Missouri University of Science and Technology
Scholars' Mine

Computer Science Faculty Research & Creative Works

Computer Science

04 Jan 2023

Scalable Skill-Oriented Task Allocation in Crowdsourcing within a Serverless Ecosystem

Biswajeet Sethi

Riya Samanta

Soumya K. Ghosh

Sajal K. Das
Missouri University of Science and Technology, sdas@mst.edu

Follow this and additional works at: https://scholarsmine.mst.edu/comsci_facwork

Part of the Computer Sciences Commons

Recommended Citation

B. Sethi et al., "Scalable Skill-Oriented Task Allocation in Crowdsourcing within a Serverless Ecosystem," *ACM International Conference Proceeding Series*, pp. 135 - 139, Association for Computing Machinery (ACM), Jan 2023.
The definitive version is available at https://doi.org/10.1145/3571306.3571399

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Computer Science Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. This work is protected by U. S. Copyright Law. Unauthorized use including reproduction for redistribution requires the permission of the copyright holder. For more information, please contact scholarsmine@mst.edu.

Scalable Skill-oriented Task Allocation in Crowdsourcing within a Serverless Ecosystem

Biswajeet Sethi*
Indian Institute of Technology Kharagpur, India
biswajeet.sethi@iitkgp.ac.in

Riya Samanta*
Indian Institute of Technology Kharagpur, India
riya.samanta@iitkgp.ac.in

Soumya K. Ghosh
Indian Institute of Technology Kharagpur, India
skg@cse.iitkgp.ac.in

Sajal K. Das
Missouri Univ. of Science and Technology, Rolla, USA
sdas@mst.edu

ABSTRACT

Allocating the most competent crowdworkers to each upcoming task is a fundamental challenge in crowdsourcing. The mechanism becomes complicated when the arriving tasks require a high level of expertise within a constrained budget. The validation of skill matching between tasks and crowdworkers adds a new dimension to the traditional problem of task allocation. In addition, in real-world scenarios, the influx of both tasks and workers is dynamic, making it nearly impossible to predict the precise amount of computational resources required for the crowdsourcing platform to operate efficiently. Serverless computing is a new pay-per-use, auto-scalable, Function-as-a-Service based model, that ensures parallel execution of lightweight event-driven functions. The developer with serverless can solely concentrate on writing application logic with zero effort on resource provision, server management, environmental configuration, and availability. Today, collaboration has become the new competition. In light of these considerations, we propose a novel framework to facilitate task allocation strategies for crowdsourcing applications, deployed within a serverless platform in order to improve performance. The results obtained are compared to the baseline, Online-Greedy, and simulations are run in both serverless and local environments.

CCS CONCEPTS

• **Information systems** → **Crowdsourcing**; • **Computing methodologies**; • **Human-centered computing** → *Empirical studies in collaborative and social computing*; • **Cloud computing**;

KEYWORDS

serverless computing, crowdsourcing, skill-oriented task allocation

ACM Reference Format:
Biswajeet Sethi, Riya Samanta, Soumya K. Ghosh, and Sajal K. Das. 2023. Scalable Skill-oriented Task Allocation in Crowdsourcing within a Serverless Ecosystem. In *24th International Conference on Distributed Computing and Networking (ICDCN 2023), January 4–7, 2023, Kharagpur, India.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3571306.3571399

*Both authors contributed equally

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
ICDCN 2023, January 4–7, 2023, Kharagpur, India
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-9796-4/23/01…$15.00
https://doi.org/10.1145/3571306.3571399

1 INTRODUCTION

Crowdsourcing (CS) has emerged as a distributed rational solution and a business-making replica. CS is a model for collaborative networked resources to accomplish a task in this connected world. Today, when data is available from the instant owner, CS has made it possible to outsource the task, once a time being performed by in-campus employees, to the nexus through an open call. In any category of CS platform, whether spatial, web-based, or volunteer services, most projects are complex and structural [17], needing several crowd-workers with a variety of skills to collaborate. A software-development project outsourced via a freelancing marketplace is one example (e.g., UpWork, Freelancer, Toptal, etc.).

