Provisioning Quality-aware Social Compute Units in the Cloud

Muhammad Z.C. Candra, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
m.candra,truong,dustdar@dsg.tuwien.ac.at

Abstract. To date, on-demand provisioning models of human-based services in the cloud are mainly used to deal with simple human tasks solvable by individual compute units (ICU). In this paper, we propose a framework allowing the provisioning of a group of people as an execution service unit, a so-called Social Compute Unit (SCU), by utilizing clouds of ICUs. Our model allows service consumers to specify quality requirements, which contain constraints and objectives with respect to skills, connectedness, response time, and cost. We propose a solution model for tackling the problem in quality-aware SCUs provisioning and employ some metaheuristic techniques to solve the problem. A prototype of the framework is implemented, and experiments using data from simulated clouds and consumers are conducted to evaluate the model.

Keywords: human-based service, social compute unit, quality of service, service cloud management

1 Introduction

Recently, we have been seeing on-demand online resource provisioning models being applied not only to hardware- and software-based computing elements but also to human-based counterpart. To date, the provisioning of human-based services (HBS) in the cloud is traditionally used to provision an individual compute unit (ICU) suitable for solving simple and self-contained human tasks. However, for solving more complex tasks, we often require a group of people working in a collaboration. We advocate the notion of a Social Compute Unit (SCU) as a construct for loosely coupled, and nimble team of individual compute units, which can be composed, deployed, and dissolved on demand.

In provisioning an SCU, quality control remains a major issue. Existing quality control approaches are traditionally relies on primitives and hard-wired techniques, which do not allow consumers to customize based on their specific requirements [1]. Still we lack effective HBS management frameworks to manage the socially connected human-based resources for fulfulling the consumers' requests.

In this work, we present a framework that focuses on the provisioning of SCUs containing socially connected ICUs obtained from the cloud, such as crowdsourcing marketplaces. We posit that provisioning SCUs using the underlying ICU

clouds could enhance the on-demand provisioning model of human-based services so that it can be utilized for solving more complex tasks. Our contribution in this paper is to provide *a flexible quality-aware SCU provisioning framework*, which is based on consumer-defined quality requirements, using ICUs obtained from the cloud. In particular, our framework provides

- an architecture for quality aware SCU provisioning, which allows different quality control mechanisms to be plugged in using provisioning APIs,
- a tool for modeling quality requirements using fuzzy concepts, and
- solution models which exemplify the quality control strategies for the framework. Specifically, we develop some algorithms, one of them based on the Ant Colony Optimization (ACO) approach, for dealing with the multiobjective quality-aware SCU formation problem.

The proposed framework is particularly useful for, e.g., (i) providing a tool for the human-based services management, which integrates the involved parties in the human-based services ecosystem, and (ii) providing a simulation testbed for studying various quality control technique for human-based services. To illustrate the usefulness of our framework, we study the feasibility of the results and compare with other simpler and common approaches using simulations.

The rest of the paper is organized as follows. Section 2 provides some background for our work. Section 3 discusses the details of our proposed framework. In Section 4, we present a provisioning solution model and describe some algorithms for dealing with the SCU formation problem. Section 5 presents our experiments to study our model. Some related works are presented in Section 6. Finally, Section 7 concludes the paper and outlines our future work.

2 Background

2.1 Human-based Compute Unit

We define two types of compute units, which are capable of delivering HBS: an individual person, i.e., Individual Compute Unit (ICU), and a composition of socially connected individuals, i.e., Social Compute Unit (SCU). These individuals can be obtained on-demand from ICU clouds. Some examples of ICU clouds include task-based crowdsourcing platforms (e.g., in [2]), collections of experts on social networks (e.g., in [3]), and enterprise ICU pools (e.g., in [4]).

The execution of human tasks may employ different patterns depending on the problem domain and on the runtime systems. Some examples of such patterns include (i) *single unit*, where ICUs works individually on different tasks, (ii) *pipeline*, where the assigned SCU members execute the task sequentially one after another, (iii) *parallel*, where the task is split into subtasks, assigned to SCU members, and the results are merged back after they finish (e.g., in [5]), (iv) *fault-tolerant*, where the task execution is made redundant and the best result is selected from the aggregation of the results (e.g., in [6]), and (iv) *shared artifacts*, where the SCU members works collaboratively over some objects shared among all SCU members (e.g., in [7]).

The effect of these patterns to the SCU provisioning is that each pattern may require different ways to measure the SCU properties. For example, given a task t, and an SCU $\mathcal{V} = \{v_1, v_2, ..., v_n\}$, the response time of the pipeline pattern may be defined as $\sum_{v \in \mathcal{V}} time(v, t)$, while the response time of the parallel pattern may be defined as $\max_{v \in V} time(v, t)$, where $time(v, t)$ is the time required by the ICU v to execute the task t.

2.2 ICU Properties

We define the following generally common ICU properties: *skill sets*, *response time*, and *cost*.

Skill Set A skill of an ICU represents a qualification that the ICU has. An ICU, v, has a set of skills, $Sk^v = \{(s_1, x_1), (s_2, x_2), ..., (s_n, x_n\}$, where $n = \mid Sk^v \mid$. The skill type, s_i, of a skill defines the kind of competency that the ICU is endowed with. The skill level x_i may be defined based on different measurement techniques. For example, the skill level can be calculated based on a qualification test or based on a stastical measurement such as the acceptance rate [2].

