A Service-Oriented Peer-to-Peer Architecture for a Digital Ecosystem

Jan Sacha, Bartosz Biskupski, Dominik Dahlem, Raymond Cunningham, Jim Dowling, and René Meier
Distributed Systems Group, Trinity College Dublin, Ireland

***Abstract*— Service-oriented computing is becoming an increasingly popular paradigm for modelling and building distributed systems in heterogeneous, decentralised, and open environments. However, proposed service-oriented architectures are usually based on centralised components, such as service registries or service brokers, that introduce reliability, management, and performance issues. In this paper, we present a fully decentralised service-oriented architecture built on top of a self-organising peer-to-peer infrastructure. This architecture is especially designed to support digital ecosystems due to its low deployment and maintenance cost and inherently decentralised nature.**[1]

INTRODUCTION

Service-Oriented Computing (SOC) is a paradigm where software applications are modelled as collections of loosely-coupled, interacting services that communicate using standardised interfaces, data formats, and access protocols. The main advantage of SOC is that it enables interoperability between different software applications running on a variety of platforms and frameworks, potentially across administrative boundaries. This is especially important in digital ecosystems, where the environment is decentralised, open, and heterogeneous, as SOC can facilitate software reuse and automatic composition and fosters rapid, low-cost development of distributed applications.

A Service Oriented Architecture (SOA) usually consists of three elements: a service provider that publishes and maintains a service, a service consumer that uses the service, and a service registry that allows service discovery by prospective consumers [1], [2]. In many proposed SOAs, the service registry is a centralised component, known to both publisher and consumer, and is usually based on the UDDI protocol. Moreover, many existing SOAs rely on other centralised facilities that provide, for example, support for business transactions, service ratings or service certification [2].

However, each centralised component in a SOA constitutes a single point of failure that introduces security and reliability risks, and may limit the systems scalability and performance.

[1]Presented at the Inaugural IEEE International Conference on Digital Ecosystems and Technologies (DEST'07), Cairns, Australia, 2007.

An approach to decentralise a SOA is to employ a Peer-to-Peer (P2P) infrastructure. A P2P infrastructure is an application-level overlay network, built on top of the Internet, where the responsibilities of nodes are approximately equal. In particular, there is no distinction between clients and servers as all nodes provide services to each other as peers. The P2P paradigm allows the construction of systems with a very large size and very high robustness, mainly due to their inherent decentralisation and redundant structures. Furthermore, P2P systems enable the utilisation of resources on a large number of machines connected to the system through the Internet.

The P2P approach is especially promising in heterogeneous and decentralised digital ecosystems, where smaller or medium-sized enterprises often cannot afford high costs of the hardware and software required to build large-scale systems. Self-managing and self-organising P2P technologies allow a number of smaller enterprises to collaborate with each other and to share the costs of system deployment and maintenance.

However, the construction of P2P applications poses a number of challenges to the system designers. Recent research [3] shows that in existing P2P systems nodes frequently join and leave, generating high churn rates, and that unreliable communication in wide-area networks cause large delays in message passing between peers. Measurements also show that many peers suffer from poor network connectivity, poor availability, or low performance [4], [5]. Furthermore, the decentralisation, large scale, and dynamism of P2P systems introduce distributed decision making problems.

In this paper, we propose a fully decentralised, large-scale SOA based on a self-organising P2P infrastructure. The main contributions of the paper are the design of the proposed SOA, and a prototypical implementation and evaluation of the underlying P2P infrastructure. Our initial evaluation shows that our approach is scalable and resilient to high peer churn rates and random failures.

The remainder of this paper is organised as follows. In section II, we analyse the requirements for a decentralised SOA and we present an overview of our architecture. The architecture is based on two P2P overlay topologies, the gradient topology overlay and the distributed hash table overlay, that we describe in detail in sections III and IV, respectively. In section V, we describe the evaluation of our approach. In section VI, we review related work, and in section VII, we conclude the paper.