Thus, in large markets like CS, corporations have recognized the scarcity of highly skilled workers and the intense competition for top talent. This necessitates the current research trends to focus on `skill-based task allocation` in the CS backdrop. Moreover, in most practical scenarios, the influx of tasks and workers is variable or dynamic, depending on factors such as working hours, technology demands, the type of available tasks, crowd-workers, etc. Hence, CS platforms strive to facilitate the on-demand and scalable distribution of assignments to human workers worldwide [11]. This makes it hard to forecast the platform's computing needs. In the era of Big Data, scalability and resource provisioning are significant concerns for any CS platform. This is why commercial applications are transitioning to micro-services and containers, leading to serverless computing.

One of the important benefits of using serverless is the availability of infinite auto-scalability, with commanding trends in data-intensive applications. Serverless lets the user execute stateless functions in the form of small chunks of reusable code. It invokes a copy of the function on request with no restriction on the number of concurrent incoming requests for the function. In the event of any sudden prong, the serverless platform is capable of scaling to any stretch with auto-provisioning of resources. It also eliminates the effort the user has to spend on server management and configuration of the runtime. One of the other important reasons behind the popularity of serverless is its pricing model. Serverless is based on a pay-per-use model. The user has to pay only for the active computation and storage that it uses.

The challenge today is to make the much-desired CS platforms more performance efficient and automated. By deploying CS applications in a serverless ecosystem, we could take advantage of all the basic benefits of serverless, such as auto-scalability, auto-provision

Table 1: Comparison of Relevant Prior Work

Authors	Allocation Mode	Scalable ?
Goel et al. (2014) [5]	Offline	×
Cheng et al.(2016)[4]	Offline	×
Liu et al. (2016) [10]	Offline	×
Jarrett et al. (2017) [7]	Offline	✓Response to volume of data
Song et al. (2020) [17]	Online	×
Ni et al. (2020) [12]	Offline	×
Samanta et al. (2021) [14]	Online	×
Liang et al. (2022) [9]	Offline	×
Samanta et al. (2022) [15]	Offline	×
Proposed Approach [THIS PAPER]	**Batch based** (hybrid of online and offline)	**Auto-scalable; Parallel computation; Pay-per-use**

of resources, and pay-per-use. Also, the stateless functions in serverless allow the developers to perform concurrent computations.

Key contributions: (1) We formulate the Task-Worker Mapping (TWM) problem. The tasks under consideration have skill requirements, along with certain definite budgets. Next, we propose Skill-oriented Allocation (SoA) algorithm for assigning crowd-workers to the tasks. (2) We propose a serverless framework to deploy our crowdsourcing application in order to improve its performance in terms of latency. Here, we define latency as the total delay in getting the final result of task-worker mapping. (3) We demonstrated the efficacy of SoA over the existing state-of-the-art method based on latency, success rate and average task waiting time. For simulation, we used a real dataset.

The remaining sections are organised as follows. Section 2 provides a concise summary of the relevant literature. Section 3 explains skill-based task allocation in CS and defines the problem. Section 4 describes the serverless framework proposed for a scalable allocation strategy. The experimental results used to evaluate the performance of the aforementioned framework are presented in Section 5. The last section is the conclusion.

2 RELATED WORK

The task allocation problem in CS has attracted researchers for more than a decade. however, only a few have considered skill orientation in the decision process. In [5], authors considered bounded budgets and non-homogeneous jobs requiring specific skills and designed an incentive-compatible technique using bipartite matching. The authors in [4] find an optimal worker-and-task assignment strategy, such that skills between workers and tasks match with each other, and workers' benefits are maximized under the budget constraint. Another work is of [10] where the authors proposed an approach to managing complex task allocation while taking into account the tasks-workers-skills tripartite graph.

