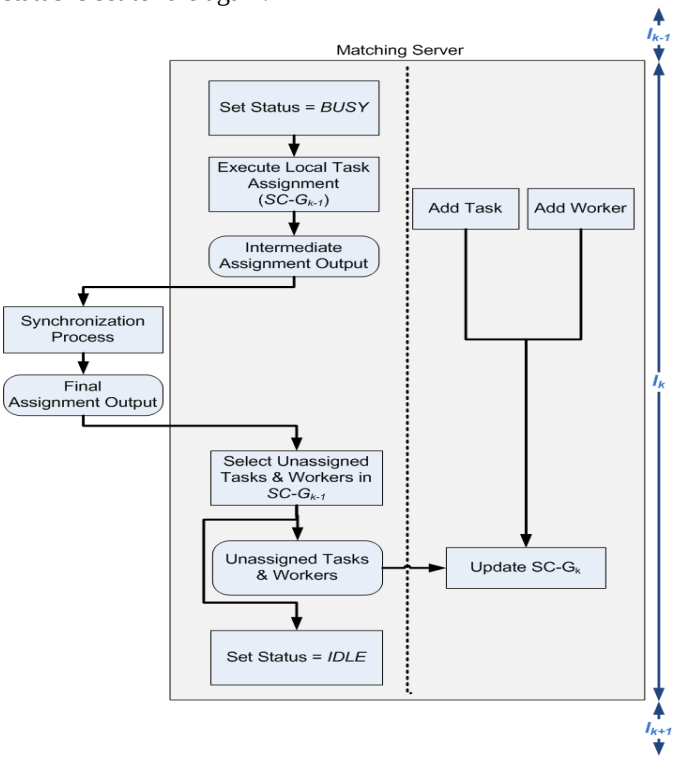



Fig. 4. Parallel Processes at M-Server *i* at a time interval $I_k$

The S-Server becomes busy when it receives an intermediate assignment result from any M-Server. It stores the intermediate results in a hash table which maps each worker to his assignment pairs received from different M-Servers. When all M-Servers have sent their intermediate results, the S-Server scans through the hash table. If a worker $w_i$ is overloaded, the S-Server chooses the best tasks based on the assignment weight criteria (i.e. the travel distance) and un-assigns the other tasks. Then, the final task assignment output is pushed concurrently to all M-Servers to update their local information. Afterwards, the S-Server's status is set back to idle.

*Example: An example is shown in Figure 3, where* $w_1$ *is replicated from the P-Server to all three M-Servers:* $MS_1$, $MS_2$, *and* $MS_3$. *The set* $\{t_1, t_2, t_3\}$ *belongs to* $MS_1$, *the set* $\{t_4, t_5, t_6\}$ *belongs to* $MS_2$, *and the set* $\{t_7, t_8, t_9\}$ *belongs to* $MS_3$. *Let us assume the local task assignment in M-Servers results assigning* $w_1$ *to* $\{t_1, t_2, t_4, t_7\}$. *Given that* $w_1.maxT=2$ *and* $f(w_1, t_7)<f(w_1, t_1)<f(w_1, t_2)< f(w_1, t_4)$, *the S-Server un-assigns* $t_2$ *and* $t_4$ *from* $w_1$ *and* $w_1$ *remains assigned to* $\{t_1,t_7\}$. *Obviously the assignment process may produce non-optimal assignments, i.e., if a task gets unassigned it is not checked if it could be assigned to another worker instead. Consequently the number of assigned tasks may be smaller than in the centralized approach.*

With this approach, assuming that the number of available tasks is greater than the number of available workers, we partition the set of tasks across M-Servers. Instead of partitioning tasks, we could also partition workers (i.e., *Random Worker Partitioning with Task Replication Approach*, *RWP-TR-A*) and adapt the assignment strategy in the S-Server. However, our experimental evaluation reveals that partitioning tasks is better in term of the number of assignments and their total cost over subsequent time intervals.

### C. Spatial Partitioning Approach (SP-A)

In *RTP-WR-A*, workers are replicated to all M-Servers. This results in overloaded workers which have to be resolved at the S-Server creating a bottleneck due to the need for waiting all intermediate results before proceeding to resolve overloading assignments. To eliminate the need of an S-Server, we propose the *Spatial Partitioning Approach* (*SP-A*) which partitions both tasks and workers based on their locations. The main idea of this approach is that each M-Server is assigned to a certain geographical region and if a task or worker is located in the geographical region of an M-Server, that M-Server receives the task or the worker. This partitioning is motivated by the fact that a worker is assigned to a task if they are located in roughly the same geographical region. Thus, spatial tasks and workers located in a specific geographical region should be processed at the same M-Server.

In this approach, the global geographical area is partitioned into disjointed sub-areas by using space partitioning techniques such as a uniform grid, a quad tree, a kd-Tree [23], or a Voronoi diagram [24]. Let us note that the technique to partitioning the space may influence the performance of system, yet this investigation is out of the scope of this paper. In the following, we use a spatial uniform grid to partition the global geographical area into equal sub-areas across M-Servers. Thus, there is no overlap between the geographical regions of M-Servers. In other words, if the M-Server ($MS_i$) is assigned to a geographical region $MS_i.R$ and there is another M-Server $MS_j$, $MS_i.R \cap MS_j.R = \emptyset$.

