Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Final release of highly available and scalable grid services
D3.2.16

Due date of deliverable: April 1^{st}, 2010
Actual submission date: April 1^{st}, 2010

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: WP3.2
Task number: T3.2.1, T3.2.2, T3.2.3, T3.2.4, T3.2.5

Responsible institution: VUA
Editor & and editor's address: Guillaume Pierre
Vrije Universiteit Amsterdam
Dept. of Computer Science
De Boelelaan 1081a
1081HV Amsterdam
The Netherlands

Version 1.0 / Last edited by Guillaume Pierre / March 25, 2010

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.0	16/02/10	Guillaume Pierre	VUA	Initial template
0.1	19/02/10	Guillaume Pierre	VUA	First parts of the content
0.2	19/02/10	Jeffrey Napper	VUA	Distributed Servers content
0.3	24/02/10	Matej Artač	XLAB	DIXI content
0.3	03/03/10	Thorsten Schüttč	ZIB	Pubsub Service
0.4	03/03/10	Corina Stratan	VUA	RSS section
0.5	16/03/10	Massimo Coppola, Emanuele Carlini	CNR	SRDS section
0.6	17/03/10	Jörg Domaschka	ULM	Added section
0.7	18/03/10	Massimo Coppola, Emanuele Carlini	CNR	Changes to SRDS section
0.8	21/03/10	Massimo Coppola	CNR	Last changes to bibliography and document.
0.81	24/03/10	Massimo Coppola	CNR	Comments from internal reviewer concerning SRDS addressed.
0.9	25/03/10	Jeffrey Napper	VUA	Edited discussion of distributed servers according to comments from internal review.
01.0	25/03/10	Guillaume Pierre	VUA	Final wrap-up according to internal review.

Reviewers:
Ramon Nou (BSC) and Benjamin Aziz (STFC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.2.1	Design and implementation of distributed servers	VUA*
T3.2.2	Design and implementation of a scalable publish/subscribe system	ZIB*
T3.2.3	Design and implementation of a service/resource discovery system	CNR*, VUA
T3.2.4	Design and implementation of a virtual node system	ULM*
T3.2.5	Cloud computing services	VUA*
T3.2.6	Distributed XtreemOS Infrastructure (DIXI)	XLAB*

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive summary

Work package WP3.2 is dedicated to building a collection of highly-available and scalable services as a support of the development of the XtreemOS infrastructure. We therefore designed and built several services that respectively address issues of distribution transparency, information dissemination, service and resource discovery, fault-tolerance, and inter-service communication. In addition, we investigated the relationships between XtreemOS and the emerging area of Cloud computing.

The present deliverable mostly aims at delivering our service implementations. These implementations can be found in the publicly-available svn repository of the XtreemOS project. We complement this code delivery with a brief description of each delivered service and its contribution to the project as a whole.

1 Introduction

Work package WP3.2 is dedicated to building a collection of highly-available and scalable services as a support of the development of the XtreemOS infrastructure. We therefore designed and built several services that respectively address issues of distribution transparency (see Section 2), information dissemination (Section 3), service and resource discovery (Section 4), fault-tolerance (Section 5), and inter-service communication (Section 6). In addition, we investigated the relationships between XtreemOS and the emerging area of Cloud computing (Section 7).

The present deliverable mostly aims at delivering our service implementations. These implementations can be found in the publicly-available svn repository of the XtreemOS project. We complement this code delivery with a brief description of each delivered service and its contribution to the project as a whole.

2 Design and implementation of distributed servers

When building a large-scale distributed service made of multiple service instances, it is important to give users a simple contact address where queries can be sent. The Distributed Servers system provides location transparent services using a single *distributed server address* that clients connect to and can thereafter be moved transparently among multiple locations [39]. Mobile IPv6 (MIPv6) route optimization transparently adjusts the clients' connections [20]: all IPv6 connections from the client are atomically changed to each location in order to avoid triangular routing. The TCP Connection Passing (TCPCP) Linux kernel module handles migrating the network stack at the server end [1]. The set of nodes that manages the distributed server address is composed of a *contact node*—to which a client first connects—and a set of *server nodes* that can accept or initiate client handoffs. The distributed server address is simply a mobile IPv6 address [11]. When a client is *handed off*, the server endpoint of all of the client's IP connections are transferred to different server for load-balancing or for client-specific processing. However, Distributed Servers can provide server mobility, inverting the mobile host functionality of MIPv6. The Distributed Servers system depends on all parties—clients and servers—possessing and using IPv6 addresses with support for Mobile IPv6, which is well supported in most modern operating systems.

