J Internet Serv Appl (2012) 3:59–66
DOI 10.1007/s13174-011-0043-x

SI: FOME - THE FUTURE OF MIDDLEWARE

Challenges in very large distributed systems

Maarten van Steen · Guillaume Pierre · Spyros Voulgaris

Received: 26 October 2011 / Accepted: 9 November 2011 / Published online: 19 November 2011

Abstract Many modern distributed systems are required to scale in terms of their support for processes, resources, and users. Moreover, a system is often also required to operate across the Internet and across different administrative domains. These scalability requirements lead to a number of well-known challenges in which distribution transparency needs to be traded off against loss of performance. We concentrate on two major challenges for which we claim there is no easy solution. These challenges originate from the fact that users and system are becoming increasingly integrated and are effectively leading us to large-scale socio-technical distributed systems. We identify the design of such integrated systems as one challenge, in particular when it comes to placing humans in the loop as a necessity to proper operation of the system as a whole. As users are so tightly integrated into the overall design, and systems naturally expand through composition, we will be facing problems with respect to long-term management, which we identify as another major challenge.

Keywords Distributed systems · Socio-technical systems · Cloud computing

M. van Steen (✉) · G. Pierre · S. Voulgaris
VU University Amsterdam, Amsterdam, The Netherlands
e-mail: steen@cs.vu.nl

G. Pierre
e-mail: gpierre@cs.vu.nl

S. Voulgaris
e-mail: spyros@cs.vu.nl

1 Introduction

There is no longer such a thing as a stand-alone computer system: computer systems are networked. Moreover, although we are now accustomed to the fact that everything is connected, we still prefer to see the distributed nature of modern computer systems more or less hidden. Distribution transparency has always been, and continues to be an important design goal.

However, distribution transparency is since long increasingly challenged as connectivity becomes more pervasive, for connectivity essentially continues to scale any distributed system by demanding that it interfaces to services and other distributed systems that were not there at design time. This continuous growth leads to a number of scalability challenges, many of which are not new, but are simply always there no matter what.

In this paper, we concentrate on new challenges that arise from the fact that many modern distributed systems will be part of our pervasive computing environment. In our view, we are beyond the point that we can speak of "end users": users are *part* of a system and as such they need to be incorporated in the design. Numerous examples come to mind concerning the contribution of users to the overall performance of a system. Perhaps the most prevalent is the use of modern social media, and the gradual move of their Web-based forms to mobile computing platforms.

We start with discussing some of the obvious challenges, but swiftly move to providing an example to illustrate what we may be facing in Sect. 3. In the succeeding two sections, we then concentrate on getting users in the loop, but also management issues. The latter may be argued belong to the "old" challenges, but problems are being aggravated as systems are moving into (a multitude of) clouds.

2 Obvious challenges

Let us look at a few challenges for large-scale distributed systems for which it can be argued that they have never been, and perhaps will never be, solved to our full satisfaction. In the following, we address just a few that we believe are very important, but there are obviously many more, often specific problems.

2.1 Distribution transparency

Distribution transparency reflects the extent to which a system appears as a whole, rather than being a collection of independent components. In other words: a high degree of transparency will let the system appear to be coherent to its users. The extreme case is when a single-system view can be offered, virtually indiscernible from a nondistributed system.

The problem with striving for distribution transparency in very large systems is that performance will degrade to unacceptable levels. The cause is obvious: being networked, we need to face failures and their recoveries that can never be fully masked. In addition, network latencies have a natural lower bound that becomes noticeable when dealing with long-haul connections.

These performance problems can be partly tackled through replication, by which components are copied and placed close to where they are needed, but as soon as read-to-update ratios decrease, replication may actually lead to a scale-down of the system if consistency is needed (see also [23]). Moreover, the CAP principle tells us that we simply cannot combine consistency and availability in the presence of network partitions [6].

There is no simple solution to achieving distribution transparency, and we will have to continue to look for application-specific solutions to achieve acceptable transparency. The last word has not been said, and the main challenge remains to discover what is actually acceptable to users.

