A Social Compute Cloud: Allocating and Sharing Infrastructure Resources via Social Networks

Simon Caton, *Member, IEEE*, Christian Haas, *Member, IEEE*, Kyle Chard, *Member, IEEE*, Kris Bubendorfer, *Member, IEEE*, and Omer F. Rana, *Member, IEEE*

Abstract—Social network platforms have rapidly changed the way that people communicate and interact. They have enabled the establishment of, and participation in, digital communities as well as the representation, documentation and exploration of social relationships. We believe that as 'apps' become more sophisticated, it will become easier for users to share their own services, resources and data via social networks. To substantiate this, we present a social compute cloud where the provisioning of cloud infrastructure occurs through "friend" relationships. In a social compute cloud, resource owners offer virtualized containers on their personal computer(s) or smart device(s) to their social network. However, as users may have complex preference structures concerning with whom they do or do not wish to share their resources, we investigate, via simulation, how resources can be effectively allocated within a social community offering resources on a best effort basis. In the assessment of social resource allocation, we consider welfare, allocation fairness, and algorithmic runtime. The key findings of this work illustrate how social networks can be leveraged in the construction of cloud computing infrastructures and how resources can be allocated in the presence of user sharing preferences.

Index Terms—Social cloud computing, social networks, cloud computing, preference-based resource allocation

1 Introduction

Cloud computing has garnered praise for many reasons, most notably due to its ability to reduce overheads and costs for consumers by leveraging economies of scale to provide infrastructure, platforms and software as services. Infrastructure providers such as amazon elastic compute cloud (EC2) rid users of the burdens associated with purchasing and maintaining computer equipment; instead compute resources can be out-sourced to specialists and consumers can obtain access to an "unlimited" supply of resources. Despite its benefits, many businesses and end users are put off by an array of (perceived) uncertainties, as identified in numerous studies (e.g., [1], [2]). Two key issues are the notions of trust and accountability between resource consumers and providers [3]. In this context, trust and accountability encapsulate several different aspects such as security, privacy, ethical practices, transparency, protection of rights, and issues concerning compensation. Addressing these concerns is a significant undertaking, and consequentially, many international research programs have emerged, covering issues such as provider certification and service level agreements.

In this paper we argue an alternative approach to establish trust and accountability in cloud platforms: a social cloud [4]; and advocate a novel preference-based approach to facilitate resource sharing.

A social cloud is "*a resource and service sharing framework utilizing relationships established between members of a social network*" [5]. It is a dynamic environment through which (new) cloud-like provisioning scenarios can be established based upon the implicit levels of trust that transcend the inter-personal relationships digitally encoded within a social network. Leveraging social network platforms as mediators for the acquisition of a cloud infrastructure can be motivated through their widespread adoption, their size, and the extent to which they are used in modern society. For example, Facebook surpassed 1 billion users in 2012,[1] and has illustrated that Milgram's 6 degrees of freedom in social networks [6] may in fact be as low as 4 [7]. Users also spend inexorable amounts of time "on" social network platforms—a recent study indicated up to 1 in every 5 minutes of time spent online by all Internet users worldwide [8]. The computational social capital available is also significant: if only 0.5 percent of Facebook users provided CPU time on their personal compute resources the potential computational power available would be comparable to a www.top500.org supercomputer [9]. Examples of such sharing include: the 25 years of cycle stealing with Condor [10], the 16 years of volunteer computing since the Great Internet Mersenne Prime Search[2] and more recently Boinc [11];

- *S. Caton and C. Haas are with the Karlsruhe Service Research Institute and the Institute of Information Systems and Marketing, Karlsruhe Institute of Technology, Karlsruhe, Germany. E-mail: {simon.caton, ch.haas}@kit.edu.*
- *K. Chard is with the Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA. E-mail: kyle@ci.uchicago.edu.*
- *K. Bubendorfer is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. E-mail: kris.bubendorfer@ecs.vuw.ac.nz.*
- *O. Rana is with the School of Computer Science and Informatics, Cardiff University, Cardiff, United Kingdom. E-mail: o.f.rana@cs.cardiff.ac.uk.*

Manuscript received 14 Jul. 2013; revised 9 Dec. 2013; accepted 13 Jan. 2014. Date of publication 27 Jan. 2014; date of current version 17 Sept. 2014. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2014.2303091

1. http://newsroom.fb.com/content/default.aspx?NewsAreaId=22.

2. The first volunteer computing project was the Great Internet Mersenne Prime Search which started in 1996, mersenne.org.

which show users are willing to donate personal compute resources to "good" causes.

Our vision of the social cloud is motivated by the need of individuals or groups to access resources they are not in possession of, but that could be made available by connected peers. In this paper, we present a social compute cloud: a platform for sharing infrastructure resources within a social network. Using our approach, users can download and install a middleware (an extension to Seattle [12]), leverage their personal social network via a Facebook application, and provide resources to, or consume resources from, their friends through a social clearing house. We anticipate that resources in a social cloud will be shared because they are underutilized, idle, or made available altruistically.

In our earliest work [4], in which we first introduced the idea of the social cloud, our proof-of-concept was a social storage cloud. That prototype relied on a virtual credit model to regulate exchange and prevent freeloading. However a key aspect of a social cloud is the notion of sharing, not selling, resources. In this paper we have revisited the allocation model and moved away from a purely economic exchange to a model that emphasizes user choice. Specifically, due to the social network basis of a social cloud, users will have explicit preferences with whom their resources are allocated to, and from whom they consume resources. To support user preferences, we implement several algorithms for bidirectional preference-based resource allocation. We compare the runtime of these algorithms finding that for large numbers of participants and frequent allocations it may be impractical to compute allocations in real-time. We also study the effects of stochastic user participation (i.e., changing supply and demand) when instant reallocation may be impossible due to constraints on migration. We therefore introduce heuristics and compare their economic performance based on metrics such as social welfare and allocation fairness.

The rest of the paper is structured as follows: Section 2 presents the concept of a social compute cloud, the challenges with its construction, and its architecture. Section 3, describes the implementation of a social compute cloud entailing the core components and preference matching algorithms. In Section 4, we evaluate our approach (via simulation), placing a focus on allocation runtime and economic performance. We outline related work in Section 5 and conclude the paper in Section 6.

2 A Social Compute Cloud

A social compute cloud is designed to enable access to elastic compute capabilities provided through a cloud fabric constructed over resources contributed by socially connected peers. A social cloud is a form of community cloud (as defined in NIST's definition of cloud computing [13]), as the resources are owned, provided and consumed by members of a social community. Through this cloud infrastructure consumers are able to execute programs on virtualized resources that expose (secure) access to contributed resources, i.e., CPU time, memory and disk/storage. In this model, providers host sandboxed lightweight virtual machines (VM) on which consumers can execute applications, potentially in parallel, on their computing resources. While the concept of a social compute cloud can be applied to any type of virtualization environment in this paper we focus on lightweight programming (application level) virtualization as this considerably reduces overhead and the burden on providers. In [14] we explored the use of a more heavyweight virtualization environment based on Xen, however the time to create and contextualize VMs was shown to be considerable.

2.1 Challenges

There are many challenges in the construction of a social cloud that need to be carefully considered. In this section, we summarize several key challenges placing a focus on: the technical facilitation of the cloud platform, the inclusion as well as interpretation of social (network) structures, the design and implementation of appropriate socio-economic models for the facilitation of exchange as well as the platform infrastructure.

Technical facilitation to enable edge users to provide resources to, and consume resources from, one another. A social compute cloud needs to traverse network address translations (NAT), handle non-static IP addresses (especially in the case of mobile users) and accommodate best effort notions of quality of service. Although the concept of a social cloud is built upon the premise that users within a social network have some level of trust for one another, the construction of a social compute cloud still requires adequate security and sandboxing mechanisms to protect resources from potentially malicious or incompetent users and also to protect user applications from potentially malicious resources. This, in combination with a need to support multiple operating systems, can be partially addressed through virtualization.

