

XtreemOS



*Enabling Linux
for the Grid*

Highly Scalable Services

XtreemOS Summer School, 10/09/09
Massimo Coppola, ISTI – CNR, Italy



Information Society
Technologies

*XtreemOS IP project
is funded by the European Commission under contract IST-FP6-033576*

Highly Scalable Services - XtreemOS

Summer School, Wadham College, Oxford -

10 Sept 09/09





- **What Highly scalable services are**
- **Directory service and Peer to peer**
- **Resource location: the XtreemOS approach**
- **The Resource Selection Service**
- **The Service/Resource Discovery Service**
- **Scalaris**
- **Down to the real thing : XML resource specification**





- **Larger and larger platforms present us several issues**
 - increasing communication latencies
 - overhead at centralization points
 - ubiquity of failures
 - need to manage a large number of resources
 - need to spread and search information at the platform level
- **These issues are critical from the operating system viewpoint**



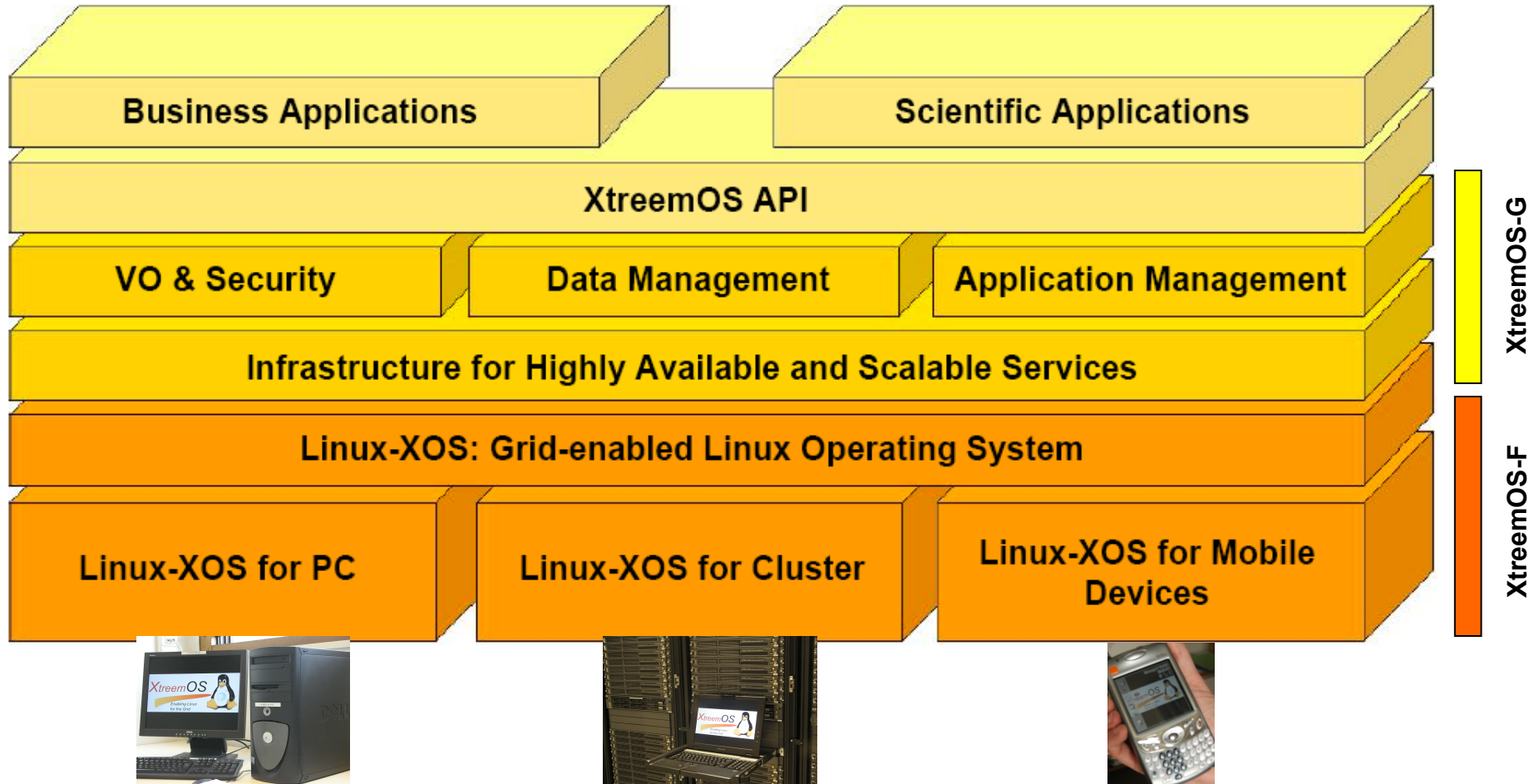


- A dedicate work-package of XtreemOS devoted to solving the platform level integration issue





XtreamOS Architecture





- **Services to store/query structured data**
 - SRDS: Service and Resource Discovery Service
 - RSS: Resource Selection Service
- **Services to communicate in a scalable fashion**
 - Publish/subscribe
- **Services to (partially) hide the effects of scale**
 - Distributed servers: hide resource distribution
 - Virtual nodes: hide node failures





An Example: Directory Services

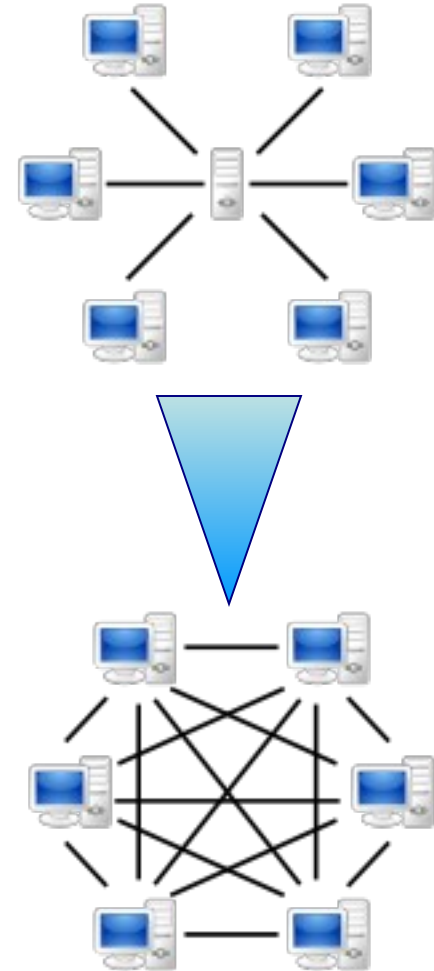