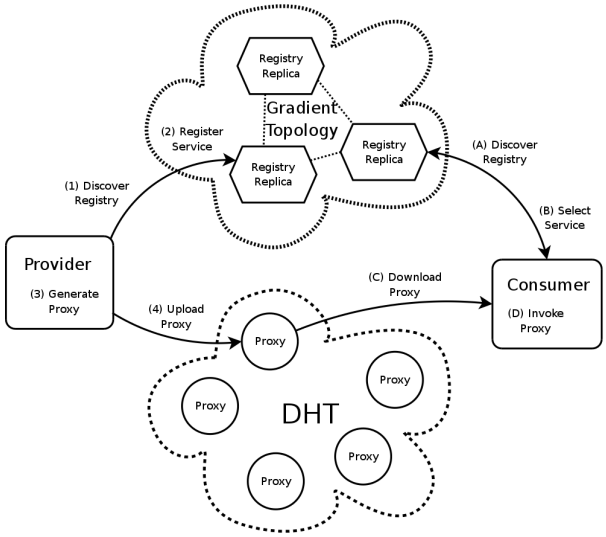

Fig. 1. Service registration (1-4), discovery (A-B), and consumption (C-D).

Architecture Overview

In a Service-Oriented Architecture, the users can be assigned two roles: a service provider or a service consumer (also called service customer). At the highest level of generality [1], [2], a SOA provides the following use cases.

- *Publish* : Introduction of a new service to the system by a service provider.
- *Find* : The discovery and selection of a service of interest to the consumer from the set of all services in the system.
- *Bind* : The usage of a service by a service customer.

In addition, SOAs usually allow service *deregistration* by a provider, and may support a service *lease* mechanism, where every registered service is assigned a lease and a service is automatically removed from the registry when its lease expires.

In this paper, we describe a decentralised service-oriented architecture, based on a self-organising P2P network, that supports the above use cases. The P2P infrastructure consists of two P2P overlays, the Distributed Hash Table (DHT) overlay, and the Gradient Topology (GT) overlay, with an additional pool of bootstrap servers used for peer initialisation. The gradient topology overlay allows service registration and discovery through a decentralised service registry, while the DHT overlay enables service consumption (see Figure 1).

Our analysis of the SOA requirements shows that the two different P2P overlays are needed for an efficient, decentralised SOA implementation. The gradient topology allows efficient selection of the most suitable peers in the P2P system to maintain a decentralised service registry. This is important, since measurements on deployed P2P systems show that peer characteristics, such as peer uptime, bandwidth, or available storage space, are highly variable, and often heavy-tailed or scale-free [4], [5], and the usage of low stability or low performance peers can lead to a poor performance of the entire system [3]. Furthermore, research on decentralised search techniques shows that handling complex search queries in highly decentralised systems is very expensive [6], and hence, more centralised approaches to data storage can offer better performance. The gradient topology allows to exploit the resource heterogeneity in a P2P system, and improve search performance, by placing the SOA registry on a subset of the most reliable peers.

The DHT overlay, unlike the gradient topology, provides an efficient and reliable routing algorithm between any two peers in the system. This allows a nearly-uniform distribution of system resources, such as data, user requests, etc., between a group of peers. Furthermore, DHTs support resource replication between peers.

Figure 1 shows a high level picture of our P2P architecture. In order to publish a service, the service provider uses the gradient topology to discover an instance of the service registry (1), registers the service (2), generates a *service proxy* (3), and uploads the proxy using the DHT overlay (4). The service proxy is an object that contains service configuration information, such as the service access protocol, service address, and potentially, service graphical user interface (GUI). Service deregistration follows a similar procedure, where the provider removes the service proxy from the DHT and removes service description from the registry.

In order to discover a service, the consumer uses the gradient topology to discover an instance of the service registry (A), as in the service registration scenario, and queries the registry for available services that satisfy some criteria (B). The registry returns a list of semantic service descriptions that satisfy the specified search criteria. The consumer may then select one of the available services and consume it. Service consumption involves service proxy download from the DHT overlay (C), using a unique service identifier contained in the service description, and service invocation through the service proxy (D), potentially using a graphical interface.

A service proxy hides the service implementation details from the service consumer, such as the communication protocol, encryption settings, or the end-point address, from the service consumer, and enables full service transparency. The provider can change the service configuration transparently from the consumer by updating the service proxy. Furthermore, the DHT overlay provides a lease and automatic clean-up mechanism that removes stale proxies from the system. This allows service consumers to avoid waiting for the service invocation timeout when the service is unavailable, since the DHT does not store proxies of unavailable services.

