Service and Resource Discovery Supports over P2P Overlays

Emanuele Carlini
IMT Lucca, Italy, and ISTI-CNR
Pisa, Italy
e.carlini@isti.cnr.it

Massimo Coppola, Patrizio Dazzi,
Domenico Laforenza, Susanna Martinelli
Institute of Information Science and
Technologies ISTI-CNR, Pisa, Italy
{m.coppola, d.laforenza, p.dazzi,
s.martinelli}@isti.cnr.it

Laura Ricci
Computer Science Dept.
University of Pisa, Italy
ricci@di.unipi.it

***Abstract*—We describe the main architecture and the design principles of the Service/Resource Discovery System (SRDS), a component of the XtreemOS Operating System. XtreemOS is a Linux extension that enables management and exploitation as single platform of computational resources provided by federated Virtual Organizations. The SRDS provides scalable and fault-tolerant directory services supporting many of the platform functionalities, ranging from dynamic resource location and job control to system and application-oriented directory services.**

The key challenge of the SRDS design is to provide the common metaphor of the directory service, meeting the scalability requirements of a Grid-aware Operating system, and at the same time enjoy extendability and configurability, especially with respect to the quality of service provided. The SRDS design combines different peer to peer structured overlay networks, exploiting their peculiar strengths. We describe the implementation and our design of the namespace abstraction as implemented on top of multiple overlay networks. Finally, we show test results of the SRDS on top of a subset of the Grid5000 platform.

I. Introduction

Exploiting a large amount of geographically scattered computing resources has been made technically feasible in recent years, thanks to the development effort spent in the Grid communities [1]. Nevertheless, daily exploitation of more and more distributed resources in a dynamically changing environment quickly becomes unpractical, due to the number of low-level tasks that have to be performed.

XtreemOS (http://www.xtreemos.eu, [2]) is a research project funded by European Commission, aiming at producing an Open-Source, Grid-enabled Operating System. The XtreemOS approach consists in developing a whole Operating System abstraction layered on top of the computing resources, so that the end-users no longer need to confront with middlewares. POSIX applications shall be transparently run over remote resources, and a specific API is available to developers willing to exploit the advanced services provided by the platform in Grid-aware applications.

9781-4244-3941-6/09/$25.00

XtreemOS goes beyond the experiences made with Grid middlewares and looks at the concrete challenges presented by scalable management of large computational platforms, dispersed among countless administrative domains. XtreemOS goal is to exploit the convergence of results in the SOA and Cloud computing research fields. In forming Virtual Organizations (VOs), XtreemOS allows to federate resources from multiple institutions and that spread over geographic networks. Large clusters, single workstations as well as mobile devices can cooperate through a common security and authorization infrastructure [3].

It is obvious that such a system has to provide efficient interconnections across its components, and allow the exploitation of individual system nodes hiding the effects of scale. Providing highly available and scalable mechanisms that allow communication, information spreading and retrieval is one of the tasks of the project. This includes the implementation of the Service/Resource Discovery System (SRDS), a directory service with resource location features suited to multi-VO environments.

Peer-to-peer (P2P) networks are almost a perfect match for these requirements, as they sport fault tolerance, scalability, low overhead. While their adoption is straightforward, exploiting them efficiently in the general case is less obvious. The SRDS has several kinds of clients, both modules of XtreemOS and applications, and a varied set of tasks (e.g. locating resources, providing

generic directory services, range-based and multidimensional constrained queries).

No single P2P approach can cope with the diversity of requirements, and still provide always the optimal tradeoff. This is especially true as in some cases the balance between the acceptable overhead and the functionalities needed is also application-dependent. In order to provide extendibility (ease to support new XtreemOS modules) as well as configurability (allow modules and applications to choose the best trade off) we designed the SRDS architecture following a layered approach, where the upper layer is a facade module used as an "extendible API", the intermediate layer allows to modularize the retrieval algorithms, and the lower layer integrates under a common abstraction different P2P overlays.

In section II we discuss related results in the literature. The rest of the paper is structured according to the SRDS architecture: in section III we define the ADS structure, section IV defines the information management level and section V describes the DHTs used in the current SRDS implementation. We then discuss in section VI the namespace feature we adopted and its implementation strategies within DHT overlays. Section VII outlines the range-query support we have developed, which exploits an optimized publication algorithm, section VIII follows with preliminary performance results on a Grid platform. Section IX summarizes our contribution and outlines our development roadmap and future research work.