- **A fundamental service for**
 - locating other services and their servers
 - collect/publish/retrieve related information
- **Information**
 - a list of attribute-value pairs for each key
 - Static (persistent) and dynamically changing (transient)
 - Multiple heterogeneous data sources
- **Requirements of robustness, performance, security**





Directory Service Evolution

- **Centralized solution**
 - MDS version 1 (LDAP based)
- **Hierarchical system**
 - MDS2.4
- **P2P solution based on DHT**
 - [Ranjian et al.]
 - [Cheema et al.]





- **Overlay networks: built on top of other networks**
 - Peers manage routing and consistency of the overlay
 - can self-repair and evolve
 - No control on the underlying routing (e.g. TCP)
 - Routing can be application and content dependent
- **Need to maintain their own organization data**
 - Countless organization choices
 - Unorganized (gossiping networks)
 - Flat-structured (all peers are the same)
vs structured (some peers have different task)
 - Hierarchical (concept of super-peer, or multi level P2P)



How does resource location work?

- **Scheduling is essential on large platforms**
 - which job should run on which resource and when
 - In XtreemOS this is the job of the AEM
 - finding resources is critical
- **optimal scheduling is NP-complete**
 - In a scalable environment we must use heuristics
- **XtreemOS heuristic:**
 1. Select a subset of “suitable nodes” (e.g., $2*N$)
 2. Choose the best N nodes to run the job on

Still selection is not easy





What is a “suitable node?”