2.2 The Internet of things

Another challenge for large-scale distributed systems is dealing with what is known as the internet of things: the pervasive presence of a multitude of IP-enabled *things*, ranging from tags on products to mobile devices to services, and so forth [2]. From a distributed-systems perspective, the challenge is to move away from the network- and *things*-oriented views and provide a view in which the collaboration of all these Internet-enabled things form, indeed, a coherent distributed system.

What we are essentially addressing here is the inherent heterogeneity of any modern large-scale distributed system.

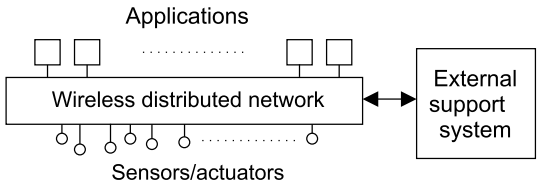

Fig. 1 An abstract view of the Internet-of-things as a distributed system

We are witnessing the integration of wireless sensor and actuator networks as a distributed computing platform with more traditional cloud-based systems to which specific computations and storage facilities are offloaded, as sketched in Fig. 1.

To come to such a perspective, we need to face considerable challenges:

- How can subsystems such as sensor networks be devised as a distributed system? Work is underway, as exemplified by approaching such networks as distributed databases [12], or providing the means to actually program them as a system [15].
- Given the potential size of the data generated by sensors and related devices, a trade-off will need to be found between in-network processing and aggregation techniques versus streaming data to the external support system. This trade-off is not an easy one. It depends on the capabilities of the distributed sensor network, the communication channel between sensor network and support system, and the support system itself. Again, the mere scale of such systems (think, for example, of hundreds of thousands of mobile devices that act as sensors), may be daunting.
- An issue that we will be addressing more in the following sections is processing of all input into useful feedback. We anticipate that notably with respect to this aspect, social computing elements will play an important role.

2.3 Building collaborative systems

Many modern distributed systems will necessarily have to be hosted by multiple administrative organizations. In fact, if large-scale decentralization is what we are aiming for, it is almost inevitable to devise *collaborative* systems. We claim that deep insights in building these type of systems such that basic security and dependability requirements are met is lacking, to say the least. In fact, we should ask ourselves whether it is even possible, as illustrated by the security problems hampering structured peer-to-peer networks [22].

The problems related to collaboration in large-scale distributed systems are not new and are being addressed by research into (algorithmic) mechanism design, a subfield within game theory. Devising mechanisms that work, scale, and can be efficiently implemented is a challenge. We foresee that much effort will be spent before seeing widely

adopted solutions. That we are not dealing with an unsolvable problem is illustrated by the success of BitTorrent.

3 New challenges: socio-technical distributed systems

As mentioned, there are many well known, or perhaps even obvious challenges to face for developing distributed systems. However, we only have to look around to see that there may be much more that we need to deal with and for which we may have to start adjusting our research agendas.

3.1 Introduction

Let us consider one important development that is taking place at this moment: the emergence of socio-technical systems. These systems *integrate* people with computer systems (comprising both hardware and software), along with a (likely mostly implicitly defined) set of rules of usage and interaction. The keyword here is *integration*: in a socio-technical system the boundary between people, technology, and usage is becoming blurred.

In this context, many distributed systems are seen to be designed to be directly used and shared by people, yet users and system are more or less separated. For example, there have been many systems for direct user-to-user communication, including those based on (mobile) telephony and various user-oriented messaging systems [25]. Also, in the last decade, we have seen an explosion of Web-based systems that offer services to end users through Web browsers, such as online transaction systems, systems for electronic commerce, and distributed systems implementing informational services (such as online timetables and route planners). More recently, we are witnessing an increasing realization of ubiquitous computing [19] as envisaged by Mark Weiser [26]. To illustrate, many public transportation systems are now equipped with RFID-based tickets and supermarkets allow customers to self-scan their groceries.