II. Related Work

Several proposals for P2P platforms exploiting a large amount of geographically scattered computing resources have been presented in recent years. *OpenDHT* [4] is a large scale *Distributed Hash Table*, deployed on the *Planet Lab* platform, a geographically dispersed volunteer-based computing platfom. OpenDHT tackles several challenging problems related to the definition of a *DHT* service on a massively distributed computing infrastructure as basic service shared among a large number of different applications and services. The cited work introduces the concept of *namespace* to distinguish data from different services/applications. It implements namespaces by embedding a two dimensional quadtree in the underlying *Bamboo DHT*. Further issues are the definition of a proper *authentication support* for the nodes joining the *DHT*, as well as defining a fair allocation policy of the DHT storage among competing clients and a mechanism to prevent client starvation.

The work [5, Jin *et al.*] proposes a directory service built over the *Chord DHT* infrastructure which acts as a *rendezvous network* connecting multiple VO on a Grid. Only a single node of each VO can become a *DHT node*, and acts as a bridge to link the local VO into the *DHT ring*. The DHT Proxy mediates between the local VO and the remote ones by publishing and deleting information from the dispersed repository, as well as by performing queries across VOs.

Some recent proposals mix *structured* and *unstructured overlay networks* to define a directory service support. For instance in the PARIS *semantic overlay network* [6] local groups of peers are organized in an unstructured overlay. In each group, the peers with the best connectivity are selected to join a *DHT* which enables communication between peers belonging to different groups.

The problem of defining of a hierarchy of namespaces within a flat DHT is strictly related to the definition of the SRDS developed within XtreemOS. Several proposals [7]–[11] of embedding structures within a flat DHT like *Chord* [12] or *Pastry* [13] have been recently presented. *Diminished Chord* [9], introduces subgroups in the Chord ring by embedding a binary tree in the Chord ring. A simple interface is defined to join a subgroup identified by X and to search a key within X. Cyclone [10] divides the identifier assigned to each node into a *suffix* which identifies a namespace and a *prefix* defining the identity of the node within the group of peers identified by the namespace. The choice of the suffix to identify namespaces enables the definition of a hierarchy within the DHT without renouncing to the properties of flat DHTs, like load balancing and uniform distribution of the peers. *Crescendo* [7] defines a hierarchical DHT over a Chord ring where *logical hierarchy* of nodes reflects their partitioning into different groups.

Along with namespaces management, the definition of a support for range queries is another key aspect of the SRDS. Some proposals exploit *Space Filling Curves* [14] to map points from a d-dimensional space of attributes over a flat 1-dimensional space, other ones are based on the definition of hierarchical structure over the DHT. Last but not least, there are approaches are based on the linearization of data by means of a *locality preserving hashing function* [15]. As shown in Sect VII, the SRDS exploits the last approach and develops it further.

III. The ADS Architecture

In order to address the challenges we mentioned in the Introduction, we designed and implemented the Service/Resource Discovery System (SRDS): a coherent framework for providing directory services which also

supports resource lookup and exploitation. In this Section we present the SRDS overall architecture giving a bird-eye-view of its main components and of the interactions between them. The main contribution of the SRDS to XtreemOS consists in providing to applications and system components general-purpose directory services which include the capability of searching for and selecting services and computational resources. SRDS performs a wide range of different tasks, ranging from simple key-based queries to range-based queries over dynamic attributes, while providing a storage service level (e.g. reliability) that is customizable according to the needs of each XtreemOS module that is an SRDS user. The submitted queries can be hard to resolve, as they can potentially involve range constraint over attributes, dynamic information on resources, and multiple fitting criteria expressing the user needs.

The problem is particularly challenging, indeed. If on the one hand efficient implementations of simple directory service functionalities over very large set of nodes are thoroughly studied, on the other hand the possibility to formulate/invoke more complex queries is still considered an open research issue. This is especially true when dynamically variable information has to be managed.

In order to allow efficient management of such a wide range of queries, the SRDS is designed to exploit a combination of different P2P approaches, so to grant the best implementation tradeoff in each situation and not to be restricted to any single solution.