Response Time For any ICU v for executing a task t, an estimated response time can be provided, i.e., $time : (v, t) \mapsto \mathbb{R}_{>0}$. This response time is also affected by the job queueing and assignment model, such as based on a maximum number of concurrent job (e.g., [2]), using a work queue approach commonly found in WfMS (e.g., [8]), or using a project scheduling approach considering the time availability of the candidates (e.g., [9]).

Cost An ICU v may specify its expected cost to perform a task t, which is modeled as a function of the task parameters $cost : (v, t) \mapsto \mathbb{R}_{>0}$. This function, for example, can be simply based on the estimated duration and the hourly cost.

2.3 Social Connectedness

The success of an SCU depends highly on its social connectedness [4]. We define a connectedness graph as an ordered pair $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents an SCU obtained from ICU clouds, and $\mathcal{E}$ represents a set of weighted undirected edges between two different ICUs in $\mathcal{V}$. We define an edge $e = \{v_1, v_2\} \in \mathcal{E}$ as an indication that v_1 and v_2 have worked together in the past. The weight of the edge, $w(e)$, is an integer number that represents the number of successful task completions substracted by the number of unsuccesful task completions. This weighting approach allows us to give penalty to, for example, malicious workers.

An SCU and its connectedness can be represented as a graph $G' = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}' \subset \mathcal{V}$, $\mathcal{E}' \subset \mathcal{E}$, so that $\mathcal{E}'$ is the maximum subset of $\mathcal{E}$ that connects all

ICUs in $\mathcal{V}'$. We measure the connectedness of G' as the average weighted degree of all nodes:

$$conn(G') = \frac{\sum_{e \in \mathcal{E}'} 2 \cdot weight(e)}{|\mathcal{V}'|} \quad (1)$$

3 Quality-aware SCU Provisioning Framework

3.1 Framework Overview

The core of our framework is the *Provisioning Middleware*, which coordinates interactions among the *ICU Cloud Manager*, the *Provisioning Engine*, and the *Runtime Engine*, as depicted in Figure 1. A scenario for an SCU provisioning starts when a consumer submits a task request to the *Runtime Engine*. This request contains the consumer-defined quality requirements for the task. This quality requirements consists of required skill levels of the SCU members, as well as their connectedness, maximum response time, and total cost. For executing this task, the *Runtime Engine* sends an SCU provisioning request to the *Provisioning Middleware* using the *hybrid service programming API* [10].

The Provisioning Middleware retrieves ICUs' properties from the ICU Cloud Manager, which maintains the functional and non-functional properties of the ICUs, as well as tracking the previous interactions between ICUs, from various ICU clouds. This ICU Cloud Manager encapsulates different APIs provided by different ICU clouds into a unified API. This ICU Cloud Manager also allows the formation of an SCU using many ICUs from different clouds.

The Provisioning Engine is responsible for controlling the quality of the SCU provisioning. A *quality control strategy* for an SCU provisioning is a strategy to control the formation and execution of an SCU, which takes the consumer requirements and the properties of ICUs on the cloud into consideration. There are two types of SCU quality control strategies covering two phases of the task life cycle: pre-runtime and runtime. At pre-runtime, an SCU quality control strategy governs the SCU formation. During runtime, a dynamic adaptation technique is employed to guarantee the required quality. Here we focus on pre-runtime quality control strategies (Section 4), and leave the latter issue for future work.

A pre-runtime quality control strategy is implemented using an algorithm and executed by the Provisioning Engine. To process a task in the queue, the Provisioning Middleware requests the Provisioning Engine to form the SCU. Then, the Provisioning Engine invokes the algorithm to create the formation. Upon receiving this formation, the Provisioning Middleware instructs the ICU Cloud Manager to instantiate this SCU and deploy it to the Runtime Engine. The SCU then executes the tasks using human interfaces provided by the Runtime Engine. When the task finished, the result is returned back to the consumer.

3.2 Consumer Requirements

In our framework, we allow consumers (e.g., human-based application owners, crowdsourcing requesters) to specify their requirements that represent con-

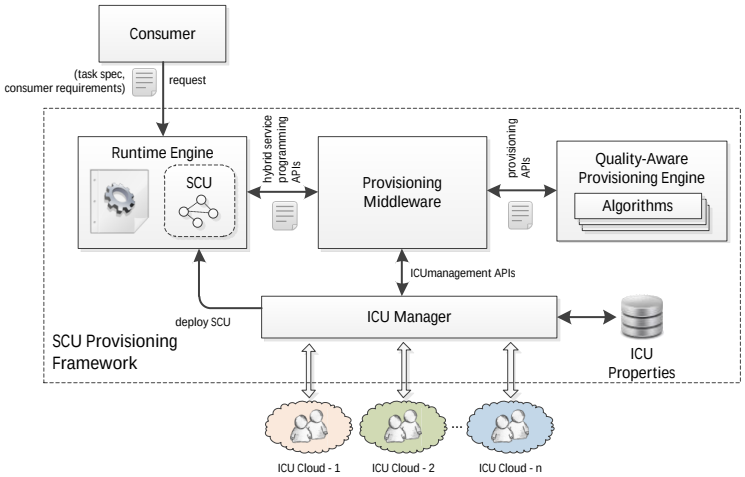


Fig. 1: SCU Provisioning Framework

straints and objectives for the SCU formation and task execution. Our model defines the consumer requirements along four dimensions: *job descriptions*, *connectedness*, *response time*, and *cost*.