Leveraging social structures to facilitate the sharing of compute resources within a social network. To utilize social structures for resource sharing, users must first allow access to their social network, and trust the platform with their social network data. Basing resource allocations upon a binary notion of friendship would be ill conceived for several reasons. First, social relationships are not simply edges in a graph. There are many different types of relationship (e.g., family, close friends, colleagues, acquaintances, etc.). Second, different users will associate different levels of trust to different relationship contexts. Third, different people have different qualities (e.g., reliability, trustworthiness, availability) and different competencies, for example users may assume that friends with computer science backgrounds are "better" or more "competent" with respect to offering compute resources. These three "relationship dimensions" mean that users may have very specific preferences with whom they interact, and these preferences may be different for consumer and provider roles.[3] A social cloud therefore requires additional metadata to augment the social graph

3. In [15], we took initial steps towards reviewing the contexts of trust for the scenario of a social cloud.

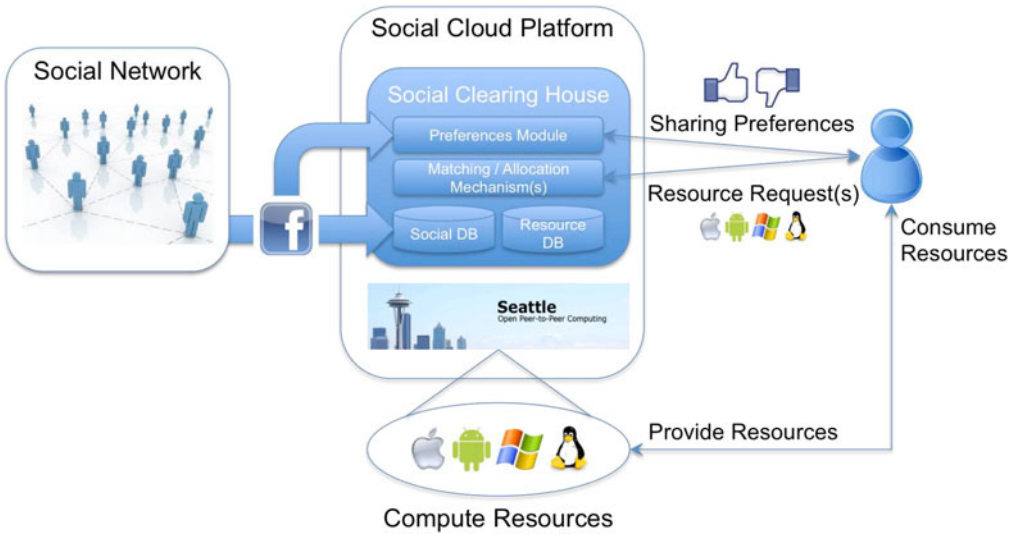

Fig. 1. A social compute cloud and its core components.

of its users so that it is possible to design a mechanism that can take into account the preferences and perceptions of users towards one another. The challenge here is the design of methods to extract these preferences either explicitly from the user or implicitly from their social network profiles.

A *socio-economic model* for resource allocation within a social compute cloud. Given that the concept of a social cloud focuses on the sharing rather than sale of resources, we do not focus on monetary models. However, a (microeconomic) system that acts as the meeting point for supply and demand is still required. The lack of an obvious economic setting makes the construction of a social compute cloud more, not less, challenging. Typically, the allocation of resources via a monetary exchange is based on the (private) valuations that users have for a good. This requires users to contemplate the value of their resources, and tasks they wish to perform, which dramatically changes their motivation(s). Using sharing preferences, the social context of exchange is accentuated. However, using only sharing preferences (or proxies thereof) as a means of determining resource allocation along with the social ties of a user dramatically complicates the allocation process. The challenge is not to perform an allocation, but to allocate resources effectively. In other words, avoiding computationally intensive allocation methods, but retaining an equilibrium between resource allocations and platform liquidity.

Platform facilitation of a social compute cloud. As the model is based upon the (altruistic) sharing of resources, the notion of a revenue model (or similar) to support the platform and its basic functionality is problematic. Users should not have to pay for the services offered by a social cloud platform. However, the platform requires computational resources to function. As we do not address this challenge in this paper, we refer instead to [16] where we present a co-operative model for the construction of a social cloud platform. A co-operative model implies that resources are provided with the intent to be shared, but also to support the platform itself in the form of infrastructure resources.

2.2 Architecting a Social Compute Cloud

In line with the challenges outlined above, we identify three areas of functionality needed for the construction of a social compute cloud: *A Social Cloud Platform*: the technical implementation for the construction and facilitation of the social cloud as well as necessary middleware to enable resource sharing between "friends" at the edges of the internet. *A socio-technical adapter*: the means to observe and interpret social ties for the elicitation or derivation of sharing preferences. *A socio-economic model*: the formulation of a social microeconomic system for the allocation of resources upon the premises of social ties, and preferences with respect to how social ties denote a user-specific willingness to consume and/or provide resources.

2.2.1 A Social Compute Cloud Platform

Like any cloud model, a platform is required to coordinate and facilitate its basic functionality (user management, resource allocation, etc.). Fig. 1 shows the high level architecture for a social compute cloud and its key components, which are explained as follows:

A *social clearing house* is an institutionalized microeconomic system that defines how supply is allocated to demand. Smith defined the key components of a microeconomic system for the purposes of exchange in [17]. However, this definition is orientated primarily for monetary-based exchanges, which is not the case here. Therefore, a social clearing house captures the following: the protocols used for distributed resource allocation, the rules of exchange, i.e., who can take part and with whom may they exchange, and the formalization of one or more allocation mechanisms. A social clearing house is therefore the central point in the system where all information concerning users, their sharing preferences, and their resource supply and demand is kept. For this reason, the social clearing house requires two databases: to capture the social graph of its users, as well as their sharing preferences, and a resource manager to keep track of resource reservations, availability, and allocations.

A *middleware* to provide the basic resource fabrics, resource virtualization and sandboxing mechanisms for provisioning and consuming resources. It should also define the protocols needed for users and resources to join and leave the system. For these purposes we selected Seattle [12], [18] as it largely provides the needed functionality. However, Seattle cannot allocate its resources based upon social ties, and was thus extended.

A *socio-technical adapter*, which in our case is a Facebook application, is needed to provide access to the necessary aspects of users' social networks, and acts as a means of authentication, for example, via Facebook connect. Once a user's social network has been acquired via the socio-technical adapter, the social clearing house requires the sharing preferences of the user to facilitate resource allocation. Therefore, a *preferences module* that provides the necessary functionality for the capture and representation of sharing preferences is required. Many aspects of a socio-technical adapter require careful consideration, and many methods can be applied to capture preferences, we discuss these in Section 2.2.2.

Matching mechanisms are socio-economic implementations of the social clearing house microeconomic system. They determine appropriate allocations of resources via users' sharing preferences across their social network, and are discussed in Sections 2.2.3 and 3.3.1.

Compute resources are the technical endowment of users that they provide to and consume from the social cloud. Here, resources largely entail personal computers, servers or clusters. However, we note that the latter is unlikely for the average user. We envisage that as the computing industry continues to invest in mobile computing devices that such devices could also be offered within a social cloud in the future. Today, however, issues such as network stability and battery life hamper their inclusion. However, despite this researchers are making notable progress in this area; see [19].

2.2.2 Social Adapters and User Preferences

To facilitate social sharing, and the construction of sharing preferences, a social cloud requires access to users' social networks. We propose using a social adapter, rather than implementing the platform as a social network application (for example a Facebook application), as we have observed that users often misunderstand the separation between social networks and their applications [20]. The most common misconception is that the social network will have access to users' data and/or resources if they offer them in a social cloud-like setting via a social network application, which is not the case. For example, Facebook applications are external to the Facebook infrastructure, and run on third party servers.

There are different ways in which the social graph required by matching mechanisms can be constructed; many platforms, following the necessary authorization, provide APIs to access the social graph and user profiles. It is, however, key to the basic assumptions of a social cloud that an element of bilateral approval has occurred in the establishment of a digital social tie. In other words, one user initiates the establishment of a digital tie, and the second user must confirm the request in order for the link to be established. This process is applied in social network platforms like Facebook, and Google+. Twitter, however, does not conform to this requirement, as a user can decide only who they follow, but not who their followers are. This is an important requirement for a social cloud, as without it we can not assume any form of pre-existent trust between participants.

Once the social network of a user has been accessed and the social database populated, the question is how to interpret the user's social ties for the purposes of allocation. There is no single unified methodology for the interpretation of social ties, and which to use is often context dependent. For our purposes, there are three obvious methods which could be applied either separately or in combination with one another: 1) ask users to rank their friends; 2) leverage methods from social network analysis to identify features of social ties that can be used to (artificially) construct preferences; and 3) use social network and interaction theories to construct a social sharing and interaction model for a social compute cloud, and tune this model over time based upon observed interactions within the social network platform and the social cloud. Each of these approaches have their advantages and disadvantages, and we do not advocate that this list is complete.