The architecture is layered and modularly decomposed. From the development point of view, the two main modules are the *Application Directory Service* (ADS), and the *Resource Selection Service* (RSS). The SRDS module is in charge of most of the query pre-processing and controls a configurable set of DHT overlays, while the RSS provides a P2P overlay specifically designed [16] to allow scalable resource location in large overlays. Both the ADS and the RSS overlays are established on the set of computing resources managed by the SRDS, each machine generally belonging to several overlays.

A. Architecture Layers

The overall architecture and the SRDS and ADS functional layers are depicted in Figure 1. We will not discuss in depth the upper *Facade* layer, whose purpose is to provide easy-to-extend multiple interface protocols, including Java-RMI, HTTP and DIXI developed inside XtreemOS project. DIXI, based on the SEDA protocol, is an event-driven communication framework, allowing

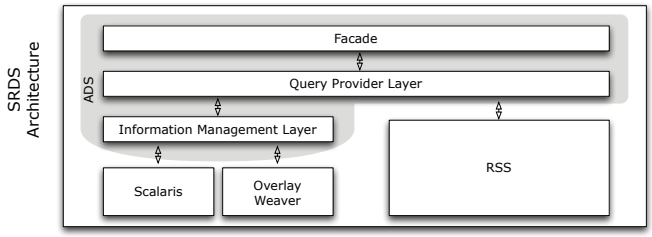


Fig. 1. SRDS overall architecture and main modules.

intra-service communication that is independent from service implementation. Some of these protocols are routinely employed in the XtreemOS system, other ones, like RMI, are solely used for testing and debugging purposes.

The intermediate *Query-Provider* (QP) layer contains a set of modules devoted to query translation, that implement the complex functionalities needed by the users in terms of simpler primitives provided by the overlay managing libraries. By providing interpretation algorithm, we can afford to implement arbitrarily complex functionalities, beyond those provided by the particular P2P overlays that we exploit. An example of complex functionality is the Job Directory Service (JDS), a directory service for XtreemOS applications running on the grid. The JDS is implemented as a set of data structures within a basic DHT, and the algorithms managing the queries (direct and indirect ones) as well as updates to those structures belong to the Query-Provider layer.

The main purpose of keeping Job information within an overlay is to decentralize and decouple it from any specific machine. DHT built-in replication and the design of the Job management System (JMS) protect from resource and JMS faults, and allow transparent migration of job control across different JMS instances (e.g. taking over application steering from XtreemOS mobile devices).

Another example is the mechanisms employed to locate resources within the platform, which is explained later in this section.

Below the previous level we find two main modules providing actual data access. The *Information Management* layer provides a common interface to DHT-like overlay networks. This layer is described in Sect. IV, and beside constituting a common interface to create, configure and access different overlays, it also provides a Namespace abstraction on top of (multiple) DHTs. The DHTs currently employed in XtreemOS are described in

Sect.V.

The second module in this layer, and the second main component of the SRDS, is the RSS, a hierarchically structured overlay network developed within the XtreemOS project. The RSS is described below.

B. Main SRDS Blocks

Overall, we can see that the ADS is heavily modularized and targeting complex functionalities over dynamically changing attributes (e.g. free memory), employing "flat" P2P overlays to store information. On the contrary, the RSS has been designed to manage constant-value attributes (e.g. memory installed on a machine) with very high scalability in a large overlay. The SRDS merges these two profiles in order to maximize its efficiency, in particular to address the resource location problem in a scalable way. Being a performance-critical operation, resource location queries are executed as a two-phase information selection process, in which the RSS and the ADS respectively act as a *"machete"*, which quickly cuts away most unfitting resources, and a *"bistoury"* that carefully removes those violating more complex requirements and dynamic constraints.

a) RSS: The RSS answers to multi-dimensional range queries over static attributes, returning a list of resources identifiers that match the query. By *static* we mean a small set of constant-valued attributes, known at overlay initialization time. On the contrary, it is foreseen that resources may leave (e.g. due to failures) and join the platform dynamically. The query is performed by a specifically designed, structured P2P overlay and a matching distributed search algorithm. A full description of the approach, developed at the Vrije Universiteit Amsterdam, as well as its validation are reported in [16]. In contrast with other P2P approaches that rely on delegation of resources, in RSS each node represents its own attributes in the overlay. The solution enjoys properties that speed up basic resource location. There is no replication overhead and no possible data inconsistency (e.g. when rebooting after an hardware upgrade). Node failures need not be accurately detected by other nodes, and no specific repair operation overhead is necessary to reconstitute the overlay. The query algorithm is designed to avoid creating imbalanced workloads. The result of a query is a large set of resource identifiers, larger by a specified fraction than the amount of resources needed to satisfy the query.