On the other hand, multi-skill-oriented task allocation in online settings is studied by [17] and [14]. The authors of [14] follow a similar plan of action as that of [17], with the addition of a willingness component for complex assignments, along with the workers' skills and the utility of assigned activities. The authors in [12] define dependency-aware spatial CS. The [9] proposed a cost-based greedy approach to minimize CS platform costs by matching a suitable team of workers for spatial tasks under multiple constraints. However, except [17] and [14], all the papers offer offline allocation. The paper [7] proposed a path for CS expansion through interoperability and scaling with no such adhered protocol.

An earlier version of this work has been accepted in *IEEE GLOBECOM 2022* [15]. The concept of the SoA algorithm is being derived from i-VTM algorithm of [15]. Table-1 gives a comparative glimpse of research work conducted in the field of CS.

On the other hand, a plentiful amount of research has been actively going on in the domain of serverless computing. Papers like [6, 9], discuss the fundamental features of serverless computing along with its opportunities and challenges. Additionally, authors in [3, 8] discuss how the scalability of serverless platforms can become the future of the industry.

To the best of our knowledge, no work has ever attempted to add an auto-scalability feature to the CS platform using a serverless backbone. In this work, we tried to overcome this research gap.

3 SKILL-ORIENTED TASK ALLOCATION IN CROWDSOURCING

In this section, we discuss the details of skill-oriented work allocation in CS and relevant formal definitions.

3.1 Batch-based Allocation Strategy

Considering the limitations of online and offline allocation approaches [2], in this work we apply a `batch-based allocation strategy` that is a hybrid of offline and online assignment. We divide the time period into $(X_0, X_1, X_2, ...X_k)$ intervals. At the start of each interval (i.e., X_j), the framework carries out the task allocation process, taking into consideration the number of tasks and crowd-workers received during X_{j-1}. Additionally, the unassigned task and crowd-worker entities, if have not surpassed the time-to-live, are pushed to X_{j+1}.

3.2 Problem Formulation

At the beginning of batch count X_i, a set of tasks T and a set of available crowd-workers C are present at the CS platform for active participation. Every task $t \in T$, has a list of skills requirement S_t and a predefined budget B_t. The task t is supposed to have arrived at time δ_t and has an estimated time-to-live of e_t. After e_t expires, the task is either removed from the system or re-posted. Each crowd-worker $c \in C$ is associated with a list of skills S_c and is supposed to incur R_c fee. Similarly, c is considered to have arrived at time δ_c and is expected to have a time-to-live till e_c. Following the justifications made in the papers [5, 17], we assume that the number of skills that every task necessitates or that a crowd-worker possesses is always part of a specified universal skill set Q with a fixed size n.

For each batch X_i, the problem is to assign crowd-workers to tasks and generate an allocation map without the net remuneration of the selected workers surpassing the total budget sanctioned. Also, the skill requirements of the tasks should be covered. We named this problem as `Task-Worker Mapping Problem`**(TWM)** which is NP-hard and can be proved by a reduction from the `Approximated Subset Sum Optimization Problem`[12, 16].

3.3 Task-Worker Mapping Mechanism

This section describes the SoA algorithm in detail. The concept of *bipartite graph* is utilized in scheming two main data structures, namely Skill-Task Mapper (G_{ST}) and Skill-Worker Mapper (G_{SW}).

Algorithm 1 Skill-oriented Allocation (SoA)

Input: Crowd-worker data-frame C', Skill-Task Mapper matrix G_{ST}
Output: Allocated Map Map', Updated G_{ST}
1: Start
2: $G'_{ST} \leftarrow G_{ST}$
3: Sort C' with respect to remuneration
4: **for all** $c \in C'$ **do**
5: Select worker ω from C' with lowest remuneration
6: $G_{SW} \leftarrow$ Generate Worker-Skill Mapper of ω
7: $col_sum \leftarrow$ Column-wise sum of G_{SW} excluding the last budget row
8: $col_max_sum \leftarrow$ Max(col_sum)
9: $t_{reco} \leftarrow argmax(col_max_sum)$
10: **if** $B_{t_{reco}} \geq R_{\omega}$ **then**
11: $t' \leftarrow t_{reco}$
12: Update $B_{t'}$
13: Add allocation (ω, t') to Map'
14: **end if**
15: Update G'_{ST}
16: **end for**
17: **return** G'_{ST}, Map'
18: End

The G_{ST} is generated by the Lambda functions (L_{task}). Similarly, the proposed SoA algorithm is also deployed in multiple Lambda functions instances, denoted as L_{SoA}. The details related to the serverless deployment are in Section-4.