The use of user generated lists, has the advantages that it is easy to implement, requires no special permissions (other than access to the list of friends), and should be closest to capturing the true preferences of the user. However, given that the average Facebook user currently has 190 friends [7], this approach would not scale as more friends joined the social cloud, as we cannot expect users to rank large numbers of friends. In contrast, the use of computational methods has the main advantage that these approaches can be scaled as the social cloud grows. The challenge, however, is in the identification of appropriate methods and indicators. These approaches also require more data from the social network platform, and are thus more invasive of a user's privacy. A simple example that can be used in a preference-like manner, are constructs like circles in Google+ or relationship lists in Facebook, as these are often created or at least curated by the user, and represent either specific (sub) groups in the social network and/or relationship types that are "similar" in some way. It is also possible to compose more complicated methods of assessing social ties with the use of indicators to assess the properties of a social tie. Identifying the best implementation(s) of the preferences module, however, remains as future work.

Two important points that cannot be overlooked, regardless of the methodology used to define preferences, are the kinds of preferences that exist, and what the absence of preferences towards a user or group of users mean. Preferences can be categorized as complete or incomplete, and being with or without ties [21], [22]. Here complete means that a preference rank exists between every connected user; and a tie implies that a user is indifferent between two users or a group of users. The case of complete preferences without ties imposes a strictly ordered preference ranking over all the other users, and is the standard case in most of the literature. If preferences either have indifferences or are incomplete, the corresponding problem of matching the preferences to an outcome becomes considerably harder (i.e., NP-hard). The absence of preferences towards users,

i.e., incomplete preferences, occurs if users either explicitly don't rank other users, thereby indicating they don't want to be matched with them, or if a user does not rank all other users due to lack of time or motivation. Both cases yield incomplete preferences, but occur for quite different reasons. To better distinguish these two cases, the implementation of a preference module has to either indicate that missing preferences mean an unwillingness to be matched or assume that missing users have the lowest preference. In other words, users either define who they are willing to share with, or "block" users.

2.2.3 Socio-Economic Model

A socio-economic model for a social compute cloud specifies which type of preference matching is used and how it is implemented. As a first step, the supply and demand, i.e., the individual requests and resource offers of users, have to be captured. In our model this is done in the social clearing house. The centralized implementation means we know the complete supply and demand in the market, which can yield better results than decentralized resource matching. The downside is that there may be additional overhead in storing and managing (updating) the corresponding information.

Given the supply and demand, a socio-economic model allows for the specification of certain "Market Design" objectives. For example, commonly used objectives include finding solutions to the matching problem which are stable (i.e., no matched user has an incentive to deviate from the solution) or optimizing the total welfare of the users, the fairness between the two sides of the market, or the computation time to find a solution. The choice of particular market objectives in turn affects which allocation and matching strategies can be considered. This can range from direct negotiation to a centralized instance that computes this matching; and both monetary and non-monetary mechanisms can be applied. Our approach considers non-monetary allocation mechanisms based on user preferences. This type of matching is successfully applied in a variety of cases, including the admission of students to colleges, and prospective pupils to schools. Depending on the specific market objective, several algorithms exist that compute a solution to the matching problem, e.g., computing a particularly fair solution or one with a high user welfare.

3 Implementation

Our implementation of a social compute cloud builds upon Seattle, an open source peer-to-peer (P2P) computing platform. Seattle was chosen as the basis for this implementation due to its lightweight virtualization middleware, which we use to enable application execution on contributed resources, and its extensible clearing house model which we extend to enable social allocation via preference matching algorithms.

3.1 Seattle

Seattle is an open source educational research platform designed to create a distributed overlay network over compute resources (servers, PCs and mobile devices) donated by its users. It features a lightweight virtualization layer (based on *Repy*—a subset of Python) that runs on a contributor's machine and enables other users to run applications across different operating systems and architectures. Importantly the virtualization layer ensures that applications are sandboxed and isolated from other programs running on the same host. Seattle is implemented in Python and the clearing house is built on the Django framework. Seattle's core components are:

Node managers act as gatekeepers for resources and are deployed on every contributed resource. The node manager ensures that users have the appropriate credentials to interact with a particular VM running on the host system. When a node manager is installed it advertises the location of the host machine to a global lookup service. A unique key for the resource is created and logged at the clearing house to associate the node manager with the donor.

Virtual machines (vessel) are sandboxed environments that provide both security and performance isolation. For example the VM stops applications from performing malicious actions and it limits usage of system resources (e.g., CPU or memory) to configurable levels. These limitations are imposed through integration with the Python parser which reads the program's parse tree and ensures that only predefined 'safe' operations are executed. Performance isolation relies on monitored resources at the API level to verify that resource limits are not exceeded. This is done by analyzing every call to the API and choosing to either accept or deny it based on resource usage.

The *clearing house* facilitates the matching process between resources donated by providers and resources required by consumers. The clearing house is a web based portal for managing users' Seattle resources. Upon registration, users can create key pairs and download a customized installer to setup a node manager and VM system on their own resources. The clearing house includes several matching algorithms such as: distributing VMs (geographically), and allocating VMs on the same network or at random.

Seattle includes a number of other services that are used for various functions. For example, Seattle includes a global lookup service that is designed to enable the discovery of contributed resources; a software updater to patch potential vulnerabilities and update parts of the system; and a range of infrastructure services such as NAT traversal, installers for bundling applications, and monitoring services that are used throughout the system.

We chose to extend the Seattle platform due to its open source and extensible architecture. The modular design of Seattle enables the use of its existing virtualization environment to execute applications on distributed resources handling aspects such as lookup, authentication and authorization. The virtualization layer is also extremely lightweight and can be downloaded, installed and configured in seconds without any significant overhead and with only a small system footprint. The clearing house model is generic and extensible providing support for user defined allocation protocols and integration of arbitrary information sources in its allocation process.

3.2 Implementing a Social Clearing House

Building upon Seattle we leverage the same base implementation for account creation and registration processes,

Fig. 2. The user preference interface.

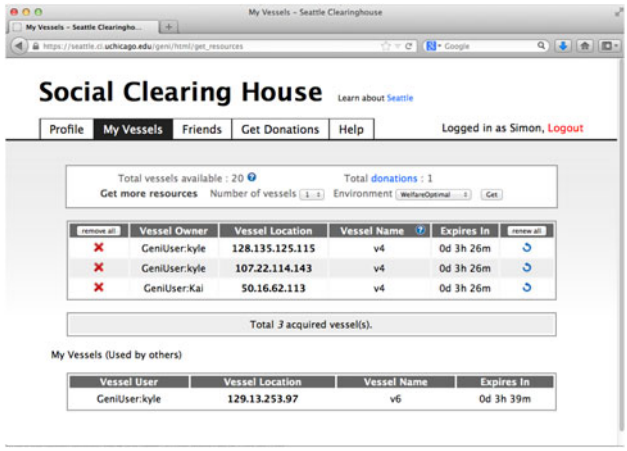

Fig. 3. The social clearing house interface showing resources being consumed and offered.

donation infrastructure, and resource acquisition mechanisms. We have extended and deployed a new social clearing house (https://seattle.ci.uchicago.edu) that leverages social information derived from users' Facebook profiles and relationships. We have implemented a service that enables users to define preferences and we have developed several new allocation mechanisms that utilize socially aware preference matching algorithms.

3.2.1 Social Network Integration

In order to access users' profile information and relationships, the social clearing house requires access to a user's Facebook profile. To do so, we have created a Facebook application for the social clearing house that requests access to profile information, friends and friend lists of registered users. The Facebook application is integrated with the clearing house through the Django social auth plugin which, when configured with a Facebook application, allows users to associate their Seattle account with their Facebook account. Authentication with Facebook uses the OAuth2 protocol to obtain an access token that allows the requesting application (the clearing house) to act on behalf of the user within the stated *scope*. The clearing house stores this access token when a user logs into the service and uses it with the Facebook APIs to obtain the profile and friend lists. The clearing house stores the list of friends for each user in an application database and periodically refreshes this information.

3.2.2 Preference Assignment

We use a simple numerical preference matching interface (see Fig. 2) that enables users to define their preference for a friend as both a consumer and a provider. The higher the value the greater the users' preference for their friend. A value of 0 indicates no preference and a negative value indicates unwillingness to interact with that friend. Assigning the same value to multiple friends indicates indifference between them. When preferences are assigned they are stored in the application database and are used to generate the overall preference model for allocation involving the user.

3.2.3 Social Resource Allocation

Seattle is based on the principle of best effort and random allocation. To reduce the search space Seattle implements a pseudo random mechanism to reduce user/donation permutations. Each user is assigned a (non-unique) port (within a range of 100) and each donated node is assigned 10 ports at random within this same range on which it is able to accept allocations. When allocating resources the clearing house filters the list of donated resources by matching port. While this is a viable approach in the standard Seattle deployment it places artificial limitations on potential matches and in our case violates the aim to provide preferred sharing. In the social clearing house we have removed this constraint and instead allow any combination of node and user as determined by the social allocation algorithms.