The P-Server is aware of the geographical regions of all M-Servers. When a spatial worker $w_i$ arrives and its location belongs to the geographic region of an M-Server $m$ (i.e., $w_i.l \in MS_m.R$), the P-Server sends a request to $MS_m$ to store $w_i$. Similarly, when a spatial task $t_j$ is received and $t_j.l \in MS_m.R$, $t_j$ is stored in $MS_m$.

*Example: Figure 5 shows an example of partitioning a certain geographical area into three sub-areas with their corresponding M-Servers. Spatial tasks and workers are partitioned across the three M-Servers. For example, $t_9$ is processed in $MS_2$ because $t_9.l \in MS_2.R$ and $w_6$ is sent to $MS_3$ because $w_6.l \in MS_3.R$. The work regions of some workers might overlap with the geographical regions of multiple servers thus the P-Server partitions workers based on their actual location. For instance, $w_4.R$ crosses $MS_2.R$ and $MS_3.R$ but $w_4.l \in MS_2.R$, so $w_4$ is added into $MS_2$. Figure 6 shows the general architecture of this approach and depicts the distribution of the sample of tasks and workers presented in Figure 5.*

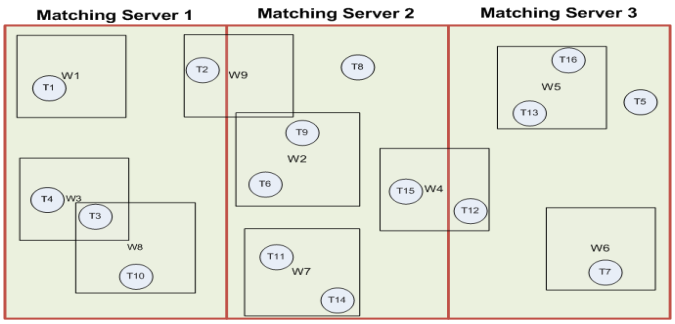



Figure 5: Example of partitioning geographical area across M-Servers

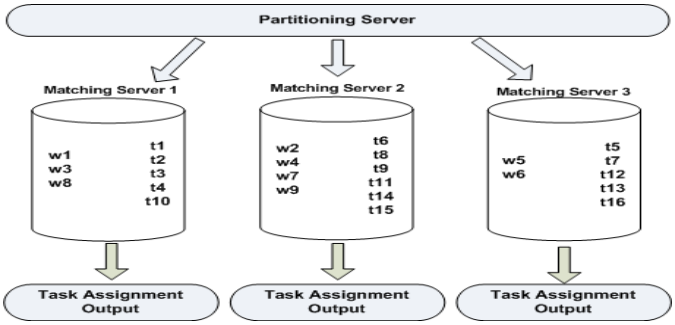



Fig. 6. Spatial Partitioning Approach

At any time instance, all M-Servers have disjointed sets of tasks and workers. Let $W_x$, be the set of workers and $T_x$ be the set of tasks in an M-Server $x$ ($MS_x$). Then any time instance for any two M-Server $MS_i$, $MS_j$, $T_i \cap T_j = \emptyset$ and $W_i \cap W_j = \emptyset$. At the beginning of each time interval $I_k$, each M-Server receives a request from the P-Server to start executing the local task assignment process. The start of task assignment in each M-Server is independent from the other M-Servers where the assignment is performed if the server is idle without considering the status of other servers as in *RTP-WR-A*. When the assignment is finished in each M-Server, the server updates its own *SC-G* without communicating with the other servers. Because there is no overlap in the distribution of tasks and workers across M-Servers, the resulted assignment pairs over M-Servers are independent. Hence, there is no need for an S-Server.

In contrast to this approach (*SP-A*), tasks and worker can be partitioned randomly without using the spatial information (*i.e., Random Partitioning Approach, RP-A*). Round-robin can be used to balance the distribution of tasks and workers. Experimentally, we will show that this approach produces the worst result. Spatial partitioning increases the effectivity of task assignment at each M-Server.

With *SP-A*, the assignment output is approximate since a worker whose work area overlaps the geographical areas of multiple servers is processed by one of them and assigned to local tasks which may not be optimal if assigning to the nearby tasks located in the neighbouring servers is better.