b) ADS: Beside simple query primitives to filter out the RSS results or DHT-like access, ADS provides specific access functions, that are tuned to solve range queries over *dynamic* resource attributes. Dynamic attributes are those that can dynamically change their values, like CPU load. Queries contain predicates over the static and dynamic attributes of the resources (e.g. we want a certain set of software and hardware resources to be currently available at the site). Solving range queries over dynamic attributes is inherently less scalable than answering queries about static values. Extensions of Distributed Hash Table (DHT) techniques and to dynamic attributes and complex queries are employed. However, in order to improve the efficiency of the refining selection process, the ADS exploits a problem-size reduction, leveraging the candidate set for the query that is provided by the RSS. Once the set of candidates has been narrowed acting on basic attributes, it is feasible to exploit DHT-based techniques to apply the more complex constraints. The ADS can also create an application-specific "directory service" using the resource identifiers received by the RSS, those related to the resources (possibly) involved in the application execution.

IV. Information Management Layer (IML)

The interface of the *Information Management Layer* (IML) is the common ground for DHT operations within *ADS*. The *IML* maps operations received by the higher level modules to concrete ones, providing a DHT layer interfaces that hides implementation peculiarities of each overlay. In particular, the IML is required to map the abstract concept of namespace (a user-allocated separate space of keys) to the DHT layer.

Consider the abstract operations received by the *IML* layer:

$$\text{operation}_{QP} = \{\ \textbf{op, key}_M\textbf{, value}_M\textbf{, Nspace, ClientType, ClientId}\ \}$$

At the IML level the namespace (Nspace) holds all information that, at the more abstract levels, might have been conveyed by any parameter for the sake of disambiguating the meaning of the key. The namespace defines thus the context in which the key is used, making explicit information that is implicit in the *SRDS primitive* invoked, in the client type or identity.

The specific name space is provided by the calling QP module (in charge of the SRDS primitive invoked by the user). Different ways of implementing the key spaces at the DHT layer will be discussed in the next section.

The abstract operation is mapped to a concrete operation of the DHT layer defined as follows:

$$\text{operation}_{DHT} = \{\ \textbf{op, key}_D\textbf{, value}_D\textbf{, auxInfo}\ \}$$

While value_M is directly mapped to value_D, because values are provided by upper level modules and go uninterpreted in IML, key translations may happen as a result of namespace implementation. For instance, to ensure uniqueness of key_D in the *DHT* space it may be defined by a concatenation of the namespace unique id, the actual key, and a unique *ID* function of the client, thus providing client separation properties whenever needed. When client separation is not needed, the *ClientType* and *ClientId* information is not exploited.

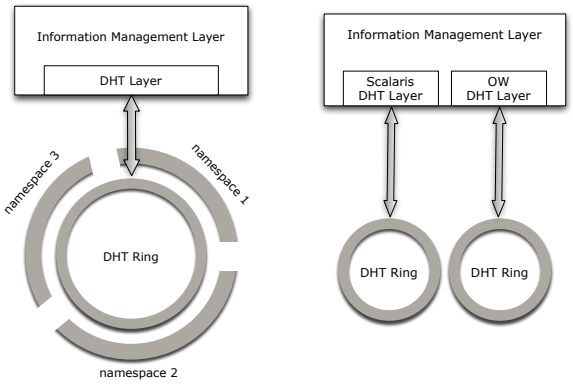
The *auxInfo* field provides hints to the *DHT* implementation about the best tuning of the overlay, whenever the tuning can be performed dynamically. It encodes inferred information from the namespace and the other parameters about the hash functions, data expiration timeouts and so on (e.g. some DHTs provide authorization mechanisms by means of user-defined secrets).

V. DHT IMPLEMENTATIONS

The DHT layer hosts different DHT implementation libraries providing low-level support for both the basic functionalities defined by the IML layer, like put/get operations. The DHT layer also wraps and provides non-standard features like namespaces, implementation details of secret-based autenthication and expiration of requests. This level has to allow a certain degree of customization, concerning for instance the hash function used, or the replication degree. Customization is needed in order to tune the DHT for different usage patterns, and to enable the ADS to support complex queries and dynamic attributes. We first describe the two DHT supports currently adopted in the ADS implementation, and in the next sections we describe the solution exploited by the ADS to support namespaces and *multi attribute range queries*.