The general process of allocation in the social clearing house is to first determine available donations with whom the requesting user has a relationship. To do this the list of all donations in the system is filtered by the list of friends for a particular user. The consumer's preferences for each possible friend is then computed by retrieving preferences stored in the database. Likewise the preferences for each of these friends for the requesting user as a consumer are computed. This information is then aggregated and sent to the matching service (Section 3.3.2) to determine an appropriate match. The social clearing house attempts to acquire available nodes from the provider to satisfy the request using Seattle's resource acquisition mechanisms. If, by the time of reservation, the chosen provider is no longer available the entire process must be re-executed. After the allocation process the user is presented with the VMs they have been allocated (see Fig. 3). Using this interface, users can renew resource reservations as well as remove them and also see who is currently using their resources.

3.3 Preference-Based Matching

Two-sided preference-based matching is much studied in economic literature, and as such algorithms in this domain can be applied in many other settings. We have selected three algorithms from the literature, and a fourth of our own implementation.

3.3.1 Matching Algorithms

For the case of complete preference rankings without indifferences there are polynomial-time algorithms that

solve the matching problem for different objective functions. In the literature, citing empirical evidence, *stability* is considered important for successful matches [22]. Stability simply means that there is no pair of users who would prefer to be matched over their current match. As there can be many different stable solutions for a given matching problem the other commonly considered objectives are welfare (i.e., the average rank each user is matched with) and fairness (i.e., if the two sides are treated equally in terms of welfare). The *deferred-acceptance (DA)* algorithm [23] is the best known algorithm for two-sided matching and has the advantages of having a short runtime and at the same time always yields a stable solution. However, it cannot provide guarantees about welfare, and yields a particularly unfair solution (one side gets the best stable solution whereas the other side gets the worst stable solution). For certain preference structures, the *welfare-optimal (WO)* algorithm [24] yields the stable solution with the best welfare score (i.e., the stable solution for which the average rank that each user is matched with is lowest) by using certain structures of the set of stable solutions and applying graph-based algorithms. DA and WO are two standard approaches used in the literature and are also considered in this paper.

As soon as either indifferences or incomplete lists are introduced, the problem of finding stable solutions with additional properties such as welfare or fairness becomes NP-hard. DA and WO can still be used in such settings, but they can no longer guarantee to find the globally best solution. In such settings, the approximation algorithm *Shift* [21] can find a stable match, with the maximum number of matched pairs for certain special cases. However, these scenarios are in general hard to approximate, and consequently the standard algorithms are not able to provide non-trivial quality bounds with respect to their objectives. Finding the optimal solution for the matching problem with respect to the most common metrics: welfare or fairness, is NP-hard [21]. DA and WO run in polynomial time ($O(n^2)$ and $O(n^4)$, respectively) and Shift's runtime is proportional to the squared length of the largest indifference group of all users.

Therefore, we proposed the use of heuristic algorithms such as a genetic algorithm (GA) in [25], and have shown that these algorithms can yield superior solutions compared to the other algorithms. The GA starts with randomly created (but stable) solutions and uses the standard mutation and crossover operators to increase the quality of the solutions. This makes the application of such heuristics the preferred choice if the quality of the allocation is the main goal. We also showed that solutions can yield even better results when combined with a threshold acceptance approach. The algorithm used in this paper, GATA, is a combination of a GA with a threshold acceptance (TA) algorithm, which further improves the solution quality. In the first step, GATA computes a solution to the matching problem by using GA, and then uses this solution as input for the TA algorithm, an effective local search heuristic that applies and accepts small changes within a certain threshold of the current solution performance.

3.3.2 Matching Service

To facilitate matching, we implemented a RESTful encapsulation of the four algorithms presented above. It can be used to either perform batch allocations for a group of users, or single allocation for an individual user. Whilst it may seem unusual to facilitate both of these settings, the reason is simple: the matching algorithms perform best when batches of users are allocated simultaneously. In reality, however, it is unlikely that large batches of users will simultaneously request resources. Rather, demand for the matching service will be stochastic. Hence both options present different tradeoffs. Individual allocations may result in resources being blocked for other users, for example those with a small number of connections. Whereas batch allocation means that users may have to wait until the next round of allocations to receive resources. Both options are inefficient in different ways: individual allocation achieves at best local optima, and can block resources for other users, but can be performed in near real time, as the computational effort is significantly lower; batch allocations could achieve the global optimum, but may require either migrations or users to wait for resources. We explore these tradeoffs and their significance in Section 4.

The social network of users is captured via the existence of preferences between users. The matching mechanisms will only consider matching two users if both have a preference for each other. If a preference exists in only one direction, i.e., A has ranked B, but B has not ranked A, we assume that B has not yet considered A, and A's preference for B will be ignored.

To invoke the matching service, a JSON string describing the user preferences is sent to the service. Upon a successful match, i.e., when one or more allocations can be found, the matching service will return a JSON string describing the matched consumer(s) and provider(s).

4 Evaluation

We showed in [4], [5] and [16] that the basic concept of a social cloud is feasible within the context of a social network and manageable for an "average" Facebook user. To evaluate a social compute cloud, we propose to study the platform's ability to allocate resources in the presence of uncertain supply and demand, various sizes of social communities and different preference structures. In Section 4.1, we study the different allocation algorithms described in Section 3.3.1 with respect to the time required to compute solutions for various sizes of social compute cloud. In Section 4.2 we investigate algorithm performance outside their typical settings with respect to batch and individual allocation and for different community sizes and preference structures.

To facilitate our evaluation, we use the social cloud simulator described in [26]. Using this simulator, we can adjust the properties of a social cloud as described above while still replicating real user resource availability distributions derived from donations and resource availability in SETI@home [27]. The data from SETI@home represents statistical clusters of users, that can be used in our simulator to define both when a resource will become available and for how long, as well as when users will request resources. As we do not know at this stage, how scalable the allocation methods

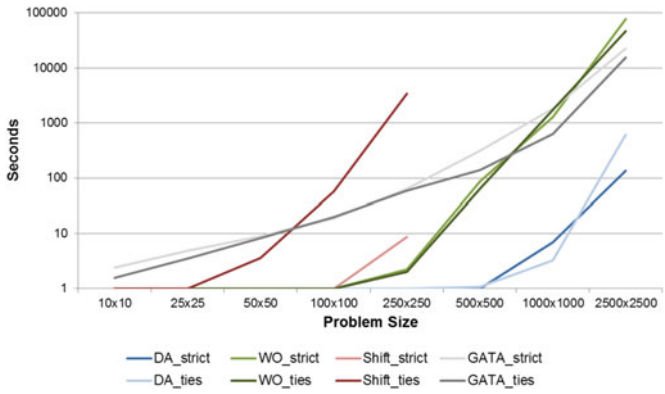

Fig. 4. Algorithm runtimes with different problem sizes.

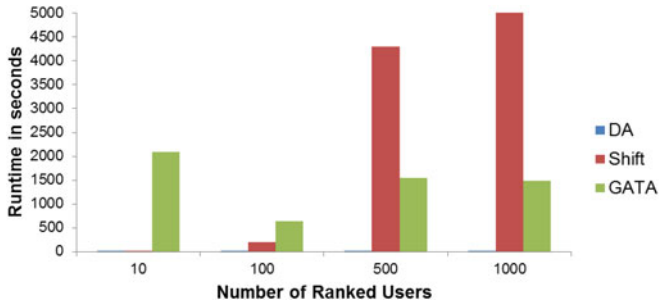

Fig. 5. Runtimes for different numbers of ranked users, for 1,000 users per side.

will be, or how easy they will be to parallelize, we run up to eight instances of the simulator in parallel on two 2.53 GHz Quad-Core Xeon processors with 24 GB main memory.

4.1 Allocation Algorithm Runtime

The runtime of an allocation algorithm has a large impact on its applicability for a social compute cloud. Given that preference-based matching is often NP-hard, algorithm runtime is an important design consideration. In this part of the evaluation, we investigate how algorithm runtime is affected by the level of preference completeness, and whether preferences are strict or not, i.e., whether indifferences are permitted. We argue that although complete and strict preferences are often assumed in the economic literature, they are unrealistic assumptions. Therefore, we investigate how relaxing these assumptions impact algorithm runtime.