The standard MIPv6 implementation in Linux is handled by a user-level daemon, called *mip6d,* that manages mobility events. Different roles in MIPv6 are handled by different configurations of the same mip6d daemon: the Mobile Host (MH) changes network addresses generating mobility events; the Home Agent (HA) manages the MH home address in the home network for connectivity of new

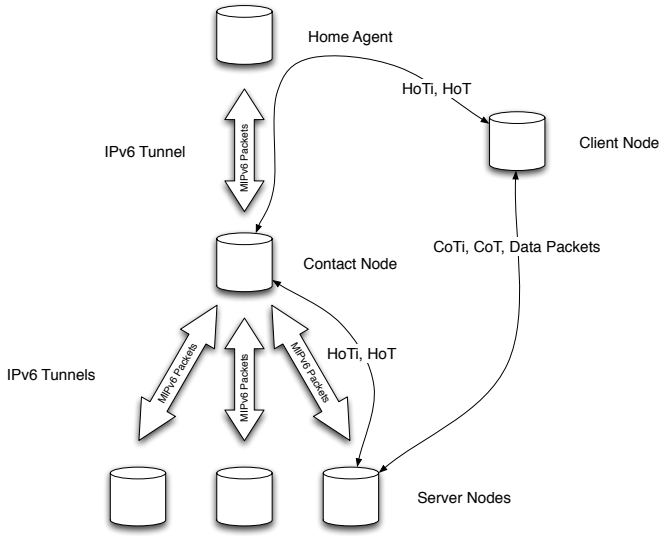


Figure 1: Overview of Distributed Servers design architecture. All MIPv6 control packets from server nodes are routed through IPv6 tunnels to a spoofed Home Agent on the Contact Node. The route optimization packets (HoTi,HoT) are then routed to the real Home Agent, which in turn routes them to the client node.

connections; and the Correspondent Node (CN) represents the other endpoint of connections with the MH during mobility events [20].

Our design places the mip6d in a controlled environment *without modifying the MIPv6 implementation* to simulate mobility of the server end of a connection to clients. The relationship between the MIPv6 and Distributed Servers terminology is straightforward. Figure 1 shows the relationships between the different servers and the client. The Home Agent (HA) is a component of MIPv6 and serves precisely the same role in Distributed Servers. The Distributed Servers *contact node* accepts new connections from clients and is registered with the HA, appearing as the Mobile Host. Other *server nodes* accept client handoffs from the contact node and other server nodes. Both the contact node and set of server nodes are configured as Mobile Hosts in the MIPv6 terminology. Clients that contact these nodes should be configured as Correspondent Nodes.

A typical client handoff proceeds as follows: (1) the TCPCP module at the donor is used to freeze all of the clients' connections and to copy the state from the corresponding network stack; (2) the donor sends the saved state to the receiver; (3) the receiver uses the TCPCP module to recreate the corresponding connections to the client; (4) the dsco daemon at the receiver injects spoofed packets to cause mip6d to initiate Mobile IPv6 route optimization; and (5) the dsco daemon at the

donor injects spoofed packets to break the clients bindings in the mip6d at the donor. The routing of MIPv6 packets is illustrated in Figure 1.

Distributed Servers provide an abstraction that allows a group of server processes to appear as a single entity to its clients, all while transferring a client's connection between servers in the group. The Distributed Servers platform was first included in XtreemOS 2.1. It uses a collection of scripts, a kernel module, and a system-level daemon to provide the Distributed Servers abstraction to applications. Applications use a separate library called Gecko to manage client handoffs between different servers [40]. The Gecko library provides a high-level API that coordinates client handoffs and communicates with the system-level dsco daemon. This daemon then injects or drops packets to manipulate the unmodified Mobile IPv6 mip6d daemon to implement Distributed Servers behavior at each server.

3 Design and implementation of a scalable publish/subscribe system

Developing large-scale distributed services is a hard problem. The two main issues are scalability and fault tolerance. Therefore, we based the PubSub service on distributed hash tables (DHT, [38]) which are a proven technology particularly suited for our scenario. They provide a key-value store with efficient lookups given the value of an item. The cost of lookups in DHTs scales logarithmically with the size of the system. So even in very large deployments low latency lookups are possible, while the throughput scales almost linearly with the number of nodes. The DHT organizes the participating nodes in a ring structure which is autonomically maintained and repaired when nodes join or leave the system.

For XtreemOS, we extended an existing key-value store, Scalaris [31], to support PubSub primitives. In contrast to other distributed key-value stores, it supports the execution of arbitrary transactions on the stored data. Scalaris is made up of three layers: (a) a P2P overlay, (b) a primitive key-value store and (c) a transaction layer. The overlay layer provides the aforementioned properties: scalability and failure tolerance. The second layer implements a simple key-value store with replication and weak consistency. The transaction layer extends the second layer with transactions and strong consistency. A lot of efforts in XtreemOS went into tuning and improving these three layers. The resulting data store is used by SRDS for storing monitoring information.

