On the Elasticity of Social Compute Units

Mirela Riveni, Hong-Linh Truong, and Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
{m.riveni,truong,dustdar}@infosys.tuwien.ac.at

Abstract. Advances in human computation bring the feasibility of utilizing human capabilities as services. On the other hand, we have witnessed emerging collective adaptive systems which are formed from heterogeneous types of compute units to solve complex problems. The recently introduced Social Compute Units (SCUs) present one type of these systems, which have human-based services as their core fundamental compute units. While, there is related work on forming SCUs and optimizing their performance with adaptation techniques, most of it is focused on static structures of SCUs. To provide better runtime performance and flexibility management for SCUs, we present an elasticity model for SCUs and mechanisms for their elastic management which allow for certain fluctuations in size, structure, performance and quality. We model states of elastic SCUs, present APIs for managing SCUs as well as metrics for controlling their elasticity with which it is possible to tailor their performance parameters at runtime within the customer-set constraints. We illustrate our contribution with an example algorithm.

Keywords: Social Compute Units, Elasticity, Adaptation, Collective Adaptive Systems.

1 Introduction

In recent years, new forms of collective adaptive systems(CASs) that consider heterogeneous types of compute units/resources(e.g., software services, human based services and smart-devices) have emerged [20]. These systems allow compute units to be flexibly added and/or removed from them, and different collectives can overlap with each other by utilizing each other's resources. Compute units within collectives are collaborative, manageable and may be given decision making responsibilities. With the advance of human computation [17] there is a possibility of forming CASs that include human-based services [21] as compute units. Social Compute Units(SCUs), introduced in [6], can be considered as one type of these collective adaptive systems. They are virtual compositions of individual human compute units, performing human computation tasks with a cloud-like behavior. SCUs are possible today because of the human resource pools that are provided by human computation platforms (e.g., crowdsourcing platforms, social networking platforms and expert networks), which have brought the possibility to investigate ways of utilizing human computation under the service oriented computing paradigm. However, due to the unpredictability of

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 364–378, 2014.

human behavior, human-based services bring considerable challenges in their management. This is especially the case with collective adaptive systems such as SCUs, where the ways to manage resources are obviously different and more complex than the management of crowd workers that work individually, and that of collaborations with fixed number of resources. In this context, traditional platforms that support virtual fixed-sized collaborations might not be as efficient as those that support SCUs with elastic capabilities that offer opportunities for variable resource numbers with variable scalable capabilities. There are several reasons for this. First, unexpected tasks might be generated at run-time which may require new type of elements with new type of capabilities. In fixed-resource collaborations, usually existing members need to learn these tasks and thus the work might be delayed and/or executed with lower quality. Next, there might be a human-compute unit that is temporarily misbehaving or its performance is degraded. Its exclusion would bring degradation of the collaboration and the performance of the collective, if another appropriate one is not employed in its place. Furthermore, due to badly planned delegations, it is often the case that some resources are overloaded while others are underutilized. The latter comes as a consequence of the problem of the reliance on human resource availability as one of the fundamental ones in social computing. In this context, the *willingness* of a human resource to execute a particular task at a specific time point is often overlooked. However, this is crucial for platforms supporting work that includes human computation because even if we assume that human resources can use "unlimited" software-based resources, e.g., using the cloud, human behavior is dynamic and highly unpredictable.

The aforementioned problems show that there is a need for management mechanisms to support elasticity by scaling in size and computing capabilities of SCUs in an elastic way. Authors in [8],[21] identify the underlying challenge in provisioning SCU elasticity to be the lack of techniques that enable proactive provisioning of human capabilities in a uniform way in large scale. Nevertheless, assuming the possibility of utilizing human-based services in a cloud-like way, systems should support runtime elastic coordination of collectives. To address the aforementioned issues, in this paper, we investigate and provide runtime mechanisms with the elasticity notion in mind, so that platforms would be able to provide *elastic capabilities of human-based compute units/SCUs*, that can be managed flexibly in terms of the number of resources, as well as their parameters such as cost, quality and performance time. Hence, our key contributions are:

- conceptualizing and modeling the SCU execution phase and states,
- defining SCU-elasticity properties and APIs,
- designing an SCU provisioning platform model with elastic capabilities.

The rest of this paper is organized as follows. In Section 2 we present a motivation example and discuss challenges in elasticity provisioning. In Section 3 we describe the SCU concept, model the execution mode of an SCU and present our platform for managing elastic SCUs. Section 4 illustrates the feasibility of our approach. We present related work in Section 5 and conclude the paper in Section 6.

2 Motivation, Background and Research Statement

Scenario. Let us consider a concrete scenario of a software development project e.g., for a health-care specific system, and assume that a software start-up company is engaged for its execution and completion. To deliver the end-artifact, these type of projects require diverse set of skills. Hence, in addition to the company employees, some specific parts of the project might need to be outsourced, e.g., to experts with experience in health-care but also to IT professionals with skills that the start-up is lacking. Hence, to solve the problem of skill deficiency, an SCU including human-based resources/services both from the software developing company but also "outside" experts is formed. The SCU utilizes software services for collaboration and task execution. On the other hand, the human-based services and the software services that they utilize are supported by an SCU provisioning and management platform that coordinates their performance.

