Social networks play an increasingly important role in people’s lives. In recent years they have expanded

in terms of the size of the population involved and in terms of the functions performed. A promising

solution for analyzing large-scale social network data is to distribute the computation workload over a

large number of nodes of a cloud. Traditionally, determining the importance of a node or a relationship

in a network is done using sampling and surveying, but in a very large network structural properties

cannot be inferred by scaling up the results from small networks. It turns out that the evaluation of social

closeness is computationally intensive.

Social intelligence is another area where social and cloud computing intersect. Indeed, the process of

knowledge discovery and techniques based on pattern recognition demand high-performance computing

and resources that can be provided by computing clouds. Case-based reasoning (CBR), the process

of solving new problems based on the solutions of similar past problems, is used by context-aware

recommendation systems. It requires similarity-based retrieval. As the case base accumulates, such

applications must handle massive amounts of history data, which can be done by developing new

reasoning platforms running on the cloud. CBR is preferable to rule-based recommendation systems

for large-scale social intelligence applications. Indeed, the rules can be difficult to generalize or apply

to some domains. All triggering conditions must be strictly satisfied, scalability is a challenge as data

accumulate, and the systems are hard to maintain because new rules have to be added as the amount of

data increases.

A system based on CBR is described in [171]. The BetterLife 2.0 system consists of a cloud layer,

a case-based reasoning engine, and an API. The cloud layer uses the Hadoop Distributed File System

clusters to store application data represented by cases as well as social network information, such as

relationship topology and pairwise social closeness information. The CBR engine calculates similarity

measures between cases to retrieve the most similar ones and stores new cases back to the cloud layer.

The API connects to a master node, which is responsible for handling user queries, distributes the

queries to server machines, and receives results.

A case consists of a problem description, a solution, and optional annotations about the path to

derive the solution. The CBR uses MapReduce; all the cases are grouped by their userId, and then a

breadth first search (BFS) algorithm is applied to the graph, where each node corresponds to one user.

MapReduce is used to calculate the closeness according to pairwise relationship weight. A reasoning

cycle has four steps: (a) Retrieve the most relevant or similar cases from memory to solve the case; (b)

reuse: map the solution from the prior case to the new problem; (c) revise: test the new solution in the

real world or in a simulation and, if necessary, revise; and (d) retain: if the solution was adapted to the

target problem, store the result as a new case.

In the past, social networks have been constructed for a specific application domain (e.g., MyExperiment

and nanoHub for biology and nanoscience, respectively). These networks enable researchers

to share data and provide a virtual environment supporting remote execution of workflows. Another

form of social computing is volunteer computing, when a large population of users donates resources

such as CPU cycles and storage space for a specific project – for example, the Mersenne Prime Search

initiated in 1996, followed in the late 1990s by SETI@Home, Folding@Home, and Storage@Home, a

project to back up and share huge data sets from scientific research. Information about these projects is available online at www.myExperiment.org, www.nanoHub.org, www.mersenne.org,

setiathome.berkeley.edu, and folding.stanford.edu.

Such platforms cannot be used in an environment where users require some level of accountability

because there are no SLAs. The PlanetLab project is a credit-based system in which users earn credits

by contributing resources and then spend those credits when using other resources. The Berkeley

Open Infrastructure for Network Computing (BOINC) aims to develop middleware for a distributed

infrastructure suitable for different applications.

An architecture designed as a Facebook application for a social cloud is presented in [76]. Methods

to get a range of data, including friends, events, groups, application users, profile information, and

photos, are available through a Facebook API. The Facebook Markup Language (FBML) is a subset

of HTML with proprietary extensions, and the Facebook JavaScript (FBJS) is a version of JavaScript

parsed, when a page is loaded, to create a virtual application scope. The prototype uses Web Services

to create a distributed and decentralized infrastructure.

There are numerous examples of cloud platforms for social networks. There are scalable cloud

applications hosted by commercial clouds (e.g., Facebook applications are hosted by Amazon Web

Services). Today some organizations use the Facebook credentials of an individual for authentication.

The new technologies supported by cloud computing favor the creation of digital content. Data

mashups or composite services combine data extracted by different sources; event-driven mashups, also

called Svc, interact through events rather than the request/response traditional method. A recent paper

[331] argues that “the mashup and the cloud computing worlds are strictly related because very often

the services combined to create new Mashups follow the SaaS model and more, in general, rely on

cloud systems.” The paper also argues that the Mashup platforms rely on cloud computing systems –

for example, the IBM Mashup Center and the JackBe Enterprise Mashup server.

There are numerous examples of monitoring, notification, presence, location, and map services based

on the Svc approach, including Monitor Mail, Monitor RSSFeed, Send SMS, Make Phone Call, GTalk,

FireEagle, and Google Maps. For example, consider a service to send a phone call when a specific email

is received; the Mail Monitor Svc uses input parameters such as User Id, Sender Address Filter, and

email Subject Filter to identify an email and generates an event that triggers the Make TTS Call action

of a Text To Speech Call Svc linked to it.

The system in [331] supports creation, deployment, activation, execution, and management of eventdriven

mashups. It has a user interface, a graphics tool called Service Creation Environment that easily

supports the creation of new mashups, and a platform called Mashup Container that manages mashup

deployment and execution. The system consists of two subsystems: the service execution platform for

mashups execution and the deployer module that manages the installation of mashups and Svcs. A

new mashup is created using the graphical development tool and saved as an XML file. It can then be

deployed into a Mashup Container following the Platform-as-a-Service (PaaS) approach. The Mashup

Container supports a primitive SLA that allows the delivery of different levels of service.

The prototype uses the Java Message Service (JMS), which supports asynchronous communication.

Each component sends and receives messages, and the sender does not block while waiting for the

recipient to respond. The system’s fault tolerance was tested on a system based on the VMware vSphere.

In this environment, the fault tolerance is provided transparently by the VMM, and neither the VMs

nor the applications are aware of the fault-tolerance mechanism. Two VMs, a primary and a secondary

one, run on distinct hosts and execute the same set of instructions such that, when the primary fails, the

secondary continues the execution seamlessly.