The PubSub service is topic based, i.e. nodes can subscribe to topics identified by a keyword and messages can be published to all subscribers of a given topic. It is implemented as an application running on top of Scalaris. It uses transactions

for maintaining the subscriber lists for each topic. The transactions guarantee that all subscribe and unsubscribe operations are executed atomically. Parts of the overlay structure are used to efficiently deliver the messages to the subscribers.

Scalaris is implemented in Erlang, a functional programming language designed for distributed and fault-tolerant systems. For the key-value store as well as the PubSub service, we provide an API based on HTTP and JSON. This API can be easily accessed from most programming languages. For Java, we provide a separate library which provides a more convenient API that hides the aspects concerning the communication between the Java process and Scalaris.

4 Design and implementation of a service/resource discovery system

The Service and Resource Directory Service (SRDS) is a meta-service allowing both modules of an XtreemOS system and users, as well as their applications, to keep track in an efficient and scalable way of the distributed status of the system.

The main task of SRDS within the XtreemOS architecture is to locate resources according to the user needs, in this interacting primarily with the Application Execution Management (AEM) service and with the Resource Selection Service (RSS). Beside the resource directory service, the SRDS manages a wide range of directory services, including the job directory service (JDS), the mobile device directory service and the XOSD directory service.

The mechanism used to locate resources within XtreemOS exemplifies the SRDS approach. When looking for resources, the SRDS leverages a combination of two distinct P2P approaches, the RSS one (very scalable and efficient in answering queries based on constant-valued attributes of resources) and a DHT network (still scalable and more suited to dynamically changing data), that provides additional information needed to refine the query results.

4.1 SRDS

The key requisites of the SRDS are those of the XtreemOS platform itself: scalability with respect to the platform size and to the number of users, in terms of service time, throughput, and reliability. The SRDS software architecture, shown in Figure 2, is layered and inherently distributed, exploiting multiple P2P techniques to meet those constraints.

- On each node of the XtreemOS platform, a lightweight service hub provides the SRDS API through the DIXI service bus, as well as through other

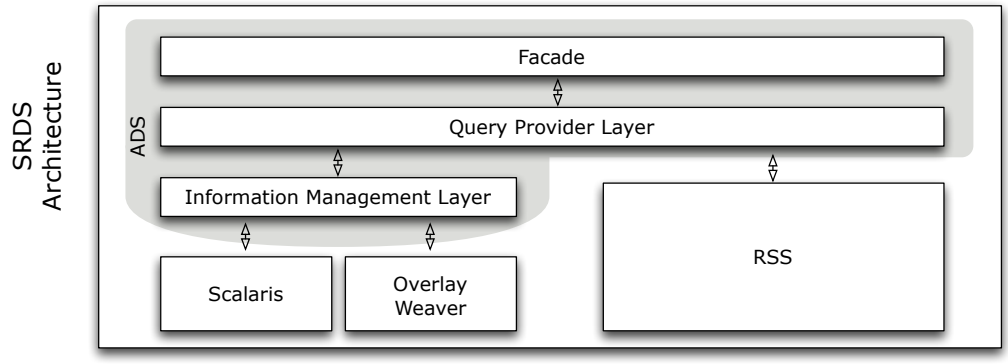


Figure 2: SRDS overall architecture and main modules, which is present in each XtreemOS enabled non-mobile device.

selectable adapters (*Facade*, in the figure). This means that each SRDS endpoint will typically receive queries originated on the same machine, but it is also able to authenticate and serve remote service requests (e.g. from mobile devices).

- The SRDS *Query-Provider Layer* transforms client requests into a combination of primitive operations over the various overlay, thus providing complex query functionalities. Both the interface and the query transformation layers are implemented in Java.

- In order to store and retrieve data, each physical machine participates in one or more P2P networks, possibly of different type. Networks are dynamically enabled according to the characteristics of each one and to the current system needs. Each query is thus forwarded through the networks from where information will be gathered. The P2P network exploited can be implemented in Java and run on the same or on a separate Java VM, or can be used with Java adapters, like it happens with Scalaris. Dynamic activation and interface adaptation to more independent DHT networks is performed by the SRDS internal *Information Management Layer* (see also Figure 3).