### D. Spatial Partitioning with Worker Replication Approach (SP-WR-A)

With *RTP-WR-A*, workers are replicated across all M-Servers which results in overloading-assigned workers so an

S-Server is required. This is avoided in *SP-A* by partitioning both tasks and workers, but disjointed subsets of workers across M-Servers prevent workers to be assigned to tasks stored in other servers. On the other hand, in *SP-A* tasks are partitioned spatially but in *RTP-WR-A* tasks are partitioned in round-robin. Spatial partitioning improves the effectivity of local task assignment in each M-Server. In *Spatial Partitioning with Worker Replication Approach* (*SPWR-A*), we propose a hybrid mechanism which spatially partitions both tasks and workers but some workers are replicated when necessary. Tasks are partitioned based on their location but workers are partitioned based on their work regions. If the worker region overlaps with the geographical area of multiple M-Servers, the worker is replicated to all of them. Otherwise, the worker is sent only to one of them. Because of the possibility of worker replication, an S-Server is required to resolve overloading workers. The architecture of this approach is shown in *Figure 7*. Generally, the architecture of this approach is similar to *RTP-WR-A*, but the functionalities of the P-Server and the S-Server are different.

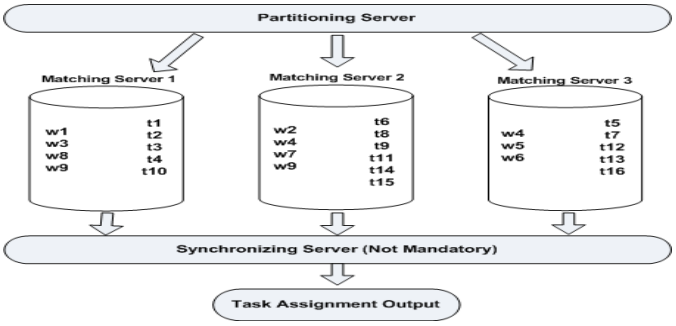



Fig. 7. Spatial Partitioning with Worker Replication Approach

*Example: The distribution of tasks and workers in Figure 7 is based on the sample set of tasks and workers given in Figure 5. For example, $w_4$.R overlaps $MS_2$.R and $MS_3$.R so the P-Server replicates $w_4$ to $MS_2$ and $MS_3$ and $w_4$ is marked as replicated.*

As in *SP-A*, the global geographical area is partitioned into disjointed sub-areas and each is assigned to an M-Server. The P-Server is aware about the geographic partitioning among M-Servers and it is used for partitioning both tasks and workers spatially. As in *RTP-WR-A*, at the beginning of a new time interval $I_k$ the P-Server checks the status of all M-Servers and the S-Server. If all of them are idle (*i.e. not involved in a task assignment process*), the P-Server sends a request to all M-Servers to execute the assignment process. If one of the servers is busy, the P-Server postpones all assignment requests to the beginning of the next time interval $I_{k+1}$.

At each M-Server, the assignment process is executed. The assignment output is categorized into two sets: *final* and *intermediate*. If an assignment pair does not contain a replicated worker, the assignment pair $<w_i, t_j>$ is categorized as final. Otherwise, the pair is added to the intermediate assignment result. The intermediate assignment result is sent to the S-Server because it includes replicated workers which may be assigned to other tasks in the other M-Servers. But, the final set of assignments is not sent to the S-Server.

With *SP-WR-A*, the S-Server performs a lighter job compared with *RTP-WR-A*. It does not receive all assignments generated from M-Servers but only the ones which contain replicated workers. When the final assignment output is generated at the S-Server, it is pushed concurrently to all M-Servers to update their local information.

The result of this approach is also approximate because tasks which are assigned to the replicated workers in M-Servers might be un-assigned in the S-Server without considering alternative workers. In this approach, the synchronization job is minimized compared with *RTP-WR-A* so the execution time of the assignment process for the entire system is improved.

## IV. Experiments

We conducted our experiments on the Amazon EC2 infrastructure. We used a varying number of machines (4, 8, or 16) each running on 64 bit CentOS 6.3 Linux OS with 2 virtual CPUs, 7.5GB memory, and 30GB disk storage. These machines were connected via a 128 Mbps network. For each approach we implemented the different servers to carry out the functionalities of the P-Server, M-Servers and S-Server using Java 1.7 as programming language. In addition, Apache Hadoop was installed across all machines as MapReduce infrastructure [22]. Additionally, we used a dedicated machine which provides the approaches with task and worker information in a continuous manner, simulating a stream. Below, we first discuss the experimental methodology and then present our results.

### A. Experimental Methodology

We conducted several experiments on real-world and synthetic datasets to evaluate the performance and the effectivity of our approaches. Real data was obtained from Gowalla and Brightkite [32], two location-based social networks where users shared their locations through check-ins. The Gowalla data was collected over the period of Feb. 2009 to Oct. 2010 and the Brightkite data was collected from Apr. 2008 to Oct. 2010. We subsequently extracted the data covering the area of California State. Each user record of a dataset was mapped to a worker $w_i$. Since each user $w_i$ had various check-ins, we set $w_i.maxT$ to the number of check-ins of $w_i$ and $w_i.R$ as the minimum bounding rectangle of those checked-in locations. Additionally, a user check-in at a certain location $l$ represents a task $t_j$ with location $l$. For the experiments on synthetic data, we randomly generated location data of workers and tasks drawn from a uniform distribution or a Gaussian mixture model. In the latter case the dataset was formed into 10 Gaussian clusters where the centers were randomly chosen from a uniform distribution. We randomly selected the value of *maxT* between 1 to 5 and the value of the spatial region $R$ was in the range of [6.9, 69] miles.