The G_{ST} is a two-dimensional matrix that is used to store skills per task requirement. To represent the columns, all of the currently available tasks T are combined. Thus, if all of the n tasks in T result in a total of m distinct skills, the size of G_{ST} will be $(m+1) \times n$. The extra last row is dedicated to storing the current budget of the tasks so that the matrix also passes the budget status while being used during the allocation process. To note, the entry (x, y) (for row 1 to m only) is set to one, if and only if a skill x is required by any task y; otherwise, it is set to zero. This G_{ST} is necessary for dynamically tracking of any task's skill requirement coverage.

The second important data structure is G_{SW} which is used for matching each crowd-worker's skills to that of the requirements of tasks. The G_{SW} is also implemented using a two-dimensional matrix. If a crowd-worker has m skills, the G_{SW} for n tasks will be $m \times n$ in size. Like G_{ST}, in G_{SW}, the entry (x, y) is set to one; if and only if skill x is required by task y; otherwise, is set to zero. To eliminate unnecessary skills, any redundant skill possessed by a crowd-worker that is not required by any of the posted tasks is ignored while constructing G_{SW}.

To begin, a local copy of G_{ST} is created and named as G'_{ST}. After sorting C, the crowd-worker asking for the least remuneration is selected (lines 3-5). Then, the worker's respective G_{SW} is constructed, and *column-wise sum* of G_{SW} is manipulated (lines 6-7). Now, the task for which the column-wise sum value is maximum is chosen and temporary stored in t_{reco} (lines 8-9). If the task's budget, $B_{t_{reco}}$ is enough to cover R_{ω} then t_{reco} is designated as the allocated task t' and accordingly the budget is refined (lines 10-12). In the case of multiple tasks qualifying for this condition, one of them is selected randomly. Next, Map' is updated and so as the G'_{ST}. Now, the data structure Map' is called the allocation map. It is implemented in the form of a dictionary having key-value pair, where key is a task name (or ID) and value is a list of tuples. Each tuple has the name (or ID) of a worker and the skill set he or she contributed. Finally, as SoA finishes execution, both Map' and G'_{ST} are returned.