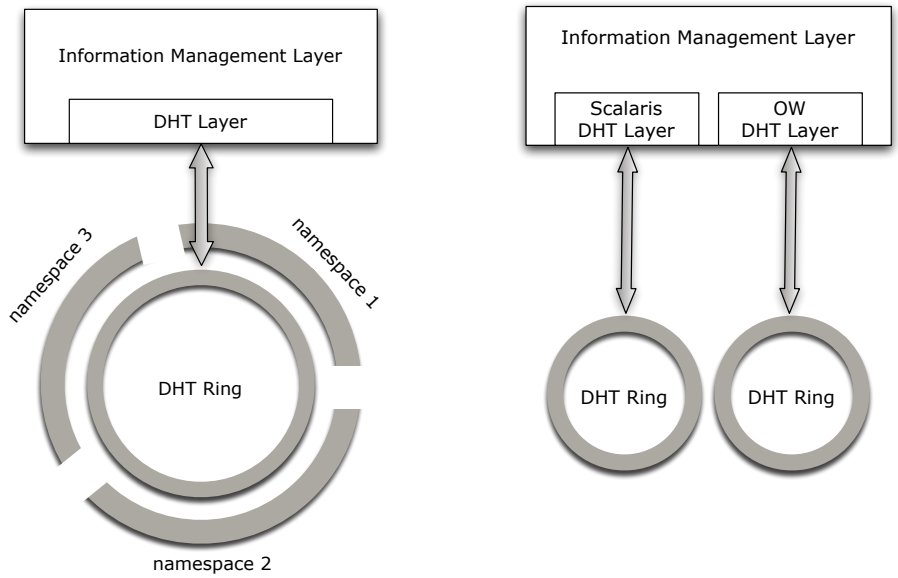
The ability to leverage a set of P2P networks, each one providing different functionalities and properties, allows to achieve a good performance/overhead trade-off for a broad set of query kinds. Besides, SRDS inherits the common scalability and fault-tolerance traits of the P2P approaches it uses.

SRDS decouples the underlying physical overlay network from the logical high level service, by using a name-space mechanism. According to the require-

ments of each SRDS client, a specific set of data (e.g. the list of active XtreemOS Jobs) is mapped into a subspace of a specific DHT, without interfering with data related to other services concurring on the same physical Distributed Hash Table (DHT).

Three different P2P overlay networks are integrated in the SRDS architecture. The Scalaris network (which also supports the publish/subscribe XtreemOS service, and is described earlier in this document) is used when a transactional behaviour is needed to control concurrent modification of the distributed data. The RSS network, also described in this document, is exploited to achieve the best scalability in resource location with respect to resource attributes which are constant during the resource lifetime. A generic P2P construction framework, OverlayWeaver [37] is used as foundation layer for general purpose directory services.

For the sake of providing greater scalability to the directory services, in the framework of the XtreemOS project a research activity focusing on P2P support for complex queries has been pursued. Here we only report the XtreemOS related part of the P2P line of research currently developed at ISTI-CNR. Two different DHT solutions have been developed in order to provide scalable multi-attribute



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

Figure 3: 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.

range-queries over a DHT overlay, and one of the two has been integrated within the SRDS.

The Distributed Digest Trie (DDT) approach [4] has been developed as a generalisation of the CONE approach [41]. DDT provides search functionalities over dynamic data, based on a distributed trie data structure and customisable digest functions. DDT improves on the CONE approach as it provides full-fledged range queries and supports customisable digest functions to tune the accuracy/overhead trade-off, where CONE behaves like a distributed heap and can only handle one-side-bounded queries. DDT has been integrated in the OverlayWeaver framework, but is not currently used within XtreemOS.

The REMED (REduce MEssages in Dht) original mechanism [16] has been added to the OverlayWeaver framework, extending the MAAN approach [3]. REMED supports range queries over dynamically changing data with reduced overhead. The optimization focuses on the updates needed for data stored within the DHT. In REMED, data update frequency is dynamically tuned according to the popularity of specific attributes as measured by the queries received, and taking into account the impact on the query results. REMED integrates with the existing MAAN query resolution mechanism, and is implemented as an additional OverlayWeaver module. Its use in the SRDS specifically allows efficient space-based retrieval of objects from the Mobile-Device Directory Service.

4.2 RSS

The role of the Resource Selection Service (RSS) is to search for resources that match a set of criteria specified in the job description files. The search in RSS is done only based on static resource attributes (e.g., the CPU frequency, the amount of memory, the version of a software library). Specifically, RSS handles queries that define a desired range for each of the static attributes that describe the resources. In a further step, SRDS filters the set of resources provided by RSS according to dynamic parameters.

RSS is a decentralized service, in which the resource nodes are organized in a P2P overlay. Each node provides information about itself (that is, its own static attribute values). The nodes are placed in a virtual multi-dimensional space, in which each attribute corresponds to one dimension. The coordinates of the nodes in this space correspond to their attribute values. In order to allow for efficient searching, the space is recursively divided into nested cells, with the nodes maintaining connections to other nodes from the neighboring cells.

The RSS overlay is maintained by using gossip protocols. The Cyclon protocol [43] has the role of periodically providing each node with references to a set of other random nodes from the system. On top of Cyclon, we use a modified version of the Vicinity protocol [44] to maintain links to nodes that belong to neighboring cells, for all the possible directions in the multi-dimensional space and for all the nesting levels.