The challenges that arise in this scenario come from the importance of performance and quality of results in paid expert units. A software solution needs to be delivered on time and in accordance with customer requirements and budget limitations. Fixed composite units with a known number of resources, including outsourced ones, often have problems with overloaded resources and may result in project delays with good quality or on time delivery of solutions with a lower quality than the desired ones. Problems such as those mentioned in the introduction also appear. However, with the availability of online resource-pools from human clouds [10], human-based services can be acquired and released from SCUs on demand, so as to best meet the customer performance and quality requirements. Hence, we assume that the "outside" experts for our software development SCU can be recruited from human clouds on demand. Under these assumptions and if the SCU supporting platform incorporates mechanisms that allow elasticity, an initial SCU will be able to adapt at runtime with respect to certain parameters, such as the number of its compute units, unit types, structure and performance. This can be particularly important in agile software development, where both the customer requirements and the development process evolve in an iterative way, and teams have high collaboration with the customer and are more responsive to change. Our hypothesis is that in consequence of these elastic capabilities, SCUs will provide higher efficiency at runtime. Thus, platforms that include mechanisms and techniques for runtime support of coordination of SCUs with elastic capabilities are crucial.

Background and Challenges. As aforementioned, our approach is based on the concept of *Social Compute Units* [6], which fundamentally represent virtual collective systems with human-based resources as compute units that are brought together to work on a common goal with a deadline. These compute units, can belong to an enterprise, they can be invoked from a crowdsourcing platform, an expert network or any platform that hosts pools of available resources for human (including social) computation. In relation to the work of the coauthors in [21], in this paper, we use the term Individual Compute Units (ICUs) for SCU members, which represent human-based services that can be programmed in a manner that

they can execute tasks on-demand. Thus, an SCU is composed on request from a customer who defines requirements and sets constraints(e.g.,budget,deadline). It has its compute (performance) power, it can be programmed (managed) and is intended to work utilizing a collaboration platform hosted on the cloud. The behavior of an SCU is cloud-like, in the sense that its duration and performance depends on its goal, customer constraints as well as events generated during its execution.

Considerable related research focus has been put on formation algorithms [1],[13] and performance optimization within fixed teams. However, SCUs have a different nature than teams, as the SCU structures and capabilities can be programmed and SCU members can be elastically managed at runtime. Thus, even if some work for teams can be utilized, there is a research gap concerning SCU elasticity during the execution phase, in terms of resource numbers but also in terms of non-functional parameters(NFPs) such as cost, reliability, performance time etc. There has been a classification of human cloud platforms, where one category of platforms is said to be focused on project governance and complex coordination between resources [10], as opposed to crowdsourcing ones where the responsibility of project governance is not entirely on the platform. Examples of these type of platforms are TopCoder[1] and workio[2]. Even though these type of platforms can manage the lifecycle of collaborations, we argue that they lack the adaptation techniques and flexibility of resource management in terms of elasticity at runtime. For example, the pricing in these cases is not set by the customer like in crowdsourcing, rather the human-based services set their own prices. Thus, there is a possibility that with these models a collective of human resources can be automatically "programmed" so that if the number and type of resources changes the cost does not exceed the customer's total budget. This is one example of NFP elasticity in terms of cost. Consequently, identifying possible elastic operations that can be triggered at critical time points present important challenges for optimizing an SCUs performance. In the context of the aforementioned scenario and what lacks in current platforms, some of the research questions that we confront are:

- Given an initial formed SCU and a set of monitored team performance metrics, what are the set of actions that can enable SCU elastic capabilities, in situations when performance is degraded and violates a threshold value for a customer set constraint?
- When is optimization(e.g, load balancing) within an SCU not enough and a reorganization needed? Which tasks need to be reassigned, when and to whom(to a resource within/out of the SCU?

To sum up, this paper investigates the following fundamental challenge: *What are the mechanisms that a human computation system needs to deploy so as to provision SCUs with elastic capabilities, both in terms of resource scaling and in terms of variable properties?*

[1] `http://www.topcoder.com/`
[2] `https://www.workio.com/`

3 Social Compute Units and Elasticity

3.1 Elastic Social Compute Units

Elastic SCUs have elastic capabilities that can be triggered at runtime to tailor their performance to best fit client requirements at runtime. With human based resources being unpredictable and dynamic, their skills, price, interest and availability can change with time and within a specific context. However as stated in [6] the concept of SCU does not have a notion of elasticity in itself, thus an SCU provisioning platform which creates, deploys and supports the execution of SCUs needs to include mechanisms for scaling it up or down as needed, and as aforementioned, with this scale an SCUs performance parameters vary as well. These mechanisms should ensure that at each time point these parameters are within desired levels and comply with customer constraints. For our purposes, we conceptually define the elasticity of SCUs as follows:

Definition 1. ***The Elasticity of Social Compute Units*** *is the ability of SCUs to adapt at runtime in an automatic or semi-automatic manner, by scaling in size and/or reorganizing and rescheduling, such that the variations in the overall performance indicators such as capability, availability, effort, productivity and cost, at each point in time are optimal within the boundaries of the customer-set constraints.*

To support elasticity for SCUs, we identify as a prerequisite to have an execution model for an SCU, as previous work identifies SCU phases but do not go into details into its execution phase. An SCU lifecycle consists of the following stages: request, create, assimilate, virtualize, deploy and dissolve [6]. The elasticity mechanisms are needed after the virtualization stage, in the *execution phase*, which we model next.

3.2 SCU Execution Model

We denote a cloud of ICUs (e.g.,from online platforms and/or enterprise internal pool) as the universal set $R = \{r_1, r_2, r_3...r_n\}$, and the set of ICUs that are members of a particular SCU as $S = \{s_1, s_2, s_3...s_n\}$, where $S \subset R$. Let the set of tasks to be executed from a specific SCU be $T = \{t_1, t_2, t_3...t_n\}$. For each task $t_i \in T$, we denote the set of matching, appropriate and possible ICUs that can perform the task t_i as $P = \{p_1, p_2, p_3...p_n\}$, where $P \subset R$. Depending on constraints the following can be valid in different situations: $P \subset S^c$, $P \subset S$ or $P = S$. To provide elasticity, ICUs from S can be released and new ICUs from P can be added to S, therefore, $|S|$ might change at runtime. We model an ICU belonging to the cloud of ICUs R, with the following set of global properties, $ICU^{gl}_{prop} = \{Id_{icu}, skillset, reputation, price, state_{global}\}$. Moreover, an ICU from the perspective of the specific SCU of which it is a member, is modeled with its local properties, as $ICU^{lscu}_{prop} = \{Id_{scu}, ICU^{gl}_{prop}, state_{local}, productivity, trust\}$, where *reputation*, *state productivity*, and *trust* are aggregate metrics that we discuss further in this section.

States. An SCU in execution mode, at a specific time point τ, can be in one of the following action-states, $SCU_{state}(\tau) = \{running, suspending, resuming, expanding, reducing, substituting, stopped\}$. These states are listed in Table 1. The mentioned states are basic/atomic ones and a combination of them makes a complex SCU execution state. For example, an SCU might be running but due to an adaptation action, at the same time multiple ICUs (a cluster of ICUs) within an SCU might be suspended, while a new ICU is being added in expanding state. In this case because *running*, *suspending* and *expanding* are all execution states of an SCU, then $running \wedge suspending \wedge expanding$ is also an SCU state. However, some states are mutually exclusive if they refer to the whole SCU and cannot be aggregated, i.e., an SCU cannot be in $running \wedge stopping$ state. If one of the atomic states refers to (a change in) individual or a cluster of ICUs, an SCU can be in $running \wedge extending$ state or for example an SCU can be in a $running \wedge reducing$ state. Thus, the aggregate states are valid in the context of the scope that a state-changing action takes place. Table 1 also shows the scope for which the state-changing actions are valid, in terms of the whole SCU, a cluster of ICUs, or ICUs only. The importance of the state of an SCU as a whole is tightly coupled with ICU states and is crucial when applying elastic strategies in two ways: 1) the state of the SCU can be a trigger for elastic operations on the SCU, and 2) it can be a desired result after applying these operations.

Table 1. Fundamental state alternatives of the SCU Execution phase

Trigger action	State	Scope	Triggering Role		
			Platform	Customer	ICU
Run	Running	SCU	√	√	
Suspend	Suspending	SCU/ICUcluster/ICU	√	√	√
Activate	Resuming	SCU/ICUcluster/ICU	√	√	√
Add	Expanding	ICUcluster/ICU	√	√	√
Exclude	Reducing	ICUcluster/ICU	√	√	√
Stop/Exclude/Add	Substituting	ICUcluster/ICU	√	√	√
Stop	Stopping	SCU	√	√	

SCU Elasticity Management. Table 1 shows ways of adaptation triggering: platform based, customer based and ICU based. To clarify, a platform that supports an SCU should have the mechanisms to support *all* of its execution states elastically. Thus all state-changing actions can be triggered in an automated way as shown in Table 1. Referring to our motivational scenario, in rare cases the customer could suspend the whole SCU of software development until he has consulted and decided for crucial changes. There are other triggering state-changing actions that the customer can also make(shown with light gray check signs). Table 1 also shows which state-changing actions can be most affected by communication and ICU feedback, which we illustrate in Section 4. We show an example for a software developing SCU in execution mode in Fig. 1. At a specific time point ICUs with *developer* skills are in running state while designers are suspended. Next, due to an event when expert information is needed(e.g.,health-care

