Want to scale in centralized systems? Think P2P

Anne-Marie Kermarrec, François Taïani

RESEARCH

Want to scale in centralized systems? Think P2P.

Anne-Marie Kermarrec[1†] and François Taïani[2*†]

*Correspondence:
francois.taiani@irisa.fr
[2]Université of Rennes 1 - ESIR / IRISA, Inria Rennes, Rennes, France
Full list of author information is available at the end of the article
†Authors in alphabetical order

Abstract

Peer-to-peer (P2P) systems have been widely researched over the past decade, leading to highly scalable implementations for a wide range of distributed services and applications. A P2P system assigns symmetric roles to machines, which can act both as client and server. This distribution of responsibility alleviates the need for any central component to maintain a global knowledge of the system. Instead, each peer takes individual decisions based on a local and limited knowledge of the rest of the system, providing scalability by design.

While P2P systems have been successfully applied to a wide range of distributed applications (multicast, routing, caches, storage, pub-sub, video streaming), with some highly visible successes (Skype, Bitcoin), they tend to have fallen out of fashion in favor of a much more cloud-centric vision of the current Internet. We think this is paradoxical, as cloud-based systems are themselves large-scale, highly distributed infrastructures. They reside within massive, densely interconnected datacenters, and must execute efficiently on an increasing number of machines, while dealing with growing volumes of data. Today even more than a decade ago, large-scale systems require scalable designs to deliver efficient services.

In this paper we argue that the local nature of P2P systems is key for scalability regardless whether a system is eventually deployed on a single multi-core machine, distributed within a data center, or fully decentralized across multiple autonomous hosts. Our claim is backed by the observation that some of the most scalable services in use today have been heavily influenced by abstractions and rationales introduced in the context of P2P systems. Looking to the future, we argue that future large-scale systems could greatly benefit from fully decentralized strategies inspired from P2P systems. We illustrate the P2P legacy through several examples related to Cloud Computing and Big Data, and provide general guidelines to design large-scale systems according to a P2P philosophy.

Keywords: cloud computing; peer-to-peer; decentralized distributed systems

1 Introduction

Fully decentralized distributed architectures, and most notably P2P systems, enjoyed a high level of interest about a decade ago, prompted by early pioneering systems such as Napster and Freenet [1], quickly followed by systems such as Gnutella, Pastry [2] or Chord [3].

The main characteristic of such systems is that they do not distinguish between clients and servers: in a P2P system, each peer can act both as a client and a server and only maintains a local and incomplete view of the rest of system. While this paradigm has been widely used for (sometimes illegal) file sharing applications, the scalability of P2P systems has been leveraged in the context of many other applications such as streaming, content delivery networks, broadcast, storage systems, and publish-subscribe systems, just to name a few areas of application.

P2P systems are *scalable by design*: the fact that each peer potentially acts as a server avoids the bottleneck of most distributed systems by causing the number of servers to increase linearly with the number of clients. This natural ability to scale is complemented by the fact that no entity is required to maintain a global knowledge of the system, a costly and difficult operation in large-scale systems. Instead, each peer only relies on *local* and *restricted* information to drive its behavior. This ensures scalability for two reasons: first, individual peers only need to process a small amount of information. Second, information usually only needs to be disseminated to a limited subset of peers, thus reducing communication costs. For instance in Pastry [2], which provides a routing functionality in a P2P overlay network, individual peers only need to maintain a small routing table of size $O(logN)$, N being the number of peers in the system. Similarly, whenever a node leaves or enters the system, only a very small number of peers, $c + O(logN)$, need to be notified and update their data structures. Chord [3] and many other P2P infrastructures exhibit similar properties.

The ability of P2P systems to function without any central authority is one of the main reasons of their success, as exemplified by systems such as Emule for file sharing or Bitcoin for virtual money. Yet, this very ability has also hampered their growth, as most web companies wish to retain full control on their users base and computing infrastructure. As these companies are completing their migration towards highly-integrated data centers and cloud infrastructures, it might seem that the time for decentralized distributed systems is over, and that P2P solutions are only marginally relevant to today's cloud-centered world, with niche applications limited to file distribution and peer-supported systems [4, 5].

In this paper we take a contrarian view to this grim assessment, and argue that although pure P2P systems might no longer be seriously considered for obvious commercial reasons, they still hold great potential for the design of future computer systems. Indeed, we advocate the renewed importance of decentralized solutions for today's cloud-based systems, highlighting how the legacy of P2P continues to live in a new guise in many of the innovative solutions proposed to tackle the challenges of extra large distributed systems. This is for instance visible in some hybrid peer-assisted solutions adopted by companies such as Spotify [6], Akamai [7], or earlier versions of Skype, which adopt a P2P infrastructure for some of their services, complemented by a central control. Spotify, for instance, indexes music on a central infrastructure, which is then potentially downloaded from other peers.

Hybrid architectures are however only one rather direct example of how P2P intuitions might be leveraged to realize large-scale distributed systems. Skype for instance has recently moved to a cloud-based infrastructure, but nevertheless still retains many of the landmarks of its P2P past (*e.g.*, supernodes) [8]. Skype's recent history exemplifies how thinking decentralized is an excellent way to achieve scalability even when the targeted infrastructure includes powerful data centers and dedicated servers. In this paper, we illustrate this connection through several examples, highlighting how the legacy of P2P is out there in many of the innovative solutions proposed to tackle the challenges of extra large, geo-replicated distributed systems.

First, we discuss key-value store systems, a popular form of distributed storage underpinning many NoSQL databases (Section 2), which are a clear legacy of P2P

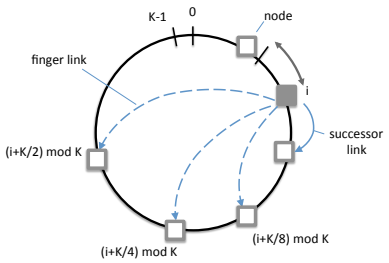

Figure 1 Principle of a ring-based DHT such as Chord [3]

Distributed Hash Tables (DHTs). In a second example (Section 3), we show that two strategies developed independently for the computation of K nearest neighbor (KNN) graphs converged to a single scalable design, although one emerged in a decentralized P2P context [9], while the other was proposed for a typical cluster-based batch processing infrastructure such as a map-reduce engine [10]. Again this example emphasizes the relevance of thinking decentralized for scalable design. Finally we reflect on the ways decentralized and P2P approaches might influence the design of very-large-scale distributed systems in the future, and try to delineate potential research paths that might realize the vision of inherently scalable computing (Section 4). We conclude in Section 5.

2 From Distributed Hash Tables to Key-value Stores

Distributed key-value stores (KVS) lie at the foundation of many NoSQL data-stores, such as Amazon's Dynamo [11], or Facebook's Cassandra [12], and play today a key role in the modern cloud ecosystem. Interestingly enough, the origin of many of today's key-value stores can be traced back to the work on distributed hash tables originally developed for P2P systems a decade ago, such as Chord [3], CAN [13], Pastry [2], and Tapestry [14].