- **If less than N of the selected nodes are unable to run the job, then scheduling fails**
 - (or it must request the SRDS for more nodes)
- **We must select “good” nodes in the first place**
 - Is the node able to run the job?
 - Static parameters (CPU family, OS version, etc.)
 - Dynamic parameters (free disk space, etc.)
 - Is the node authorized to run the job?
 - Does the node's policy authorize this job?





What is a suitable P2P approach?

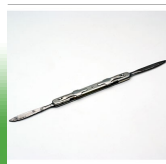
- **Resource selection is based on many parameters**
 - static resource attributes : memory size, disk space, kind of CPU
 - dynamically varying attributes: memory free, current load, available software licenses ...
 - the set of relevant values does change
 - with the resource
 - with the specific request
- **No single P2P approach up to now can cope with all the issues at the same time**
- **Different Services handle the different tasks**





A combination of Services

- Different services, exploiting different P2P techniques, are combined to provide the best tradeoff
- A two phase selection scheme is adopted
 - (first the machete, then the bistoury)





Resource Selection Service (RSS)





Why the Resource Selection Service

- **Current resource location solutions rely on delegation**
 - Each node registers its properties with some registry node(s) that implement the selection
 - Centralized, hierarchical, DHT-based
- **Delegation is a bad idea in the large case**
 - Unnecessary load due to periodic revalidations of registered values
 - Inconsistency between the actual and registered values
 - Imbalanced workload





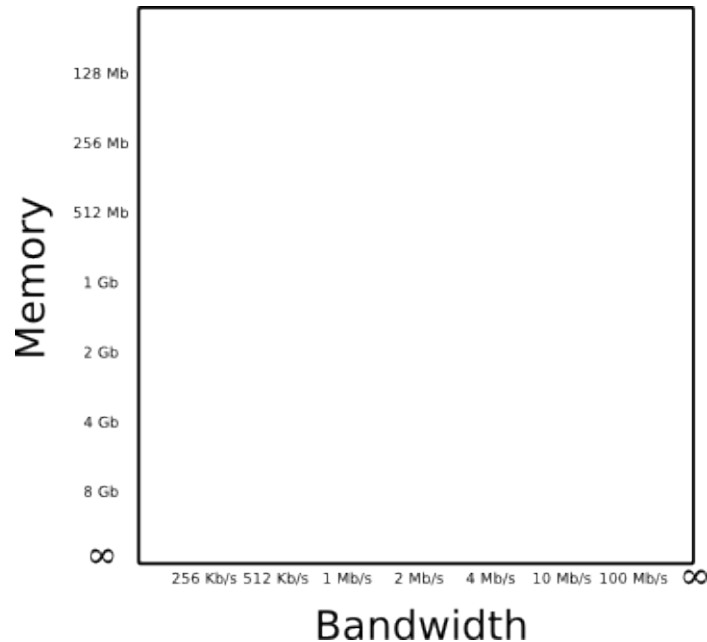
- **RSS develops a hierachically structured P2P overlay**
 - very efficient range search for static-valued attributes
 - the attribute number, i.e. the number of network dimensions, has to be bounded by a constant
 - the attributes are fixed at network initialization and common to all nodes
 - (there is work in progress on these topics)
- **We use the RSS to efficiently select a tractable number of candidates, then we exploit DHT-based techniques to refine the selection**





RSS space structure

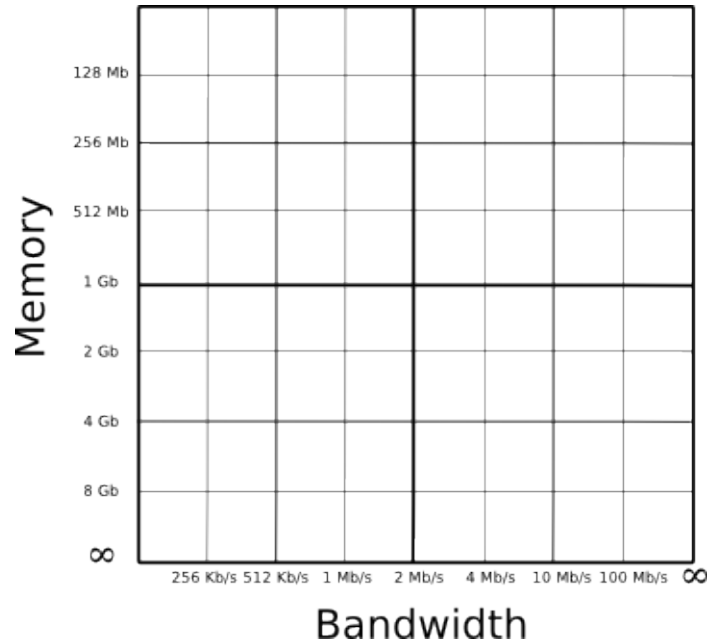
- Each attribute represents a dimension of a hyper-cube