As said, the design of these distributed systems has treated end users as being separated from the system, in the sense that they are assumed to literally *use* a system. As a result, much effort has always been put into improving distribution transparency: masking, where appropriate, that data, processes, and control are indeed distributed across a system. We believe that we are facing a major challenge for future large-scale systems as they become increasingly ubiquitous. We can no longer afford to separate the users from a system in the design process. Instead, a seamless integration of users and computer system is what we will have to deal with.

3.2 An example scenario

To illustrate what we might be dealing with, consider the following scenario. Bob likes to listen to music and is interested in discovering things he has never heard before. His taste is rather diverse and covers various music genres, for which reason he has subscribed to different periodicals in order to pick up the latest and to be recommended about reference recordings. Using these and other information sources, Bob buys CDs, buys songs from various Internet sources, and makes use of online audio-streaming services such as Spotify [9].

What we have just described is a fairly traditional way of how users currently find information and how they buy music, or other media for that matter. What is already quite novel is the use of modern distributed systems for audio streaming (Spotify uses peer-to-peer technology to increase availability and to guarantee high streaming rates). However, there is considerable room for improvement which is already and steadily underway. For example, magazines are increasingly becoming online, CDs will most likely be replaced entirely by digital-only solutions, and even high-end audio equipment is becoming increasingly network enabled.

If we take these developments one step further, a future scenario in which users and distributed systems are in a closed loop, could be the following.

Bob is offered a personalized electronic magazine that is pushed to his tablet. The magazine makes no distinction between music styles: there is no need to do so as it is personalized. The system knows which information to push as it has learned what to recommend from observing past purchases, but also from what Bob has actually been listening to. In addition, it has discovered the best way to present new material that will most likely be appreciated. Next to such recommendations, there is ample opportunity to effectively and efficiently discover new content by browsing as the system has also found out what the optimal search space is for Bob (of which size and content may be dependent on current context).

The information pushed to a user comes from various sources, notably other people publishing about music in the same way that magazines today are full of CD reviews. The system automatically discovers to which reviewers a user reacts (e.g., by music that is being bought). Through content analysis it also effectively organizes itself as a publish-subscribe system, which is continuously being fine tuned.

Yet there is more. In this particular example, to improve the discovery process, the system automatically builds a social network around music taste, linking Bob to his potential peers. At first instance, this is done to improve the efficiency of searching for content, along the lines we have described previously [24]. More advanced decentralized evolutionary algorithms which optimize on which attributes to base link

formation are, however, preferred [7]. At second instance, this social network will bring Bob into contact with others (if so desired) to discuss music. Most important, however, is that in the end he will be looking at a personalized library of his favorite music, which is continuously being filled with new music to his liking, and at a rate that fits his way of living.

An important observation is that the boundary between user and computer system is vanishing. In this example, Bob is continuously reacting to what the system is offering, and the system, in turn is continuously reacting to Bob. Moreover, he is not alone: the system as a whole can operate only by virtue of other users and their behavior, be they listeners, reviewers, bloggers, and so on.

3.3 Its consequences

Obviously, there are many challenges to face by just considering this example scenario. As a starting point, one possible high-level architecture is sketched in Fig. 2. Essentially, the distributed system consists of a series of local media systems installed at a user's premises. We envisage that the local system acts as a thin client, but this is actually not important. The core of the entire system is formed by a myriad of services that are now shown to be running in the cloud. Much of the distributed systems research has concentrated on developing these services, often for cluster-based computer systems. The real challenge now lies in developing additional components that monitor user interaction, can do a proper analysis, and adjust itself to individual usage. Moreover, many of the services shown will need to be integrated and distributed between the cloud and the local media system to make sure that performance criteria are met.

In effect, we claim that socio-technical distributed systems actually form a feedback control loop, and that a major challenge lies in effectively closing that loop. Let us consider the various parts that comprise this architecture.