Due to imprecise nature of human work, defining a precise constraint can be troublesome for consumers. Here, we propose to model quality requirements using *fuzzy concept* [11,12]. For example, instead of saying "I need an ICU with a translation $qualification \geq 0.75$", the consumer could say "I need an ICU with a *good* translation skill". For a given fuzzy quality q (e.g., *good*), we could measure the *grade of membership* of an ICU using the function $\mu_q : \mathbb{R}_{\geq 0} \rightarrow [0..1]$. We apply this fuzzy concept to model the consumer requirements with respect to job descriptions and connectedness.

Job Description A task request contains a set of *job descriptions*, or *jobs* for short. For each job, the consumer defines the meta-information (e.g., title, description, and presentation) and the required skill set. Our framework provisions an SCU for the task, where each SCU member with the required skill set fulfills a job in the task. Table 1 depicts an example of job requirements for a task, which requires two SCU members: one translator and one reviewer.

Given a task with a set of jobs $\mathcal{J} = \{j_1, j_2, ..., j_n\}$ for an SCU with size n, the Provisioning Engine attempts to find a set of ICUs $\mathcal{V}' = \{v_1, v_2, ..., v_n\}$, which maximizes $\mu_{j_i}(v_i)\ \forall i \in [1..n]$. μ_{j_i} represents the aggregated grade of membership on the *intersection* of the fuzzy sets of all required fuzzy qualities in the job, i.e., given $j_i = \{(t_1, q_1), (t_2, q_2), ..., (t_m, q_m)\}$, $\mu_{j_i}(v) = \wedge_{(t_k,q_k)\in j_i}\{\mu_{q_k}(x_k^v)\}$, where x_k^v is the numerical skill level of ICU v for skill type t_k. Here, we use the `min` operation as the interpretation of fuzzy set intersection [12].

Connectedness The required connectedness of the SCU being formed is calculated using Equation 1. This requirement is also defined using a liguistic variable, e.g., the consumer may say "I want to have an SCU with *fair* connectedness".

Jobs	Required Skill Sets	
	Skill Types	Fuzzy SkillLevels
Job #1	– Translating DE to EN – Acceptance Rate	Good Fair
Job #2	– Reviewing Translation – Acceptance Rate	Good Very Good

Table 1: An example of job requirements for an SCU

Given a connectedness requirement q_{conn} (e.g., *fair*), the Provisioning Engine composes an SCU $\mathcal{V}' = \{v_1, v_2, ..., v_n\}$ with a connectedness graph $G' = (\mathcal{V}', \mathcal{E}')$, which maximizes $\mu_{q_{conn}}(conn(G'))$.

Maximum Response Time The *maximum response time* of the task t, $maxRT \in \mathbb{R}_{>0}$, is the time limit specified by the consumer within which the task execution by the SCU must finish. For example, given a task t with parallel subtasks and the maximum response time $maxRT$, the Provisioning Engine selects SCU members $\mathcal{V}' = \{v_1, v_2, ..., v_n\}$, which satisfies $\max_{i=1}^{n} time(v_i, t) \leq maxRT$.

Cost Limit The consumer defines *cost limit* of the task t, $costLimit \in \mathbb{R}_{>0}$ which represents the maximum total cost payable to the SCU members, i.e., give a task t with cost limit $costLimit$, the composed SCU members $\mathcal{V}' = \{v_1, v_2, ..., v_n\}$, must satisfy $\sum_{i=1}^{n} cost(v_i, t) \leq costLimit$.

Objectives Furthermore, consumers may also define the objective of the SCU formation. We support the following four goals: *maximizing skill levels*, *maximizing connectedness*, *minimizing maximum response time*, and *minimizing cost*. An *objective* is an ordered 4-tuple, $\mathcal{O} = (w_s, w_{cn}, w_t, w_c)$, each respectively represent the weights of skill levels, connectedness, response time, and cost for measuring the objective value of a provisioning solution, where $w_s + w_{cn} + w_t + w_c > 0$.

Given the aforementioned constructs, we define a task request as a 3-tuple, $t = (\mathcal{J}, \mathcal{C}, \mathcal{O})$, where $\mathcal{J} = \{j_1, j_2, ..., j_n\}$, $j_i = \{(t_1, q_1), (t_2, q_2), ..., (t_m, q_m)\}$, $\mathcal{C} = (q_{conn}, maxRT, costLimit)$, and $\mathcal{O} = (w_s, w_{cn}, w_t, w_c)$.

4 Quality Control Strategies

Here we focus on pre-runtime quality control strategies, which deal with the formation of SCU prior to runtime. We formulate the SCU formation problem, which takes the quality requirements from the consumer into consideration, and propose some algorithms to solve it.

Given an SCU $\mathcal{V}$ socially connected in a graph $G = (\mathcal{V}, \mathcal{E})$, and a task request $t = (\mathcal{J}, \mathcal{C}, \mathcal{O})$, we define the *SCU formation problem* as a problem of finding $\mathcal{V}' \subset \mathcal{V}$ as members of SCU for executing task t which minimizes $\mathcal{O}$ subject to $\mathcal{C}$ and skill set requirements in $\mathcal{J}$. In the following we discuss some building blocks required to solve the SCU formation problem.