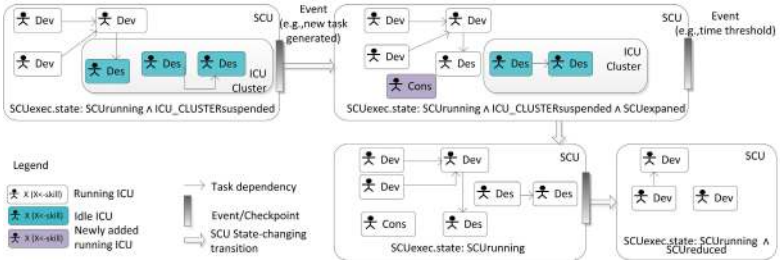

Fig. 1. An illustrative example of an SCU in execution: expanding and reducing states

information in our scenario), the SCU is expanded by including ICU with specific expertise and consultancy skills while a designer-ICU is resumed. At another time point each ICU is running, while before dissolving, the SCU is reduced as ICUs with designer and consultancy skills have finished their tasks. Adaptation actions on an SCU can change its execution model not only in terms of the state but also in terms of its execution structure. These changes are interdependent with task structure changes and ICU state changes.

Basic ICU and SCU Metrics. The decision to apply an elastic adaptation action depends on events that are triggered by two level monitoring of global and local metrics, namely to detect: 1) a violation of preset threshold values for overall SCU performance, and 2) which ICUs have affected the SCU's performance degradation. The focus of this paper is not to investigate extensive metrics, as many are context dependent. Thus, in this section we list and define some basic ones that we identify to be useful for SCUs at runtime.

Project Effort and *Productivity* have been listed as performance measures for software projects [12]. Modified versions of these metrics can be reused for SCUs on software and other goals. Thus, we define the *SCU Effort* as the sum of the average time spent by each ICU on each assigned task. The SCU task completion ratio, gives the fraction of completed tasks within those assigned. However this does not always mean that the results of all completed tasks are also

Table 2. Notation and description of basic ICU metrics and parameters

Metrics	Description
n_{req}	Number of willingness requests sent from the scheduler to an ICU
n_{ack}	Number of willingness acknowledgments sent to the scheduler by an ICU
n_{reasgn}	Number of tasks reassigned to an ICU
$n_{sucreasgn}$	Number of successfully executed reassigned tasks by an ICU
$n_{approved}(s_i)$	Total number of successfully executed/approved tasks for an ICU
$\tau(s_i, t_x)$	Processing time for task x executed by an ICU
$c(s_i, t_x)$	Cost for task x when executed by an ICU
$c_(s_i^{nw}, t_x)$	Cost for task x when reassigned to a new ICU

approved. Thus, we also consider the number of valid or approved tasks, which we use for calculating the productivity of an SCU. We define *SCU Productivity* as the ratio of approved tasks to *SCU Effort*, giving an average number of tasks-per-time-unit value. The *SCU Reputation* is a weighted sum of the reputation score of each ICU regarding its expertise for the skill for which is included in the SCU. We model the *SCU Reputation* in this way because some ICUs in a specific SCU are more crucial than others by executing more critical tasks. We define the, $reputation(s_i)$ as a function of *(Success Rate, Approved Tasks, Timeliness, Reliability, SocialTrust)*. The *SCU Cost* is an aggregate sum of the cost of each ICU for each task according to its type and skill-type requirements. The metrics are given in Table 3, where $s_i \in S$ and $t_x \in T$. See Table 2 for notation on individual metrics, some of which we use in calculating those in Table 3. The described metrics are dynamic and a platform supporting elastic SCUs should be able to monitor and utilize them in runtime adaptation strategies.

From all that was discussed, we can now characterize the elastic profile of an SCU within time τ, as $SCU_{exec}(\tau) = \{SCU_{size}(\tau), SCU_{structure}(\tau), SCU_{state}(\tau), SCU_{effort}(\tau), SCU_{productivity}(\tau), SCU_{cost}(\tau), SCU_{reputation}(\tau)\}$.