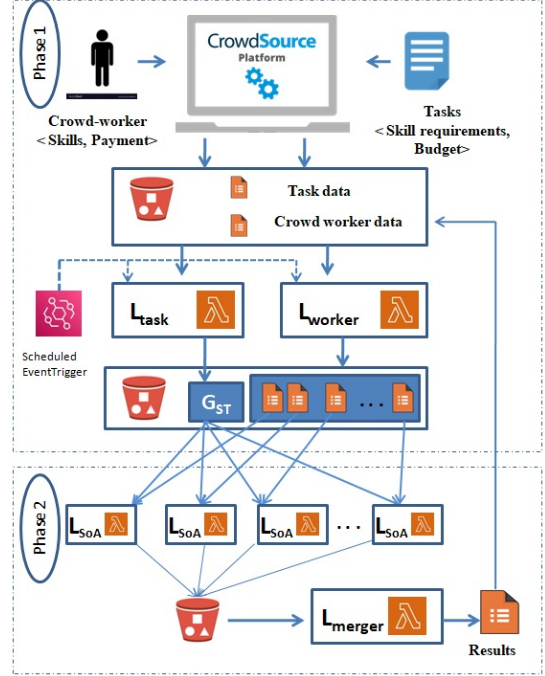


Figure 1: Proposed Serverless Task-Worker Mapping Framework

THEOREM 1. *SoA runs in $O(C \times T)$ time.*

PROOF. Every $t \in T$ is expected to have at least one separate skill need. In T, no two tasks will have exactly the same skill requirements or none at all. In SoA, if C is the set of crowd-workers provided, then its sorting would be in $O(C(log(C))$. Next, in the outer loop of length $|C|$, G_{SW} matrix of size $|S_c| \times |T|$ is formed for each worker in C. The worst case scenario is when a worker's skill set size is the same as S_T. Therefore, creation of G_{SW} takes $O(|T \times S_T|)$ time and column-wise summation also takes $O(|T \times S_T|)$ time. Moreover, finding the argmax is of $O(|T|)$ costs. As a result, the total time complexity of the Skill-oriented Allocation (SoA) Algorithm is $O(C(log(C)) + C \times (2(T \times S_T) + T))$. It is already assumed that the count of skills required for any task or that a crowd-worker holds are always part of a fixed-size universal skill set Q. The SoA algorithm has a time complexity of $O(C(log(C)) + (C \times T))$ which is effectively $O(C \times T)$, where C is the set of participating crowd-workers and T is the set of tasks placed on the CS platform. □

4 PROPOSED SERVERLESS APPROACH FOR TASK ALLOCATION IN CROWDSOURCING

The primary purpose of CS platform is to enable the assignment of workers to suitable tasks. Most of the existing CS platforms are pull-based, in which workers pick the tasks, e.g., Amazon MTurk. In this work, we explored a push-based system [18] in which the platform identifies potential workers and assigns tasks to them. The proposed framework works in two phases, as shown in **Figure-1**. We used AWS Lambda [1] as our serverless computation unit, Amazon Simple Storage Service (Amazon S3) for storage and Amazon Eventbridge for scheduled triggers to functions.

4.1 Phase 1

Phase 1 has the following steps:

- Step 1: The CS platform takes the crowd-worker and task data as input.
- Step 2: The data from the platform is pushed into a `AWS S3` bucket as two different objects.
- Step 3: Using `AWS Amazon EventBridge`, Lambda functions are auto-scheduled, to fetch task and crowd-worker data concurrently at some specified interval(also denoted as batch).
- Step 4: The crowd-worker data is fetched by a `Lambda` function, L_{worker} for required splitting. The splitting mechanism of L_{worker} is based upon horizontal slicing. In parallel the task file is fetched by another `Lambda` function L_{task} for generating the G_{ST}.
- Step 5: The L_{task} processes the task file and produces G_{ST} and pushes it to the AWS buckets. In the mean time, L_{worker} slice the crowd-worker data horizontally to n parts and uploads the partitioned crowd-worker data as objects to `S3`.
  - Each crowd-worker data bucket has been mapped with a corresponding `Lambda` function L_{SoA}.
  - Here, each push of partitioned crowd-worker data to a bucket acts as a trigger to the corresponding `Lambda` function L_{SoA}.

The outcome of phase 1 are G_{ST} and partitioned smaller crowd-worker data set. The main purpose of phase 1 is to pre-process the incoming task and crowd-worker dataset and make it ready for the SoA to act on. The entire task data set is converted into a bipartite graph (G_{ST}). The worker dataset, on the other hand, is split up into many files to allow SoA instances to run concurrently. Moreover, the G_{SW} is actually a sub-graph of the G_{ST}. Thus, after the G_{ST} has loaded for each instance of SoA, it is simple to construct the G_{SW}.

4.2 Phase 2

Phase 2 has the following steps:

- Step 1: Input to each L_{SoA} function is the G_{ST} and one of the sub crowd-worker data set files generated from phase one.
- Step 2: After execution, each `Lambda` function sends the processed partitioned crowd-worker data and modified $G^{'}_{ST}$ as separate objects to a `S3` bucket.
- Step 3: Another `Lambda` function L_{merger} fetches all the objects from the bucket and processes data to produce the final result. The final result is stored back in the original `S3` bucket as in the **Figure-1**.

The main purpose of phase 2 is to perform concurrent execution of SoA in multiple instances and to generate the final result from the intermediate maps.