4.1 Assignments

The *ICU Cloud Manager* maintains a socially connected ICUs $G = (\mathcal{V}, \mathcal{E})$ obtained from various ICU clouds. Given a task t with a set of jobs $\mathcal{J}$, our goal is to create assignments $\mathcal{A} = \{(j_1, v_1), (j_2, v_2), ..., (j_n, v_n)\}$, $\forall j_i \in \mathcal{J}$, $v_i \in \mathcal{V}$.

The goal of an algorithm for solving the quality-aware SCU formation problem is to find $\mathcal{A}$ in the search space $\mathcal{J} \times \mathcal{V}$. Due to the size of $\mathcal{V}$ obtained from ICU clouds, this search space can be extremely huge. Therefore, we filter out non-feasible assignments based on the feasibility of competency, deadline, and cost. Formally, for each job j, we search only in $\mathcal{V}' \subset \mathcal{V}$, where $\mu_j(v) > 0$ and $time(v, t) \leq maxRT$ and $cost(v, t) \leq costLimit$, $\forall v \in \mathcal{V}'$.

However, this filtering does not guarantee a full feasibility of complete assignments on all jobs. To guide our heuristic algorithms for selecting assignments towards a feasible solution while minimizing the objective, we define two algorithm control mechanisms: *the local fitness* which represents the fitness of an assignment relative to other possible assignments for the same job, and *the objective value of a solution* which represents the fitness of a complete solution. The formulation of these mechanisms is stimulated by the necessity to measure the heuristic factors and solution quality in ACO approaches[13]. However, as we show in Section 4.4, these mechanisms can also be used by other heuristics.

4.2 Local Fitness

The local fitness of an assignment is defined based on a partially selected assignments, starting form an empty set of assignments when the algorithm begins. Given a task t with the objective weighting factors $\mathcal{O} = (w_s, w_{cn}, w_t, w_c)$, a set of selected partial assignments up to job number $i-1$, $\mathcal{A}_{i-1}$, that already contains a set of ICUs $\mathcal{V}_{i-1}$, and a set of possible assignments for the subsequent job j_i, $\mathcal{A}_i^P$, the *local fitness* λ for an assignment $a_{i,j} = (j_i, v_j)$, $a_{i,j} \in \mathcal{A}_i^P$, is defined as

$$\lambda(a_{i,j} \cup \mathcal{A}_{i-1}) = \frac{\lambda_s \cdot w_s + \lambda_{cn} \cdot w_{cn} + \lambda_t \cdot w_t + \lambda_c \cdot w_c}{w_s + w_{cn} + w_t + w_c} \tag{2}$$

where

$$\begin{aligned}
\lambda_s(a_{i,j} \cup \mathcal{A}_{i-1}) &= \mu_{j_i}(v_j), \\
\lambda_{cn}(a_{i,j} \cup \mathcal{A}_{i-1}) &= \frac{conn(v_j \cup \mathcal{V}_{i-1}) - conn(\mathcal{V}_{i-1})}{\gamma_{conn} + conn(v_j \cup \mathcal{V}_{i-1}) - conn(\mathcal{V}_{i-1})}, \\
\lambda_t(a_{i,j} \cup \mathcal{A}_{i-1}) &= \frac{\gamma_{time}}{\gamma_{time} + time(v_j \cup \mathcal{V}_{i-1}, t) - time(\mathcal{V}_{i-1}, t)}, \\
\lambda_c(a_{i,j} \cup \mathcal{A}_{i-1}) &= \frac{\gamma_{cost}}{\gamma_{cost} + cost(v_j, t)}.
\end{aligned}$$

where γ is an adjustable parameter, e.g., we can use the consumer-defined $costLimit$ as γ_{cost}. Note that these local fitness values are normalized, i.e., $\lambda : \mathcal{A}^P \mapsto [0..1]$. The elements in $\mathcal{A}_i^P$ can be defined based on the ICUs filtering described in Section 4.1.

4.3 Objective Value of Solution

For each solution, i.e., a complete set of assignments $\mathcal{A}$ for all jobs in $\mathcal{J}$, we could measure the normalized *objective value* returned by the function $f : \mathcal{A}^D \mapsto [0..1]$, $\mathcal{A}^D = \mathcal{J} \times \mathcal{V}$. Given a task t with the objective weighting factors $\mathcal{O} = (w_s, w_{cn}, w_t, w_c)$, the objective function $f(\mathcal{A})$ for $\mathcal{A} = \{(j_1, v_1), (j_2, v_2), ..., (j_n, v_n)\}$, is defined as follows:

$$f(\mathcal{A}) = 1 - \frac{f_s(\mathcal{A}) \cdot w_s + f_{cn}(\mathcal{A}) \cdot w_{cn} + f_t(\mathcal{A}) \cdot w_t + f_c(\mathcal{A}) \cdot w_c}{w_s + w_{cn} + w_t + w_c}, \tag{3}$$

where

$$\begin{aligned} f_s(\mathcal{A}) &= \wedge_{(j_i,v_i)\in\mathcal{A}}\{\mu_{j_i}(v_i)\}, \\ f_{cn}(\mathcal{A}) &= \mu_{conn}(\mathcal{V}_\mathcal{A}), \\ f_t(\mathcal{A}) &= \frac{\gamma_{time}}{\gamma_{time} + SCURT(\mathcal{V}_\mathcal{A}, t)}, \text{ and} \\ f_c(\mathcal{A}) &= \frac{\gamma_{cost}}{\gamma_{cost} + \sum_{(j_i,v_i)\in\mathcal{A}} cost(v_i, t)}. \end{aligned}$$