2.1 In the beginning were Distributed Hash Tables

A DHT consists in storing (key, value) pairs on a (typically large) set of distributed nodes, while maintaining an appropriate routing overlay to rapidly find the machine holding a particular key. Which keys are allocated to each machine is determined by an appropriate hash function, with each machine in charge of an area of the hash space, thus resulting in a form of consistent hashing [15]. Individual DHTs vary in the type of routing overlay they use: DHTs such as Chord [3] use a ring extended with forward fingers (Fig. 1), while others such as CAN [13] or Pastry [2] use a prefix routing graph. This basic set of two mechanisms (hash space partitioning and routing overlay) is complemented with a number of additional protocols to handle nodes joining and leaving (either graciously, or through failures, including catastrophic ones [16]), and load-balancing (in case of a skewed distribution of keys in the hash space, or particularly popular keys).

For instance, in Chord [3], keys and node IDs are encoded over a fixed size of m bits, taking value from 0 to $2^m - 1$, and computation over this key space are done modulo 2^m. Each node stores a routing table containing m entries: the k^{th} routing

entry of node x is the first node whose ID is equal or greater than $id(x) + 2^{k-1}$ (modulo 2^m). The resulting finger links subsume the ring of key IDs: only taking into account the first routing entry of each node yields a ring in which each node points to its successor (the node with an ID $\geq id(x) + 1$).

Chord uses a simple routing algorithm to locate a key key on the ring: a node n trying to locate a key key forwards the request to its closest preceding finger nodes $n.finger[k]$, i.e. the finger node so that $n.finger[k] \in [n, key]$ and $n.finger[k+1] \notin [n, key]$. $n.finger[k]$, then repeats the operation recursively, until the procedure returns the node n^{-1}_{key} preceding the key on the key ring. The node storing key is the successor of n^{-1}_{key}, $n^{-1}_{key}.finger[0]$. Maintaining consistent routing information in a very large system is difficult and costly, so Chord does not assume that finger links are necessarily always up to date or consistent. The above routing mechanism however continues to work as long as successor links ($n^{-1}_{key}.finger[0]$) are correct: in the worst case, routing might degrade to a linear complexity as a routing request travels the ring in search of the node holding a key, but the system continues nevertheless to function.

Chord further includes dedicated protocols to handle the joining and leaving (through failure or otherwise) of participating nodes, including a stabilization protocol to overcome potential topological corruptions following concurrent failures and modifications.

As a result of this scheme, Chord provides the routing infrastructure required to implement a DHT in a fully decentralized manner. This is typical of other similar systems such as Pastry, or CAN that maintain a routing table of small size compared to the number of nodes.

2.2 From DHTs to industrial key-value stores

The above basic working of a DHT provides the foundation on which richer storage services can be built, with improved performance and consistency mechanisms tailored to the specific needs of individual systems. One first important evolution away from DHTs was the introduction of *one-hop routing* [11], to meet the stringent latency requirements of deployed systems. This is achieved in systems such as Cassandra [12] or Dynamo [11] (and its Erlang counterpart Riak[1]), by replicating the full routing information on each node as a speed-up mechanism over the typical $O(log(N))$ routing of DHTs. Because DHTs can tolerate obsolete routing information when nodes join or leave, these systems can too, a crucial property in large systems in which nodes failures and reconfigurations are common.

The potential downside of this strategy is the loss of extreme scalability: the size of the system is constrained by how much routing information can be stored on a single node. The approach is however reported to work well to up to a few hundred nodes, and can be extended with hybrid techniques and hierarchical designs. Riak, for instance, proposes a multi-data-center replication scheme for fault tolerance purposes, in which a source Riak instance is periodically mirrored into a sink Riak cluster.

[1] http://docs.basho.com/riak/latest/, accessed 2 June 2015

A second line of extension uses specific data-structures for keys and/or values. For instance, using lists or dictionaries for values creates a flexible storage structure organized in rows by columns that is reminiscent of relational databases (although generally without any of the ACID properties relational databases usually provide). Adding time-stamps (*e.g.*, as in BigTable [17]) or version numbers (as in Dynamo [11]) to values provides versioning, while adding timestamps to keys makes the data-store akin to a multi-version database.

Finally, a third line of extension adds additional querying capabilities, such as range queries [18, 19], by combining additional routing links, and well-chosen hash functions [20].

The main legacy of DHTs in these recent systems is the use of consistent hashing to distribute data uniformly over a large number of machines. The resulting systems, although they are designed to run in data centers on a few hundreds or thousands of machines, rather than on swarms of millions of home-based machines, remain inherently peer-to-peer in that they avoid any central component. They are also able in most cases to fall back on peer-based routing and reconciliation approaches (using mechanisms such a gossip-based anti-entropy [21]) to overcome failures and provide elastic growth, a crucial strength in highly dynamic cloud environments.

2.3 The challenge of consistency

Some of the reasons why decentralized key-value stores based on DHTs successfully upgraded from an initial P2P ecosystem to cloud computing is because they scale particularly well over many machines (routing typically takes at most log(n) steps, or O(1) with one-hop routing), are resilient to ongoing failures (a key requirement in large infrastructures), and can rapidly scale up or down by simply adding or removing machines.

One weakness, however, of basic decentralized key-value stores is their poor support for strong consistency guarantees. The need for fault tolerance and availability generally implies some form of redundancy of the same (key, value) pair over multiple machines. In the absence of additional mechanisms, concurrent modifications of a key are therefore not guaranteed to be atomic or even sequentially consistent [22, 23]. One possible strategy is to limit the concurrency the system can be subject to, which maps well to applications in which individual values are only manipulated by one single user at a time, as for instance the cart of an on-line shopping site. Another solution is to layer fault-tolerant consistency protocols on top of a basic DHT engine [24, 25] such as Paxos [26, 27], providing strong consistency between the replicas of individual key pairs, and delivering atomicity properties for single-key accesses.

Scatter [25] for instance uses a basic ring-based DHT (as in Figure 1) in which individual nodes are replaced by groups of nodes running the state-machine replication algorithm Paxos [26, 27]. To support the reconfiguration of these groups (*e.g.*, to accommodate shifting workloads, node failures, or new resources), Scatter further stacks a two-phase commit protocol on top of Paxos (a combination proposed earlier by Leslie Lamport and Jim Gray [28], and also found in Google's Spanner [24]). By combining the known ingredients of (i) a basic DHT, (ii) a fault-tolerant consensus

protocol, and (iii) a distributed transaction feature, Scatter exploits both the scalability of DHT and the strong consistency guarantees of fault tolerant distributed protocols.

The above examples illustrate how early ideas first experimented in the context of fully decentralized P2P systems continue to live on, sometimes in a different guise, and often combined with additional mechanisms, in today's systems designed for data centers and cloud computing.

Looking forward, future distributed systems are very likely to execute increasingly on multiple data centers, at a global scale, while taking into account scalability, performance, and the inherent limitations of the FLP (Fischer, Lynch, Paterson) [29] and CAP (Consistency, Availability, and Partition Tolerance) [30] impossibility results. These challenging requirements mean that the benefits of decentralized designs are unlikely to disappear anytime soon. They are more likely to continue to live on in new combinations as distributed systems adapt to the growing demands of scale, performance, and resilience, and to the opportunities brought about by technological advances (the lower latency of solid state drives over traditional hard drives being one such example). An open question is therefore how the various mechanisms we have touched upon could be better unified to help developers configure, compose, and extend existing platforms, and take informed decisions on how best to obtain desired features.