A query can be initiated by any node in the system, and is routed through the overlay links until it reaches the cells that overlap with the required ranges. This routing mechanism is important for scalability, as the number of hops needed to reach a matching cell does not depend on the system size. Also, due to the use of gossip protocols, the system is tolerant to node failures. More details about the design and performance of RSS can be found in [9].

The nodes participating to an RSS overlay belong to the same Virtual Organization; in a multi-VO grid there should be a separate RSS overlay for each VO. In order to prevent attacks, RSS provides authentication mechanisms to ensure that the overlay only contains node belonging to the respective Virtual Organization. The nodes are authenticated with X.509 certificates and all messages exchanged among them are digitally signed.

RSS is implemented in Java, its parameters (including the set of static attributes) can be modified through a configuration file. Most of the communication within the overlay is done through TCP and, for efficiency, the nodes maintain persistent connections to their neighbors.

5 Design and implementation of a virtual node system

In a distributed application some processes may have a key role so that their failure would be critical to the entire application. In case the application has interaction with the outside world also availability suffers.

We propose replication as a solution to both issues. Replication increases availability of services on one side and, on the other side, provides reliability of critical components. The Virtual Nodes replication framework provides replication support for Java services.

Replication is expensive in terms of performance. Thus, it is important to fine-tune replication strategies towards the needs of the application to be replicated. Therefore, Virtual Nodes comes with high configurability as its key feature. Basically, there are three main parameters to be set at server-side. The number and location of replicas determines the degree of resilience to node failures. The replication strategy defines the operations the replicated service is allowed to execute

and also decides on the achievable throughput and computation power required. Finally, the middleware adapter sets the API clients have to use in order to access the service.

Virtual Nodes supports two basic replication strategies: active replication and passive replication. In passive replication requests are only executed by a single and fixed replica (called *leader*) which pushes modifications of the application state to the other replicas (called *backups*). Passive replication imposes barely any restrictions on the code of the replicated application so that it can be widely applied. As a downside, it requires sequential execution of requests. Active replication in turn is more rigid with regards to implementation restrictions. It requires that the implementation of the application be deterministic. This is due to the fact that all replicas execute requests independently and still have to have consistent states. On the upside, it allows concurrent execution of request, as long as scheduling of threads is deterministic. Virtual Nodes comes with a set of deterministic scheduling algorithms that allow a fine-grained tuning of concurrency. A consequence of concurreny is higher throughput when using active replication.

Virtual Nodes is one of very few replication frameworks that does neither require a fixed configuration of the number of replicas nor the locations they run on. This means, that it is possible to add and remove replicas at run-time and to even instantiate them on locations that have not been known at startup of the very first replica. All location- and configuration-related issues are handeled within the replica group in a distributed manner. An instance of Virtual Nodes is completely self-contained in a way that it also replicates its management information. This makes it independent from third-party services that may constitute a single-point of failure.

Finally, Virtual Nodes is opaque towards the middleware API the client application uses. It is only required to implement a middleware adapter that maps invocations of the middleware API to calls of the framework. For the time being, Virtual Nodes comes with adapters for Java RMI and the Distributed XtreemOS Interface which cover nearly all applications being used in XtreemOS. Both middleware adapters provide a powerful yet simple user interface to service providers (i.e., administrators) and are fully transparent to service users. In particular, the code of a client application does not need to be modified when the service is replicated with Virtual Nodes.

The usability of Virtual Nodes was shown by replicating an off-the-shelf POP3 service in less than a week. Furthermore, the successfull integration of Virtual Nodes with DIXI in order to replicate XtreemOS' application execution management (AEM) infrastructure makes Virtual Nodes a key component to the reliability of XtreemOS.

6 Distributed XtreemOS Infrastructure (DIXI)

During the development of XtreemOS components, most notably the AEM, a need emerged for a framework and a message bus that would offer quick prototyping of the services, provide a staging environment for the XtreemOS services, simplify a development of a distributed system, composed of services, provide a communication layer between services running on the same node and provide communication between the nodes. To accommodate these requirements, we developed the Distributed XtreemOS Infrastructure (DIXI). The related WP3.2 task was focused on meeting additional requirements expressed by the developers to adopt the framework. In this section we summarise the features of the framework, while the detailed description can be found in [48].

The framework consists of two main parts: the development helper tool, and the runtime environment. The development helper tool's purpose is to analyse the implementation of any DIXI hosted components' code, and produce such auxiliary code that can be derived from the service interfaces. The auxiliary code enables that user's services can easily be integrated into the DIXI framework, and thus instantly be able to communicate and cooperate with any other service within the framework.