$\mathcal{V}_\mathcal{A}$ is the set of ICUs in assignments $\mathcal{A}$, i.e., for any $\mathcal{A} = \{(j_1, v_1), (j_2, v_2), ..., (j_n, v_n)\}$, $\mathcal{V}_\mathcal{A} = \{v_1, v_2, ..., v_n\}$. For $f_s(\mathcal{A})$, we again apply `min` function as the interpretation of intersection operation $\wedge$. The function $SCURT(\mathcal{V}_\mathcal{A}, t)$ returns the aggregated response time of all ICUs in $\mathcal{V}_\mathcal{A}$, which determined by, e.g., the response time of each ICU and the SCU pattern employed (see Section 2.1). The goal of an SCU formation algorithm is to minimize $f(\mathcal{A})$.

4.4 Algorithms

We have established the building blocks required for solving the SCU formation problem. Here, we present some algorithms to solve the SCU formation problem.

Simple Algorithms We present two simple algorithms that can be used to find a solution of the SCU formation problem based on the *first come first selected (FCFS)* and the *greedy* approach.

FCFS Approach This approach resembles the approach traditionally used in task-based crowdsourcing model: the first ICU who 'bids' wins the task. Assuming that a standby ICU is interested in taking a task, we select the first earliest available ICU for each job. In the case where there are some ICUs with the same earliest availability, we pick one randomly.

Greedy Approach Initially we construct a solution by selecting assignments for each job that has the highest local fitness value. Afterwards, we gradually improve the solution by changing an assignment at a time. Improvement is done by randomly selecting a job, and randomly selecting another ICU for that job. If the new assignment improve the objective value of the solution, we replace the

associated old assignment with this new better one. This procedure is repeated until a certain number of maximum cycle is reached. The greedy approach makes a locally optimized choice for each job at a time with a hope to approximate the global optimal solution.

Ant Colony Optimization Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of some ant species[13]. In the ACO technique, artificial ants tour from one node to another node in the solution space until a certain goal is achieved. The tour is guided by the pheromone trails, which are deposited by the ants to mark the favorable path. The nodes visited in a complete tour represent a solution. Once all ants have finished a tour, the process is repeated for a specified number of cycles or until a certain condition is met. The best solution of all cycles is selected as the solution of the problem.

In our SCU formation problem, given a requested task with a set of ordered jobs $\mathcal{J}$, a node is a tuple (j, v), where $j \in \mathcal{J}$ and $v \in \mathcal{V}$. An ant starts a tour by selecting an initial node (j_1, v_1) and travels to the next nodes (j_2, v_2), ... (j_n, v_n) until all jobs $j_i \in \mathcal{J}$ are assigned. Each node has a probability to be selected determined by the pheromone trail and heuristic factor of the node.

Several variants of ACO algorithms have been proposed. Here, we develop our algorithm based on three variants: the original Ant System (AS) [14], $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System (MMAS) [15], and Ant Colony System (ACS) [16]. Generally, the ACO approach is depicted in Algorithm 1.

When traveling through the nodes, at each move i, an ant k constructs a partial solution $\mathcal{A}_i^k$ consisting all visited nodes for job 1 to i. When ant k has moved $i-1$ times, the probability it moves to another node (j_i, v_j) is given by

$$p_{i,j}^k = \begin{cases} \dfrac{(\tau_{i,j})^\alpha \cdot (\eta_{i,j})^\beta}{\sum_{(j_i,v_w)\in\mathcal{A}_i^{P'}} \left((\tau_{i,w})^\alpha \cdot (\eta_{i,w})^\beta\right)} & \text{if } (j_i, v_j) \in \mathcal{A}_i^{P'}, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $\mathcal{A}_i^{P'} = \mathcal{A}_i^P - \mathcal{A}_{i-1}^k$, i.e. the set of possible assignments for job j_i containing only ICUs that are not yet included in $\mathcal{A}_{i-1}^k$; $\tau_{i,j}$ is the pheromone value of the node (j_i, v_j) at current cycle; and the heuristic factor $\eta_{i,j} = \lambda(a_{i,j} \cup \mathcal{A}_{i-1}^k)$ as defined in Equation 2. The relative importance of pheromone and heuristic factor are determined by parameter α and β. ACS variant uses a modified transition rule, so-called *pseudorandom proportional rule* as shown in [16].

At the end of each cycles, pheromone trails on all nodes are updated. At each cycle t, given the number of ants $nAnts$, the basic pheromone update formula for a node (j_i, v_j), which is proposed by the original AS variant [14], is given by

$$\tau_{i,j}(t) = (1-\rho) \cdot \tau_{i,j}(t-1) + \sum_{k=1}^{nAnts} \Delta\tau_{i,j}^k \tag{5}$$

where $\rho \in (0..1]$ is the *pheromone evaporation* coefficient, and $\Delta\tau_{i,j}^{k}$ is the quantity of pheromone laid by ant k on the node (j_i, v_j), which is given by

$$\Delta\tau_{i,j}^{k} = \begin{cases} {}^{Q}/_{f(\mathcal{A}^k)} & \text{if } (j_i, v_j) \in \mathcal{A}^k \wedge \mathcal{A}^k \text{ is feasible,} \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

where $\mathcal{A}^k$ is the solution found by ant k and Q is an adjustable parameter. $\mathcal{A}^k$ is feasible if it does not violate any constraints $\mathcal{C}$. We exclude solutions that violate one or more constraints so that only feasible solutions are promoted by the ants. The pheromone update for MMAS and ACS variant has the same principle but different formula as presented in [15] and [16].