3 Gossip-based versus centralized KNN graph construction

A second area in which the intuitions developed for P2P environments seem to hold strong potential for more centralized systems is Big Data. We illustrate this point in the case of K nearest neighbors (KNN) graphs. Constructing the KNN graph of a set of items is a critical operation in many domains, ranging from data-mining and search to machine-learning, image processing, and collaborative filtering. When applied to (user-based) collaborative filtering [31], a KNN computation helps predict the interests of a given user by collecting the opinion of other users that are similar to her/him. Such a mechanism nicely translates into a P2P environment, and over the last 15 years a number of works have proposed P2P protocols to construct KNN graphs with applications to recommendation, search and query extension [32, 9, 33, 34, 35]. It turns out that the underlying design of these approaches is in fact very close to highly efficient KNN algorithms recently proposed for standalone machines [10]. This convergence highlights how strategies developed for decentralized peer-to-peer systems apply to much more centralized systems.

In (user-based) collaborative filtering [31], a KNN graph connects each user u to a user v if v is one of the k nearest neighbors of u in the considered application. The similarity between users is computed on the profiles of each user, for instance the lists of items that users have rated (*e.g.*, movies in a movie recommender system), or vectors of features for images, using one of several similarity metrics developed for informational retrieval such as the cosine similarity metric [31], and the Jaccard index. A brute force KNN computation has an $O(N^2)$ complexity, N being the number of vertices in the graph, and designing low-complexity KNN algorithms remains an open problem. While KNN graphs have played a crucial role in many applications, they are now increasingly applied to huge databases. Consequently,

as often in a Big Data context, scalability is of utmost importance. The challenge is to cope with many users and items, at acceptable costs and speeds. Traditional centralized approaches achieve this by constructing the KNN graph offline and exploiting elastic cloud platforms to massively parallelize the recommendation jobs on numerous nodes [36, 37]. However, accounting for dynamics requires periodic recomputations which turn out to be very costly [36, 38, 39].

Instead of deploying increasingly larger back-end servers to compute KNN graphs, alternatives have been recently proposed that exploit sampling to reduce drastically the dimension of the problem while achieving close to accurate results. The goal of this section is to show that approaches proposed independently, for centralized [10] and for decentralized [32, 35, 40, 33, 41] systems, exploit a similar sampling strategy to construct KNN graphs that is motivated by the same scalability concerns. In both cases, the crucial element for scalability is the strong locality of the algorithms, which consider each vertex of the constructed KNN graph using only a local and restricted knowledge of the system.

3.1 Gossip-based KNN graph construction

Gossip-based (or epidemic) protocols [21] have been widely used in the context of fully decentralized systems because of their robustness to churn and dynamics, their scalability, and their versatility [42, 43, 44]. The scalability of gossip-based protocols comes from the fact that each node takes individual decisions based only on a local knowledge of the system, while still allowing the whole system to eventually converge towards a desired state.

Several gossip-based protocols have been proposed to construct KNN graphs in a fully decentralized manner. These protocols can be parameterized to build both random topologies [45] and organized structures (rings, trees, torus) [32, 35, 40], and can been used to cluster peers sharing similar properties into a KNN graph, with applications to file sharing [46, 47, 48], link prediction [49], publish-subscribe systems [50], top-k processing [9], search [51], and recommenders [33, 41, 52, 53].

For instance, Tribler [51], a decentralized search engine implemented on top of the BitTorrent protocol, extracts users preferences and provides them with recommendations after a few search queries. Tribler relies on a gossip protocol to form the neighborhood, i.e. the set of similar users that should be considered to compute recommendations. Similarly, PocketLens [52] is a decentralized recommender algorithm developed by the GroupeLens research group. This system can rely on several architectures including fully decentralized ones to compute node neighborhoods. Finally, the approach presented by Kermarrec, Leroy, Moin, and Thraves [53] proposes a new collaborative filtering user-based random walk approach customized for decentralized systems, specifically designed to handle sparse data. Neighborhoods are formed using a gossip protocol instrumented with a modified Pearson's correlation metric to connect each user to a set of similar users.

Figure 2 shows the typical organization of a P2P KNN graph construction protocol, as originally proposed by Vicinity [32, 35] (and with some important variations by T-Man [40]), and then reused by other works, such as Gossple [9, 33, 34]. A protocol such as Vicinity or Gossple maintains a dynamic implicit social network, *i.e.* a directed graph linking peers (representing users) with similar interests. The

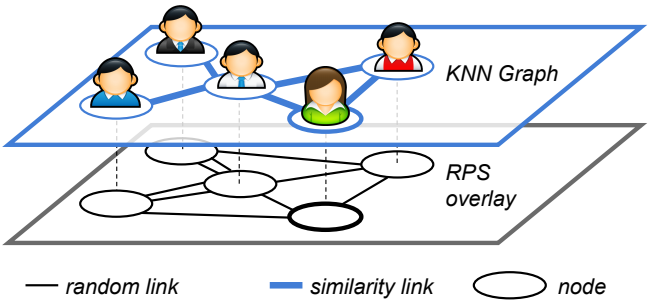

Figure 2 P2P KNN graph construction, as originally proposed by Vicinity [35]

protocol achieves this without relying on any central component by building a P2P overlay network in which each peer has two sets of neighbors: a (dynamic) set of neighbors picked uniformly at random in the network, and the KNN set (Figure 2). These two sets of neighbors are maintained by two co-existing gossip protocols that periodically sample the network, gossip node profiles, and connect similar users. The lower-layer random peer sampling protocol (RPS) [45] ensures connectivity by building and maintaining a continuously changing random topology. The upper-layer clustering protocol [32] uses this overlay to provide nodes with the k most similar candidates to form their KNN neighborhood.

More precisely, each protocol maintains at each node two *views*, a data structure containing references to other nodes: the RPS view and the KNN view. Each entry in each view contains *(i)* the neighbor's ID, *(ii)* its IP address, *(iii)* its profile [2], as well as *(iv)* a timestamp to date the last contact with this neighbor. Periodically, each protocol selects the entry in its view with the oldest timestamp [45] and sends it a message containing its profile with part (or all) of its view.

In the RPS protocol, the peer receiving the message updates its RPS view by keeping a random sample of the union of its own RPS view and the received view. This constitutes a continuously changing random graph. In the KNN protocol, the receiving peer selects the K closest peers found in both its current KNN view, its current RPS view, and the received KNN view, i.e. the K peers whose profiles are closest to its own according to the similarity metric.

This provides a two-layer overlay network as depicted on Figure 2. Note that the KNN graph could be constructed by using the RPS view only, since the RPS protocol provides a continuously changing sample of the nodes in the system. Doing so would however be very slow, converging in $O(N)$ steps. The second gossip protocol, which exploits the KNN view, speeds up the convergence on the assumption that a neighbor of a neighbor in the current KNN estimation is probably a good candidate to consider for the KNN view of the local node.