4.3 Discussion

For each n sliced worker files produced in the first phase, phase 2 of the framework runs n number of instances of L_{SoA} function. This allows concurrent processing of sliced crowd-worker data and G_{ST}. Each instance of the L_{SoA} produces a task allocation map and a modified G_{ST}, giving in total $2n$ outputs. For simplicity, we denote the map generated from k^{th} L_{SoA} function as $Map^{'}_{k}$ and the respective Skill-Task Mapper as $G^{'}_{ST_k}$. Each intermediate map is basically a `dictionary` where the key represents the task and the values represent the respective allocated crowd-workers along with their contributed skills. The L_{merger} then merges all the maps $\{Map^{'}_{1}, Map^{'}_{2}, ...Map'_{n}\}$ to form the penultimate mapper Map_{σ}. For any task assigned with multiple crowd-workers contributing the same skills, the crowd-worker with the lowest remuneration criteria is selected and the rest are unassigned resulting in Map.

On a similar note, a global picture of G_{ST} is also extracted to overview the current status and to remove any redundancy, which could be a frequent possibility [2] in our case as we are replicating the task information in all the L_{SoA} instances in the form of G_{ST}. However, the crowd-workers' information is horizontally sliced in subsequent data sets to reduce latency and to improve processing time by enabling parallel execution of mapping algorithms (i.e., SoA). At the end of the current batch, not only is the Map published to the CS platform, but this Map together with G_{ST}, is resubmitted to phase 1 for consideration in the next batch cycle. This aids in detecting unassigned crowdworkers and unfinished tasks having a certain residual budget, giving them another chance in the upcoming batch cycles if their time-to-live is still valid.

5 PERFORMANCE EVALUATION

5.1 Dataset

For simulation purposes, we employ a real dataset (Meetup) referenced in the work [17]. Here, $|T|$ be the number of tasks which is set to 1234, Q, Universal skill set. There are 554 distinct skills in Q. $|S_t|$ be the number of skills required by each task; varying between 5 and 10. B_t, the mean budget of the task is set to \$428, with a standard deviation of \$255. $|C|$ is the number of crowd-workers which is set to 3275. $|S_c|$, the number of skills that each worker masters, varying between 1 and 5 and R_c be the mean payment incurred by any crowd-worker is set to \$40, with a standard deviation of \$50.

5.2 Experimental Results

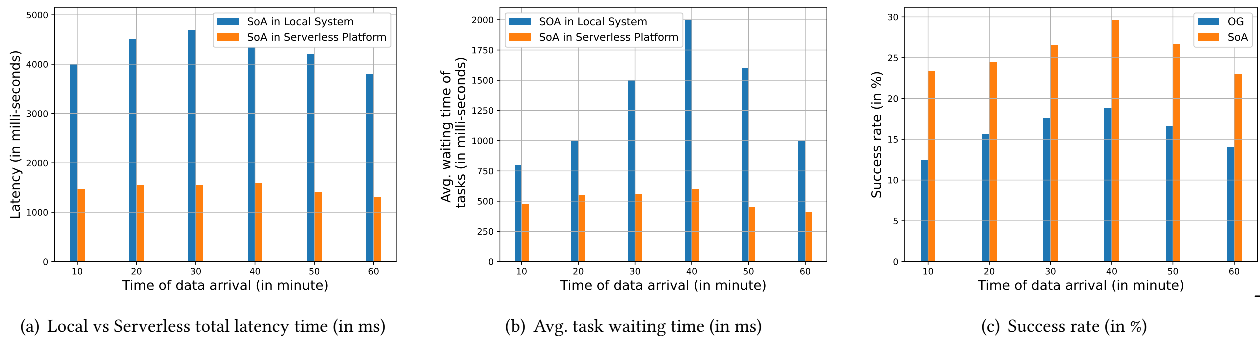
The Online-Greedy (OG) algorithm [17] is considered the baseline for comparing our proposed `SoA` algorithm. It is to be noted that the time complexity of OG is $O((|C| + |T|)^2)$ whereas that of SoA's is $O(|C| \times |T|)$. Hence, inherently, SoA is more efficient than OG. Next, to validate the proposed serverless framework, we executed the SoA algorithm on both the serverless platform and local computer environments. For the latter part, SoA is executed as a single instance at the start of every batch cycle. The task and crowd-worker data is assumed to arrive following the `Normal Distribution`[13] having a mean of 30 minutes and standard deviation of 10 minutes. The batch size is set to 10 minutes. Every single simulation is done for 1 hour, and 100 such simulations are run to get the average for each performance metric.