A. Scalaris

Scalaris [17] is a P2P overlay that offers transaction-based access to the basic DHT operation. Transactional behaviour is useful when criteria of atomicity are needed to ensure data consistency. For example, the internal DHT key space that register namespaces inside SRDS must be updated avoiding concurrent interferences. The Scalaris internal architecture is also layered. A low-level DHT provides basic put and get, a second layer on top provides availability of data by symmetric replication. Symmetric replication partitions nodes into classes and distributes same item replicas always on nodes of different classes. A third layer implements ACID transactional data accesses on groups of read and write operations.



(a) Single DHT Ring (b) Multiple DHT Rings

Fig. 2. Different implementations of the DHT layer, exploiting either a single DHT ring to hold the information of multiple namespaces (key space partitioning) or a distinct DHT ring for each namespace.

B. Overlay Weaver

Overlay Weaver (OW, [18]) is an implementation of various P2P overlays which aims at separating high level services such as DHT, multicast and anycast from the underlying key-based routing (KBR) level. The OW routing layer architecture follows the KBR concepts but leaves behind the KBR monolithic approach, decomposing the routing layer in a set of independent modules, (e.g. communications, routing and query algorithms). The routing module is defined by three layers: the routing layer (bottom), the service layer and the application layer (top).

VI. NAMESPACE SUPPORT WITHIN THE ADS

As far as it concerns the namespace implementation, two different approaches may be adopted. They are sketched, with reference to the Chord DHT [12], in Fig.2. The two approaches can also be merged to achieve a better tradeoff.

In the implementation of Figure 2(a), a single DHT is used to implement the IML functionalities. Providing a separate instance of the ADS for each application is still possible, as more instances of the ADS can share the local DHT ring instance. In this case, namespace information is used to dynamically select hashing and replication characteristics for the given set of keys. We can map namespaces to additional parameters of the DHT implementation that optimize its behaviour, provided that the DHT allows us to dynamically choose

those parameters. An example is tuning the hash function to enhance support of *range queries*, e.g. by choosing a *linear affine function*, or a *space filling curve mapping*, in order to map locality for a given set of keys to locality over the DHT overlay.

As a concrete example of a single ring supporting multiple spaces, let us consider a distributed directory storing the *network coordinates* of the nodes of the system. Network coordinate embedding systems [19] embed latency such as *Round-Trip-Times* between nodes into some geometric space so that unmeasured RTTs can be estimated using distance computation in that space. The network coordinates of each are stored by the ADS in the underlying DHT, which should support both *direct* and *inverse queries*. The QP module managing the task defines three distinct namespaces: *IP*, *X* and *Y*. The first one enables to retrieve the network coordinates paired with a node whose IP is known. The *X* and *Y* namespaces support inverse queries where the Key is respectively the value of the X and Y coordinate of the host in the cartesian space. Inverse queries are submitted by nodes searching for their nearest neighbours in the networks, where the neighborhood is proportional to the estimated latency. A mechanism to support range queries is defined in these namespaces, in order to support those queries and find all the hosts belonging to a portion of the cartesian space.

The main drawback of a single-ring based solution is that it will not be possible to tune all of its parameters according to the namespaces, e.g. P2P ring repair strategies will have to be common. An alternative solution is shown in Figure 2(b) where *a different ring* is paired with each namespace. Ring creation can happen on-demand, and all parameters and policies of the DHT ring are customized for its specific use at ring set-up time, that is when that specific overlay network is created. In *XtreemOS*, rings dedicated to essential key spaces always remain active, while smaller rings supporting application specific tasks, or temporary Virtual Organizations, can have a shorter lifespan (that of an application or a VO) as well as a lower initialization cost. In this solution the required amount of state and communication effort scales linearly with the number of rings. The need of running intensive communication protocols independently for each application would lead to a significative waste of bandwidth.

The current version of the ADS merges the two approaches, and exploits two different rings, one based on Scalaris and one on the Chord DHT [12] provided by *Overlay Weaver* [18]. A set of namespaces can be defined on the same DHT, although in the current version of the system the namespaces are still statically defined.