Crucially, the construction of the KNN graph is local (only the profile related to a peer and its neighbors are present on a given peer). There are no global data structures; instead, each peer receives for one of its neighbors a set of candidates

[2] The profile of a node is application dependent and represents the interests on a peer on which the similarity with other peers will be computed.

to compute similarity metrics. A second key characteristic of P2P KNN graph construction protocols is their sampling-based approach: each peer selects a set of candidates based only on a partial sample of the network.

3.2 KNN-Descent: a sampling-based centralized KNN

In a recent work, Dong, Moses, and Li have proposed a simple yet effective centralized algorithm, called *KNN-Descent*, that approximates a KNN graph under arbitrary similarity measures [10]. The main originality of their approach over previous work is the fact that the algorithm is local and sample-based.

The basic algorithm follows the very same philosophy of gossip-based protocols such as T-Man [40], Vicinity [32] or Gossple [33], namely *a neighbor of a neighbor is also likely to be a neighbor*. This means that provided there already exists an approximation of a KNN graph, the approximation can be iteratively improved.

KNN-Descent starts with a random approximation of the KNN graph, which is very similar to the RPS overlay of Vicinity or Gossple. Then, the algorithm compares each vertex of the graph with its current neighbor's neighbors until no further improvement can be made.

KNN-Descent further extends this basic strategy with a number of optimizations designed to minimize system costs (by maximizing local accesses) and speed up the computation. A first optimization uses what the authors have termed a local join: given a vertex v and its neighbors N_v, KNN-Descent computes the similarity between each pair p, q such that $p \in N_v$, and $q \in N_v$. In other words, each neighbor of v computes its similarity with each other neighbor of v. The KNN of v's neighbors are updated accordingly. A second optimization is introduced to reduce the number of similarity computations: pairs that were already compared during previous iterations are ignored. This is done by only comparing two vertices in a local join operations if at least one of them has been updated (this is indicated by a specific flag). Finally, KNN-Descent leverages the fact that little improvement is typically observed in the last iterations of the algorithm. KNN-Descent therefore implements an early termination mechanism and stops the algorithm when the number of KNN updates in neighborhoods falls bellow a given threshold.

Note that the KNN-Descent algorithm does not provide an accurate KNN graph but instead an approximation.

3.3 Comparing P2P KNN construction and KNN-Descent

The KNN constructed by P2P approaches such as Vicinity or Gossple, and that of KNN-Descent both rely on exactly the same philosophy, a philosophy pioneered by gossip-based algorithms. All these approaches are both local and sample-based, they all start from a random approximation, provided by a random sample in KNN-Decent and by the RPS overlay in Vicinity and Gossple, and progress by greedy iterations to progressively converge towards a KNN graph (possibly approximated in the case of KNN-Descent).

The main difference is that because P2P KNN approaches operate in a fully decentralized way, where each vertex is a machine on a network, they optimize their KNN views one pair of nodes at a time by traversing directed edges in the KNN-graph. By contrast, KNN-Descent first computes an undirected graph from its current KNN estimation, and then uses a local join operation on a node's neighbors, for each node in

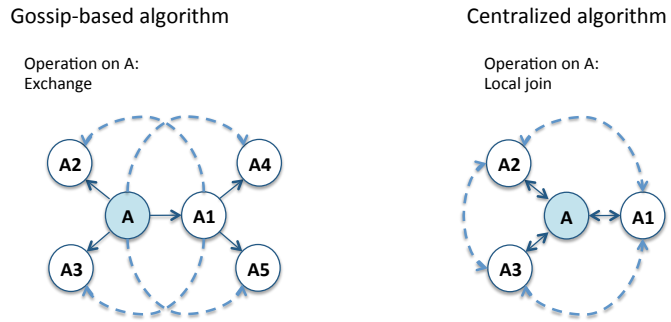

Figure 3 Local optimization of the KNN graph in decentralized and centralized approaches

this graph. This local join operation compares all pairs of a node's neighbors in one iteration, and thus increases the memory locality of the algorithm. This difference is illustrated in Figure 3. In this figure, solid lines represent the current estimation of the KNN graph, and the dashed lines the new potential neighbors considered during the next iteration. The left diagram illustrates the workings of a typical P2P KNN protocol. In this example, Node A currently has the nodes $\{A_1, A_2, A_3\}$ in its neighborhood, and will consider the nodes A_4 and A_5 (A_1's neighbors) as potential new neighbors. Similarly, A_1 will consider A_2 and A_3 (A's current neighbors) as potential new neighbors. The behavior of KNN-Descent on A's neighborhood is shown on the right. Rather than working with a directed graph, KNN-Descent first reverses all edges (shown as solid double arrows). The resulting undirected graph can then be exploited to realize a local join by looping through a node's neighbors in pairs: for instance, in the case of A's neighbors, KNN-Descent will consider whether A_1 might become one of A_2's neighbors, and reciprocally (double dashed arrow), and then loop over the pairs (A_1, A_3) and (A_2, A_3). This local join mechanism allows KNN-Descent to compute similarity values at most only once per iteration. It has however no impact on the actual set of edges being considered compared to a strategy that would simply traverse the edges of the undirected graph, as in the P2P case. This is an optimization primarily motivated by performance considerations on a standalone machine, or on a highly integrated cluster.

The use of a reverse graph does increase, however, the set of edges considered in one iteration by KNN-Descent. As a result, KNN-Descent tends to converge more rapidly than a pure P2P KNN network, but at the cost of maintaining a reverse graph, which can be a costly operation in a fully distributed environment, in which network communication is orders of magnitude slower than local memory access. The other difference is that P2P KNN construction protocols such as Vicinity or Gossple are guaranteed to eventually provide an accurate KNN graph while KNN-Descent provides an approximation of the graph. This is because in P2P KNN graph construction protocols the operation of the RPS ensures that nodes that were *forgotten* over several operations are eventually considered as potential candidates.

4 Discussion and perspectives

The examples we have discussed illustrate how algorithms that had initially been designed for fully decentralized systems have led to highly scalable solutions de-

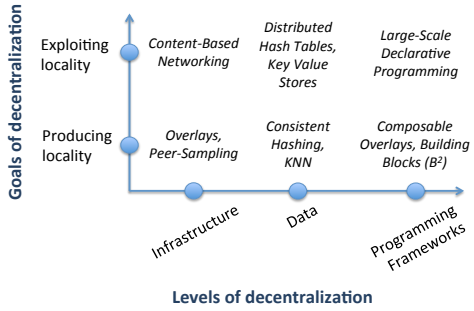

Figure 4 Decentralization objectives, and the levels where they apply

ployed on much more centralized infrastructures, in which all machines execute within the same data center or cluster. We think this is because the extreme nature of peer-to-peer and fully decentralized systems forces designers to explore radical solutions that, when reused in other contexts, provide *scalability by design*.

If we try to tease out the ingredients empowering these radical solutions, we find two key elements behind the scalability of decentralized P2P algorithms:

1 These algorithms seek to create *locality*. In particular, they avoid global structures or knowledge which are difficult to construct and maintain. For instance, one of the simplest forms of this principle can be found in a random peer-sampling service (RPS) [45]. An RPS generates a highly connected overlay topology with a short diameter (i.e. $O(log(n))$) using mechanisms limited to neighboring nodes.