Table 3. Example metrics of SCU performance

SCU Metrics	**Definition**
SCU Total Completed Tasks	$CT(scu_i) = \sum_{i=1}^{\lvert S\rvert} n_{completed}(s_i)$
SCU Approved Tasks	$AT(scu_i) = \sum_{i=1}^{\lvert S\rvert} n_{approved}(s_i)$
SCU Success Rate	$ST(scu_i) = AT(scu_i)/CT(scu_i)$
SCU Effort	$Effort(scu_i) = \frac{1}{CT(scu_i)} \sum_{s=1}^{\lvert S\rvert} \sum_{x=1}^{m} \tau(s_i, t_x)$
SCU Productivity	$Productivity(scu_i) = AT(scu_i)/Effort(scu_i)$
SCU Reputation	$Reputation(scu_i) = \sum_{i=1}^{\lvert S\rvert} w_{expertise} * reputation(s_i)$
SCU Cost	$Cost(scu_i) = \sum_{i=1}^{\lvert S\rvert} \sum_{x=1}^{m} c_(s_i, t_x)$

Elasticity APIs. To be able to provide SCU elasticity capabilities, which include ICUs having the aforementioned (and other domain-dependent) properties, we need to have common APIs for their description and management. Currently we develop APIs which we categorize in ICU-description APIs for manipulating ICU profiles, ICU-scheduling APIs for ICU management and elastic operations,

Table 4. Example API, abstract methods for ICU manipulation

Scheduling methods	Description
abstract AddICU()	adds an ICU to the SCU
abstract void SuspendICU(SCU scu)	brings an ICU to idle state, still included in the SCU
abstract void ExcludeICU(SCU scu)	excludes an ICU form the SCU
abstract void ResumeICU(SCU scu)	restart an ICU and its associated tasks
abstract void ReserveICU(Task t)	reserves an alternative ICU for an already assigned task
abstract void SubstituteICU()	substitutes an ICU with a reserved one
public List <ICU> getAllICUinSCU(SCU scu)	returns ICUs within the SCU
public List<ICU>getSuspendedICUs(SCU scu)	returns suspended ICUs within an SCU
public List<ICU>getIdleICUs(SCU scu)	returns idle ICUs in an SCU
public List<ICU>getReservedICUs(Task t)	maintains an ordered list of top appropriate ICUs for a certain task (ICUs might be in/out of the specific SCU)

and communication operations. Table 4 describes some specific methods that we develop to be utilized in strategies providing SCU elastic capabilities.

3.3 Elastic SCU Provisioning Platform

Figure 2 shows a model of our concept of an elastic SCU provisioning platform, that utilizing our SCU execution model, metrics and API is able to support elastic SCU management. Thus, the platform supports the following behavior: a customer/SCU consumer submits a project/request with multiple tasks to it. When submitting tasks and request for SCU formation, the client specifies functional and non-functional ICU requirements such as: skill, reputation and cost. In addition he specifies overall SCU constraints, such as total budget and deadline. The platform integrates an SCU formation component with ICU selection/ranking algorithms. The resource selection and initial task assignment is not in our focus. The SCU creation/formation component's output is an initial SCU created by selecting ICUs from human cloud providers. This SCU is "fed" to a *controller*-a component that hosts monitoring and adaptation algorithms utilizing APIs for elasticity control, which provide SCU runtime management. The challenge of this component, is to monitor and adapt the SCU in accordance to customer set constraints, such that the SCU gives the maximum performance and quality within the preset boundaries for time related, cost and quality related indicators. Different scheduling and ICU management algorithms can be plugged into the platform, which would support the SCU during its lifecycle.

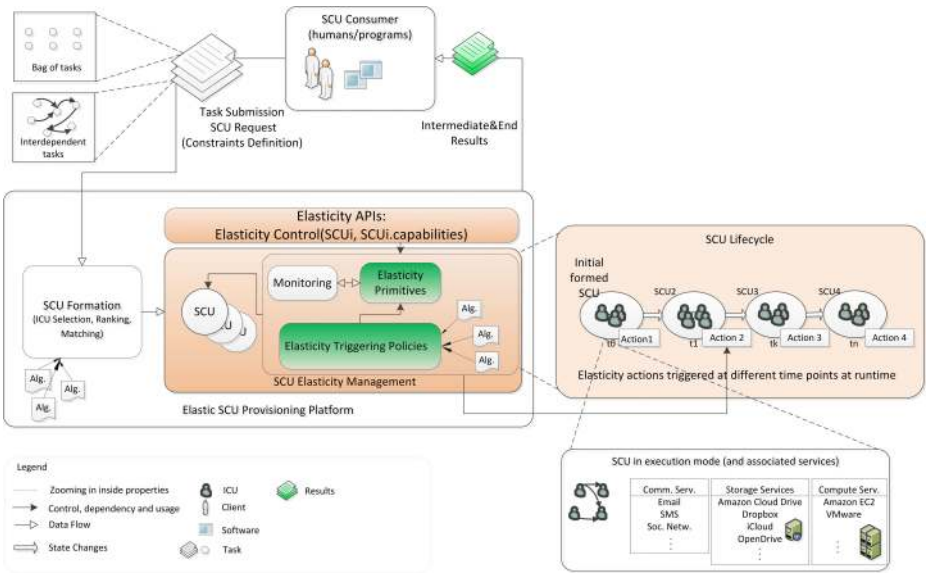

Fig. 2. Conceptual platform model supporting elastic SCUs

4 Illustrating Example

In this section we show the benefit of having explicit state management, metrics and elasticity for supporting elastic SCU. We present the way our framework can simplify the complexity of the development of elasticity strategies for SCUs. Typically, an elasticity strategy for an SCU is a domain-specific problem. In the following, we illustrate how an *ICU Feedback-based elastic SCU management strategy* can be implemented.