(1) Local vs Serverless total latency: Figure-2(a)compares the latency observed for executing SoA in both the proposed serverless framework and in a local system. The overall latency, Lat_{sl} in serverless can be represented by the following.

$Lat_{sl} = Lat_{processing} + Lat_{merge}$

where, $Lat_{processing} = Lat_{fetch} + Lat_{split} + Lat_{upl}$

For total latency calculation, we considered $Lat_{processing}$ in three different splits. Lat_{fetch} is the latency to fetch the crowd-worker data and task data to the computation unit from the CS platform. Here, task and crowd-worker data fetching take place concurrently. Hence, we considered the maximum value of both. $Lat_{fetch} = \max(Lat^{task}_{fetch}, Lat^{cw}_{fetch})$ Here, Lat^{task}_{fetch} is the latency to fetch the arriving task data from the CS platform to the serverless ecosystem by L_{task} and Lat^{cw}_{fetch} is the latency to fetch the arriving crowd-worker data from the CS platform to the serverless ecosystem by L_{worker}. Lat_{split} is the time taken to split the crowd-worker data and segregate it to respective S3 buckets. And Lat_{upl} is the sum of processing time at L_{SoA} and latency to upload generated maps to



(a) Local vs Serverless total latency time (in ms) (b) Avg. task waiting time (in ms) (c) Success rate (in %)

Figure 2: (a) total latency and (b) waiting time of SoA in serverless and local settings and (c) success rate for SoA and OG in local setting.

the S3 bucket for further processing. Lat_{merge} is the sum of time to fetch intermediate maps from the S3 bucket, processing time to generate the final maps from the intermediate maps and to upload the final results to the S3 bucket.

In the local system, SoA is executed in a sequential flow as a single instance. The G_{ST} is generated in-house for all tasks received during a given batch and is passed as arguments along with the entire crowd-workers dataset. The output is a single allocation map $Map^{'}$ and a $G^{'}_{ST}$ for every batch cycle. Thus, overall latency, Lat_{loc} in local system is the execution of the algorithm. On comparing the total latency in both the local and serverless ecosystems, we observed that the latency in a serverless platform varies in the range of 1500-1600 ms, whereas locally, the latency was found to be in the range of 4000-5000 ms.

(2) Average task waiting time: Figure-2(b) shows the average time spent by the tasks in the waiting queue before successful allocation. For serverless settings, the average waiting time of a task varied between 400 and 600 ms, whereas for local set-up, the average waiting time was as high as 2000 ms. The incoming batch task for the range of 30 to 40 experiences the longest wait times as tasks and workers arriving during this period are at their maximum.

(3) Success rate: The success rate is the average percentage of completed tasks out of the available tasks on the platform. Figure-2(c) shows that the average success rate of SoA is approximately 25% whereas that of OG is 16%. Thus, SoA performed with an efficiency of **36%** compared to OG.

6 CONCLUSION

Considering performance as a major concern, in this paper, a novel serverless framework to promote CS skill-oriented task allocation is proposed. The adaption of serverless computing has been shown to help run the allocation process in terms of light-weight concurrent stateless functions, leading to a performance of 2.5x better compared to the local implementation. Besides, the core features of the serverless platform have even made the job of developers simple and easier. Further, we modelled the Task-Worker Mapping (TWM) problem and proposed the SoA algorithm as a solution. Evaluating with a real dataset (Meetup), we observed a task allocation success rate of about 9% compared to the baseline. Again compared to local set-up, we achieved an efficiency of 70% in case of average waiting time for task allocation.

REFERENCES

[1] 2022. AWS Lambda - Serverless Compute. https://aws.amazon.com/lambda/. Last accessed 21 June 2022.