2 Once locality has been established, these algorithms *exploit* it with decentralized mechanisms that are able to provide global services (multicast) or answers (*e.g.*, the k most similar items to a query) from lightweight local computations.

These two elements are the main goals of any decentralization (shown on the vertical axis labeled *Goals of decentralization* in Figure 4). These goals are however very generic, and can be instantiated at many levels of a system's distributed architecture. We see strong opportunities at at least three of these levels (horizontal axis in Figure 4): at the *infrastructure* level, in terms of *distributed data*, and in *programming frameworks*.

These levels should not be taken as hard and well-delineated layers, but more as useful props to capture the shifting organization of modern large-scale distributed systems. The *infrastructure* level seeks to cover the communication layer (naming, multicast), fundamental mechanisms (remote invocation, distributed events), and base services (service discovery, directory, membership) of a distributed system. By *distributed data*, we mean the strategies used to distribute data in a large-scale system while accounting for performance and scalability. Finally, *Programming Frameworks* aim to provide generic programmatic structures that guide developers when realizing a broad range of applications. Frameworks usually embody patterns, guidelines, and rules into a predefined but flexible architecture that developers can extend and modify to fit their needs.

In the following, we discuss how decentralization, and the two goals we have discussed, could be implemented at these three levels, first discussing *Infrastructure* and *Data* together (Section 4.1), before moving on to *Programming Frameworks* (Section 4.2).

4.1 Infrastructure and data

In a centralized system, data can be stored and processed in the same location. While this centralization clearly yields strong performance benefits, the scalability of this design is limited by the capabilities of a single machine. By contrast, a P2P design can scale horizontally at will, but this scalability comes with side effects : data is scattered at the extreme, with elements stored and processed at every user machine. This dispersion is key to scalability but potentially inefficient for computations that require non-local data. We argue that the local nature of P2P algorithms can be leveraged to mitigate this problem by either (1) creating locality or (2) exploiting locality both at the data and infrastructure levels.

Creating locality In a P2P system, computers at the edge are used to contribute to the system. This yields a natural one-to-one mapping between a machine and a user. This natural mapping can be leveraged to create locality in a number of user-centric applications, in which data is inherently attached to users. For instance in recommendation systems, users are not only the target of the service in the sense that they need to be provided with some (recommended) items, but also produce the data used for computing recommendations, typically in the form of profiles, such as the list of music files, pictures, movies, or news items they have downloaded, posted or liked.

The central position of users in these applications can be leveraged to guide the distribution of both *data* and *computation* on the underlying infrastructure and *create locality* along these two dimensions. How this distribution occurs is flexible, and allows for hybrid designs in which parts of a system are decentralized while others are not. This flexibility offers a variety of design choices that developers can adapt to the context of their application. For instance in a file sharing system, all files can be stored at the user that created them. In a recommendation system, each user might be responsible for processing her/his own data, while this data is stored on a centralized infrastructure, as in the Hyrec recommender system [54]. Spotify [6] illustrates the reverse case, in which data is indexed on centralized servers (computation) but data transfers occur in a peer-to-peer fashion between users. User interactions may also be exploited to influence data placement, as in the work of Pujol *et al.* [55], where the data of a social network is placed according to how users interact with one another.

Finally, the constraints that a P2P system imposes, such as favoring local computations and limiting communication, turn out to be sometimes particularly beneficial to performance, and can be transposed to the design of cloud-based implementations. Locality (of computation and communication) for instance has driven the design of the distributed graph embedding algorithm proposed by Kermarrec, Leroy, and Trédan [49], but the resulting algorithm is not tied to a P2P deployment. In this work, a force-based model is used to embed a graph into a high dimensional

space by associating each node with some coordinates that reflect its position in the graph. The embedding yields distances between nodes that carry more semantics than plain hop counts, and can be used within further applications (search, recommendations). Starting from random coordinates, each node updates its coordinates by being attracted by the neighbors in the graphs and repulsed by all other nodes of the system. By applying a fully decentralized algorithm not all nodes are considered for repulsion but only a random sample. It appears that this does not only provide scalability but also limits the influence of remote nodes on the system's stability (which might in this example actually disrupt the system). Applying such an algorithm in a cloud-based environment will yield the same benefits, with potential applications to graph processing. For instance, updating a KNN graph can be easily distributed by first partitioning the graph, and then updating only parts of it, thus limiting the need for communication between distributed units that are working on weakly connected parts of the graph.

Leveraging locality User-based applications can rely on the link between users and data to create a natural form of locality. Unfortunately, this natural locality is not always present, and must instead be injected into some systems to reap the full benefits of a decentralized design. This is apparent for instance in the domain of DHTs, when comparing Pastry [2] to earlier designs such as Chord [3] or CAN [13]. Contrary to the initial designs of Chord and CAN, Pastry takes into account the geographical proximity of participating nodes when managing its underlying overlay. As a result, Pastry avoids routing messages through geographically distant nodes when connecting geographically close neighbors, yielding much better performance than approaches that ignore the physical locations of nodes. This illustrates how, in a P2P DHT, the relative link between a node's logical and physical locations can have tremendous consequences for the DHT's overall performance. If both locations are only weakly correlated or worse not correlated at all, messages routed through the overlay might bounce between nodes that lay geographically far from one another, with drastic consequences for latency and network traffic.

Interestingly, this type of locality-driven strategy, which seeks to align the logical behavior of a distributed system with the shape of its physical deployment, can be transposed to cloud-based infrastructures, with substantial performance gains as exemplified by Camdoop [56]. Camdoop exploits a direct-connect network topology to link servers in a low-diameter graph at the physical level, which is particularly beneficial to data aggregation for map-reduce applications. We think that Camdoop along with other efforts in the area of rack-space computing [57, 58] nicely illustrate how the mechanisms designed in the context of P2P systems to leverage or create locality hold a huge potential to design efficient and highly scalable cloud infrastructures.

4.2 Programming frameworks

The third level of decentralization in which we see strong opportunities are *programming frameworks for decentralized systems*. This observation is prompted by the sheer number of existing decentralized solutions [59, 60]. This success makes it paradoxically hard for practitioners to orient themselves in this large domain.

In particular, practitioners cannot rely on any unified set of tools or programming abstractions for decentralized systems to help them make sense of the many subtleties and options and the field. As a result, they cannot easily reuse, compose, or adapt existing solutions to fit their needs, and have limited opportunities to share knowledge and ideas, which in turn limits the adoption of novel decentralized techniques.

We think this situation mirrors that of traditional distributed systems in the eighties, when developing a distributed application often meant coding at the levels of sockets and packets, if not lower. Many middleware technologies have since then been developed to improve this situation, by raising the level of abstraction at which developers of distributed systems must work. This includes distributed objects [61], component-based middleware [62], modular distributed platforms [63, 64, 65, 43], aspects [66], reflection [67], and models at runtime approaches [68, 69] just to name of few. These technologies have in common that they seek to offer well-encapsulated modular entities (interfaces, components, aspects) that foster reuse and composition. They thus advocate a principled approach to distributed software development, to ensure desirable software properties, such as reuse, composability, and maintainability.