Fig. 4 shows the runtime of each algorithm relative to the problem size, i.e., how many users are on each market side (consumers $\times$ providers), and the number of (indifference) groups that users have. Here, "strict" indicates that preferences are strict, i.e., there are no indifferences, and "ties" indicates that users have indifferences in their preferences. The size of an indifference group is not restricted, i.e., there can be one group containing all users or multiple groups. This is determined at random.

The figure shows that runtime increases significantly with the size of the problem space. Theoretical results suggest that DA runs in $O(n^2)$, and WO in $O(n^4)$, indeed a sharp increase in runtime can be observed especially for WO, yet even DA takes several seconds to compute for larger problem sizes. Furthermore, the heuristic GATA exhibits a longer runtime than DA and WO for smaller problem sizes, but its relative runtime improves if users have indifferences, i.e., if the setting is more realistic. The longer runtime for smaller instances is, most likely, due to the initialization steps necessary for the GA. Fig. 4 illustrates the considerable overhead of the approximation algorithm (Shift), as its runtime depends on the length of indifference lists. As the length of the largest indifference list is not limited, we were only able to calculate up to a problem size of 250×250 users. Hence, especially in realistic settings, it is not a feasible option.

For users with a large number of friends participating in a social compute cloud, it may not be feasible to rank everyone. Therefore Fig. 5 depicts the scenario where users rank different numbers of their friends, and do not provide a complete set of rankings. The scenario assumes 1,000 users on each side and varies the number of friends ranked by each user (to create a realistic setting, we allowed up to 10 users per indifference group). Note that WO is not depicted, as it has not been developed for incomplete preferences with ties.

The runtime for DA and Shift increases if users explicitly provide rankings for more users. For Shift the increase is considerable. GATA, however, seems to take longer if only a few users are ranked. This is most likely because the heuristic has to repeatedly check if new potential solutions are acceptable to the users, i.e., if the users of each matched pair appear in each others' preference list. By increasing the number of users, new, valid solutions can be calculated more easily and quickly, yet it takes longer to evaluate the solutions. These results show that in addition to the number of users, the number of preference rankings also influences computation time. The results illustrate that appropriate actions are needed when users do not rank all their friends. They can either be ranked with the lowest preference (beneficial for GATA) or left out (beneficial for DA and WO). Note, however, that in case unranked users are left out, the algorithms cannot guarantee to find a solution of maximum size, i.e., some users might be left unmatched whereas they would be matched if unranked users receive the lowest preference rank.

Overall, we see that the even the fastest algorithms which always yield stable results take at least several seconds to compute. If the allocation has to be computed very frequently, as is potentially the case in a social cloud, we require faster algorithms.

4.2 Considerations for Stochastic Participation

The matching algorithms discussed in related literature are usually assumed to be batch jobs. In this case, allocations are computed after certain time intervals, e.g., every hour. For example, economic studies of allocation mechanisms in cloud computing often assume that allocations are computed hourly, often referring to Amazon EC2 where users buy resources based on hourly usage. In the case of our social compute cloud, we can say that the allocation is calculated every x hours, where x is the predetermined lease period of a compute vessel.

While this type of allocation computation yields good results for the supply and demand given at the time of the computation, it is unclear what happens in the case of new

or changing supply and demand. For example, when users offer/request new resources or retract offers/requests in between two calculation intervals. It is clear that if allocations are only (re)computed at predetermined time intervals, resources will be idle and requests will be left (or become) unsatisfied. As far as we are aware, existing preference-based matching literature does not consider such settings. We therefore propose the following solutions for dynamic supply and demand:

The "optimal" solution is *immediate rematching* of the entire supply and demand. In this case, no (new) supply would be idle if an allocation was available, and the resulting allocation would always be stable (assuming we use a stable matching algorithm). However, this places additional requirements on the system. First, computing resources (e.g., VM's) would have to be migratable at any given point in time, and, second, the runtime of the allocation mechanism has to be short. For our implementation, this is currently unachievable, as Seattle does not yet support migration. Hence, this approach should be considered the best benchmark from a system perspective, i.e., with respect to performance criteria such as stability, welfare, and fairness.

The "worst" solution would be to *disregard* any new incoming supply or demand until the next time the allocation is computed. In this case, new supply/demand would be idle until the next batch allocation, even if there were corresponding demand/supply.

An intermediate solution would be to use allocation heuristics that assume that the currently matched supply/demand cannot be migrated until either the next time the overall allocation is determined (i.e., when the leases expire), or until the requester or supplier drops out of the matching. We consider two cases of heuristics for this case: 1) *Random*: allocate resources to/from random friends of the user; and 2) *Greedy*: allocate the best available match for the incoming, requester/provider based upon their sharing preferences.

Note that, both algorithms are likely to yield unstable solutions, i.e., the consumer-provider pairs in the market at the end of the lease period would not be the pairs that a stable allocation algorithm would yield. However, if we assume that matched pairs cannot be migrated in between two batch allocations, this does not, per se, affect the practical stability in between batch computations. We also note that a lack of system stability may mean that some users are blocked as all valid allocations have been made, and none of their friends have or need resources. This is why stability is such a critical metric.

A final approach is to check if there is a match that would yield a stable solution, but not require other users to be reallocated. We argue that this approach would be subsumed by the above approaches, as the probability of achieving a stable solution is low, and in the absence of a stable match, another approach would be applied.

To study stochastic supply and demand, we simulated the four approaches mentioned above to study how they support the system with new supply/demand in between two batch-allocation computations. Intuitively, immediate rematching should yield the best solutions, whereas leaving resources idle should be worst. Random and Greedy should be somewhere in between.

Fig. 6 shows the simulation results for each approach. We simulated 200 users (100 on each side) with incomplete preferences and indifferences. This is a realistic setting as not each user can be expected to rank all other users. Each user is drawn an (un)availability distribution from the SETI@-home distribution, which determines when and how long they will be (un)available. Only available users are taken into account for resource matching. At time points 155, 265 and 410, the batch allocation algorithm is run for the current supply/demand.

Fig. 6a shows that the number of matched users fluctuates over time as users come and go based on their (un) availability pattern. It can be seen that, most of the time, the "optimal" matching yields the highest number of matched pairs, and both Random and Greedy yield fewer matched pairs. This can be explained by the lack of choice that incoming users have: in Random and Greedy, only the currently unmatched users are suitable for matching, whereas the "optimal" rematching can consider all available users at that time. This gives the "optimal" matching more flexibility in finding suitable matches for users. Fig. 6a also shows the baseline scenario in which allocation only happens at predetermined time intervals. In this case, new requests and offers are only considered at predetermined time intervals (here: time points 155, 265 and 410), and if matched users become unavailable, the corresponding request/offer is freed without being automatically reallocated. Hence, the baseline scenario depicts the worst case, a quasi-static scenario where intermediate demand is not considered. It can be easily seen in Fig. 6a that not considering intermediate supply/demand can lead to a significant amount of unused, unallocated resources, and that even simple heuristics for intermediate matching can increase the number of matched pairs considerably.

Figs. 6b, 6c, and 6d show the results for stability, welfare and fairness respectively. In these figures welfare represents the average rank for a friend with whom a user is matched and fairness shows the welfare distribution between consumers and providers. As more preferred options have a lower preference score (the most preferred option has score 1), a lower welfare score is preferable. A positive fairness indicates better average welfare for the provider, whereas negative fairness indicates a better average welfare for consumers.

These figures show that immediate rematching performs well with respect to welfare (allocating users close to their highest preference) and fairness (balancing the two market sides), and always yields stable outcomes. This is partially expected, as stability is enforced through the algorithm. Furthermore, both Random and Greedy strategies lead to a worse welfare score, as only unallocated resources are considered for matching whereas the "optimal" algorithm allows rematching of previously allocated resources. The Greedy strategy seems to provide better welfare than the Random strategy (Fig. 6b), and at the same time is computationally as efficient. While the runtime for GATA per allocation is around 10 seconds, both Random and Greedy run almost instantly (milliseconds). We get similar results for