Specifically, we conducted our experiments using the following four datasets: In *S1-SYN-UD*, tasks and workers were uniformly distributed. *S2-SYN-SD* demonstrates a skewed data distribution where workers were distributed

uniformly but tasks were distributed normally. The third dataset *S3-SYN-SD* is similar to *S2-SYN-SD* but the Gaussian centers of workers were chosen close to the geographical borders of M-Server areas. In the last dataset *S4-RD*, workers were obtained from the Brightkite data and tasks were obtained from the Gowalla data.

In the experiments, we compared the performance of our approximate-solution approaches (*RTP-WR-A*, *SP-A*, and *SP-WR-A*) with the exact-solution approaches, Centralized-System Approach (*CS-A*) and MapReduce-based Approach (*MR-A*). Additionally, we evaluated two variations of the proposed approaches: *RWP-TR-A* which is a variation of *RTP-WR-A* and *RP-A* which is a variation of *SP-A*. We evaluated all approaches based on three metrics: $M_1$) *the execution time speed-up of the approach compared to CS-A*, $M_2$) *the percentage of assigned tasks compared with the exact solution*, and $M_3$) *the percentage of assignment cost per assignment pair compared to the exact solution* where $M_3 = \frac{(\sum_{X-A}\text{assignment cost}/\sum_{X-A}\text{assigned tasks})}{(\sum_{CS-A}\text{assignment cost}/\sum_{CS-A}\text{assigned tasks})}$, *X-A*∈ {*MR-A*, *RWP-TR-A*, *SP-A*, *SP-WR-A*, *RTP-WR-A*, *RP-A*}. Since *MR-A* and *CS-A* are the ground truth when calculating the metrics $M_2$ and $M_3$, their values were 1.0.

By default the length of a time interval (period) was set to 1 minute, but was extended when the running assignment was not completed.

### B. The effect of Arrival Rate

For every dataset (*S1-SYN-UD*, *S2-SYN-SD*, *S3-SYN-SD*, or *S4-RD*), we performed three experiments in which we varied the arrival rate of tasks and workers per time interval at the spatial crowdsourcing system. For the synthesized datasets, the received number of tasks per interval was 8K, 9K, or 10K while the number of workers was 2K, 3K or 5K correspondingly. Because of the limited number of real data, the received number of tasks was 1000, 1200, or 1500 while the number of workers was 200, 400, or 500. The approaches ran through 10 time intervals. We extracted the average of the three metrics (i.e., execution time speed up, assignment ratio and assignment cost ratio) over the three experiments for each dataset. In this set of experiments all distributed approaches used four M-Servers.

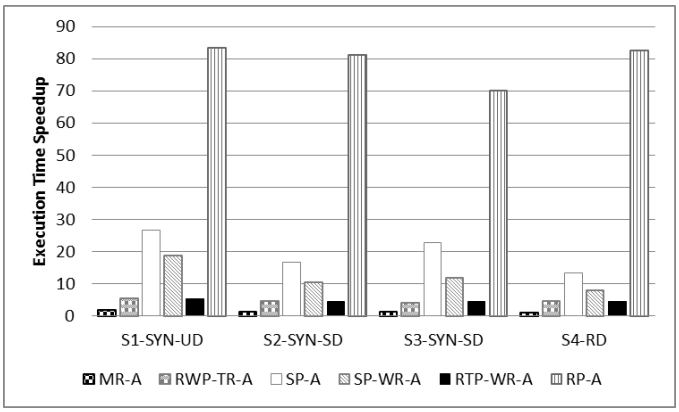



Fig. 8: Avg. of Execution Time Speedup over 10 Time Intervals in Four Datasets with Three Arrival Rates

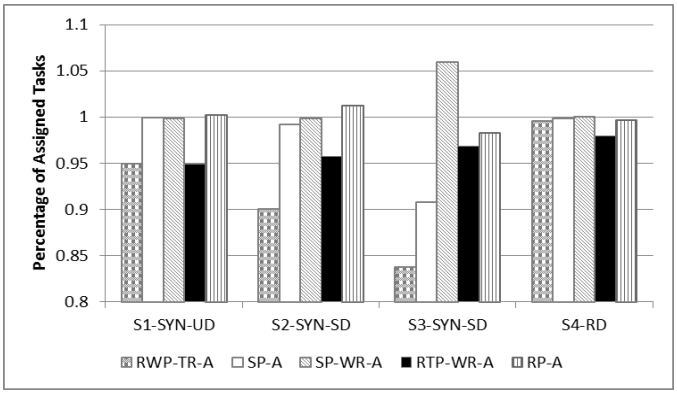



Fig. 9: Avg. of Percentage of Assigned Tasks compared with CS-A or MR-A over 10 Time Intervals in Four Datasets with Three Arrival Rates