3.3.1 The local media system

The local media system consists of many different types of hardware and software components, and the complete set of components may vary and change over time. The challenge here is to make the composition easy. Hardware components include networked audio/video equipment and their control devices, e-reader devices, notification devices, and secured payment equipment. Many of these components may be integrated into, for example, modern tablets, but may also be separated in the form of smartphones or dedicated devices.

Managing the hardware is a challenge, but with protocols like UPnP we have already come a long way. Nevertheless, there is still considerable room for improvement as anyone who has ever installed a networked audio/video system will acknowledge. Moreover, by just considering the fact that people are mobile, we will be facing intricate networking problems and content access issues. However, we believe that many of these hardware-related issues are being tackled and do not form the big challenges for the near future.

Also, managing the core software components is an art that has been mastered quite successfully. Most components can automatically check for updates and often only a confirmation from the user is sufficient to start the process. Although there is obviously room for improvement, and certainly the science of systems management and administration is arguably underdeveloped (see, for example, [3]), we do not see the traditional management as one of the biggest challenges.

3.3.2 Monitoring user behavior

If we are to build the type of adaptive systems that we envisage are necessary in the future, a major challenge lies in accurately and effectively measuring and monitoring user behavior. In our example, we need to collect statistics on:

- Which music has been purchased
- Which music is being listened to, when, how long, and in which context
- Which information on music has been read, when, and how long

What can easily complicate matters is the fact that in our example several people may be colocated and listening to the same music. Likewise, in the case of a video-streaming system, we need to take collectively watching a movie into account. Furthermore, anticipating that the system will also make recommendations, we need to get feedback from the user on the quality of those recommendations. That feedback may be implicit (by testing to see if recommendations are followed up), or asked directly from the user.

The challenge in monitoring a user's behavior is to do this as nonintrusive as possible. Ideally, a user would not even notice that the system is continuously measuring actions and reactions. However, life is not as simple as one may think. Moran and Nakata [14] make clear that what they term *ubiquitous monitoring* is in need of much research. For example, the mere fact that users know that they are being monitored changes their behavior. Capturing a user's behavior and predicting the effects of monitoring that behavior is a challenge by itself.

What does this mean for designing *very large-scale* distributed systems? One thing to realize is that we may be dealing with millions of users spread across the Internet. If we are to accurately capture and analyze user interactions, we will have to think carefully of where we place components to do this. Ubiquitous computing architectures seem to benefit from a more centralized organization [5], and in

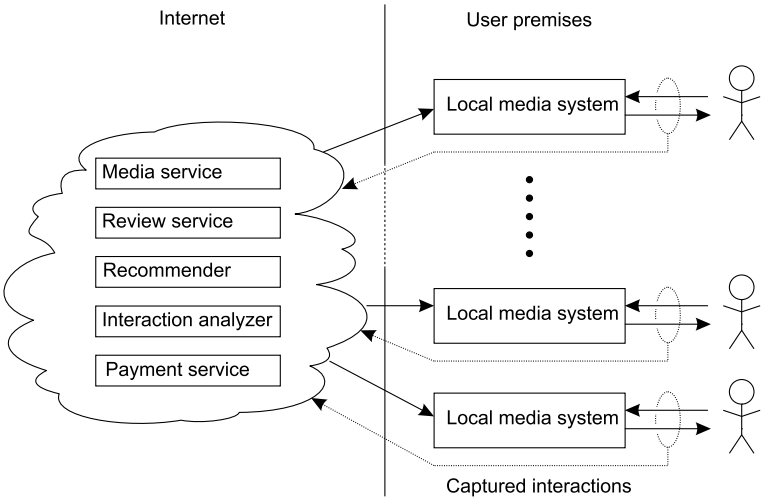

Fig. 2 A high-level architecture of a socio-technical distributed system

combination with processing lots of input data, placing the analysis system locally seems to be a good option. In any case, simply assuming that an interaction analyzer is placed entirely in the cloud is not the way to go. At the same time, we foresee that it is necessary to collect data from other user interactions in order to improve the ubiquitous computing experience. Not only that, we need to take into account that users are mobile (although perhaps less in the scenario that we sketched).