RSS space structure

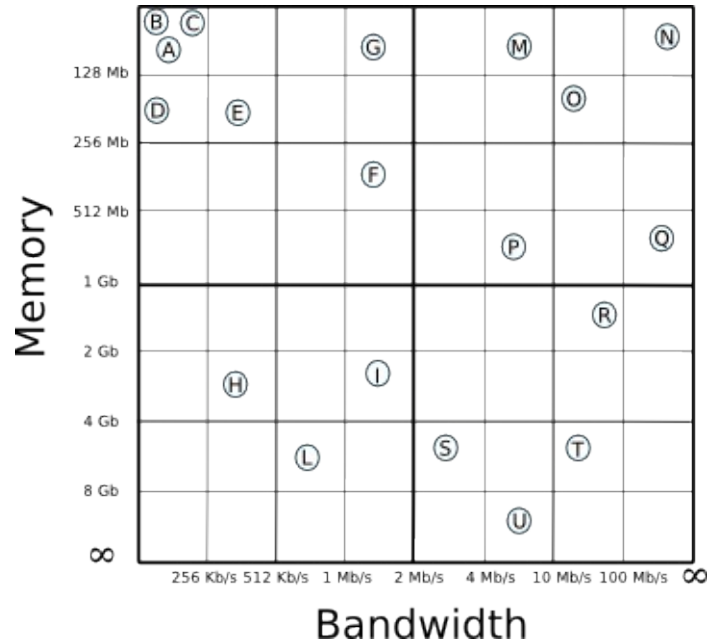
- **Each dimension is split in two recursively**
 - Using any boundary that “makes sense”





RSS space structure

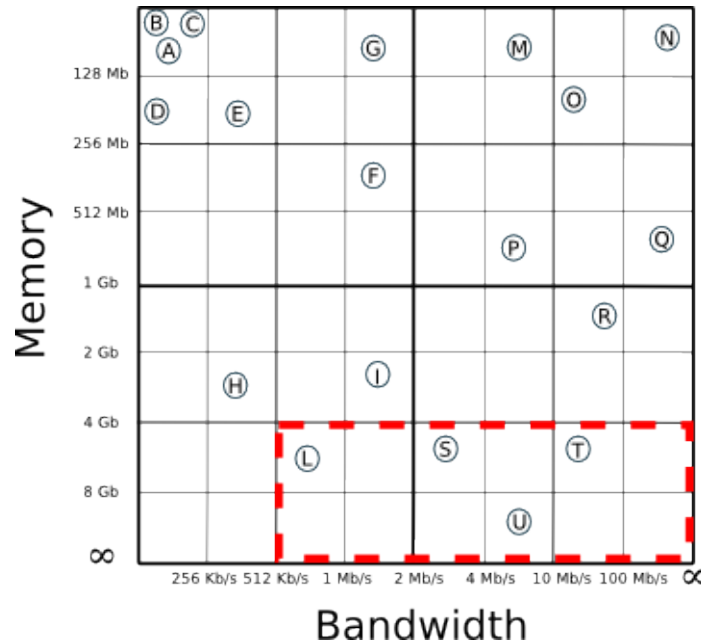
- Each node has one coordinate in the hyper-cube