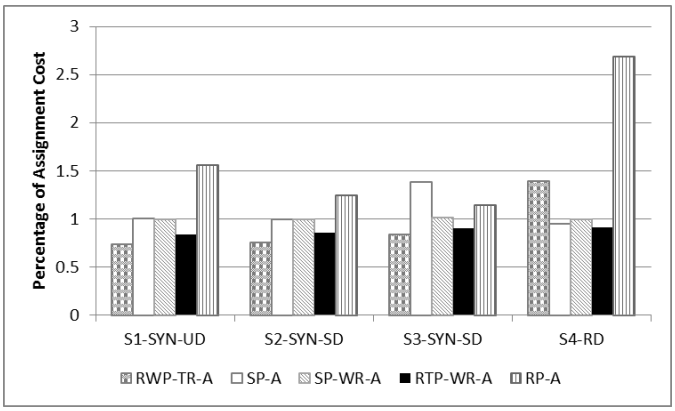



Fig. 10: Avg. of Percentage of Assignment Cost per Assignment Pair over 10 Time Intervals in Four Datasets with Three Arrival Rates

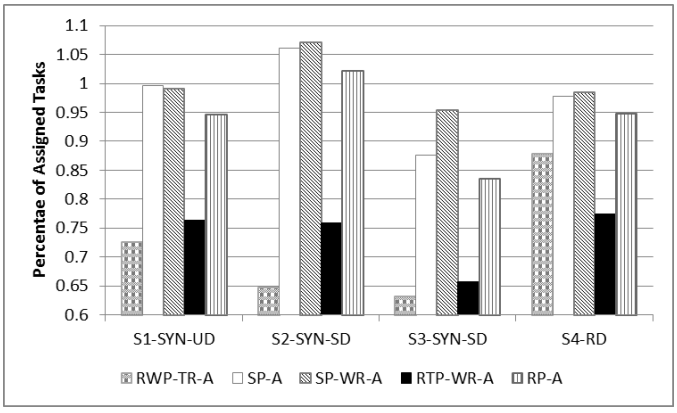



Fig. 11: Avg. of Percentage of Assigned Tasks in the 1st Time Interval in Four Datasets with Three Arrival Rates

First we show the efficiency and effectivity of the distributed approaches (*MR-A*, *RWP-TR-A*, *SP-A*, *SP-WR-A*, *RTP-WR-A*, and *RP-A*) compared with the centralized approach (*CS-A*) in terms of the execution time speed-up (*Figure 8*), the percentage of task assignment (*Figure 9*) and their cost along 10 time intervals (*Figure 10*). It is clearly visible that our approximate approaches achieved a very high speed-up. In addition, we noticed that on one hand the average number of assignments was nearly the same for most of the approaches in the four datasets and on the other hand they were different in the assignment cost. Most of the proposed

approaches were able to achieve more than 95% (sometimes even 105%) of the assignments of the centralized approach (*CS-A*). This can be attributed to two reasons: a) the high rate of arriving tasks and workers which provides a plethora of alternatives when executing the assignment and b) running the system over consecutive time intervals gives the system chances to re-assign tasks, which were un-assigned in the S-Server, to their best matches among old or new-arrived workers.

*RP-A* was much faster than *CS-A*. Even though it partitions tasks and workers randomly across M-Servers, it was able to generate a number of assignments almost similar to *CS-A* but increasing the cost of *CS-A* up to three times. This renders this approach infeasible to maintain a comparable assignment cost percentage to *CS-A*. On the other hand, the distributed solution offered by *MR-A* only provided a marginal speed-up.

With *RWP-TR-A* and *RTP-WR-A*, the average of $M_1$ for both approaches was 4.6, which shows that the two approximate approaches did not provide a good improvement in terms of the execution time. The random partitioning and the existence of an S-Server hinder the system achieving a high speed-up. In the two datasets (*S1-SYN-UD* and *S4-RD*), $M_2$ was similar for both approaches but in the other datasets *RTP-WR-A* outperformed *RWP-TR-A*. This shows that the un-assignments occurred at the S-Server of *RWP-TR-A* were higher than the un-assignments at *RTP-WR-A*. In terms of $M_3$, *RTP-WR-A* displayed a stable behaviour while *RWP-TR-A* performed differently in different datasets. However in *S4-RD* and *S1-SYN-UD*, $M_2$ of *RTP-WR-A* was almost equivalent to *CS-A* and $M_3$ of *RTP-WR-A* became 30% higher than *CS-A* in *S4-RD* while it was significantly less in *S1-SYN-UD*. The latter dataset indicates that *RTP-WR-A* utilized the new arriving tasks and workers in addition to the old un-assigned tasks and workers for choosing the least-cost assignments.