Gradient Topology

In this section, we describe the design of the gradient P2P topology, and we show how this topology is used to support a decentralised service registry.

In the gradient P2P topology, each peer is assigned a *utility* value and the highest utility peers are connected with each other and form the so called core of the system, while lower utility peers are located gradually farther from the core. Peer utility is a metric that measures the ability of a peer to contribute resources and maintain system infrastructural services, such as the SOA registry. The key property of the gradient topology is that it enables an efficient search algorithm, called *gradient search*, that allows the discovery of the highest utility peers in the system.

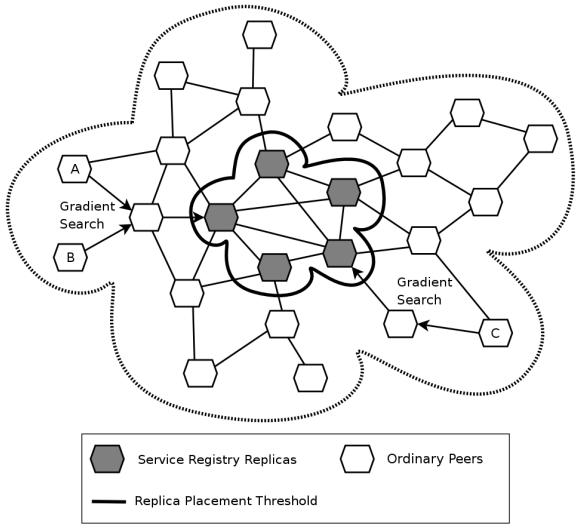

Fig. 2. SOA registry replication and discovery in the gradient topology. Peers A, B, and C access registry replicas, hosted by peers in the core, using gradient search.

In our approach, the SOA registry is distributed between a number of peers in the system for reliability and performance reasons. Hence, there are two types of peers: super-peers that host registry replicas, and ordinary peers that maintain no replicas. A utility *threshold* is defined as a criteria for registry replica placement, i.e., peers with their utility above the selected threshold host registry replicas. Figure 2 shows a sample P2P gradient topology, where the service registry is located at the core peers determined by the replica placement threshold.

Peer utility is defined as a function of peer's local parameters, such as available storage space, bandwidth, latency, CPU performance, average availability, and open IP address. Each of these parameters can be calculated by a peer locally, without the use of global system knowledge. However, setting appropriate replica placement thresholds is more challenging, as it requires the estimation of system-wide characteristics of peer utility values in order to know what constitutes high utility in a running system. If the utility threshold is static (e.g. hardwired), it may happen that no peer in the system has utility above the threshold, or that the threshold is very low and hence sub-optimal. Moreover, due to the system's dynamism, the utility threshold should be continuously adapted to the system's current state and peer availability.

In the gradient topology, peers periodically execute a decentralised *aggregation* algorithm that allows the estimation of system-wide peer utility characteristics. The aggregation algorithm continuously maintains estimates of the following global system properties: the current number of peers in the system, N, the maximum peer utility in the system, Max, and a cumulative *histogram* of peer utility values, H. The cumulative utility histogram with B bins of width w is defined as

$$H(i) = \Big|\{p \mid U(p) \geq i \cdot w\}\Big|$$

for $1 \leq i \leq B$, where $U(p)$ is the utility of peer p. The histogram approximates the cumulative peer *utility distribution*, D, in B points, i.e., $H(i) = D(i \cdot w)$ for $1 \leq i \leq B$, where D is defined as

$$D(u) = \Big|\{p \mid U(p) \geq u\}\Big|.$$

The aggregation algorithm is executed periodically by every peer in the system and is based on gossipping [7]. At one step of the algorithm, a peer sends a message to a random neighbour, and receives messages sent by neighbours in the previous step. A sequence of steps, called an aggregation epoch, leads to a new approximation of N, Max, and H. The details of the aggregation algorithm are described in [8], and due to space limitations, are not covered in this paper.