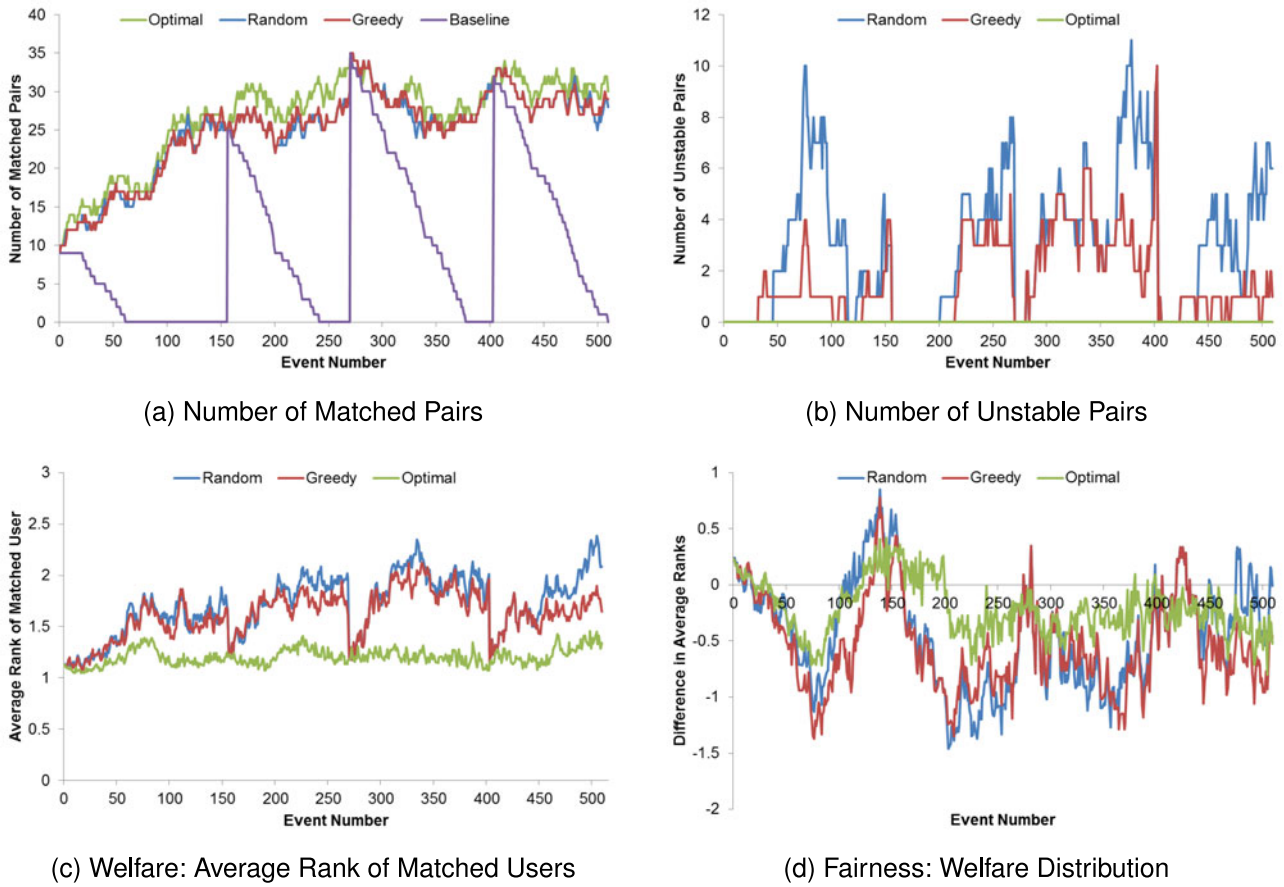

Fig. 6. Comparison of matching heuristics for intermediate supply and demand.

the number of unstable pairs, which are most often lower for the Greedy strategy than the Random strategy.

Fig. 6d shows that both the Greedy and Random strategies seem to be more beneficial for consumers, indicated by the lower fairness scores, especially compared to the optimal matching. The number of unstable pairs introduced by Greedy or Random is comparably low, given that each new allocated pair has to be compared to all existing matched pairs for instability. (For example, if a new pair is added to 30 existing matched pairs, in the worst case the new pair could introduce up to 60 unstable pairs). This finding, along with the fact that Greedy can sometimes yield worse results for welfare as well, is not surprising as it primarily aims to give new, incoming users their highest priority, without considering the preferences of other users. Such a strategy can yield matches that are good for the incoming user, yet are not optimal for the system. For example, if there are two open offers, the Greedy strategy would allocate the most preferred one to the incoming user, whereas it could be better for the market to reserve this particular offer for the next incoming user with a higher ranking.

Overall, the results suggest that approaches for the intermediate rematching of supply/demand are necessary, and that on average the Greedy heuristic performs well with respect to welfare, despite one side being favored in the matching. This is especially interesting if immediate rematching is not technically feasible.

4.3 Discussion

The social compute cloud facilitates preference-based sharing of computational infrastructure using several different preference matching algorithms. Our prototype implementation leverages the Seattle virtualization middleware to enable execution of user applications on remote resources and our deployed clearing house enables users to define preferences and provides several matching algorithms to obtain a short term resource lease. Our results show the qualities of the algorithms and the tradeoffs that arise from factors like runtime, allocation mode, and allocation quality.

Section 4.1 shows a tradeoff between the allocation quality (in terms of the objective function) and the runtime of the respective algorithms. This is particularly clear for larger problem sizes. DA is the most time-efficient algorithm to obtain a stable allocation, whereas the other algorithms' runtimes increase considerably with problem size. However, it is well-known that the stable allocation calculated by DA is highly unfair in the sense that it favors one side over the other. As we show in [25], solution quality with respect to fairness or welfare can be significantly increased by using the other algorithms. Hence, using algorithms such as GATA is preferable for smaller settings or when there are no time constraints, whereas using fast algorithms such as DA might be better if allocations need to be made more frequently.

Section 4.2 shows convincing evidence that we must consider supply and demand changing between batch

allocation times. While these results show that the solution quality tends to be very good and even close to optimal it is clear that continuously running the algorithms might not be feasible due to their computational overhead. Hence, we need fast heuristics that are able to deal with changing supply and demand, although these heuristics usually lack the solution quality of the other algorithms. For small problem sizes, it might still be feasible to run algorithms such as the GATA in a continuous setting.

One potential strategy to improve both allocation quality and runtime would be to compute an initial solution with a fast algorithm, e.g., DA, and then leverage users' provided computational power to improve solution quality. This would give users an incentive to provide resources for a cooperative infrastructure [16].

Another issue with the matching algorithms considered in this paper is that they currently support only one-to-one matchings, i.e., they do not yet support multi-unit allocations. In some settings, users might contribute or request multiple units of resources (e.g., several VMs to run a compute-intensive job). One strategy to deal with this problem would be to extend the algorithms to handle these cases. Another simple strategy would be to include users with multiple units of supply or demand as multiple entries in the matching problem, and solve it with the algorithms discussed in this paper.

In this paper, we have focused on methods for the allocation of resources within a social cloud. However, once users have been allocated one or more resources, they need to use their social cloud to create dynamic clouds and build distributed applications. Seattle already provides a demokit where applications similar to these can be implemented (see: https://seattle.poly.edu/wiki/EducatorsPage). For instance, a social cloud user could easily instantiate a MapReduce framework running on socially connected nodes. A master worker scenario similar to that discussed in [28] could also be constructed. Finally, peer-to-peer like social content delivery networks (see [29], [30]) for the sharing or distribution of large scientific data could be implemented. A core aspect of our future work is the construction of a Seattle-based toolkit for social cloud applications.

5 Related Work

With the increasing pervasiveness of social network platforms, adoption of social network structures for different types of collaboration is becoming more common. Key examples are: community and scientific portals like PolarGRID [31] and ASPEN [32]; social science gateways [33], [34]; social storage systems like Friendstore [35] and omemo.com; network and compute infrastructure sharing web sites such as fon.com; models to share insurance policies amongst social peers (friendsurance.de); and where social networks emerge due via collaboration, e.g., [36], [37].

McMahon and Milenkovic [38] proposed social volunteer computing, an extension of traditional volunteer computing, where consumers of resources have underlying social relationships with providers. This approach is similar in nature to a social compute cloud, but it does not consider the actual sharing of resources, as there is no notion of bilateral exchange.

Ali et al. [39] present the application of our social cloud model to enable users in developing countries to share access to virtual machines through platforms like Amazon EC2. In effect they subdivide existing allocations to amortize instance cost over a wider group of users. Using a cloud bartering model (similar to our previous virtual credit model), the system enables resource sharing using social networks without the exchange of money and relying on a notion of trust to avoid free riding. Like our approach, they use a virtual container (LXC) to provide virtualization within the existing virtual machine instance, however our approach using Seattle's programming level virtualization provides a much more lightweight model at the expense of flexibility.

Mohaisen et al. [40] present an extension to our definition of a social cloud. The authors investigate how a social compute cloud could be designed, and propose extensions to several well known scheduling mechanisms for task assignments. Their approach considers resource endowment, and physical network structure as core factors in the allocation problem, which are different considerations for resource allocation. They analyse the potential of a social cloud via simulation, using several co-authorship and friendship networks as input. They observe how a social cloud performs based upon variations in load, participation and graph structure.