Algorithm 1: Ant-based Solver Algorithm

```
initialize graph and pheromone trails
repeat
    A_ants ← ∅
    for i = 0 to nAnts do
        A ← find a tour for ant_i
        A_ants ← A_ants ∪ A
    update pheromone trails
until ∃A ∈ A_ants f(A) = 0 or is stagnant or max cycles reached
```

5 Evaluation

5.1 Implementation

We have implemented a prototype of our proposed provisioning framework as depicted in Figure 1. The implementation contains three independent components, namely *Provisioning Middleware*, *Provisioning Engine*, and *ICU Cloud Manager*. The Provisioning Engine is implemented using the quality control strategies discussed in Section 4. For simulation purpose, we populate the ICU cloud with a simulated pool of ICUs. Furthermore, we have also develop a prototype *consumer application* which capable to submit SCUs provisioning requests to the Provisioning Middleware. These components are loosely-coupled and talk to each other through specified APIs implemented using SOAP-based Web services.

In our experiments, we focus on the following aspects of the SCU provisioning: (i) we study our pre-runtime quality control strategy based on the three aforementioned algorithms and analyze the perfomance and result, and (ii) we study the ACO approach to have an insight of (a) the effect of different algorithm parameters (b) the performance and result of the three different ACO variants.

5.2 Experiment Setup

Our prototype ICU manager maintains a work queue for each ICU. Each ICU can only execute a single job at a particular time. We experiment with *parallel*

pattern (see Section 2.1), where subtasks, i.e., jobs, are assigned to the SCU members and executed in parallel. We generate 500 ICUs on our simulated cloud. We define 10 types of skills, and each ICU is randomly endowed with these skill types. The consumer application generates task request with random parameters. Each job in a task has some skills set requirements with the required fuzzy quality uniformly distributed over four fuzzy quality levels: *poor*, *fair*, *good*, and *very good*. In this experiment, we use the *trapezoidal* membership functions adopted from [17], which support *over-qualification* when assigning SCU members.

5.3 Experiment Result

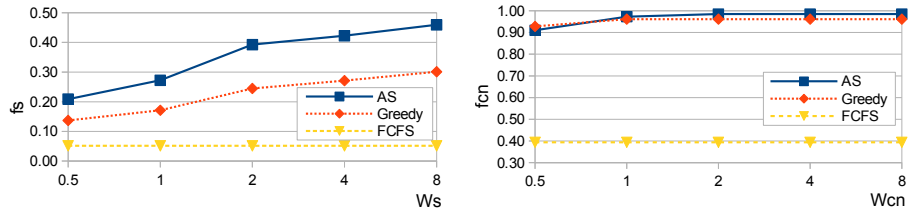
To study our pre-runtime quality control strategy, we configure our consumer application to randomly generate and submit 100 task requests. The requests are queued by the Provisioning Middleware in first-in-first-out manner. The Provisioning Middleware then requests the Provisioning Engine to form an SCU for each task request. We repeat the same setup three times to test the Provisioning Engine configured using the three implemented algorithms: the FCFS algorithm, the greedy algorithm, and the original variant of Ant System (AS) algorithm.

Table 2 shows a comparison of average results from all task requests. The AS algorithm outperforms the others with respect to the aggregated objective, i.e., minimizing $f(\mathcal{A})$. The AS algorithm also provides SCU team formation with better skill levels. However, as expected, the FCFS algorithm gives the fastest running time. But considering the nature of human tasks, few seconds running times of the AS algorithm and the greedy algorithm are reasonable. This fast performance is not without cost, since the FCFS algorithm concludes a solution too fast considering the response time only, it results in some constraint violations. Fortunately, due the filtering of the search space (see Section 4.1), violations on skill level constraints do not occur.

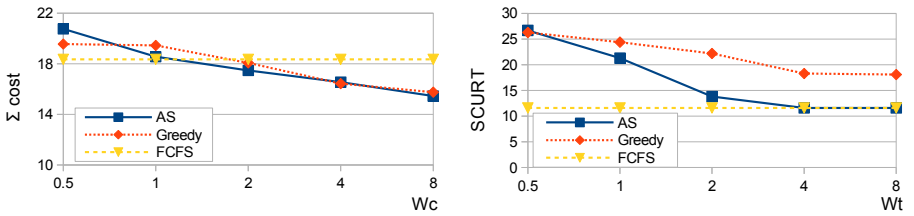
Algo	Objective Values $\overline{f}$	Skill Levels $\overline{f_s}$	Response Times $\overline{SCURT}$	Violation	Algo Time
FCFS	0.4501	0.0810	6.06	4%	0.9117 ms
Greedy	0.3468	0.2130	11.87	0%	0.1219 s
AS	0.3147	0.3228	10.90	0%	6.6565 s