Summarizing, we face the need to have many local, centralized computing elements that need to be interconnected, and each being able to support user mobility. Whether these computing elements are personalized and can be carried with their owner, or are part of the environment yet capable of handling different users is an open question.

3.3.3 Analyzing user behavior

The truly difficult part as we see it, lies in the analysis of user behavior and subsequently taking the right measures to optimally adapt the system. For our example scenario, this analysis may boil down to developing advanced recommender systems that take current context into account (such as time of day and location). For distributed systems research, the challenge is to develop recommender systems enhanced with the information from other users. Finding out which information is needed, and from whom is not obvious. To illustrate, we have found, quite surprisingly, that for certain recommender systems it does not really matter whether data is used from a group of similar peers, or data from a randomly selected group [17]. Note that such findings may have a huge impact on the design of a distributed system: whereas the first approach may require intricate distributed algorithms, the second approach may be easy to implement using fairly straightforward techniques such as gossiping and randomized peer selection [8].

Of course, the real challenge in (distributed) user analysis lies in exploring the trade-offs between privacy and recommendation/adaptation. Without any information from a user, building a system that optimally adapts toward that user is virtually impossible. On the other hand, once information on a user is known and deployed in a decentralized way to enhance the quality of recommendations and adaptations, we may easily be giving out information that should have stayed private. This is a well-known problem (see, e.g., [13, 21]), yet we envisage that it will remain one of the major challenges for large socio-technical distributed systems.

In the following, let us zoom into two major challenges: incorporating the users as part of system design, and overall systems management.

4 Making users part of the system

A great portion of future challenges in distributed systems stems from the fact that users are becoming *part* of the system, with all the unpredictability and spontaneity that this brings. In our previous example, the system needs to figure out how it can perform best for a single user, for which purpose it mainly needs only that user's input. We foresee that future systems will need to take input from many more users to reach acceptable levels of performance. In fact, with Web 2.0, blogs, and social networks, the user has been promoted from a passive consumer of information to an active player, shaping services and determining content and interactions.

A keyword here is *crowdsourcing*, which refers to the delegation of a task to a large, undefined set of people, in the form of an open call. Wikipedia is probably the most widely known example of crowdsourcing, with close to 4,000,000 articles (in the English site alone) written collaboratively by over 600,000 people. Wikipedia essentially provides the framework for registering the *collective knowledge base* of

hundreds of thousands of people, and making it easily accessible to the whole humanity.

In our example system, a typical form of crowdsourcing will be collecting the opinions and reviews of streaming content and distributing those reviews to listeners. An important challenge will be to properly channel the reviews: Bob may gradually discover that Alice provides reviews of his liking, and that he would very much prefer receiving her reviews only and not those of Chuck. Again, the system will have to discover these preferences, and perhaps also recommend other reviewers to Bob based on his feedback. In this way, a fully automated, personalized reviewer space appears from which Bob can collect informed recommendations.

There are other forms of crowdsourcing, where the contribution of users is not as straightforward as in Wikipedia or the sketched review subsystem. Take, for instance, Twitter. Many services are built around it to give trends and news organized by topic or location simply by aggregating the tweets people post. It is not any single individual's tweet that makes the trends, but the massive retweeting by many users. What we have is a hugely decentralized system where information is processed to produce aggregate results on the fly. What is interesting and challenging in this case, is the fact that humans are taking some action (posting tweets) driven by some motivation (will to express themselves), and the data are combined appropriately to produce some additional result (realtime trends and news).

A more radical form of crowdsourcing is currently being explored and deployed using smartphones. One example is a smartphone application that traces the locations of its users, and uses data aggregation and analysis to split users in groups of related hangout patterns. Given that, it can subsequently send feedback to users on which spots of the city appear to be vibrant with respect to people of their own group, for instance due to an event that might interest them. The application harnesses people's *mobility patterns* and hangout habits to *produce* a service. In a similar vein, Alex Pentland's group from MIT have successfully been automatically capturing face-to-face social interactions to optimally organize groups of people [18] or to obtain insight in what they term societal "health state" [11].