Peers use the aggregates to calculate the adaptive utility thresholds. We define the *top-k threshold*, as a utility value, t_K, such that K highest utility peers have their utilities above or equal t_K, and all other peers have utilities below t_K. Given the utility distribution function, D, the threshold is described by formula

$$D(t_K) = K.$$

Top-k threshold allows the restriction of the number of replicas in a dynamic system, where peers are continuously joining and leaving, since it has the property that exactly K peers in the system are above the threshold. Top-k threshold can be estimated using a B-bin and w-width utility histogram, H, by

$$t_k \approx w \cdot \max_{1 \leq i \leq B} (H(i) \geq K).$$

Similarly, we define the *proportional threshold*, as a utility value, t_F, such that a fixed fraction F of peers in the system have their utility values above or equal t_F and all other peers have utilities below t_F

$$D(t_F) = F \cdot N$$

where N is the number of peers in the system. The proportional threshold can be approximated using a utility histogram

$$t_F \approx w \cdot \max_{1 \leq i \leq B} (H(i) \geq F \cdot N).$$

As the system grows or shrinks in size, the proportional threshold has an advantage over the top-k threshold, as the number of replicas in the system is increased or decreased accordingly, and the ratio of replicas to ordinary peers remains constant.

In order to generate and maintain the gradient topology, each peer periodically executes a self-organising neighbour selection algorithm [9]. The *neighbourhood set* of a peer p consists of two parts: a *similarity-based* set, S_p, that contains neighbours with similar utility to peer p, and a *random* set, R_p, that contains a random sample of peers in the system. The former set generates the gradient topology structure and enables gradient search, while the latter set improves the topology's robustness, reducing the partition probability to a negligible value, and decreases the network diameter. Random set is also used by the aggregation algorithm for gossipping.

In addition to the neighbour sets, a peer p maintains a cache U_p that stores an estimated utility value, $U_p(q)$, for each neighbour q.

The gradient structure of the topology allows an efficient search heuristic, called gradient search, that enables the discovery of high utility peers in the system. The goal of the search algorithm is to deliver a message from any peer in the system to a high utility peer in the core, i.e., to a peer with its utility above a given threshold. In gradient search, each peer greedily forwards messages to its highest utility neighbour, i.e., to a neighbour q whose utility is equal to

$$\max_{x \in S_p \cup R_p} U_p(x).$$

Thus, messages are forwarded along the utility gradient, as in hill climbing and other similar techniques. The algorithm exploits the information contained in the topology to limit the search space to a relatively small subset of peers and to achieve a significantly better search performance than traditional search techniques, such as flooding or random walking that require the communication with potentially all peers in the system [9]. Gradient search also reduces message loss rate by preferentially forwarding messages to high utility, and hence stable, peers.

In order to determine which peers in the gradient topology host SOA registry replicas, each peer periodically calculates the replica placement threshold, using the aggregates provided by the aggregation algorithm, and compares it with its own utility. If a peer's utility is higher than the replica placement threshold, the peer creates a local replica. The replica is removed, when the peer's utility falls below the replica removal threshold.

In order to access the SOA registry, a peer that does not store a local replica calculates the replica placement threshold, using the utility histogram, and performs gradient search that returns the address of a peer above the threshold that stores a registry replica.

The registry replica synchronisation is handled by higher level protocols on top of the gradient topology. Since the number of registry replicas is significantly lower than the total number of peers in the system, it is possible for the registry to maintain a full list of all replicas in the system. Updates on the registry may be reactively propagated between all replicas when issued, or may be timestamped and periodically disseminated using a gossip algorithm. Such a gossip-based approach is further facilitated by the fact that the replicas are hosted by high utility, stable, and well-connected peers located at the core of the gradient topology.

Distributed Hash Table

This section describes the Distributed Hash Table (DHT) overlay and its usage for service proxy storage.

A DHT is a P2P system that distributes *values,* such as objects, chunks of data, system resources, or user requests, between peers in the system. Each value is associated with a *key,* and the system provides an efficient and deterministic mapping from keys to values. DHTs support three operations: an *insert* operation that associates a given key with a given value and stores the key-value pair on a peer in the system; a *lookup* operation that retrieves the value associated with a given key; and a *delete* operation that removes from the system a given key together with its associated value.

Every peer in a DHT is assigned a *unique identifier* and is responsible for the maintenance of a part of the key space. Usually, a peer is responsible for the keys that are numerically closest to its own identifier.

The three operations, insert, lookup, and delete, require multi-hop routing between peers to access keys and values stored by the system. Typically, DHTs support routing from a peer to any other peer in the system in $O(logN)$ overlay hops, while every peer maintains $O(logN)$ neighbours, where N is the number of peers in the system.