The second main part consists of the runtime libraries that enable the deployment of the services, their hosting in the staging environment, and the actual ability to exchange service messages throughout the distributed environment. In this respect the framework acts as a messaging bus. The stages hosted within the same memory space use memory message queues. To gain inter-process and inter-host communication capabilities, it also includes a special stage which supports the message transportation using plain TCP/IP socket connectivity as well as the SSL communication to gain privacy and the ability to properly authenticate the client and server.

The stages implemented and running as services within the DIXI environment represent a top layer which, unless required otherwise, is not aware of the specifics of the target service location or the kind of transport that would be required to invoke the service call and get the result back. This is made possible because the framework performs the needed look-up of the hosts running the target service, and directing the messages to their proper destination. In this process it takes advantage of the high availability services such as Scalaris to publish the services' presence. Intrinsically, we have added mechanisms to control individual stages' lifetime, and the configuration of the framework has become scripting-friendly.

The runtime part is complete with the libraries that can be used by the clients that are not a part of DIXI themselves. The bindings are provided for Java and C. Further, the clients can use their own implementation of the communication modules, since the service messages are exchangeable with the DIXI hosts using

Java binary serialization, or using XML. Initially the clients acted as a special case of DIXI services, requiring a server socket to be accessible from the VOs. For the final release, we enabled a proxy service permitting the utility of real clients connecting from NAT or from behind a firewall. On top of the client programs developed with usage of the client libraries, the users can also employ a DIXI console which contains commands directly obtained from the services' API.

The DIXI has therefore become an integral part of XtreemOS, hosting many of the services itself and bridging others with wrappers.

7 Cloud computing services

Contrary to other tasks within WP3.2, the task on Cloud computing services did not primarily aim at producing any specific service. Rather, it was meant as a support for investigating the relationships between XtreemOS and the then-emerging area of Cloud computing. The results of this investigation can be found in deliverable D3.2.15 [47]. Although this was not the prime goal of the task, one side product of our investigations is the port of the open-source HBase NoSQL data service to XtreemOS..

Relational databases (RDBMS) such as MySQL and Oracle have been popular for decades thanks to their conceptual simplicity, the great expressive power of the SQL query language, and the performance improvements that have been brought by decades of development. However, their great expressive power also makes it very difficult to *scale* them up by using large numbers of computers instead of a single powerful database server. Because users are assumed to be likely to query any set of data items from the database through a single query, distributed RDBMSs often rely on *full replication* to distribute their computation across multiple machines. Read queries can thus be addressed to any replica, and scale very well. On the other hand, update queries must in one way or another be propagated to all replicas. This means that, when using N database replicas, each replica must process $\frac{1}{N} \times ReadQueries + WriteQueries$. When the number of write queries alone grows beyond the capacity of a single replica server, no additional improvement can be brought by adding extra replicas. The only solution to scale an application further is to use data partitioning [45]. Partitioning data manually is a difficult process, so developers prefer to rely on automatic data partitioning.

A new family of scalable database systems is being developed for Cloud computing environments, exemplified by Amazon.com's SimpleDB [2], Google's Bigtable [6], Yahoo's PNUTS [7] and Facebook's Cassandra [21]. Although all these systems are slightly different from each other, they all rely on the same underlying principles. These systems scale nearly linearly with the number of servers they are using, thanks to the systematic use of automatic data partitioning. On the other

hand, they do not support the SQL language and rather provide a simpler query language. Data are organized in tables, which can be queried by primary key only. Similarly, these systems do not support join operations. As restrictive as such limitations may look, they do allow to build useful applications. Scalable database systems typically provide weak consistency guarantees such as eventual consistency [42] or single-record transactions, but one can apply stronger consistency as an added layer on top [46].

To demonstrate how XtreemOS can be a great platform for PaaS Cloud computing, we ported the HBase system [18] (an open-source clone of Bigtable) to XtreemOS. This provides XtreemOS with a scalable database service that can be used by Grid applications to store and query their structured data.

8 Conclusion

In conclusion, WP3.2 has delivered a collection of services that allowed the construction of XtreemOS as a successful software development project.

In addition to producing code, it should be noted that WP3.2 has also been very successful in terms of academic publications. Over the course of the project the work package has published 3 journal articles [8, 35, 39], 1 book chapter [24], 11 papers in international conferences [4, 9, 14, 15, 25, 26, 27, 28, 32, 34, 36], 13 papers in international workshops [5, 10, 12, 13, 16, 17, 19, 22, 23, 29, 31, 33, 49] and 1 paper in a non peer-reviewed venue [30], notwithstanding papers currently under review or soon to be submitted.

References

[1] Werner Almesberger. TCP connection passing. In *Ottawa Linux Symposium*, July 2004.

[2] Amazon.com. Amazon SimpleDB. `http://aws.amazon.com/simpledb`.