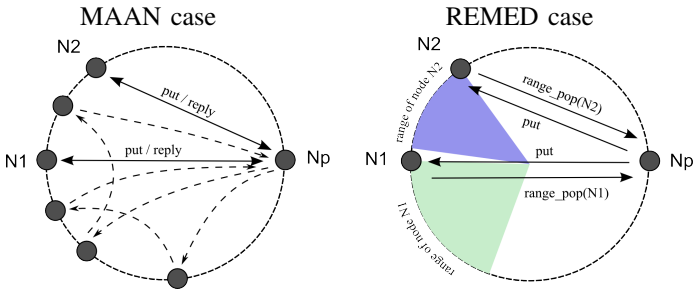


Fig. 3. Successive publications of a resource with one attribute. When the value changes from v_1 to v_2, the storage node switches from $N1 = H(v_1)$ to $N2 = H(v_2)$. (*left*) Without caches, routing is always performed in the OW implementation of CHORD, in order to locate N1 and N2. (*right*) In REMED, routing is skipped if N1 and N2 are found in cache, i.e. values mapped to N1 and N2 were used recently. Reply messages update the cache of the provider node Np with managed range and popularity information.

VII. Range Queries within ADS

The ADS supports *multi attribute range queries* by exploiting the REMED approach [20], based on the MAAN approach [15]. MAAN builds up a publication and query mechanism over a Chord DHT [12]. A data object published by the ADS is defined as a collection of *attribute-value pairs*. Each attribute is coupled with a function that maps values to keys in the DHT space.

Following [15] the key is obtained using a *static locality preserving hash function*, and the publication process is based on replicas of the same object to be stored under the keys of all its attribute values.

A provider node publishes a data object, described by a set of attributes values $(a_1, v_1)..(a_k, v_k)$, by inserting in the DHT ring the whole object description, for each different attribute a_i, at the position given by $H(v_i)$.

Hence object descriptors are spread and replicated into the overlay by a factor equal to the number of attributes.

A generic multi-attribute range query is structured like a set of pairs attribute-constraint. Each constraint is a range of values the attribute must lie within.

Given an attribute constraint $[l, u]$, the resources that satisfy it lie in the contiguous portion of DHT space, bounded by $H(l)$ and $H(u)$, as a consequence of H being a locality preserving hashing function.

The constraint corresponding to the smallest portion of DHT space is called the most *selective*, and its attribute, called *dominant*. If $[l, u]$ is the range associated to the most selective constraint, in order to resolve a lookup query the query is first sent to the node managing $H(l)$ (the lower bound value). The request is then forwarded to

the nodes in the range $H(l)$ to $H(u)$, traversing them all and accumulating on the way the identifiers of resources that satisfy *all* query constraints. When the query reaches the node handling $H(u)$, the set of matches computed for the query is returned to the query initiator.

We developed the REMED optimized approach [20] on top of the MAAN basic publication/query process, to be applied in the ADS resource location to enhance the handling of dynamic attributes. Two optimizations of the basic publication/query process has been defined.

The first REMED optimization pairs each node with a *soft-state* cache of the routing results obtained during the publication phases. We reuse routing results discovered in previous iterations to reduce the number of messages over the network. In figure 3 we see that with respect to the standard MAAN approach, REMED collects through reply messages from target nodes that allows to skip routing for attribute values managed by recently visited nodes.

The second strategy defines the *popularity* of an attribute as the frequency with which it is chosen as dominant. The popularity of a published object, that is its the popularity according to its attribute values, strictly depends on the distribution of queries submitted to the system. As a consequence, the popularity of attribute values can also vary during system lifetime.

We defined a dynamic adaptive solution, where we update with lower frequency resources associated with low popularity attributes (i.e. rarely used in query resolution), and continuously refine popularity estimates in a distributed fashion. Popularity is estimated locally at target nodes, and gathered by caching information provided by the target nodes via reply messages, see figure 3. Cached information is then exploited according to the temporal locality of the dynamic attributes.

For example, if we consider a scenario where an unique attribute is dominant for all the queries, the updates for all the other non-dominant attributes become useless. As a more concrete case, updates are less frequent for those replicas which are rarely used because of the values of some of their attributes. On a large platform, machines with a small memory installed are likely to be unpopular, but if many queries start to concentrate on them, they will be updated in a more and more timely manner.