A number of DHT topologies have been proposed, where the earliest and most popular to date systems include Chord [10] and Pastry [11]. An open-source implementation of Pastry, called Bamboo [3], is available and has been deployed on a global P2P testbed called PlanetLab [12]. Bamboo has been shown to handle higher peer churn rates than other state-of-the-art DHT systems [3].

In Bamboo, each peer is assigned a 160-bit unique identifier that is generated randomly when the peer joins the system. Randomisation ensures that peer identifiers are uniformly distributed in the key space, which provides natural load-balancing between peers and ensures that no peer becomes a bottleneck in the system. A peer identified by id selects the neighbours at position i in its neighbourhood table in such a way that the identifiers of these neighbours share a prefix of length i with id. This neighbour selection algorithm generates a topology structure that allows efficient routing between peers and guarantees the worst case cost of insert, lookup, and delete operations to be $O(logN)$ messages.

Message routing is performed in the following way. At the first hop, a message is forwarded to a peer whose identifier has the same first digit as the message destination d. At the second hop, the message is forwarded to a peer whose identifier has the same two leading digits as d. At hop i, the message is forwarded to a peer whose identifier shares a prefix of i digits with d. This way, the message is guaranteed to reach destination d in $O(logN)$ hops. Figure 3 shows an example scenario where peer *0000* is routing two messages to peers *0111* and *1100* respectively.

In the service-oriented architecture presented in this paper, the DHT overlay is used for service proxy storage. Each proxy is associated with a unique service identifier, generated by the service registry, that allows the proxy upload, download, and removal. Furthermore, the Bamboo overlay supports a lease mechanism that sets a lease duration for every proxy stored in the DHT and that automatically removes proxies that have expired, i.e., proxies that have not been renewed by the service provider before the lease expiry time. This mechanism ensures that the proxies stored in the system are up to date.

Moreover, for reliability and performance reasons, each proxy is replicated on a number of peers. Bamboo supports a replication algorithm, where each value stored by the system, such as a service proxy, is stored by K peers that have numerically closest identifiers to the key associated with the replicated value.

Future work will investigate the usage of the DHT overlay

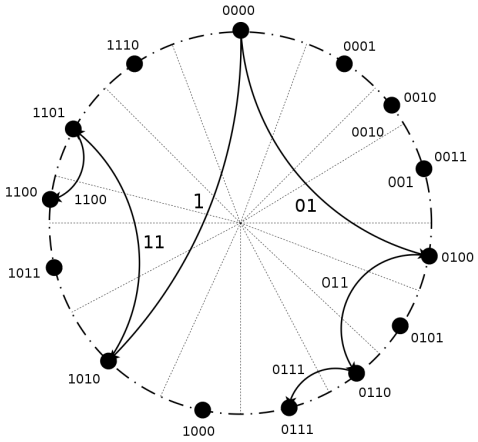

Fig. 3. Peer *0000* sending messages to peers *0111* and *1100*.

to store additional SOA information, such as user public keys and service certificates, and to route service invocation and response messages over the DHT.

EVALUATION

This section describes our initial evaluation of the proposed architecture. In the following two subsections, we evaluate the main two components of the architecture, the gradient topology overlay and the distributed hash table overlay.

Gradient Topology Evaluation

We evaluate the algorithms used in the gradient topology by performing experiments in which we measure the precision of the aggregation algorithm and the performance of gradient search. We measure the average error in histogram estimation, Err_H, number of peers in the system estimation, Err_N, and the maximum utility estimation, Err_{Max}. In the same experiments, we compare the performance of gradient search with random walking [13], [14]. For both search algorithms, we measure two properties. We calculate the average number of hops in which the algorithms deliver a search message from a random peer in the system to a high utility peer above 1% utility threshold, and we measure the search failure rate, i.e., the percentage of search messages that are lost.

The results of our experiments show that the aggregation algorithm offers a good approximation of the system properties and that gradient search exhibits significantly better performance than random walking, in terms of both number of hops and failure probability.

Due to space limitations in this paper, we can only give a brief overview of our experiments. A detailed description of the experiments can be found in [8] and [9].