[3] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 184. IEEE Computer Society, 2003.

[4] D. Carfi, M. Coppola, D. Laforenza, and L. Ricci. DDT: A distributed data structure for the support of P2P range query. In *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, nov. 2009.

[5] E. Carlini, M. Coppola, P. Dazzi, D. Laforenza, S. Martinelli, and L. Ricci. Service and resource discovery supports over p2p overlays. In *Proceedings of International Conference of Ultra Modern Telecommunications (ICUMT)*, 2009.

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable : a distributed storage system for structured data. In *Proceedings of The 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 205–218, 2006.

[7] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. In *Proceedings of the 34th International Conference on Very Large Data Bases*, pages 1277–1288, 2008.

[8] Massimo Coppola, Yvon Jegou, Brian Matthews, Christine Morin, Luis Pablo Prieto, Oscar David Sanchez, Erica Y. Yang, and Haiyan Yu. Virtual organization support within a grid-wide operating system. *IEEE Internet Computing*, 12(2):20–28, February 2008.

[9] Paolo Costa, Jeff Napper, Guillaume Pierre, and Maarten van Steen. Autonomous resource selection for decentralized utility computing. In *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems (ICDCS)*, Montreal, Canada, June 2009.

[10] Paolo Costa, Guillaume Pierre, Alexander Reinefeld, Thorsten Schütt, and Maarten van Steen. Sloppy management of structured P2P services. In *Proceedings of the Third Workshop on Hot Topics in Autonomic Computing (HotAC)*, June 2008.

[11] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). RFC 2460, December 1998.

[12] J. Domaschka, H. P. Reiser, and F. J. Hauck. Towards generic and middleware-independent support for replicated, distributed objects. In *Proceedings of the 1st workshop on Middleware-application interaction*, March 2007.

[13] J. Domaschka, A. I. Schmied, H. P. Reiser, and F. J. Hauck. Revisiting deterministic multithreading strategies. In *Proceedings of the 9th International Workshop on Java and Components for Parallelism, Distribution and Concurrency*, March 2007.

[14] Jörg Domaschka, Thomas Bestfleisch, Franz J. Hauck, Hans P. Reiser, and Rüdiger Kapitza. Multithreading strategies for replicated objects. In *Proceedings of the 9th International Middleware Conference*, December 2008.

[15] Jörg Domaschka, Franz J. Hauck, Hans P. Reiser, and Rüdiger Kapitza. Deterministic multithreading for java-based replicated objects. In *Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2006.

[16] D. Laforenza E. Carlini, M. Coppola and L. Ricci. Reducing traffic in dht-based discovery protocols for dynamic resources. In *CoreGrid ERCIM Working Group Workshop on Grids, P2P and Service Computing in conjunction with EuroPAR 2009*, 2009.

[17] Marco Fiscato, Paolo Costa, and Guillaume Pierre. On the feasibility of decentralized grid scheduling. In *Proceedings of the Workshop on Decentralized Self-Management for Grids, P2P, and User Communities (Selfman)*, Venice, Italy, October 2008.

[18] HBase. An open-source, distributed, column-oriented store modeled after the Google Bigtable paper. `http://hadoop.apache.org/hbase/`.

[19] Mikael Hoegqvist, Seif Haridi, Nico Kruber, Alexander Reinefeld, and Thorsten Schütt. Using global information for load balancing in DHTs. In *Proceedings of the Workshop on Decentralized Self Management for Grids, P2P, and User Communities*, October 2008.

[20] D. Johnson, C. Perkins, and J. Arkko. Mobility support in IPv6. RFC 3775, June 2004.

[21] Avinash Lakshman, Prashant Malik, and Karthik Ranganathan. Cassandra: A structured storage system on a P2P network. In *Keynote talk at the ACM SIGMOD international conference on Management of Data*, 2008.

[22] Guillaume Pierre, Thorsten Schütt, Jörg Domaschka, and Massimo Coppola. Highly available and scalable grid services. In *Proceedings of the Third Workshop on Dependable Distributed Data Management*, March 2009.

[23] S. Plantikow, A. Reinefeld, and F. Schintke. Transactions for distributed Wikis on structured overlays. In *Proceedings of the 18th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, October 2007.

[24] A. Reinefeld, F. Schintke, T. Schütt, and S. Haridi. A scalable, transactional data store for future Internet services. In G. Tselentis, J. Domingue, A. Galis, A. Gavras, D. Hausheer, S. Krco, V. Lotz, and T. Zahariadis, editors, *Towards the Future Internet - A European Research Perspective*, pages 148–159. IOS Press, 2009.

[25] Alexander Reinefeld and Thorsten Schütt. Out-of-core parallel heuristic search with MapReduce. In *Proceedings of the High-Performance Computing Symposium (HPCS)*, June 2009.