[2] Abdullah Alfarrarjeh, Tobias Emrich, and Cyrus Shahabi. 2015. Scalable Spatial Crowdsourcing: A Study of Distributed Algorithms. *Proceedings - IEEE International Conference on Mobile Data Management* 1 (2015), 134–144. https://doi.org/10.1109/MDM.2015.55

[3] Kyle Chard and Ian Foster. 2019. Serverless Science for Simple, Scalable, and Shareable Scholarship. In *2019 15th International Conference on eScience (eScience)*. 432–438. https://doi.org/10.1109/eScience.2019.00056

[4] Peng Cheng, Xiang Lian, Lei Chen, Jinsong Han, and Jizhong Zhao. 2016. Task assignment on multi-skill oriented spatial crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2201–2215. https://doi.org/10.1109/TKDE.2016.2550041 arXiv:1510.03149

[5] Gagan Goel, Afshin Nikzad, and Adish Singla. 2014. Allocating tasks to workers with matching constraints: truthful mechanisms for crowdsourcing markets. In *Proceedings of the 23rd International Conference on World Wide Web*. 279–280.

[6] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[7] Julian Jarrett and M. Brian Blake. 2017. Interoperability and scalability for worker-job matching across crowdsourcing platforms. *Proceedings - 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2017* (2017), 3–8. https://doi.org/10.1109/WETICE.2017.8

[8] Hyungro Lee, Kumar Satyam, and Geoffrey Fox. 2018. Evaluation of production serverless computing environments. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 442–450.

[9] Zhejun Liang, Wenan Tan, Jiu Liu, and Kai Ding. 2022. Multi-skill collaboration-based task assignment in spatial crowdsourcing. In *International Conference on Computer Application and Information Security (ICCAIS 2021)*, Vol. 12260. SPIE, 42–48.

[10] Jiaxu Liu, Haogang Zhu, and Xiao Chen. 2016. Complicated-skills-based task assignment in spatial crowdsourcing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9998 LNCS (2016), 211–223. https://doi.org/10.1007/978-3-319-47121-1_18

[11] Xiaoye Miao, Huanhuan Peng, Yunjun Gao, Zongfu Zhang, and Jianwei Yin. 2022. On Dynamically Pricing Crowdsourcing Tasks. *ACM Transactions on Knowledge Discovery from Data (TKDD)* (2022).

[12] Wangze Ni, Peng Cheng, Lei Chen, and Xuemin Lin. 2020. Task allocation in dependency-aware spatial crowdsourcing. *Proceedings - International Conference on Data Engineering* 2020-April (2020), 985–996. https://doi.org/10.1109/ICDE48307.2020.00090

[13] Jagdish K Patel and Campbell B Read. 1996. *Handbook of the normal distribution*. Vol. 150. CRC Press.

[14] Riya Samanta, Soumya K Ghosh, and Sajal K Das. 2021. SWill-TAC: Skill-oriented Dynamic Task Allocation with Willingness for Complex Job in Crowdsourcing. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–6.

[15] Riya Samanta, Vaibhav Saxena, Soumya Ghosh, and Sajal K. Das. 2022. Volunteer Selection in Collaborative Crowdsourcing with Adaptive Common Working Time Slots. In *2022 IEEE Global Communications Conference: Selected Areas in Communications: Social Networks (Globecom2022 SAC SN)*. Rio de Janeiro, Brazil.

[16] Steven S Skiena. 2020. *The algorithm design manual*. Springer International Publishing.

[17] Tianshu Song, Ke Xu, Jiangneng Li, Yiming Li, and Yongxin Tong. 2020. Multi-skill aware task assignment in real-time spatial crowdsourcing. *GeoInformatica* 24, 1 (2020), 153–173. https://doi.org/10.1007/s10707-019-00351-4

[18] Akash Yadav, Joydeep Chandra, and Ashok Singh Sairam. 2021. A Budget and Deadline Aware Task Assignment Scheme for Crowdsourcing Environment. *IEEE Transactions on Emerging Topics in Computing* 6750, c (2021), 1–14. https://doi.org/10.1109/TETC.2021.3062843