These paradigms of harnessing collective knowledge, opinions, mobility patterns, social interactions etc., lead to systems that capture *collective intelligence*. These systems have become known as under the term *human-based computation*, which consists in capturing human brain functions that are still very difficult or infeasible for computers to perform, giving humans a relative motivation. Perhaps the simplest example is *reCAPTCHA*, which presents users with a challenge of reading a word to prevent automated programs from accessing content intended only for humans, while as a by-product enables the accurate digitization of old books and newspaper archives. Many other examples have followed, such as *GWAP* (Game With A Purpose), which is a simple online game that is surprisingly effective in accurately and concisely tagging huge collections of photos by its players.

In these systems, humans are completely embedded inside the system. In fact, the traditional model of computers performing the calculations and humans just seeing them has been completely inverted. Computers play the role of the coordinator, while the actual computation is performed by humans. What is common to these systems is their smart design which combines attractive *incentives* for people to voluntarily participate, as well as mechanisms to filter out low quality input and harvest the useful data.

It would be exaggerated to claim that all future distributed systems will be aiming at putting humans in the loop to harness collective intelligence. However, it is becoming evident that computers are becoming increasingly connected and pervasive, and they have invaded our lives in almost all respects, including our social activities, lifestyle, etc. This suggests that for a multitude of future distributed systems, humans will be part of the system by default, and applications that take human feedback into account will always have an extra advantage. Therefore, we believe that designing massive scale distributed applications that encourage human participation, and at the same time interact to the user in a form that enables the automated processing, analysis, and usage of human input, will be a new type of challenge for future systems. Clearly, this is radically different than conventional challenges of computation, memory, storage, and bandwidth, which we do not consider here.

5 Managing cloud-based distributed systems

An interesting observation about the large-scale distributed systems described so far is that they are most likely not provided by a single administrative entity. The electronic magazines, recommender systems, online stores, social networks, and so on clearly belong to a multitude of independently managed systems designed to interoperate with each other. Besides the obvious challenge of designing standard interfaces to allow such compositions to happen, one may notice that nobody is in charge of "the entire system." As a matter of fact, even the choice of composing one subsystem with another is not explicitly made by an administrator, but it derives from the collective behavior of users.

From the point of view of large-scale distributed application providers, this means that any user may couple any application with any other one at any time. As more links are being built between different applications, it becomes increasingly irrelevant to consider them in isolation from each other. For example, Bob may use the music-related tweets he receives to train his personal recommendation system. Any change in the semantics of Twitter, or the properties of tweet

messages have a direct impact on the behavior of the recommender.

Such increasing levels of interdependencies between seamingly independent subsystems introduce a new challenge for large-scale distributed systems: In such conditions, how does one guarantee correctness, fault-tolerance, performance, privacy, etc.? To paraphrase Leslie Lamport's famous quote, we could say that a very large distributed system is one in which the failure of an external service you did not even know existed can render your own application unusable.

5.1 Management for correctness and fault tolerance

Managing a large distributed system is a tremendously complex task. This was true already when a system was assumed to work in isolation from other systems, but it becomes even harder as soon as the system is coupled with others. In particular, it is not sufficient that each individual subsystem manages its own correctness to be able to necessarily derive the same property for the entire system.

A good example is the Amazon EC2 outage from April 2011 [1]. This incident originated in a faulty router configuration which disturbed the normal system operation. However, this event triggered several interrelated systems to take corrective measures. Although each such measure makes perfect sense when studying each subsystem in isolation, the combined effect of these actions turned out to be disastrous and to *increase* the magnitude of the problem rather than solve it. What we are witnessing here is the emergent behavior of a distributed system in terms of complex networks [10, 16].