VIII. Experiments

The SRDS, along with all its subcomponents, with the exceptions of Scalaris, is implemented in Java. Java is a suitable from a performance standpoint, as the main overhead is due to network latencies, and it ensures straightforward compatibility with other Java-based XtreemOS components surrounding the SRDS, including the RSS and the DIXI framework.

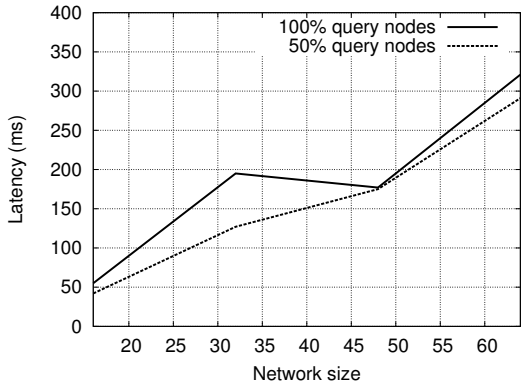


Fig. 4. SRDS scalability with 50% and 100% of the nodes querying information.

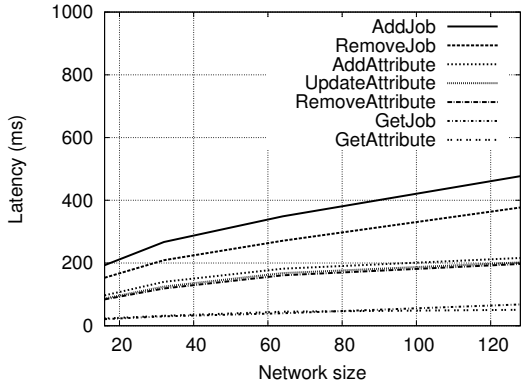


Fig. 5. Latency of JDS operations.

SRDS has been extensively under testing since soon after the beginning of the XtreemOS project. We tested SRDS on a range of different platforms. Simulations of thousands of nodes have been done for evaluating the range-query resolution algorithm [20]. The whole SRDS has been tested both on small-size clusters and over Grid5000 platform, with up to 500 nodes on several sites across France. Tests have regarded both extensive latency measurements and functionality validations.

Figure 4 shows the SRDS behaviour when all nodes provide information every 30s, and a large fraction of them run queries every 100ms. Nodes belong to one or two different clusters of the same Grid5000 site.

To further test the scalability of the SRDS we measured the latency of a subset of JDS operations (Figure 5). Here nodes belong to two different clusters of a Grid5000 site. The complexity of the measured opera-

tions varies, but is rather homogeneous, e.g. *RequestJob* is a single DHT get, while *AddJob* requires a sequence of put/get operations. The test has been repeated with different network sizes and the time interval between each request is 200 milliseconds. The measurements include also the delay of the RMI interface (implemented in the ADS Facade) used to deliver requests to single nodes. Along with the JDS data, all the nodes performed publications of different data over the DHT at a fixed rate (every 30 seconds), in order to realistically simulate a loaded overlay.

IX. Conclusions and Future Work

We described *SRDS*, the service and resource discovery support developed for the *XtreemOS* [21] distributed operating system. It provides scalable and customisable information query support over large platforms, exploiting a combination of different P2P approaches to enhance flexibility and run-time configurability of the system.

We are currently evaluating the functionalities of *SRDS* for different *XtreemOS clients* and on large computing platforms. We plan to investigate a set of solutions for the the dynamic creation of *namespaces*, leveraging the transactional features of the Scalaris DHT. We also plan to to study further strategies for the support of multi-attribute range and neighborhood queries, both with respect to efficiency and to the use of customized hashing functions to improve query performance and load balancing.

Acknowledgment

The authors acknowledge the support of Project FP6-033576, Building and Promoting a Linux-based Operating System to Support Virtual Organizations for Next Generation Grids (2006-2010).

References

[1] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds. San Francisco, CA, USA: Morgan Kaufmann, 1999.

[2] T. Cortes, C. Franke, Y. Jegou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld, "XtreemOS: a Vision for a Grid Operating System," XtreemOS Consortium, Tech. Rep. 4, May 2008.