RSS space structure

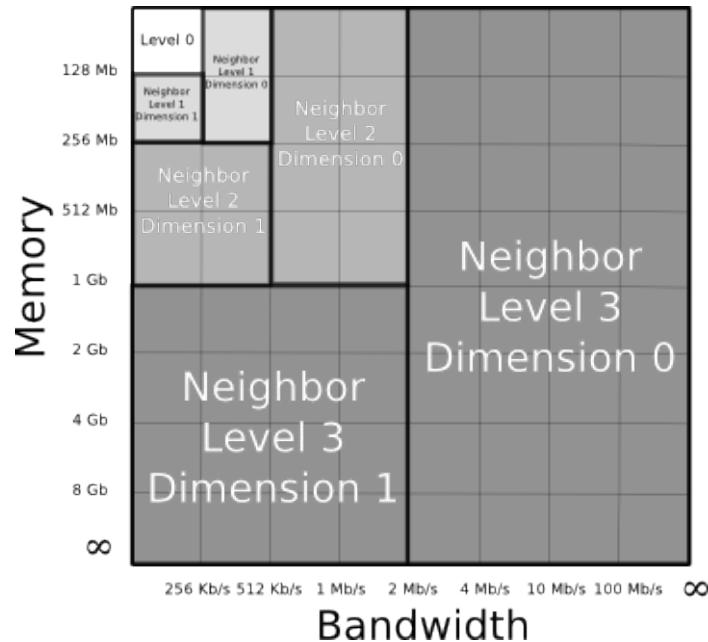
- A query is defined as a region of the hyper-cube
 - Some dimensions can be left unspecified





RSS overlay structure

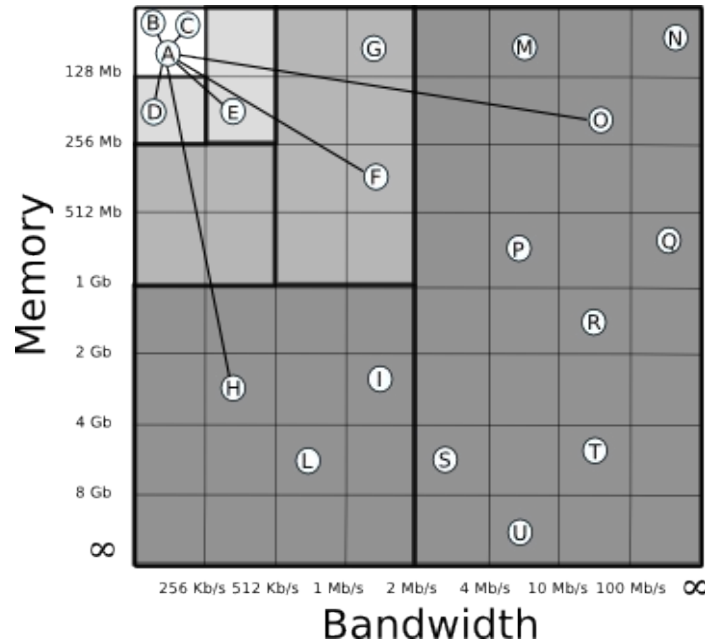
- Each node defines cells around itself





RSS overlay structure

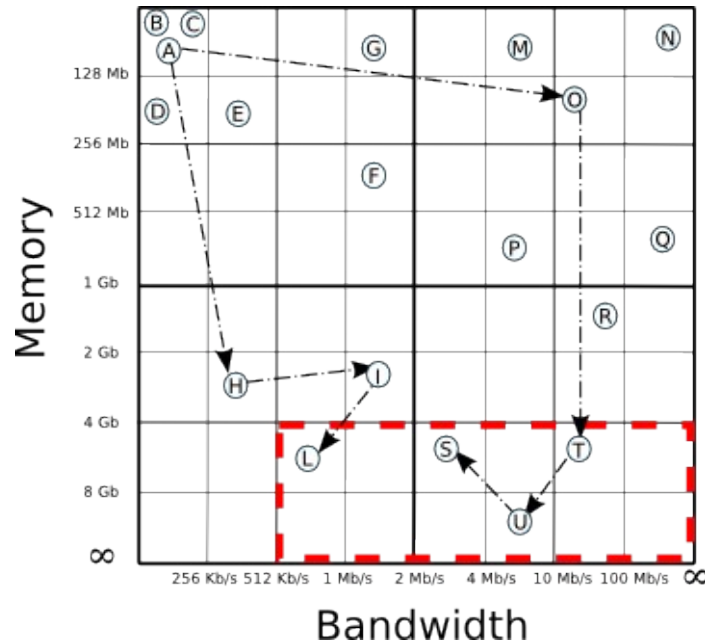
- **And maintains a link to one node in each cell**
 - The number of links is **linear** to the dimensionality