Addressing this important challenge requires to be able to predict the effect of any management action such as updating a configuration file, and starting or stopping a given machine. Recent examples show that we are far from fully understanding the implications of such actions, even within a single administration domain. Extending this type of prediction to be able to reason about the effect on other external yet coupled systems is a difficult research challenge that we will have to address. This type of prediction will likely be made even more challenging by the fact that each system administrator will keep parts of her system's internal details secret. Note that a perfect prediction may not be necessary: it may be sufficient to build systems such that they can observe if an action has positive or negative effects on the global system, and continuously *converge* toward a desired state.

Management for fault tolerance may also require efficient mechanisms to detect the faulty components in a large-scale distributed system. Such components may be more complex to monitor than individual misbehaving machines: for example we may need to efficiently detect entire applications which do not comply with some minimum standards of quality. Facing this type of issues, a defensive action from a large-scale composition of applications could be to collectively identify the faulty parts of the system and make sure they cannot harm the rest (for example, by excluding them from the system as a whole). An alternative could be to harness the collective intelligence of the users in order to perform complex system management tasks. Such actions would obviously have to be carried out implicitly by the users.

5.2 Management for performance

Any local, small-scale application typically has dozens of configuration parameters which can potentially influence its performance. A large-scale distributed application probably has thousands or more. Managing such an application for performance consists in finding the set of parameter values which maximize performance. In essence, we can see performance as a function of the system's definition, its configuration parameters, and the execution environment (current workload, network condition, etc.). It should be clear that no human administrator can be expected to continuously tune hundreds of configuration parameters. An important challenge will therefore be to build self-configuring systems. Important steps in this direction have already been made [4] but they are restricted to systems with no more than a few dozen parameters. We are still very far from truly self-configuring large-scale systems.

6 Next steps: speculative solutions?

Any modern computer system can by now be considered as a distributed system. Where hiding the details of distribution was for long a challenging goal, we have now reached a stage in which challenges are shifting. We have identified a few "obvious" ones: next to attaining distribution transparency, building the inevitable (very large) Internet of things, and realizing truly collaborative systems.

In line of these challenges for very large-scale systems, we envisage that the loop between user and core system will become increasingly tighter, leading to very large-scale socio-technical systems. The challenges for these types of systems are very demanding: we need to design the systems making use of the information and even computing capabilities of humans. Users are thus moving into the design space.

And while this move is taking place, distributed systems themselves are moving toward the cloud, where they are composed (possibly by users), and maintained in a fully decentralized manner crossing many administrative bounds. What this means for (design for) maintenance and management is something we can only guess. The relative lack of science in systems management will definitely need to be addressed.

So here we have it: users are moving into the design arena and are becoming first-class citizens in our distributed systems, and in doing so, we are facing huge management problems that cannot be solved using administrators. In fact, we may even need to harness the collective intelligence of those same users in order to make next steps.

What to do? The answer is perhaps indeed building self-managing systems. But if we take a look at *how* we are currently doing this, it would seem that we're just running a bit faster with more people along avenues that have grown wider, but are otherwise quite familiar to us. Maybe it is time we start running along different paths.

What about trying to manage *less* and, instead, let the system tell us what really needs to be managed? Tuning a system is important, but maybe (fine-)tuning should be moved to a second plan first. In our own work, we have been testing how to deploy fully decentralized evolutionary algorithms to find the best way for semantically clustering users, if only to reduce search spaces. We throw in the parameters and let evolution do its work. Results are worse compared to when we design solutions using our own expertise, but the results are still quite satisfactory. It's a start.

Along these lines, seeking solutions that will give us just a bit more grip on emergent behavior, perhaps without completely understanding how those solutions intrinsically work, may be an important step that we, as middleware and systems community, may need to take more often. Loosening our grip of control may be the most challenging step to take. Along these lines, finding and adopting solutions that generally work, but never fix the problem, may also be something we need to consider more seriously. The work by Qin et al. reported at SOSP back in 2005 [20] may show to be an inspirational starting point.