Unfortunately these techniques are often only moderately relevant to highly-scalable decentralized mechanisms, as they rarely capture the challenges inherent to large-scale systems, which are left to the developers to solve. This is either because they say little about a system's organization beyond local interactions, or tend to encourage medium-scale architectures, which emphasize punctual interactions and explicit bindings, a philosophy that is ill aligned with the dynamicity and unpredictability of very-large-scale distributed systems.

Prompted by this diagnostic, some pioneering works have been proposed over the last 15 years, to ease the development of large-scale and decentralized systems. Mace [70] and Macedon [71] for instance use a form of data-flow programming inspired from datalog to implement peer-to-peer systems. Similarly, OverML [72] offers a set of languages to describe at a high level how an overlay should be constructed, which data each node should maintain, and what kind of messages should be exchanged. A number of works have in the same way sought to systematize the design and implementation of epidemic protocols, an emblematic family of highly scalable algorithms [42, 44, 73, 74, 75].

In spite of their promises, these first attempts do not yet fulfill the expectations of a systematic and generic framework for the programming of decentralized systems. They rarely allow developers to think about distributed applications as a woven composition of decentralized mechanisms (*e.g.*, overlay topologies, routing paths, markers), or to reason in a systematic manner about the fundamental aspects of these decentralized mechanisms such as their spatial extent, their interactions (bindings, events, regulations) and their life cycle. Similarly, these first attempts provide very few mechanisms for recursion in the structures they produce (a recurring property of composable systems), for example by allowing a network to exist within one another, while feeding on the data and context provided by its host network.

These gaps open a number of exciting research paths to simplify the deployment of large-scale decentralized systems, and ease their application within cloud-based

infrastructures. Most fruitful seem to be efforts [76] that take inspiration from successful approaches in medium-scale distributed systems (such as models at runtime [68, 69], distributed macro-programming languages [77, 78], reflection [67], declarative networking [79, 80]) and adapt them to the specifics and existing ecosystems of highly scalable decentralized algorithms.

5 Conclusion: Want to scale? Adopt the P2P mindset.

Today's distributed computer systems have reached scales never heard of before, be it in terms of the number of machines they host, the number of cores these machines contain, the amount of data they store and process, or the number of users they serve. The need for scalable and future-proof solutions to support these systems is therefore more crucial than ever. In this paper, we have argued that such scalable solutions should adopt a P2P strategy to succeed.

Contrary to the original vision of peer-to-peer systems, most modern distributed computations occur in highly integrated data centers, and are increasingly made available at various abstraction levels (IaaS, PaaS, SaaS) through cloud computing technologies. One could be led to believe that peer-to-peer technologies are therefore no longer relevant in today's world. In this paper, we have argued otherwise: as data centers and cloud platforms grow in size and number, we believe the decentralized approaches originally proposed to leverage the computing power of home computers still hold a strong potential to implement large-scale globally distributed systems. Our key argument is that decentralized designs will in the long term become increasingly applied to very-large-scale data center systems.

This is because, regardless of whether the targeted system is a centralized, server-based or fully decentralized one, designing algorithms that are efficient in a P2P system is an excellent way of providing *scalability by design*. Interestingly, if one can do the big things, one can do the little things as well. P2P algorithms can be transposed easily and directly to centralized systems. The reverse is more complicated, a scalable centralized algorithm has usually to be adapted to decentralized systems.

These observation chimes in with other works on scalable computing platforms and models, for instance on sublinear time algorithms [81], or sample-based querying engines [82], which aim to compute accurate results using only a partial view of a system. Decentralized and P2P designs can be understood as a practical embodiment of this intuition, which, we think, will continue to live on in tomorrow's distributed computer systems.

Competing interests
The authors declare that they have no competing interests.

Author's contributions
Both authors contributed equally to the paper. Authors' names are in alphabetical order.

Acknowledgements
This work has been partially supported by the ERC project GOSSPLE 204742, by the French ANR project SocioPlug (ANR-13-INFR-0003), and by the DeSceNt project granted by the Labex CominLabs excellence laboratory (ANR-10-LABX-07-01).

Author details
[1]Inria, Rennes, France. [2]Université of Rennes 1 - ESIR / IRISA, Inria Rennes, Rennes, France.

References

1. Rodrigues, R., Druschel, P.: Peer-to-peer systems. Commun. ACM **53**(10), 72–82 (2010). doi:10.1145/1831407.1831427
2. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg. Middleware '01, pp. 329–350. Springer, London, UK, UK (2001). http://dl.acm.org/citation.cfm?id=646591.697650
3. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. SIGCOMM '01, pp. 149–160. ACM, New York, NY, USA (2001). doi:10.1145/383059.383071. http://doi.acm.org/10.1145/383059.383071
4. Huang, Y., Chen, Y.-F., Jana, R., Jiang, H., Rabinovich, M., Reibman, A., Wei, B., Xiao, Z.: Capacity analysis of mediagrid: a p2p iptv platform for fiber to the node (fttn) networks. IEEE Journal on Selected Areas in Communications **25**(1), 131–139 (2007)
5. Yin, H., Liu, X., Zhan, T., Sekar, V., Qiu, F., Lin, C., Zhang, H., Li, B.: LiveSky: Enhancing CDN with P2P. ACM Transactions on Multimedia Computer Communication Applications **6**, 16–11619 (2010)
6. Kreitz, G., Niemelä, F.: Spotify – large scale, low latency, P2P music-on-demand streaming. In: IEEE Tenth International Conference on Peer-to-Peer Computing, P2P 2010, Delft, The Netherlands, 25-27 August 2010 (2010)
7. Zhao, M., Aditya, P., Chen, A., Lin, Y., Haeberlen, A., Druschel, P., Maggs, B., Wishon, B., Ponec, M.: Peer-assisted content distribution in Akamai NetSession. In: 2013 Internet Measurement Conference (IMC'2013), pp. 31–42 (2013). ACM
8. Kaufman, M.: Skype / NSA. http://www.listbox.com/member/archive/247/2013/06/entry/6:271/-20130623090855:0B714E0A-DC06-11E2-9F35-8CD4CCA160A2/. (e-mail, forwarded by Dave Farber to ip@v2.listbox.com), accessed 2 June 2015 (2013)
9. Bai, X., Guerraoui, R., Kermarrec, A.-M., Leroy, V.: Collaborative personalized top-k processing. ACM Trans. Database Syst. **36**(4) (2011)
10. Dong, W., Moses, C., Li, K.: Efficient k-nearest neighbor graph construction for generic similarity measures. In: Proceedings of the 20th International Conference on World Wide Web. WWW '11, pp. 577–586. ACM, New York, NY, USA (2011). doi:10.1145/1963405.1963487. http://doi.acm.org/10.1145/1963405.1963487
11. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles. SOSP '07, pp. 205–220. ACM, New York, NY, USA (2007). doi:10.1145/1294261.1294281. http://doi.acm.org/10.1145/1294261.1294281
12. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system, vol. 44, pp. 35–40. ACM, New York, NY, USA (2010). doi:10.1145/1773912.1773922
13. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. SIGCOMM '01, pp. 161–172. ACM, New York, NY, USA (2001). doi:10.1145/383059.383072. http://doi.acm.org/10.1145/383059.383072
14. Zhao, B.Y., Kubiatowicz, J., Joseph, A.D.: Tapestry: a fault-tolerant wide-area application infrastructure. Computer Communication Review **32**(1), 81 (2002)
15. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. STOC '97, pp. 654–663. ACM, New York, NY, USA (1997). doi:10.1145/258533.258660. http://doi.acm.org/10.1145/258533.258660
16. Haeberlen, A., Mislove, A., Druschel, P.: Glacier: highly durable, decentralized storage despite massive correlated failures. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2. NSDI'05, pp. 143–158. USENIX Association, Berkeley, CA, USA (2005). http://dl.acm.org/citation.cfm?id=1251203.1251214
17. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7. OSDI '06, pp. 15–15. USENIX Association, Berkeley, CA, USA (2006). http://dl.acm.org/citation.cfm?id=1267308.1267323
18. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. ACM SIGCOMM Computer Communication Review **34**(4), 353–366 (2004)
19. Guerraoui, R., Handurukande, S.B., Huguenin, K., Kermarrec, A.-M., Le Fessant, F., Riviere, E.: Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles. In: Proceedings of the Sixth IEEE International Conference on Peer-to-Peer Computing. P2P '06, pp. 12–22. IEEE Computer Society, Washington, DC, USA (2006). doi:10.1109/P2P.2006.19. http://dx.doi.org/10.1109/P2P.2006.19
20. Vilaça, R., Oliveira, R., Pereira, J.: A correlation-aware data placement strategy for key-value stores. In: Proceedings of the 11th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems. DAIS'11, pp. 214–227. Springer, Berlin, Heidelberg (2011). http://dl.acm.org/citation.cfm?id=2022090.2022107
21. Demers, A., Greene, D., Houser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: Proc. of the 6th Annual ACM Symp. on Principles of Dist. Comp., pp. 1–12. ACM, ??? (1987)
22. Herlihy, M.P., Wing, J.M.: Linearizability: a correctness condition for concurrent objects. ACM Trans.