As ICUs within an SCU are inherently dynamic and unpredictable, we cannot always fully rely on the system-based availability information concerning an ICU and fully automated task assignment and scheduling might not always be the most suitable approach, especially when there is a possibility of unexpected generation of tasks at runtime. Hence, we propose an SCU adaptation strategy that uses ICU acknowledgments for their *willingness* to work on specific tasks. More specifically, these acknowledgments are sent in response to system requests for availability guarantees for the execution of tasks that need reassignment. This strategy supports elasticity in the sense that it departs from the idea that a customer knows in advance which and how many ICUs will contribute and the final cost for his "project". However, the customer budget is kept within its limits as the cost may vary within these limits, just as the size and structure of the assembled SCU may vary with time until the final result is returned.

Our example of elastic SCU mechanism is a semi-automatic task scheduling strategy where part of the coordination for task re-assignment is delegated to ICUs. With this approach a task is being re-assigned to a more available ICU,

on an ICUs own approval and when certain conditions apply (e.g, when a threshold is reached). Thus, the task reassignment decisions are partly based on feedback from ICUs and in this way the elastic SCU management is influenced from "human in the loop" decentralized coordination. With this example, we show how new SCU metrics can be derived and how APIs for elastic capabilities can be used.

Deriving New SCU Metrics. By utilizing APIs for obtaining SCU metrics at runtime, one can calculate the willingness of an ICU and the willingness confidence score as: (See Table I for notations):

$$Willingness = \frac{n_{ack}}{n_{req}}, Success_{reasgn} = \frac{n_{sucreasgn}}{n_{reasgn}}, WCnf = \frac{n_{ack}}{n_{req}} \times \frac{n_{sucreasgn}}{n_{reasgn}}.$$

We derive the willingness confidence value from the basic indicators, *ICU willingness,* and the *rate of success in executing the reassigned tasks.* The willingness confidence score WCnf, is computed from the number of acknowledgments that an ICU has sent to the scheduler in response to its Requests for Willingness, and the number of successfully completed tasks that are assigned to it as responses to these acknowledgments. Thus, it is an indicator about the reliability of the alternative ICUs guarantee about its willingness to work.

Programming an Elasticity Strategy Using Elasticity APIs. Considering worker willingness, provides a way to measure and control the unpredictability/reliability of ICUs by asking them for task-execution guarantees because it provides a way to compare their "statements" with their actual behavior. This is what the value of *Willingness Confidentiality* indicates. In this strategy we assume that each incoming task is assigned to the ICU at the top of a ranked list which is returned by a ranking algorithm, and references to the first x most appropriate ICUs from the ranked list are stored as reserves/alternatives for each task. The algorithm can be summarized with the following steps:

1. When a preset threshold, related to a task which is already assigned to the most appropriate ICU matching the requirements is reached, e.g.,the tasks waiting-time in an ICUs task-queue, the scheduler sends a request for execution willingness to the next top x number of ICUs that it has references to (reserves from the initial ranked list), which at the same time are idle, or their task queues are smaller than that of the ICU to which the task was initially assigned. With this request for willingness, it notifies them that there is a task that they can work on. This request is a resource availability-check; it is a request for a resource's *willingness* to work on a specific task as a form of a worker-side commitment or guarantee that the task will be executed by it.
2. Each ICU that receives this request and is ready and wishes to work on the task then sends the scheduler a willingness acknowledgment(Ack)/feedback to this request.
3. The scheduling component reassigns the task on threshold to the alternative resource that has sent a willingness acknowledgment and that is idle or has the smallest task queue. Priority is given to less loaded ICUs that are already members of the SCU.

Algorithm 1. Task-reassignment with ICU-side assurance

Require: scuTasks for SCU
Require: customer constraints on NFP
1: **for all** tasks in T **do**
rank matching ICUs and return the first 10 appropriate
2: $P \leftarrow getReservedICUList(Taskt)$ ▷ store reserve ICUs
3: assign task t to top ranked ICUs r
4: **if** r is not an element in SCU **then** do
5: $SCU \leftarrow addICU()$ ▷ add ICU r to SCU x and update its profile
6: **if** $task.taskQueueTime == task.timeThreshold$ **then** do
7: **if** $r == idle$ **then** do
8: $SCU \leftarrow removeICU()$ ▷ reduction: remove ICU r from SCU
9: **for all** ICU in P **do**
10: $getICUState(ICUICUid)$
11: **if** ICU_ STATE==idle AND $icuReserve.tQueue() < r.tQueueSize()/2$ **then** do
12: $willingnessReqMessage()$
13: **for all** $icuReserve.sentAck == true$ in ascending order of $icuResource.taskQueue$ **do**
14: **if** resource belongs in SCU **then** do
15: $substituteICU()$ ▷ re-assign task to SCU member and update its profile
16: **break**
17: $substituteICU()$ ▷ re-assign task to external ICU and update its profile
18: $SCU \leftarrow addICU()$ ▷ expansion: include resource y in SCU