We ran our experiments on a Pentium 4 machine using a Java-based discrete event simulator, which we developed in order to evaluate our P2P algorithms. An individual experiment consists of a set of peers, connections between peers, and messages passed between peers. We assume all peers are mutually reachable, i.e., any pair of peers can establish a connection. We also assume that it takes exactly one time step to pass a message between a pair of connected peers.

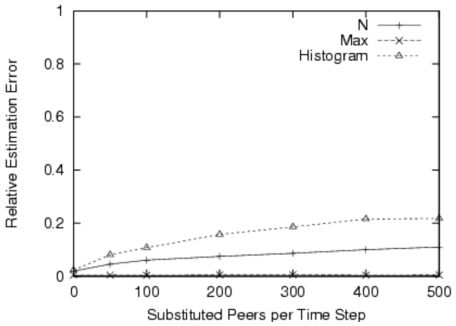

Fig. 4. Average estimation error of the number of peers in the system (N), maximum utility (Max), and the utility distribution (Histogram) as a function of peer churn rate.

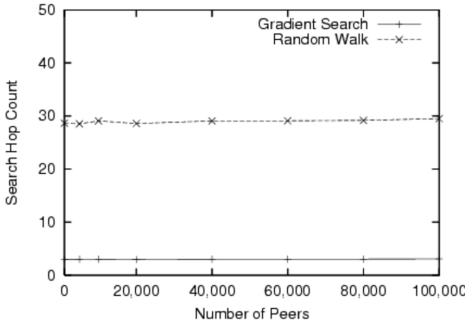

Fig. 5. Average route length (hop count) when searching for peers above the utility threshold. Gradient search is compared with random walking.

We do not model network congestion, however, in order to reflect network heterogeneity, we limit the maximum number of concurrent peer connections according to Pareto distribution (power law).

The simulated P2P network is under constant churn. Every peer is assigned a session duration determined probabilistically with Pareto distribution, where the expected session time is inverse proportional to the simulated peer churn rate. When leaving the system, 10% of peers do not perform the leave procedure, i.e., they crash. Furthermore, a centralised server is used to bootstrap peers that are joining the system.

We start each individual experiment from a network consisting of a single peer. The number of peers is increased by one percent at each time step, until the network grows to the size specified for the experiment. Afterwards, the network is still under continuous peer churn, however, the rate of arrivals is equal to the rate of departures and the number of peers in the system remains constant. Each peer periodically performs the neighbour selection, aggregation, and routing algorithms at every time step of the simulation. Additionally, at each step, a number of randomly selected peers emit search messages that are routed using gradient search or random walking. All peers attempt to either deliver search messages they hold in the buffers, if their utility is higher than the specified search threshold, or forward messages to neighbours selected by the current search policy. Messages are lost if their TTL value drops to zero, or if the peer that stores them in its buffer leaves the system.

Figure 4 shows the average relative error of the aggregation

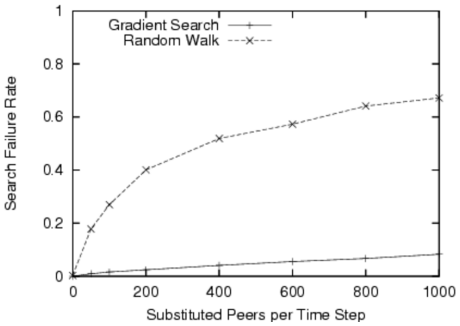

Fig. 6. Average failure rate when searching for peers above the utility threshold. Gradient search is compared with random walking.

algorithm, Err_N, Err_{Max}, and Err_H, as a function of the peer churn rate. The variance is not shown as it is approximately two orders of magnitude lower than the plotted values. The maximum average error does not exceed 20% for the highest churn rates.

Figure 5 shows the average hop count of search messages delivered to peers above the utility threshold as a function of the number of peers in the system. Figure 6 shows the average failure rate when searching for peers above the utility threshold as a function of peer churn rate. Both figures demonstrate superior performance of gradient search over random walking.

Distributed Hash Table Evaluation

A number of Distributed Hash Table systems have been analysed, including Chord [10], Pastry [11], and Symphony [15], and DHTs have been shown in both theoretical and experimental evaluations to exhibit good performance and stability in the presence of high peer churn. In particular, DHTs usually provide routing between any peers in the system in $O(logN)$ hops, given the system size N. Many DHTs have also demonstrated scalability to millions of participating peers.