[3] B. Aziz, A. Arenas, J. Bicarregui, B. Matthews, and E. Yang, "A Formal Security Requirements Model for a Grid-Based Operating System," in *BCS-FACS Christmas 2007 Meeting: Formal Methods in Industry*, ser. Electronic Workshops in Computing Series. British Computing Society, 2007.

[4] S.Rhea, B.Godfrey, B.Karp, J.Kubiatowicz, S.Ratnasamy, S.Shenker, I.Stoica, and H.Yu, "OpenDHT: A Public DHT Service and Its Uses," in *ACM SIGCOMM'05*, Philadelphia, USA, 2005.

[5] H.Jin, Y.Tao, S.Wu, and X.Shi, "Scalable DHT-based Information Service for Large-scale Grids," in *5th ACM Conference On Computing Frontiers*, Ischia, Italy, 2008.

[6] C.Comito, S.Patarin, and D.Talia, "PARIS: A Peer-to-peer Architecture for Large-scale Semantic Data Sharing," *International Journal of Computer Systems Science and Engineering (IJCSS)*, vol. 23, no. 2, pp. 59–76, February 2008.

[7] P.Ganesa, K.Gummadi, and H. Molina, "Canon in G Major: Designing DHT with Hierarchical Structure," in *24th International Conference on Distributed Computing Systems, ICDCS'04*, Tokyo, Japan, March 2004.

[8] P.Di, K.Kutzne, and T.Fuhrmann, "Providing KBR Service for Multiple Applications," in *International Workshop on Peer-to-Peer Systems, IPTPS'08*, Tampa Bay, USA, February 2008.

[9] D.Karger and M.Ruhl, "Disminished Chord: A Protocol for Heterogeneous Subgroup Formation in Peer-to-Peer Networks," in *International Workshop on Peer-to-Peer Systems, IPTPS'04*, La Jolla, USA, 'February 2004.

[10] M.S.Artigas, P. G. Lopez, J. P. Ahullo, and A. Skarmeta, "Cyclone: A Novel Design Schema for Hierarchical DHTs," in *5th IEEE International Conference on Peer-to-Peer Computing*, Kostanz Germany, August September 2005.

[11] V.Ramasubramanian and E.G.Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," in *International Workshop on Peer-to-Peer Systems, IPTPS'04*, La Jolla, USA, February 2004.

[12] I.Stoica, M.Robert, D. Nowell, D.Karger, F.Kaashoek, F.Dabek, and H.Balakrishnan, "Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking, TON*, vol. 11, no. 1, pp. 17–32, February 2003.

[13] R.Antony and P.Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[14] R.Ranjan, A.Harwood, and R.Buyya, "Peer-to-Peer Based Resource Discovery in Global Grids: A Tutorial," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 2, pp. 6–33, 2008.

[15] M.Cai, M.Frank, J.Chen, and P.Szekely, "MAAN: a Multi-attribute Addressable Network for Grid Information Services," in *4th Int.l Workshop on Grid Computing*, 2003, pp. 184–191.

[16] P.Costa, J.Napper, G.Pierre, and M. Steen, "Autonomous Resource Selection for Decentralized Utility Computing," in *9th International Conference on Distributed Computing Systems, ICDCS'09*, Montreal, Canada, June 2009.

[17] T.Schütt and F.Schintke and A.Reinefeld, "Scalaris: Reliable Transactional P2P Key/Value Store," in *7th ACM SIGPLAN workshop on ERLANG*, Victoria, Canada, 2008.

[18] S.Kazuyuki, T.Yoshio, and S.Satoshi, "Overlay Weaver: An Overlay Construction Toolkit," *Computer Communications*, vol. 31, no. 2, pp. 402–412, February 2008.

[19] F.Dabek, R.Cox, F.Kaashoek, and R.Morris, "Vivaldi: a Decentralized Network Coordinate System," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, October 2004.

[20] E. Carlini, M. Coppola, D. Laforenza, and L. Ricci, "Reducing Traffic in DHT-based Discovery Protocols for Dynamic Resources," in *CoreGRID Workshop on Grids, P2P and Service Computing*, Delft, Netherlands, August 2009, colocated with EuroPar, to appear in Springer CoreGRID series.

[21] G. Pierre, T.Schütt, J. Domaschka, and M.Coppola, "Highly Available and Scalable Grid Services," in *3rd Workshop on Dependable Distributed Data Management*, Nuremberg,Germany, March 2009.