When multiple reserve ICUs send acknowledgments that they are ready to execute the task, the reassignment decision is made based on the information from the Acks combined with monitoring information about their task queues and logged information about the WCnf score. This type of scheduling combines the freedom of choosing tasks that workers have in crowdsourcing environments, with policy based assignment of tasks. It is these ICU-side guarantees that are combined with task queue analysis, that can avoid problems such as *delegation sinks*. We outline the steps of this strategy in Alg 1. Alternatively, the request for willingness can be sent immediately after the task's initial assignment so that when a threshold is reached the scheduler only checks the task queues of ICUs.

Executing an Elasticity Strategy. We implemented the algorithm using methods described in the API section. We created tasks with different skill requirements and modeled an ICU with a single skill for simplicity, and assigned each of them different costs. When a decision is made about which tasks are going to be reassigned to which ICU, the new cost calculation includes the prices of each of the new ICUs, as follows:

$$Cost_{adapt}(scu_i) = Cost_{previous}(scu_i) - \sum_{i=1}^{m}\sum_{x=1}^{j} c(s_i, t_x) + \sum_{i=1}^{m}\sum_{x=1}^{j} c_(s_i^{nw}, t_x),$$

where $Cost_{adapt}(scu_i) \leq$ *Allowed Budget*. Due to space limitations and to the fact that it is not our goal to show how good this strategy is, we provide a supplement material[3].

Generally, the results show that SCU productivity raises with the number of ICUs and the same effort, while it declines if the effort is high for a low number of tasks and a small number of ICUs.

5 Related Work

Resource Management and Adaptation. Work on a retainer model for crowdsourcing environments and examples of its application are presented in [4],[3]. The model is designed for recruiting guaranteed workers by paying them a small additional amount, and in this way keeping them in reserve and in ready state for handling real-time tasks. The similarity of our ICU-feedback based strategy is in that our scheduler keeps references to the top x number of resources that are previously ranked as most suitable for a specific task. Hence, these resources are the reserve resources in our approach. However, the difference in our approach is that no prior payment is made for reservation of these resources, rather the scheduler sends them a notification asking for feedback for their willingness to execute a task that is already assigned to another resource but for which a threshold is reached. Our model is not concerned with initial task assignment and it is not intended for crowdsourcing tasks, although ICUs may be invoked from a crowdsourcing platform. Authors of [15] present a programming language and framework called CrowdLang for systems that incorporate human computation, and what is of interest to us is that they provide cross-platform integration of resources, in this way making a human cloud possible. There is a considerable amount of work conducted on adaptation and more interestingly on self-adaptation strategies. For example, authors in [16] have presented an architecture that includes a self-adaptation framework for service-oriented collaboration systems. The part that this work relates to, is their approach on identifying worker misbehavior patterns (e.g., as a result of uncontrolled task delegations) and providing a solution of reassigning tasks to other alternative resources by taking into account their task-queue size. Our strategy differs from theirs in that tasks are not delegated if ICUs are not willing to accept tasks. Rather, the task reassignment is managed with *consent* from alternative ICUs. [9] describes a delegation model and related algorithms that concern trust updates. The authors mention adoption as a process where the delegation is initiated by the "delegatee". Our algorithm stands in between delegation and adoption.

Collaborative Communities and Teams. The concept of the SCU that we utilize in our work is presented by one of the coauthors in [6]. However, while this is the fundamental work introducing the SCU, it describes its life-cycle and does not go into details into the SCU execution phase as this was not its aim. This is tackled in [19], where researchers have looked into a specific case of incident management to investigate how SCUs and their evolution(adaptation) perform better

[3] `dsg.tuwien.ac.at/research/viecom/prototypes/viecas`

over traditional process management. Resource discovery in crowdsourcing and team formation strategies and algorithms have been the subject of investigation in many works, such as [1], [2], [14], [13],[5]. The algorithms in these works can be utilized for SCU formation and some also for ICU selection when an SCU needs to be extended. Task executing collaboration models and runtime collaborations are also investigated in works such as [18]. However, the mentioned works focus on fixed teams without elasticity assumptions.

Elasticity. The notion of elasticity is treated in several domains and contexts and has especially gained importance with the advance of cloud computing. In [8] authors discuss the reasons, challenges and their approach toward virtualizing humans and software under the same service-based model that will enable elastic computing in terms of scaling both software and human resources. The concept of elasticity in Cloud computing, is being extended to concepts like application [22] and process [7] elasticity, e.g., in [7], the authors identify resource, cost and quality elasticity as being crucial in modeling processes in service oriented computing. Mechanisms and a middleware to support scaling services in and out from applications utilizing SaaS are presented in [11].