*SP-A* scored the second place in terms of $M_1$ after *RP-A*. Even though both *SP-A* and *RP-A* partition tasks and workers, there was a large difference in $M_1$. This is a result of two reasons a) in *RP-A* the random partitioning in round robin manner results in a balanced load between M-Servers but the spatial partitioning in *SP-A* yields imbalanced load, b) $M_1$ was calculated as the average of execution time for all M-Servers. In terms of $M_1$, *SP-WR-A* was the third after *SP-A* due to the existence of an S-Server. *SP-WR-A* was much faster than *RWP-TR-A* and *RTP-WR-A* because the replication of workers is only performed when necessary. The average of $M_2$ and $M_3$ of *SP-A* and *SP-WR-A* were nearly similar to *CS-A* except in *S3-SYN-SD* in which $M_2$ of *SP-A* decreased by 9% with a 30% higher cost (i.e. $M_3$) than *CS-A*. This displays the main drawback of *SP-A* when most of worker areas overlap with multiple M-Servers. On the other hand, $M_2$ and $M_3$ of *SP-WR-A* were almost similar to *CS-A* in *S3-SYN-SD*.

*Figure 11* shows $M_2$ of the approaches in the first time interval compared with *CS-A*. Note that some approaches achieved a low assignment percentage. For example, in *S2-SYN-SD*, $M_2$ of *RWP-TR-A* and *RTP-WR-A* was 0.65 and 0.76 in the first time interval but they increased to 0.90 and 0.95, respectively, after 10 time intervals.

### C. The effect of Number of M-Servers

We varied the number of M-Servers (4, 8 or 16) to analyse the behaviour of the approximate approaches when scaling the number of servers. *Figures 12* and *13* compare the average of $M_1$ and $M_2$, respectively, of *MR-A*, *SP-A*, *SP-WR-A* and *RTP-WR-A* in different cluster settings. The results were calculated over 10 time intervals for a set of experiments using all datasets with multiple arrival rates as discussed in the previous subsection.

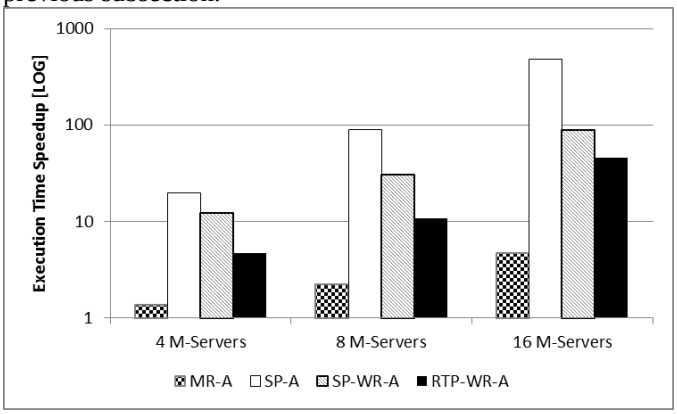



Fig. 12: Avg. of Execution Time Speedup of *MR-A*, *SP-A*, *SP-WR-A* and *RTP-WR-A* in different clusters (4, 8, or 16 servers)

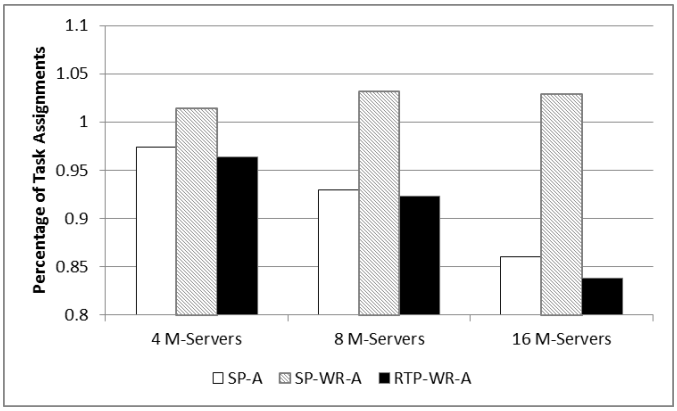



Fig. 13: Avg. of Assigned Tasks for *SP-A*, *SP-WR-A* and *RTP-WR-A* in different clusters (4, 8, or 16 servers)

*Figure 12* shows $M_1$ in logarithmic scale. When increasing the number of servers, $M_1$ of *SP-A* increased exponentially while increased linearly for *SP-WR-A*. $M_1$ of *SP-A* was 20, 89, and 480 for the number of servers 4, 8, and 16. *SP-WR-A* occupied the second place with the $M_1$ values 12, 31, and 89. *RTP-WR-A* scored the third place with 5, 11, and 46 displaying a small change when scaling the system. This marginal increase is due to having an S-Server whose job increases when adding an M-Server. On the other hand, *MR-A* showed the smallest speedup factor when scaling the cluster of servers due to the increase of intercommunication between servers when partitioning input data and merging intermediate data.

In *Figure 13*, we notice that *SP-WR-A* was able to maintain a high $M_2$ comparable to *CS-A* regardless of the number of M-Servers. On other hand, the effectivity of *SP-A* and *RTP-WR-A* decreased when the number of servers increased. *SP-A*

achieved 0.97, 0.93, and 0.86 of $M_2$ when the number of M-Servers was 4, 8, and 16, respectively. While running *SP-A* on 16 server in *S3-SYN-SD*, $M_2$ reached to 0.75 which displays the infectivity of this approach when most of the workers existing around the borders of M-Servers and having a large number of M-Servers. *RTP-WR-A* scored 0.96, 0.92, and 0.83 of $M_2$ when the number of M-Servers was 4, 8, and 16, respectively. This shows that the un-assignments happening in the S-Server in *RTP-WR-A* increases as the number of M-Servers increases.