RSS overlay routing

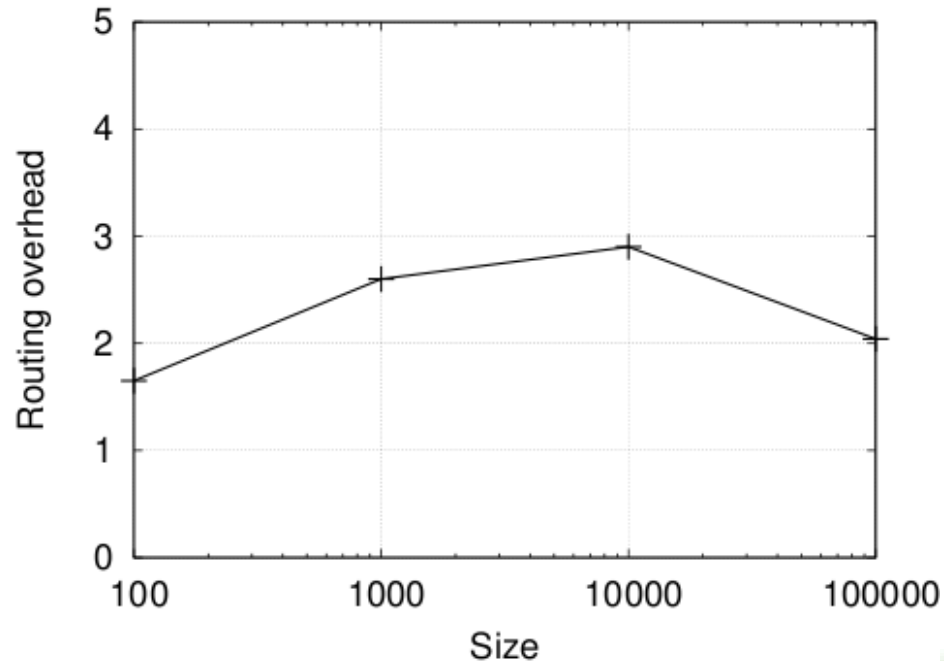
- **Queries are easily routed across these links**
 - Note that **nodes select themselves** along the route





Scalability vs. Number of nodes

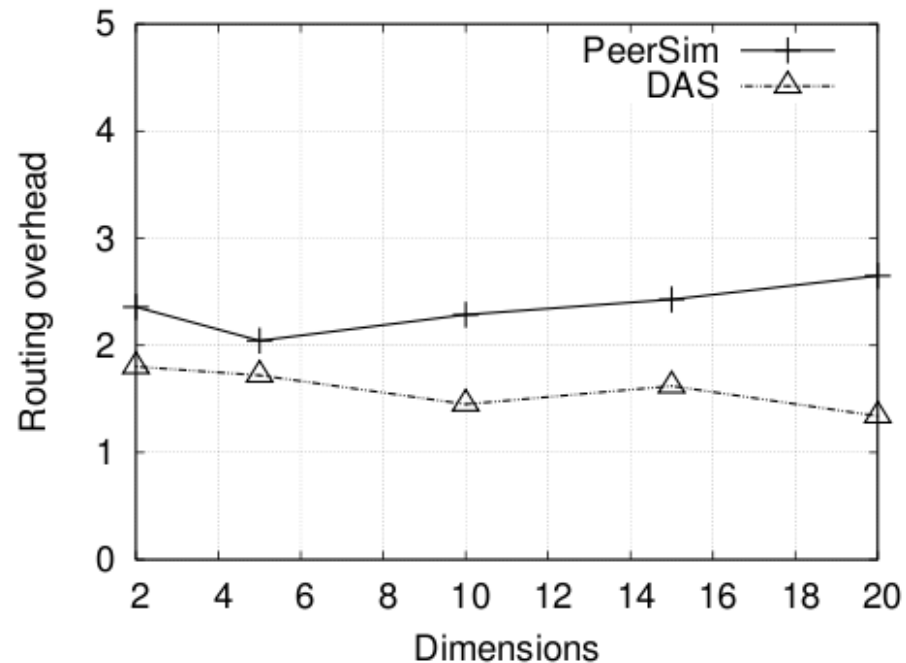
- **Routing overhead vs. system size**
 - Number of nodes that do **not** match a query on its route