6 Conclusion

Our research focus in this work was to provide mechanisms for effective provisioning of SCUs with elastic capabilities and their efficient runtime management. We have modeled an SCU at runtime and provided exemplary algorithm that utilizes operations for provisioning of elastic capabilities. We have shown that platforms supporting human computation in collective collaborations are more reliable by working based on the elasticity concept of scalability in terms of both resources and their parameters. Our future work includes further development of an SCU execution framework, which will include the presented model, metrics, API and algorithms so as to be able to deploy our approach in real environments.

Acknowledgments. This work is supported by the Vienna PhD School of Informatics (http://www.informatik.tuwien.ac.at/teaching/phdschool) and by the EU FP7 FET SmartSociety project(http://www.smart-society-project.eu/) under the Grant agreement n.600854.

References

1. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Power in unity: forming teams in large-scale community systems. In: CIKM, pp. 599–608 (2010)
2. Anagnostopoulos, A., Becchetti, L., Castillo, C., Gionis, A., Leonardi, S.: Online team formation in social networks. In: Proceedings of the 21st International Conference on World Wide Web, WWW 2012, pp. 839–848. ACM, New York (2012)
3. Bernstein, M.S., Brandt, J., Miller, R.C., Karger, D.R.: Crowds in two seconds: enabling realtime crowd-powered interfaces. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST 2011, pp. 33–42. ACM, New York (2011)

4. Bernstein, M.S., Karger, D.R., Miller, R.C., Brandt, J.: Analytic methods for optimizing realtime crowdsourcing. CoRR abs/1204.2995 (2012)
5. Dorn, C., Dustdar, S.: Composing near-optimal expert teams: A trade-off between skills and connectivity. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 472–489. Springer, Heidelberg (2010)
6. Dustdar, S., Bhattacharya, K.: The social compute unit. IEEE Internet Computing 15, 64–69 (2011)
7. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of elastic processes. IEEE Internet Computing 15(5), 66–71 (2011)
8. Dustdar, S., Truong, H.L.: Virtualizing software and humans for elastic processes in multiple clouds- a service management perspective. IJNGC 3(2) (2012)
9. Hexmoor, H., Chandran, R.: Delegations and Trust. International Journal of Computational Intelligence, Theory and Practice 3(2), 95–108 (2008)
10. Kaganer, E., Carmel, E., Hirschheim, R., Olsen, T.: Managing the human cloud. MITSloan Management Review 54(2), 23–32 (2013)
11. Kapuruge, M., Han, J., Colman, A., Kumara, I.: ROAD4SaaS: Scalable business service-based saaS applications. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 338–352. Springer, Heidelberg (2013)
12. Kasunic, M.: A Data Specification for Software Project Performance Measures: Results of a Collaboration on Performance Measurement. Technical report. Carnegie Mellon University, Software Engineering Institute (2008)
13. Lappas, T., Liu, K., Terzi, E.: Finding a team of experts in social networks. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2009, pp. 467–476. ACM, New York (2009)
14. Lopez, M., Vukovic, M., Laredo, J.: Peoplecloud service for enterprise crowdsourcing. In: 2010 IEEE International Conference on Services Computing, pp. 538–545 (2010)
15. Minder, P., Bernstein, A.: Crowdlang: programming human computation systems. Technical report (JAN (2012)
16. Psaier, H., Juszczyk, L., Skopik, F., Schall, D., Dustdar, S.: Runtime behavior monitoring and self-adaptation in service-oriented systems. In: Proceedings of the 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2010, pp. 164–173. IEEEComputerSociety, Washington, DC (2010)
17. Quinn, A.J., Bederson, B.B.: A taxonomy of distributed human computation
18. Sagar, A.B.: Modeling collaborative task execution in social networks. In: Potdar, V., Mukhopadhyay, D. (eds.) CUBE, pp. 664–669. ACM (2012)
19. Sengupta, B., Jain, A., Bhattacharya, K., Truong, H.-L., Dustdar, S.: Who do you call? Problem resolution through social compute units. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service Oriented Computing. LNCS, vol. 7636, pp. 48–62. Springer, Heidelberg (2012)
20. SmartSociety: Hybrid and diversity-aware collective adaptive systems: When people meet machines to build a smarter society,
`http://www.smart-society-project.eu/`
FP7 FET,EU Funded Project (accessed: December 20, 2013)
21. Truong, H.-L., Dustdar, S., Bhattacharya, K.: Programming hybrid services in the cloud. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service Oriented Computing. LNCS, vol. 7636, pp. 96–110. Springer, Heidelberg (2012),
`http://dblp.uni-trier.de/db/conf/icsoc/icsoc2012.html#TruongDB12`
22. Zhang, X., Kunjithapatham, A., Jeong, S., Gibbs, S.: Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. Mob. Netw. Appl. 16(3), 270–284 (2011)