Program. Lang. Syst. **12**(3), 463–492 (1990). doi:10.1145/78969.78972
23. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. Computers, IEEE Transactions on **100**(9), 690–691 (1979)
24. Corbett, J.C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J.J., Ghemawat, S., Gubarev, A., Heiser, C., Hochschild, P., Hsieh, W., Kanthak, S., Kogan, E., Li, H., Lloyd, A., Melnik, S., Mwaura, D., Nagle, D., Quinlan, S., Rao, R., Rolig, L., Saito, Y., Szymaniak, M., Taylor, C., Wang, R., Woodford, D.: Spanner: Google's globally-distributed database. In: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. OSDI'12, pp. 251–264. USENIX Association, Berkeley, CA, USA (2012). http://dl.acm.org/citation.cfm?id=2387880.2387905
25. Glendenning, L., Beschastnikh, I., Krishnamurthy, A., Anderson, T.: Scalable consistency in scatter. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. SOSP '11, pp. 15–28. ACM, New York, NY, USA (2011). doi:10.1145/2043556.2043559. http://doi.acm.org/10.1145/2043556.2043559
26. Lamport, L.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
27. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. **16**(2), 133–169 (1998). doi:10.1145/279227.279229
28. Gray, J., Lamport, L.: Consensus on transaction commit. ACM Transactions on Database Systems (TODS) **31**(1), 133–160 (2006)
29. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985). doi:10.1145/3149.214121
30. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002). doi:10.1145/564585.564601
31. Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative Filtering Recommender Systems. Now Publishers Inc., Boston - Delft (2011)
32. Voulgaris, S., van Steen, M.: Vicinity: A pinch of randomness brings out the structure. In: MIDDLEWARE (2013)
33. Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A.-M., Leroy, V.: The gossple anonymous social network. In: Proc. of the ACM/IFIP/USENIX 11th Int. Conf. on Middleware. Middleware'10, pp. 191–211 (2010)
34. Bai, X., Bertier, M., Guerraoui, R., Kermarrec, A.-M., Leroy, V.: Gossiping personalized queries. In: Proc. of the 13th Int. Conf. on Extending Database Technology. EDBT'10, pp. 87–98. ACM, New York, NY, USA (2010)
35. Voulgaris, S., van Steen, M., Iwanicki, K.: Proactive gossip-based management of semantic overlay networks. Concurrency and Computation: Practice and Experience **19**(17), 2299–2311 (2007)
36. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: WWW (2007)
37. Dean, J., Ghemawat, S.: Mapreduce: a flexible data processing tool. Commun. ACM **53**(1), 72–77 (2010)
38. Linden, G., Smith, B., York, J.: Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Computing (2003)
39. Meisner, D., Sadler, C.M., Barroso, L.A., Weber, W.D., Wenisch, T.F.: Power management of online data-intensive services. SIGARCH Comput. Archit. News (2011)
40. Jelasity, M., Montresor, A., Babaoglu, Ö.: T-man: Gossip-based fast overlay topology construction. Computer Networks **53**(13), 2321–2339 (2009)
41. Boutet, A., Frey, D., Guerraoui, R., Jégou, A., Kermarrec, A.-M.: Whatsup: A decentralized instant news recommender. In: IPDPS, pp. 741–752 (2013)
42. Eugster, P., Felber, P., Le Fessant, F.: The "art" of programming gossip-based systems. SIGOPS Oper. Syst. Rev. **41**, 37–42 (2007)
43. van Renesse, R., Minsky, Y., Hayden, M.: A gossip-style failure detection service. In: Proc. of the IFIP Int. Conf. on Distributed Systems Platforms and Open Distributed Processing. Middleware'98, pp. 55–70. Springer, London, UK (1998)
44. Kermarrec, A.-M., van Steen, M.: Gossiping in distributed systems. SIGOPS Oper. Syst. Rev. **41**, 2–7
45. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: Gossip-based peer sampling. ACM Trans. Comput. Syst. **25** (2007)
46. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In: Europar, pp. 1143–1152 (2005)
47. Handurukande, S.B., Kermarrec, A.-M., Le Fessant, F., Massoulie, L., Patarin, S.: Peer Sharing Behaviour in the eDonkey Network, and Implications for the Design of Server-less File Sharing Systems. In: EuroSys, pp. 359–371 (2006)
48. Voulgaris, S., Kermarrec, A.-M., Massoulié, L., van Steen, M.: Exploiting semantic proximity in peer-to-peer content searching. In: FTDCS, pp. 238–243 (2004)
49. Kermarrec, A.-M., Leroy, V., Trédan, G.: Distributed social graph embedding. In: CIKM, pp. 1209–1214 (2011)
50. Baldoni, R., Beraldi, R., Quéma, V., Querzoni, L., Piergiovanni, S.T.: Tera: topic-based event routing for peer-to-peer architectures. In: DEBS, pp. 2–13 (2007)
51. Tribler. (2010). http://www.tribler.org, accessed 2 June 2015
52. Miller, B.N., Konstan, J.A., Riedl, J.: Pocketlens: Toward a personal recommender system. ACM Trans. Inf. Syst. **22**(3), 437–476 (2004)
53. Kermarrec, A.-M., Leroy, V., Moin, A., Thraves, C.: Application of random walks to decentralized recommender systems. In: OPODIS, pp. 48–63 (2010)
54. Boutet, A., Frey, D., Kermarrec, R.G.A.-M., Patra, R.: Hyrec: Leveraging browsers for scalable recommenders. In: ACM/IFIP/USENIX Int. Middleware Conf. (Middleware'14). ACM, ??? (2014)
55. Pujol, J.M., Erramilli, V., Siganos, G., Yang, X., Laoutaris, N., Chhabra, P., Rodriguez, P.: The little engine(s) that could: Scaling online social networks. IEEE/ACM Trans. Netw. **20**(4), 1162–1175 (2012)