## V. Related Work

Crowdsourcing attracted researchers from different communities [11][12][13][14] and commercially used in the industry [7][8][9][10]. Few studies focused on spatial crowdsourcing [15] [16]. In [15], a crowdsourcing platform was proposed which utilizes location as a parameter to distribute tasks among workers. In [16], their crowdsourcing platform employed a location-based service (e.g., Foursquare) to find the appropriate people to answer a query given over Twitter but it did not assign to workers any spatial task, for which the worker should go to that location and perform the corresponding task.

The taxonomy of spatial crowdsourcing and the problem of spatial task assignment were studied in [1] and the issue of trust in spatial crowdsourcing was studied in [2]. Both [1] and [2] studied the problem in server-assigned mode. Instead of server mode, the worker-assigned mode in spatial crowdsourcing was explored in [3] and the proposed techniques considered the dynamic location of workers while travelling to tasks. In [3], the shortest path to other spatial tasks was computed from the worker location at the current visited task so the task assignment problem was solved differently and mapped to the Travel Salesman Problem. The current existing work focuses on the task assignment problem without considering a scalable solution.

Boutsis et al [17] proposed a crowdsourcing platform to assign appropriate workers to tasks to increase the probability of getting on-time completed tasks and high-quality responses to tasks but the location parameter was utilized implicitly. For scalability, workers and tasks located in the same geographical area were processed in the same server which is similar to our second approach (*SP-A*) but location parameter was not considered in the assignment. In our scalable approaches, spatial partitioning is used because of the prominent constraints for spatial tasks and workers. The spatial task assignment considers travel distance however it can be extended naturally to other parameters such as task deadline and worker expertise.

Spatial task assignment is mapped to a weighted b-matching problem which can be converted to a weighted maximum flow. Scaling the problem in MapReduce is possible [4][5]. In [4], the maximum flow problem in a network flow graph was studied in MapReduce, where an exact-solution approach was designed. In [5], two approximate algorithms were proposed to solve *b*-matching problem in MapReduce. We compared our approximate approaches with the exact MR-based approach to study the effectivity of our approximate solutions.

The auction algorithm [35] is a variant of task assignment problem where workers (e.g., robotic agents) assign themselves to tasks while attempting to maximize a certain collective benefit function. It solves the assignment in decentralized fashion (i.e., worker-assigned) but our spatial task assignment is based on a server-assigned strategy where the information about workers and tasks are collectively managed by servers. In [36] the auction algorithm is extended to deal with spatial tasks and mobile workers. Beside the difference in the task assignment strategy, they did not focus on improving the performance of the task assignment to be feasible in an online large-scale setting.

Partitioning a graph is a common problem which is mainly categorized as offline or online. Offline partitioning requires full graph information to distribute the graph. Spectral clustering algorithm [34] is an example of offline partitioning which results with optimal graph partitioning but lengthy execution time. Online partitioning aims to find a near-optimal solution by distributing vertices and edges with only limited graph information. There are usually two approaches in online partitioning: edge cut or vertex cut. PowerGraph [33] is an online partitioning framework which employs several vertex-cut algorithms to provide edge balanced partitions. Online and offline partitioning require knowledge about the graph structure but our distributed approaches partition the problem information without constructing the graph itself.

## VI. Conclusion

In this study we considered the problem of spatial crowdsourcing in a distributed environment. We showed that a single centralized system is not capable to cope with a rising number of workers and tasks arriving in short amount of time. We thus evaluated several approaches that differ in the distribution and handling of the incoming data. In a cluster of 16 servers, the most efficient of these approaches were shown to be up to 88 times faster while yielding comparable effectivity to the centralized approach. In contrast, a Map Reduce based implementation could only achieve a speedup by a factor 5. To the best of our knowledge this is the first study of suitable approaches for the problem of distributed spatial crowdsourcing.

As future work, we plan to study different space partitioning techniques for assigning the geographical areas of the matching servers. We also plan to study the theoretical bounds of our approximated solutions. Moreover, we plan to incorporate the historical information of the previous partitions and task assignments in our distributed approaches.

## Acknowledgment

This research has been supported in part by NSF grants IIS-1320149 and CNS-1461963, the USC Integrated Media Systems Center, and unrestricted cash gifts from Google and Northrop Grumman. The opinions, findings, and conclusions or recommendations expressed in this material are those of the

authors and do not necessarily reflect the views of any of the sponsors such as NSF.

## REFERENCES


[1] L.Kazemi and C. Shahabi. "Geocrowd: enabling query answering with spatial crowdsourcing." In *SIGSPATIAL*, pp. 189-198. ACM, 2012.