Tan et al. [41] present a similar idea to the basic concept of a social cloud. The authors, although not extending beyond a conceptualisation, motivate the philosophy of a social cloud with the core use case of sharing and exchanging resources within a social network or community to tackle big data problems.

Gracia-Tinedo et al. [42], [43], [44] propose a friend-to-friend cloud storage solution, i.e., dropbox via a social network: F2Box. They analyze and discuss how to retain a reliable service whilst using the best effort provisioning of storage resources from friends. They identify that a pure friend-to-friend system cannot compare in terms of quality of service with traditional storage services. Therefore, they propose a hybrid approach where reliability and availability can be improved using services like Amazon S3. This approach provides a valuable consideration in the realisation of a social cloud, but is not necessarily transferable to our setting.

There have been several publications on economic models for a social cloud that have developed independently. Zhang and van der Schaar [45] and we [46] discuss different types of incentives users face during their participation in a social cloud, and describe the challenges of providing the right incentives to motivate participation. While in another study [16], we investigated how the infrastructure of a social cloud can be co-operatively provided by the participating members, and present an economic model that takes individual incentives and resource availability into account.

Kuada and Olesen [47] propose opportunistic cloud computing services (OCCS): a social network approach for the provisioning and management of enterprise cloud resources. Their idea is to provide a governing platform for enterprise level social networking platforms consisting of interoperable Cloud management tools for the platform's

resources, which are provided by the enterprises themselves. The authors present the challenges and opportunities of an OCCS platform, but there is no indication that they have yet built an OCCS. Similarly, Diaspora,[4] and My3 [48] apply similar concepts to host the social network on resource provided by their users.

Gayathri et al. [49] and Chen and Roscoe [50] discuss the security implications in the construction of a social cloud. They respectively pay special attention to, and provide counter measures for, how a social cloud can be used to circumvent copyright as well as perform other illicit actions. Whilst the consideration of security implications are critical for the success of a social cloud, it is not yet a focal point in our work.

6 Conclusions and Future Work

In this paper, we have presented a social compute cloud: a platform that enables the sharing of infrastructure resources between friends via digitally encoded social relationships. Using our implementation, users are able to execute programs on virtualized resources provided by their friends. To construct a social compute cloud, we have extended Seattle [12], [18] to access users' social networks, allow users to elicit sharing preferences, and utilize matching algorithms to enable preference-based socially-aware resource allocation.

Preference-based resource matching is (in a general setting) an NP-hard problem, makes often unrealistic assumptions about user preferences and most state of the art algorithms run in batch modes. Therefore, we investigated what happens when we apply these algorithms to a social compute cloud under the assumption that resource supply and demand do not fit to a batch allocation model. By applying methods to allocate resources in between Amazon EC2-like periodic allocations, we were able to quickly (in milliseconds) allocate resources temporarily, and then globally optimize resource allocation at the next batch allocation period. Our results are promising and indicate how the allocation of resources could take place in a production social compute cloud.

As future work, we will include additional ways for users to provide their preferences, as well as methods to detect them automatically from their social network. Where examples of the latter include: clustering based on homophily (aspects of similarity), relationship lists and Granovetter-like [51] indicators for relationship strength. This would also enable further, and potentially more realistic settings for experimenting with the allocation algorithms. In terms of the social cloud platform we will further extend the sandbox to provide additional system calls and social access control so that users can give extended/restricted access rights to groups, for example enabling command line access for family members. These extensions would increase the number of possible applications that could be executed within the social cloud and also further extend the social integration of the system. Finally, we aim to investigate how users use and interact with the resources of their friends, and move our implementation towards a production ready system.

4. https://joindiaspora.com/.

Acknowledgments

This work was partially funded by the Karlsruhe House of Young Scientists (KHYS). We would like to thank Justin Cappos and the Seattle team for their assistance in resolving technical issues, and many valuable discussions. We acknowledge support by Deutsche Forschungsgemeinschaft and Open Access Publishing Fund of Karlsruhe Institute of Technology.

Additional Resources

Find the project on Facebook: http://www.facebook.com/SocialCloudComputing.

References

[1] M. Armbrust et al., "A View of Cloud Computing," *Comm. ACM*, vol. 53, no. 4, pp. 50-58, 2010.
[2] F. Gens, "New IDC IT Cloud Services Survey: Top Benefits and Challenges," IDC exchange, http://blogs.idc.com/ie/?p=730, 2009.
[3] D. Bradshaw, G. Folco, G. Cattaneo, and M. Kolding, "Quantitative Estimates of the Demand for Cloud Computing in Europe and the Likely Barriers to Up-Take," http://ec.europa.eu/information_society/activities/cloudcomputing/docs/quantitative_estimates.pdf, July 2012.
[4] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, "Social Cloud: Cloud Computing in Social Networks," *Proc. IEEE Third Int'l Conf. Cloud Computing (CLOUD)*, pp. 99-106, 2010.
[5] K. Chard, K. Bubendorfer, S. Caton, and O. Rana, "Social Cloud Computing: A Vision for Socially Motivated Resource Sharing," *IEEE Trans. Services Computing*, vol. 5, no. 4, pp. 551-563, Jan. 2012.
[6] S. Milgram, "The Small World Problem," *Psychology Today*, vol. 2, no. 1, pp. 60-67, 1967.
[7] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four Degrees of Separation," *CoRR*, vol. abs/1111.4570, 2011.
[8] comScore, "Its a Social World: Top 10 Need-to-Knows about Social Networking and Where Its Headed," http://www.comscore.com/Insights/Presentations_and_Whitepapers/2011/it_is_a_social_world_top_10_need-to-knows_about_social_networking, 2011.
[9] K. John, K. Bubendorfer, and K. Chard, "A Social Cloud for Public eResearch.," *Proc. Seventh IEEE Int'l Conf. Science*, 2011.
[10] M.J. Litzkow, M. Livny, and M.W. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. Eighth Int'l Conf. Distributed Computing Systems*, pp. 104-111, 1988.
[11] D.P. Anderson, "Boinc: A System for Public-Resource Computing and Storage," *Proc. Fifth IEEE/ACM Int'l Workshop Grid Computing*, pp. 4-10, 2004.
[12] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Seattle: A Platform for Educational Cloud Computing," *Proc. 40th Technical Symp. ACM Special Interest Group for Computer Science Education (SIGCSE '09)*, 2009.
[13] P. Mell and T. Grance, "The Nist Definition of Cloud Computing," Technical Report 800-145, Nat'l Inst. of Standards and Technology http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, Sept. 2011.
[14] A. Thaufeeg, K. Bubendorfer, and K. Chard, "Collaborative eResearch in a Social Cloud," *Proc. IEEE Seventh Int'l Conf. E-Science (e-Science)*, pp. 224-231, Dec. 2011.
[15] S. Caton, C. Dukat, T. Grenz, C. Haas, M. Pfadenhauer, and C. Weinhardt, "Foundations of Trust: Contextualising Trust in Social Clouds," *Proc. Second Int'l Conf. Cloud and Green Computing (CGC)*, pp. 424-429, 2012.
[16] C. Haas, S. Caton, K. Chard, and C. Weinhardt, "Co-Operative Infrastructures: An Economic Model for Providing Infrastructures for Social Cloud Computing," *Proc. 46th Ann. Hawaii Int'l Conf. System Sciences (HICSS)*, 2013.
[17] V.L. Smith, "Microeconomic Systems as an Experimental Science," *The Am. Economic Rev.*, vol. 72, no. 5, pp. 923-955, 1982.
[18] Y. Zhuang, A. Rafetseder, and J. Cappos., "Experience with Seattle: A Community Platform for Research and Education," *Proc. The Second GENI Research and Educational Workshop*, 2013.