Table 2: Results and performance comparison

Furthermore, we are also interested in studying the quality control behavior with respect to the objective weightings, $\mathcal{O} = (w_s, w_{cn}, w_t, w_c)$, as defined by the consumer. Figure 2 shows results of our experiment using task requests with varying objective weightings and SCU size. On each experiment shown on the subfigures, we vary one weight from 0.5 to 8 and fix the others. The results show that the AS algorithm honors the consumer defined weights better compared to the other two. The sensitivity of the FCFS algorithm is flat on all cases, because it does not consider the objective weightings during the formation. The sensitivity levels of the cost weight w_c of the greedy algorithm and the AS algorithm are



(a) Skill levels (f_s vs w_s) (b) Connectedness (f_{cn} vs w_{cn})

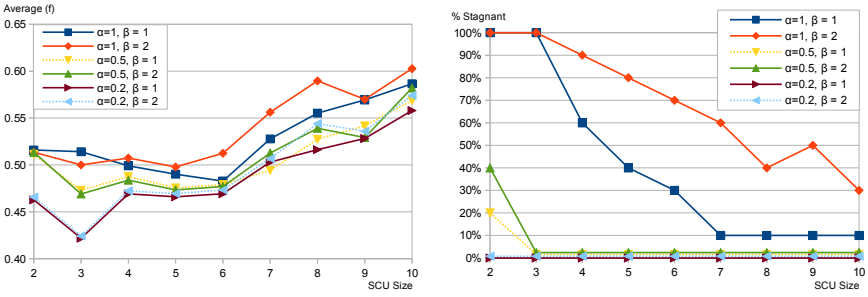


(c) Cost (f_c vs w_c) (d) Response time (f_t vs w_t)

Fig. 2: Sensitivity on objective weightings

similar, due to the fact that the local fitness value for cost λ_c contributes linearly to the objective value of the cost f_c. For the connectedness sensitivity, the AS algorithm cannot be seen clearly outperforms the greedy algorithm, because the formed SCU almost reach the upper limit of f_{cn}, i.e., 1.

Knowing that the AS algorithm provides better results in many aspects, we carry out further experiments to understand the behavior of our ACO approach. First, we study the effect of the ACO parameters to the perfomance and to the quality of the resulted SCU formation. In our experiment, we use the AS variant and fix the pheromone evaporation factor low, $\rho = 0.01$. If ρ is set too high, it will cause the pheromone trails to be negligible too fast. Then, we vary the relative importance of pheromone and heuristic factor, α and β. Figure 3a shows how different α and β yield different results with respect to the average aggregated objective value of the best SCUs formed. Furthermore, we run the experiments



(a) Objective values average (b) Stagnant behavior

Fig. 3: Influence of α and β

for 8 ants in 2000 cycles and see whether a stagnant behavior occurs as shown in Figure 3b. A cycle is said to be stagnant when all ants result in the same SCU formation; hence, causing the exploration of the search space to stop. Our experiments show that the combination of $\alpha = 0.2$ and $\beta = 1$ gives best results.

Furthermore, we extend the experiment further using the same α and β parameters to the other two ACO variants. We are interested in finding out which ACO variants give faster conclusion to a good SCU formation. We run the experiment using 8 ants and 10000 cycles as shown in Figure 4. The result shows that the MMAS variant gives better SCU formations (less objective values) in less number of cycles than the others.

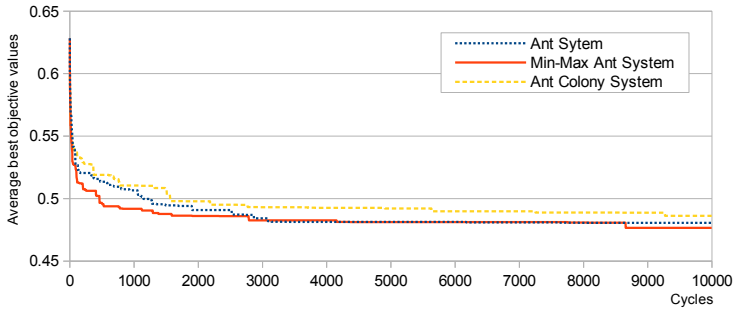


Fig. 4: Comparison on results of ACO variants

Different quality control strategies implemented by different algorithms cater different needs. Here we show an ACO based algorithm provides better results in some aspects. However, there is no "one size fits all" strategy. For example, the FCFS approach may be preferable in some circumstances where the response time is the most important factor and the consumer only cares about skill constraints, which happens in typical microtask crowdsourcing systems. The usefulness of our framework is therefore also to support multiple strategies.

6 Related Work

HBS Management Framework Recently, the issue of quality management in human-based services has attracted many researchers. The issue becomes even more crucial when we deal with online and open labor markets such as crowdsourcing marketplaces [1,18]. Several works have also been introduced to deliver managed human-based services frameworks such as [19,20].

Many techniques have also been introduced for executing human tasks in a workflow management system using organizational human-based services, such as [21,22]. Some works such as [23] goes further to allow the execution of workflows or business processes using the cloud of human-based services.

Our work endorses the notion of Social Compute Unit (SCU), which allows the execution of human tasks not only by a single human-based service but also by a composition of socially connected human-based services. Furthermore, we abstract open (e.g., crowdsourcing) and organizational pool of human-based

services as ICU clouds, and therefore, we envision the execution of organizational human-based workflow using open ICU clouds such as crowdsourcing platforms.