56. Costa, P., Donnelly, A., Rowstron, A.I.T., O'Shea, G.: Camdoop: Exploiting in-network aggregation for big data applications. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012, pp. 29–42 (2012)
57. Chowdhury, M., Zaharia, M., Ma, J., Jordan, M.I., Stoica, I.: Managing data transfers in computer clusters with orchestra. In: SIGCOMM '11
58. Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., Stoica, I.: Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling. In: EuroSys '10
59. Taiani, F., Lin, S., Blair, G.S.: Gossipkit: A unified componentframework for gossip. Software Engineering, IEEE Transactions on **40**(2), 123–136 (2014). doi:10.1109/TSE.2013.50
60. Babaoglu, O., Canright, G., Deutsch, A., Caro, G.A.D., Ducatelle, F., Gambardella, L.M., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., Urnes, T.: Design patterns from biology for distributed computing. ACM Trans. Auton. Adapt. Syst. **1**, 26–66 (2006)
61. Common Object Request Broker Architecture (CORBA). http://www.omg.org/spec/CORBA/, accessed 2 June 2015
62. Seinturier, L., Merle, P., Rouvoy, R., Romero, D., Schiavoni, V., Stefani, J.-B.: A component-based middleware platform for reconfigurable service-oriented architectures. Software: Practice and Experience, (2011). doi:10.1002/spe.1077
63. Hiltunen, M.A., Schlichting, R.D.: The cactus approach to building configurable middleware. In: in Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000 (2000)
64. van Renesse, R., Birman, K., Hayden, M., Vaysburd, A., Karr, D.: Building adaptive systems using ensemble. Software Practice and Experience **28**(9), 963–979 (1998)
65. Bhatti, N.T., Hiltunen, M.A., Schlichting, R.D., Chiu, W.: Coyote: a system for constructing fine-grain configurable communication services. ACM Trans. Comput. Syst. **16**, 321–366 (1998)
66. Colyer, A., Clement, A.: Large-scale aosd for middleware. In: Proc. of the 3rd International Conference on Aspect-oriented Software Development. AOSD'04, pp. 56–65. ACM, New York, NY, USA (2004)
67. Fleury, M., Reverbel, F.: The JBoss extensible server. In: ACM/IFIP/USENIX Int. Middleware Conf. (Middleware'03), pp. 344–373 (2003)
68. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.-M.: Modeling and validating dynamic adaptation. In: Models in Software Engineering (MODELS'10). LNCS, vol. 5421, pp. 97–108 (2009)
69. Morin, B., Barais, O., Nain, G., Jezequel, J.-M.: Taming dynamically adaptive systems using models and aspects. In: Proc. of the 31st Int. Conf. on Soft. Engineering. ICSE '09, pp. 122–132. IEEE Computer Society, Washington, DC, USA (2009)
70. Killian, C.E., Anderson, J.W., Braud, R., Jhala, R., Vahdat, A.: Building distributed systems using mace. In: Proceedings P2P 2009, Ninth International Conference on Peer-to-Peer Computing, 9-11 September 2009, Seattle, Washington, USA, pp. 91–92. IEEE, ??? (2009). doi:10.1109/P2P.2009.5284502. http://dx.doi.org/10.1109/P2P.2009.5284502
71. Rodriguez, A., Killian, C.E., Bhat, S., Kostic, D., Vahdat, A.: MACEDON: methodology for automatically creating, evaluating, and designing overlay networks. In: 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California, USA, Proceedings, pp. 267–280. USENIX, ??? (2004). http://www.usenix.org/events/nsdi04/tech/rodriguez.html
72. Behnel, S., Buchmann, A.: Models and languages for overlay networks. In: Databases, Information Systems, and Peer-to-Peer Computing. Lecture Notes in Computer Science, vol. 4125, pp. 211–218. Springer, ??? (2007). doi:10.1007/978-3-540-71661-7_21. http://dx.doi.org/10.1007/978-3-540-71661-7_21
73. Rivière, E., Baldoni, R., Li, H., Pereira, J.: Compositional gossip: A conceptual architecture for designing gossip-based applications. ACM SIGOPS Operating System Review **41**(5), 43–50 (2007)
74. Princehouse, L., Birman, K.: Code-partitioning gossip. SIGOPS Oper. Syst. Rev. **43**(4), 40–44 (2010). doi:10.1145/1713254.1713264
75. Lin, S., Taïani, F., Bertier, M., Blair, G., Kermarrec, A.-M.: Transparent componentisation: high-level (re)configurable programming for evolving distributed systems. In: Proceedings of the 2011 ACM Symposium on Applied Computing. SAC '11, pp. 203–208. ACM, TaiChung, Taiwan (2011). doi:10.1145/1982185.1982233
76. DIONASYS: Declarative and Interoperable Overlay Networks, Applications to Systems of Systems. (2014). http://www.chistera.eu/projects/dionasys, accessed 2 June 2015
77. Mainland, G., Morrisett, G., Welsh, M.: Flask: staged functional programming for sensor networks. In: ICFP '08: Proceeding of the 13th ACM SIGPLAN International Conference on Functional Programming, pp. 335–346. ACM, New York, NY, USA (2008). doi:10.1145/1411204.1411251
78. Newton, R., Morrisett, G., Welsh, M.: The regiment macroprogramming system. In: IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, pp. 489–498. ACM, New York, NY, USA (2007). doi:10.1145/1236360.1236422
79. Loo, B.T., Condie, T., Garofalakis, M., Gay, D.E., Hellerstein, J.M., Maniatis, P., Ramakrishnan, R., Roscoe, T., Stoica, I.: Declarative networking. Commun. ACM **52**(11), 87–95 (2009). doi:10.1145/1592761.1592785
80. Chu, D., Popa, L., Tavakoli, A., Hellerstein, J.M., Levis, P., Shenker, S., Stoica, I.: The design and implementation of a declarative sensor network system. In: SenSys '07: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, pp. 175–188. ACM, New York, NY, USA (2007). doi:10.1145/1322263.1322281
81. Rubinfeld, R., Shapira, A.: Sublinear time algorithms. Electronic Colloquium on Computational Complexity (ECCC) **18**, 13 (2011)
82. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., Stoica, I.: Blinkdb: queries with bounded errors and bounded response times on very large data. In: Eighth Eurosys Conference 2013 (EuroSys)'13, Prague, Czech Republic, pp. 29–42 (2013)