[26] Hans P. Reiser, Franz J. Hauck, Jörg Domaschka, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Consistent replication of multithreaded distributed objects. In *Proceedings of the 25st IEEE Symposium on Reliable Distributed Systems (SRDS)*, 2006.

[27] Hans P. Reiser, Rüdiger Kapitza, Jörg Domaschka, and Franz J. Hauck. Fault-tolerant replication based on fragmented objects. In *Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS)*, June 2006.

[28] Jan Sacha, Jeff Napper, Corina Stratan, and Guillaume Pierre. Adam2: Reliable distribution estimation in decentralised environments. In *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS)*, June 2010.

[29] T. Schütt, A. Reinefeld, F. Schintke, and C. Hennig. Self-adaptation in large-scale systems: A study on structured overlays across multiple datacenters. In *Proceedings of the Workshop on Architectures and Languages for Self-Managing Distributed Systems (SELFMAN@SASO09)*, September 2009.

[30] T. Schütt, F. Schintke, and A. Reinefeld. Scalable Wikipedia with Erlang. In *Google Scalability Conference*, June 2008.

[31] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: Reliable transactional P2P key/value store. In *ACM SIGPLAN Erlang Workshop*, September 2008.

[32] Thorsten Schütt, Alexander Reinefeld, Florian Schintke, and Marie Hoffmann. Gossip-based topology inference for efficient overlay mapping on data centers. In *Proceedings of the 9th IEEE International Conference on Peer-to-Peer Computing (P2P)*, September 2009.

[33] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. Structured Overlay without Consistent Hashing: Empirical Results. In *Proceedings*

of the 6th Workshop on Global and Peer-to-Peer Computing (GP2PC), May 2006.

[34] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. A structured overlay for multi-dimensional range queries. In *Proceedings of the Euro-Par conference*, August 2007.

[35] Thorsten Schütt, Florian Schintke, and Alexander Reinefeld. Range queries on structured overlay networks. *Computer Communications*, 31(2), February 2008.

[36] Tallat M. Shafaat, Monika Moser, Ali Ghodsi, Thorsten Schütt, Seif Haridi, and Alexander Reinefeld. Key-based consistency and availability in structured overlay networks. In *Proceedings of the 3rd International ICST Conference on Scalable Information Systems (INFOSCALE)*, June 2008.

[37] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay Weaver: An Overlay Construction Toolkit. *Computer Communications*, 31(2):402–412, 2007.

[38] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[39] Michał Szymaniak, Guillaume Pierre, Mariana Simons-Nikolova, and Maarten van Steen. Enabling service adaptability with versatile anycast. *Concurrency and Computation: Practice and Experience*, 19(13):1837–1863, September 2007.

[40] Willem van Duijn. A versatile anycast framework for distributed servers. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, February 2008. `http://www.globule.org/publi/VAFDS_master2008.html`.

[41] G. Varghese and G. M. Bhagwan, P.and Voelker. Cone: Augmenting dhts to support distributed resource discovery. In *19th ACM Symposium on Operating Systems Principles, SOSP poster session*, 2003.

[42] Werner Vogels. Eventually consistent. *Communications of the ACM*, 52(1):40–44, 2009.

[43] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *J. Network Syst. Manage.*, 13(2), 2005.

[44] Spyros Voulgaris and Maarten van Steen. Epidemic-style management of semantic overlays for content-based searching. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 1143–1152. Springer, 2005.

[45] Zhou Wei, Jiang Dejun, Guillaume Pierre, Chi-Hung Chi, and Maarten van Steen. Service-oriented data denormalization for scalable web applications. In *Proceedings of the 17th International World Wide Web Conference*, Beijing, China, April 2008.

[46] Zhou Wei, Guillaume Pierre, and Chi-Hung Chi. CloudTPS: Scalable transactions for Web applications in the cloud. Technical Report IR-CS-053, Vrije Universiteit, Amsterdam, The Netherlands, February 2010. `http://www.globule.org/publi/CSTWAC_ircs53.html`.

[47] XtreemOS. On the feasibility of cloud computing functionality for grid applications. Deliverable D3.2.15, December 2009.

[48] XtreemOS. Distributed xtreemos infrastructure (dixi). Deliverable D3.2.17, March 2010.

[49] Erica Y. Yang, Brian Matthews, Amit Lakhani, Yvon Jegou, Christine Morin, Oscar David Sanchez, Carsten Franke, Philip Robinson, Adolf Hohl, Bernd Scheuermann, Daniel Vladusic, Haiyan Yu, An Qin, Rubao Lee, Erich Focht, and Massimo Coppola. Virtual organization management in XtreemOS: an overview. In *Proceedings of the CoreGRID Symposium*, August 2007.