[2] L. Kazemi, C. Shahabi, and L. Chen. "GeoTruCrowd: trustworthy query answering with spatial crowdsourcing." In *SIGSPATIAL*, pp. 304-313. ACM, 2013.

[3] D. Dingxiong, C. Shahabi, and U. Demiryurek. "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing." In *SIGSPATIAL*, pp. 314-323. ACM, 2013.

[4] H. Felix, R. H. Yap, and Y. Wu. "A Mapreduce-based maximum-flow algorithm for large small-world network graphs." In *ICDCS*, pp. 192-202. IEEE, 2011.

[5] G. D. F. Morales, A. Gionis, and M. Sozio. "Social content matching in mapreduce." In *VLDB* Endowment 4, no. 7 (2011): 460-469.

[6] L. R. Ford and D. R. Fulkerson. "Maximal flow through a network." Canadian journal of Mathematics 8, no. 3 (1956): 399-404.

[7] "Amazon Mechanical Turk", http://www.mturk.com/

[8] "CrowdFlower", http://www.crowdflower.com/

[9] "CrowdCloud", http://www.crowdcloud.com/

[10] "MicroWorkers", http://microworkers.com/

[11] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. "CrowdDB: answering queries with crowdsourcing." In *SIGMOD*, pp. 61-72. ACM, 2011.

[12] K.-T. Chen, C.-C. Wu, Y.-C. Chang and C.-L. Lei. "A crowdsourceable QoE evaluation framework for multimedia content." In *MM'09*, pp. 491-500. ACM, 2009.

[13] A. Sorokin and D. Forsyth. "Utility data annotation with amazon mechanical turk." Urbana 51, no. 61 (2008): 820.

[14] R. Snow, B. O'Connor, D. Jurafsky, and A. Y. Ng. "Cheap and fast---but is it good?: evaluating non-expert annotations for natural language tasks." In *EMNLP*, pp. 254-263. Association for Computational Linguistics, 2008.

[15] F. Alt, A. S. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. "Location-based crowdsourcing: extending crowdsourcing to the real world." In *NordiCHI*, pp. 13-22. ACM, 2010.

[16] M. F. Bulut, Y. S. Yilmaz, and M. Demirbas. "Crowdsourcing location-based queries." In *PERCOM Workshops*, pp. 513-518. IEEE, 2011.

[17] I. Boutsis and V. Kalogeraki. "On task assignment for real-time reliable crowdsourcing." In *ICDCS*, pp. 1-10. IEEE, 2014.

[18] C.-J. Ho and J. W. Vaughan. "Online Task Assignment in Crowdsourcing Markets." In *AAAI*. 2012.

[19] R. C.-W. Wong, Y. Tao, A. W.-C. Fu, and X. Xiao. "*On efficient spatial matching*." In *VLDB* Endowment, 2007: 579-590

[20] M. L. Yiu, K. Mouratidis, and N. Mamoulis. "Capacity constrained assignment in spatial databases." In SIGMOD, pp. 15-28. ACM, 2008.

[21] J. Dean, and S. Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51, no. 1 (2008):107-113.

[22] "Apache Hadoop", http://hadoop.apache.org/

[23] H. Samet. Foundations of multidimensional and metric data structures. Morgan Kaufmann, 2006.

[24] F. Aurenhammer. "Voronoi diagrams—a survey of a fundamental geometric data structure." In *CSUR* 23, no. 3 (1991):345-405.

[25] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. "Network flows: theory, algorithms, and applications." (1993).

[26] "mturk tracker", http://www.mturk-tracker.com/

[27] "The Crowd in the Cloud: Exploring the Future of Outsourcing", http://www.lionbridge.com/files/2012/11/Lionbridge-White-Paper_The-Crowd-in-the-Cloud-final.pdf

[28] V. Chvatal. "Linear programming". Macmillan, 1983.

[29] T. Brunsch, K. Cornelissen, B. Manthey, and H. Röglin. "Smoothed analysis of the successive shortest path algorithm." In SODA, pp. 1180-1189. ACM SIAM, 2013.

[30] A. Mehta and D. Panigrahi. "Online matching with stochastic rewards." In FOCS, pp. 728-737. IEEE, 2012.

[31] A. Mehta "Online matching and ad allocation." Theoretical Computer Science 8, no. 4 (2012): 265-368.

[32] J. Leskovec, and A. Krevl. "Stanford large network dataset collection." http://snap.stanford.edu/data/index.html (2014).

[33] J. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs." In OSDI, vol. 12, no. 1, p. 2. 2012.

[34] A. Ng, M. I. Jordan, and Y. Weiss. "On spectral clustering: Analysis and an algorithm." Advances in neural information processing systems 2 (2002): 849-856

[35] D. Bertsekas, and David A. Castañon. "Parallel synchronous and asynchronous implementations of the auction algorithm." Parallel Computing 17, no. 6 (1991): 707-732.

[36] B. Moore, and K. Passino. "Distributed task assignment for mobile agents." Automatic Control, IEEE Transactions on 52, no. 4 (2007): 749-753.