Formation Techniques One of the main focus of our work is in the domain of team formation optimization. Some approaches for team formation based on the fuzzy concept have been proposed, e.g., [9,17]. Other works, such as [24,25,26,27], also take the social network of the team member candidates into consideration. Our work differs from the aforementioned works in the following aspects: (i) we model constraints and objectives in four dimensions: skills, social connectedness, response time, and cost, (ii) we utilize the fuzzy concept not only to model skills but also to model the social connectedness, and (iii) we employ Ant Colony Optimization to compose the team members.

7 Conclusions and Future Work

In this paper we present our framework for the quality-aware provisioning of SCU using ICU clouds. Our framework contains the Provisioning Engine which executes quality control strategies. We propose some algorithms for pre-runtime quality control strategies, which deals with the SCU formation request considering the consumer-defined quality requirements and the ICUs properties obtained from the cloud. We conduct experiments to study the characteristics of the algorithms, which could be utilized to cater different system needs.

Our work presented in this paper is part of our ongoing research in the field of human-based service. We plan to develop other quality control strategies such as runtime adaptation techniques to govern the human-based services during runtime. Furthermore, we are also interested in investigating quality control strategies for human-based tasks on busines processes using the ICU clouds.

Acknowledgements

The first author of this paper is financially supported by the Vienna PhD School of Informatics. The work mentioned in this paper is partially supported by the EU FP7 FET SmartSociety (`http://www.smart-society-project.eu/`).

References

1. Allahbakhsh, M., Benatallah, B., Ignjatovic, A., Motahari-Nezhad, H.R., Bertino, E., Dustdar, S.: Quality control in crowdsourcing systems: Issues and directions. IEEE Internet Computing **17**(2) (2013) 76–81
2. Amazon: Amazon mechanical turk. Website (2013) `http://www.mturk.com/`.
3. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: WWW, ACM (2012) 839–848
4. Sengupta, B., Jain, A., Bhattacharya, K., Truong, H.L., Dustdar, S.: Who do you call? problem resolution through social compute units. In: ICSOC. Springer (2012) 48–62

5. Kulkarni, A.P., Can, M., Hartmann, B.: Turkomatic: automatic recursive task and workflow design for mechanical turk. In: CHI. (2011) 2053–2058
6. Varshney, L.: Privacy and reliability in crowdsourcing service delivery. In: SRII Global Conference (SRII), 2012 Annual, IEEE (2012) 55–60
7. Spillers, F., Loewus-Deitch, D.: Temporal attributes of shared artifacts in collaborative task environments. (2003)
8. Jin, L.j., Casati, F., Sayal, M., Shan, M.C.: Load balancing in distributed workflow management system. In: ACM SAC, ACM (2001) 522–530
9. Baykasoglu, A., Dereli, T., Das, S.: Project team selection using fuzzy optimization approach. Cybernet. Syst. **38**(2) (2007) 155–185
10. Truong, H.L., Dustdar, S., Bhattacharya, K.: Programming hybrid services in the cloud. In: ICSOC. Springer (2012) 96–110
11. Zadeh, L.A.: The concept of a linguistic variable and its application to approximate reasoning—i. Information sciences **8**(3) (1975) 199–249
12. Bellman, R.E., Zadeh, L.A.: Decision-making in a fuzzy environment. Management science **17**(4) (1970) B–141
13. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. Computational Intelligence Magazine, IEEE **1**(4) (2006) 28–39
14. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE TSMC **26**(1) (1996) 29–41
15. Stutzle, T., Hoos, H.H.: Max-min ant system. Future generations computer systems **16**(8) (2000) 889–914
16. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. TEC **1**(1) (1997) 53–66
17. Strnad, D., Guid, N.: A fuzzy-genetic decision support system for project team formation. Applied Soft Computing **10**(4) (2010) 1178–1187
18. Ipeirotis, P.G., Horton, J.J.: The need for standardization in crowdsourcing. In: Proceedings of the CHI'11 Conference on. (2011)
19. Minder, P., Seuken, S., Bernstein, A., Zollinger, M.: Crowdmanager-combinatorial allocation and pricing of crowdsourcing tasks with time constraints. In: Workshop on Social Computing and User Generated Content. (2012) 1–18
20. Dow, S., Kulkarni, A., Klemmer, S., Hartmann, B.: Shepherding the crowd yields better work. In: ACM CSCW, ACM (2012) 1013–1022
21. Agrawal, A., Amend, M., Das, M., Ford, M., Keller, C., Kloppmann, M., König, D., Leymann, F., et al.: WS-BPEL extension for people (BPEL4People). V1. 0 (2007)
22. Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. EJOR **134**(3) (2001) 664–676
23. La Vecchia, G., Cisternino, A.: Collaborative workforce, business process crowdsourcing as an alternative of bpo. In: Current Trends in Web Engineering. Springer (2010) 425–430
24. Rangapuram, S.S., Bühler, T., Hein, M.: Towards realistic team formation in social networks based on densest subgraphs. In: WWW, ACM. (2013) 1077–1088
25. Kargar, M., An, A., Zihayat, M.: Efficient bi-objective team formation in social networks. In: ECML PKDD'12, Springer-Verlag (2012) 483–498
26. Cheatham, M., Cleereman, K.: Application of social network analysis to collaborative team formation. In: CTS, IEEE (2006) 306–311
27. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: ACM SIGKDD, ACM (2009) 467–476