The evaluation of Bamboo is described in [3]. Research has shown that Bamboo can achieve better performance and higher resilience to peer churn than other existing state-of-the-art DHT systems.

Related Work

An approach to web service discovery that uses a decentralised search engine, based on a P2P network, is described in [16]. In this approach, services are described by a set of keywords and positioned in a multi-dimensional space that is mapped onto a DHT and partitioned between peers. A similar approach, described in [17], partitions the P2P system into a set of ontological clusters, using a P2P topology based on hypercubes, in order to efficiently support complex search queries. However, both these approaches are based on P2P networks that do not reflect peer heterogeneity in the system, unlike our gradient topology, and do not address the problem of high utility peer discovery in a decentralised P2P environment. A number of general search techniques have been developed for unstructured P2P systems (e.g., [13] and [14]), however, these techniques do not exploit any information contained in the underlying P2P topology, and hence achieve lower search performance than the gradient heuristic that takes advantage of the gradient topology structure.

Conclusions

In this paper, we have proposed a decentralised service-oriented architecture built on top of a self-organising peer-to-peer infrastructure. The P2P infrastructure consists of a gradient topology that enables the selection of the most suitable peers for a service registry maintenance, and a distributed hash table overlay that is used for service proxy storage. Service proxies encapsulate service access points, communication protocols, and graphical interfaces. The P2P infrastructure allows an efficient implementation of the SOA operations, i.e., service registration and deregistration by the service provider, and service discovery and consumption by the service consumer. Our evaluation shows that the proposed P2P infrastructure scales to a large number of peers and can successfully manage high peer churn rates.

Acknowledgement

The work described in this paper was partly supported by the "Information Society Technology" Programme of the Commission of the European Union under research contract IST-507953 (DBE).

References

[1] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet Comput.*, vol. 9, no. 1, pp. 75–81, January 2005.

[2] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proc. of the 4th Int. Conference on Web Information Systems Engineering*, 2003, pp. 3–12.

[3] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *Proc. of the USENIX 2004 Annual Technical Conference*, 2004, pp. 127–140.

[4] S. Sen and J. Wong, "Analyzing peer-to-peer traffic across large networks," *IEEE/ACM Trans. Networking*, vol. 12, pp. 219–232, 2004.

[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. of Symposium on Operating Systems Principles*, 2003, pp. 314–329.

[6] J. Li, B. T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris, "On the feasibility of peer-to-peer web indexing and search," in *Proc. of the 2nd Int. Workshop on Peer-to-Peer Systems*, ser. LNCS, no. 2735. Springer, 2003, pp. 207–215.

[7] D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of the 44th IEEE Symposium on Foundations of Computer Science*, 2003, pp. 482–491.

[8] J. Sacha, J. Dowling, R. Cunningham, and R. Meier, "Using aggregation for adaptive super-peer discovery on the gradient topology," in *Proc. of the 2nd IEEE Int. Workshop on Self-Managed Networks, Systems & Services*, ser. LNCS, no. 3996. Springer, 2006, pp. 77–90.

[9] J. Sacha, J. Dowling, R. Cunningham, and Meier, "Discovery of stable peers in a self-organising peer-to-peer gradient topology," in *Proc. of the 6th IFIP Int. Conference on Distributed Applications and Interoperable Systems*, ser. LNCS, no. 4025. Springer, 2006, pp. 70–83.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, 2001.

[11] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of the 18th Int. Conference on Distributed Systems Platforms*. Springer, 2001, pp. 329–350.

[12] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: An overlay testbed for broad-coverage services," *Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, July 2003.

[13] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proc. of the 22nd Int. Conference on Distributed Computing Systems*. IEEE, 2002, pp. 5–14.

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. of the 16th Int. Conference on Supercomputing*. ACM, 2002, pp. 84–95.

[15] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed hashing in a small world," in *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003, pp. 127–140.

[16] C. Schmidt and M. Parashar, "A peer-to-peer approach to web service discovery," *World Wide Web*, vol. 7, no. 2, pp. 211–229, June 2004.

[17] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl, "A scalable and ontology-based p2p infrastructure for semantic web services," in *Proc. of the 2nd Int. Conference on Peer-to-Peer Computing*, 2002, pp. 104–111.