[19] H.T. Dinh, C. Lee, D. Niyato, and P. Wang, "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches," *Wireless Comm. and Mobile Computing*, vol. 13, pp. 1587-1611, 2013.
[20] R. Thal, "Representing Agreements in Social Clouds," master's thesis, Karlsruhe Inst. of Technology, 2013.
[21] M. Halldórsson, K. Iwama, S. Miyazaki, and H. Yanagisawa, "Improved Approximation Results for the Stable Marriage Problem," *ACM Trans. Algorithms*, vol. 3, no. 3, article 30, 2007.
[22] A. Roth, "Deferred Acceptance Algorithms: History, Theory, Practice, and Open Questions," *Int'l J. Game Theory*, vol. 36, no. 3, pp. 537-569, 2008.
[23] D. Gale and L. Shapley, "College Admissions and the Stability of Marriage," *Am. Math. Monthly*, vol. 69, pp. 9-15, 1962.
[24] R.W. Irving, P. Leather, and D. Gusfield, "An Efficient Algorithm for the Optimal Stable Marriage," *J. ACM*, vol. 34, no. 3, pp. 532-543, http://doi.acm.org/10.1145/28869.28871, July 1987.
[25] C. Haas, S. Kimbrough, S. Caton, and C. Weinhardt, "Preference-Based Resource Allocation: Using Heuristics to Solve Two-Sided Matching Problems with Indifferences," *Proc. 10th Int'l Conf. Economics of Grids, Clouds, Systems, and Services (Under Rev.)*, 2013.
[26] C. Haas, S. Caton, D. Trumpp, and C. Weinhardt, "A Simulator for Social Exchanges and Collaborations—Architecture and Case Study," *Proc. Eight IEEE Int'l Conf. eScience (eScience '12)*, 2012.
[27] B. Javadi, D. Kondo, J. Vincent, and D. Anderson, "Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of Seti@home," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, no. 11, pp. 1896-1903, Nov. 2011.
[28] S. Caton and O. Rana, "Towards Autonomic Management for Cloud Services Based upon Volunteered Resources," *Concurrency and Computation: Practice and Experience*, vol. 24, pp. 992-1014, http://onlinelibrary.wiley.com/doi/10.1002/cpe.1715/pdf, 2012.
[29] K. Chard, S. Caton, O. Rana, and D.S. Katz, "A Social Content Delivery Network for Scientific Cooperation: Vision, Design, and Architecture," *Proc. Third Int'l Workshop Data Intensive Computing in the Clouds (DataCloud '12)*, 2012.
[30] K. Kugler, K. Chard, S. Caton, O. Rana, and D.S. Katz, "Constructing a Social Content Delivery Network for eScience," *Proc. Third Int'l Workshop Analyzing and Improving Collaborative eScience with Social Networks (eSoN '13)*, 2013.
[31] Z. Guo, R. Singh, and M. Pierce, "Building the Polargrid Portal Using Web 2.0 and Opensocial," *Proc. Fifth Grid Computing Environments Workshop (GCE '09)*, pp. 1-8, 2009.
[32] R. Curry, C. Kiddle, N. Markatchev, R. Simmonds, T. Tan, M. Arlitt, and B. Walker, "Facebook Meets the Virtualized Enterprise," *Proc. 12th Int'l IEEE Enterprise Distributed Object Computing Conf. (EDOC '08)*, pp. 286-292, 2008.
[33] W. Wu, H. Zhang, and Z. Li, "Open Social Based Collaborative Science Gateways," *Proc. 11th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid)*, pp. 554-559, 2011.
[34] W. Wu, H. Zhang, Z. Li, and Y. Mao, "Creating a Cloud-Based Life Science Gateway," *Proc. IEEE Seventh Int'l Conf. E-Science (e-Science)*, pp. 55-61, 2011.
[35] D.N. Tran, F. Chiang, and J. Li, "Friendstore: Cooperative Online Backup Using Trusted Nodes," *Proc. First Int'l Workshop Social Network Systems (SocialNet '08)*, 2008.
[36] D.D. Roure, C. Goble, and R. Stevens, "The Design and Realisation of the Myexperiment Virtual Research Environment for Social Sharing of Workflows," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 561-567, 2009.
[37] G. Klimeck, M. McLennan, S.P. Brophy, G.B. Adams, and M.S. Lundstrom, "nanoHUB.org: Advancing Education and Research in Nanotechnology," *Computing in Science and Eng.*, vol. 10, pp. 17-23, 2008.
[38] A. McMahon and V. Milenkovic, "Social Volunteer Computing," *J. Systemics Cybernetics and Informatics*, vol. 9, no. 4, pp. 34-38, 2011.
[39] Z. Ali, R. Rasool, and P. Bloodsworth, "Social Networking for Sharing Cloud Resources," *Proc. Second Int'l Conf. Cloud and Green Computing (CGC)*, pp. 160-166, 2012.
[40] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, "Socialcloud: Using Social Networks for Building Distributed Computing Services," arXiv preprint arXiv:1112.2254, 2011.
[41] W. Tan, M.B. Blake, I. Saleh, and S. Dustdar, "Social-Network-Sourced Big Data Analytics," *IEEE Internet Computing*, vol. 17, no. 5, pp. 62-69, Sept./Oct. 2013.
[42] R. Gracia Tinedo, M. Sánchez;Artigas, A. Moreno Martinez, and P. Garcia Lopez, "Friendbox: A Hybrid F2F Personal Storage Application," *Proc. IEEE Fifth Int'l Conf. Cloud Computing (CLOUD)*, pp. 131-138, 2012.
[43] R. Gracia-Tinedo, M. Sánchez;Artigas, and P. Garcia Lopez, "F2box: Cloudifying F2F Storage Systems with High Availability Correlation," *Proc. IEEE Fifth Int'l Conf. Cloud Computing (CLOUD)*, pp. 123-130, 2012.
[44] R. Gracia-Tinedo, M.S. Artigas, and P. Garcia Lopez, "Analysis of Data Availability in F2F Storage Systems: When Correlations Matter," *Proc. IEEE 12th Int'l Conf. Peer-to-Peer Computing (P2P)*, pp. 225-236, 2012.
[45] Y. Zhang and M. van der Schaar, "Incentive Provision and Job Allocation in Social Cloud Systems," *IEEE J. Selected Areas in Comm.*, vol. 31, no. 9, pp. 607-617, Sept. 2013.
[46] C. Haas, S. Caton, and C. Weinhardt, "Engineering Incentives in Social Clouds," *Proc. 11th IEEE/ACM Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid '11)*, pp. 572-575, 2011.
[47] E. Kuada and H. Olesen, "A Social Network Approach to Provisioning and Management of Cloud Computing Services for Enterprises," *Proc. Second Int'l Conf. Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING '11)*, pp. 98-104, 2011.
[48] R. Narendula, T.G. Papaioannou, and K. Aberer, "My3: A Highly-Available P2P-Based Online Social Network," *Proc. IEEE Int'l Conf. Peer-to-Peer Computing (P2P)*, pp. 166-167, 2011.
[49] K. Gayathri, T. Thomas, and J. Jayasudha, "Security Issues of Media Sharing in Social Cloud," *Procedia Eng.*, vol. 38, pp. 3806-3815, 2012.
[50] B. Chen and A. Roscoe, "Social Networks for Importing and Exporting Security," *Large-Scale Complex IT Systems, Development, Operation and Management*, pp. 132-147, 2012.
[51] M.S. Granovetter, "The Strength of Weak Ties," *The Am. J. Sociology*, vol. 78, no. 6, pp. 1360-1380, 1973.

Simon Caton received the BSc (Hons) degree in computer science and the PhD degree in computer science from Cardiff University in 2010. He is a senior researcher at Karlsruhe Institute of Technology. He is the coordinator of the Young Investigator Group: Social Cloud: social network-based collaboration environments. His research interests include social cloud computing, social network analysis, autonomic computing, electronic markets, and service level agreements. He is a member of the IEEE.

Christian Haas received the diploma in business engineering from the Karlsruhe Institute of Technology in 2010, and the Master of Science degree in computer science from the Georgia Institute of Technology in 2008. He is currently working toward the PhD degree at the Karlsruhe Institute of Technology, Germany. His research interests include incentive engineering, two sided matching, and simulation for social clouds. He is a member of the IEEE.

Kyle Chard received the BSc (Hons) degree in computer science and the BSc degree in mathematics and electronics. He received the PhD degree in computer science from Victoria University of Wellington in 2011. He is a senior researcher at the Computation Institute, University of Chicago and Argonne National Laboratory. His research interests include distributed metascheduling, grid and cloud computing, economic resource allocation, social computing, and services computing. He is a member of the IEEE.

Kris Bubendorfer is the program director for networking engineering and a senior lecturer in the School of Engineering and Computer Science at Victoria University of Wellington. His PhD thesis was on mobile agent middleware, and his long term research interests are centered around distributed computing—including grid, cloud, market oriented, and social computing, plus he has wider interests in privacy, digital provenance, and reputation. He is a member of the IEEE.

Omer F. Rana received the PhD degree in neural computing and parallel architectures from the Imperial College of Science, Technology & Medicine, London University, United Kingdom. He is a professor of performance engineering at the School of Computer Science & Informatics, Cardiff University, United Kingdom. His research interests include high performance distributed computing, data mining, and multi-agent systems. He served as the deputy director of the Welsh e-Science Centre and has also worked as a software developer in the United Kingdom. He is a member of the IEEE.