References

1. Amazon Web Services (2011) Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region. http://aws.amazon.com/message/65648/
2. Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. Comput Netw 54(15):2787–2805
3. Bergstra J, Burgess M (eds) (2007) Handbook of network and system administration. Elsevier, Amsterdam
4. Chen H, Jiang G, Zhang H, Yoshihira K (2010) A cooperative sampling approach to discovering optimal configurations in large scale computing systems. In: Proc. 29th IEEE international symposium on reliable distributed systems
5. Dey A (2010) Context-aware computing. In: Krumm J (ed) Ubiquitous computing fundamentals. CRC Press, Boca Raton, pp 321–352
6. Gilbert S, Lynch N (2002) Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2):51–59
7. Huijsman R-J (Aug. 2011) An investigation into evolving distributed systems. Master's thesis. VU University Amsterdam
8. Jelasity M, Voulgaris S, Guerraoui R, Kermarrec A-M, van Steen M (2007) Gossip-based peer sampling. ACM Trans Comput Syst 25(3)
9. Kreitz G, Niemelä F (2010) Spotify—large scale, low latency, P2P music-on-demand streaming. In: Proc. 10th international conference on peer-to-peer computing, Aug. 2010. IEEE Comput Soc, Los Alamitos, pp 266–275
10. Lewis TG (2009) Network science: theory and practice. Wiley, New York
11. Madan A, Cebrian M, Moturu S, Farrahi K, Pentland A (2011) Sensing the 'Health state' of our society. Technical Report TR-663, MIT, Cambridge, MA
12. Madden SR, Franklin MJ, Hellerstein JM, Hong W (2005) TinyDB: an acquisitional query processing system for sensor networks. ACM Trans Database Syst 30(1):122–173
13. McSherry F, Mironov I (2009) Differentially private recommender systems: building privacy into the Netflix prize contenders. In: Proc. 15th international conference knowledge discovery and data mining (KDD), June 2009. ACM, New York, pp 627–637
14. Moran S, Nakata K (2010) Ubiquitous monitoring and user behaviour: a preliminary model. J Ambient Intell Smart Environ 2(1):67–80
15. Mottola L, Picco GP (2011) Programming wireless sensor networks: fundamental concepts and state of the art. ACM Comput Surv 43(3):19
16. Newman M (2010) Networks, an introduction. Oxford University Press, Oxford
17. Ogston E, Bakker A, van Steen M (2006) On the value of random opinions in decentralized recommendation. In: Proc. 6th international conference on distributed applications and interoperable systems. Lecture notes in computer science, vol 4025, pp 84–98. Springer, Berlin
18. Olguín DO, Pentland AS (2010) Sensor-based organisational design and engineering. Int J Organ Des Eng 1(1/2):69–97
19. Poslad S (2009) Ubiquitous computing: smart devices, environments and interactions. Wiley, New York
20. Qin F, Tucek J, Sundaresan J, Zhou Y (2005) Rx: treating bugs as allergies—a safe method to survive software failures. In: Proc. 20th symposium on operating system principles, Oct. 2005. ACM, New York, pp 235–248
21. Ramakrishnan N, Keller BJ, Mirza BJ, Grama AY, Karypis G (2001) Privacy risks in recommender systems. IEEE Internet Comput 5:54–62
22. Urdaneta G, Pierre G, van Steen M (2011) A survey of DHT security techniques. ACM Comput Surv 43(2)
23. van Steen M, Pierre G (2010) Replicating for performance: case studies. In: Charron-Bost B, Pedone F, Schiper A (eds) Replication, theory and practice. Lecture notes in computer science, vol 5959. Springer, Berlin, pp 73–89. Chapter 5
24. Voulgaris S, van Steen M, Iwanicki K (2007) Proactive gossip-based management of semantic overlay networks. Concurr Comput 19(17):2299–2311
25. Wams J, van Steen M (2004) Unifying user-to-user messaging systems. IEEE Internet Comput 8(2):76–82
26. Weiser M (1991) The computer for the 21st century. Sci Am September:67–83