Scalability vs. dimensionality

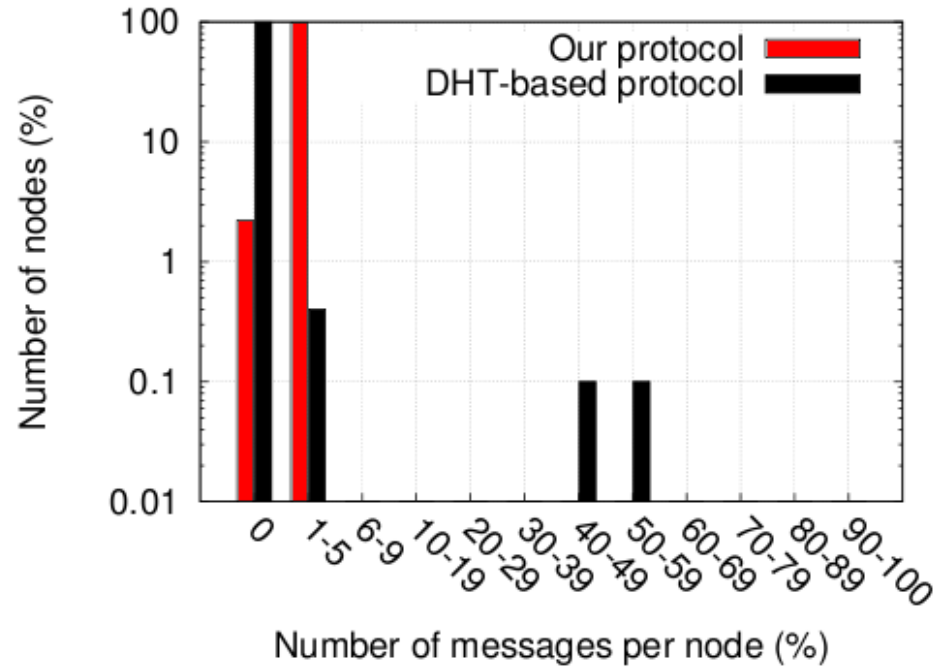
- **Routing overhead vs. number of dimensions**
 - We can support as many attributes as we want





Load balancing

- RSS balances the query load across all nodes





- **And now for something completely different...**

Service/Resource Discovery Service





- **Exploit a distributed platform to provide a single service : key-value store**
- **Decentralization**
 - the nodes collectively form the system without any central coordination.
- **Scalability**
 - efficiency up to thousands or millions of nodes.
- **Fault tolerance**
 - provide enough reliability even as nodes continuously join, leave, and fail.





- **Any one node coordinates with only a few other nodes in the system**
 - routing table are usually $\Theta(\log n)$ of the n participants
- **Provide an abstract keyspace**
 - e.g. the set of 128-bit strings.
- **Keyspace partitioning scheme**
 - split ownership of the keyspace among the participating nodes
 - each node manages a portion (contiguous or not) of the keyspace





- **Algorithms and data structure are needed for**
 - routing
 - accessing information
 - building / repairing the overlay (routing table maint.)
 - provide fault tolerance (e.g. multiple owners for any given keyspace portion ü replica management)
 - validating information (security, expiration, atomicity ...)





- **Many different DHTs have been developed with varying requirement / choices**
 - CAN Chord, Pastry, Tapestry
 - Implementations: Bamboo, OverlayWeaver, Scalaris
- **DHT limitations**
 - updating data costs
 - they provide key-based search:
 - range queries, multi-attribute queries are needed
 - adding support for value-based search reduces performance and scalability
 - other features have a cost: FT, atomicity



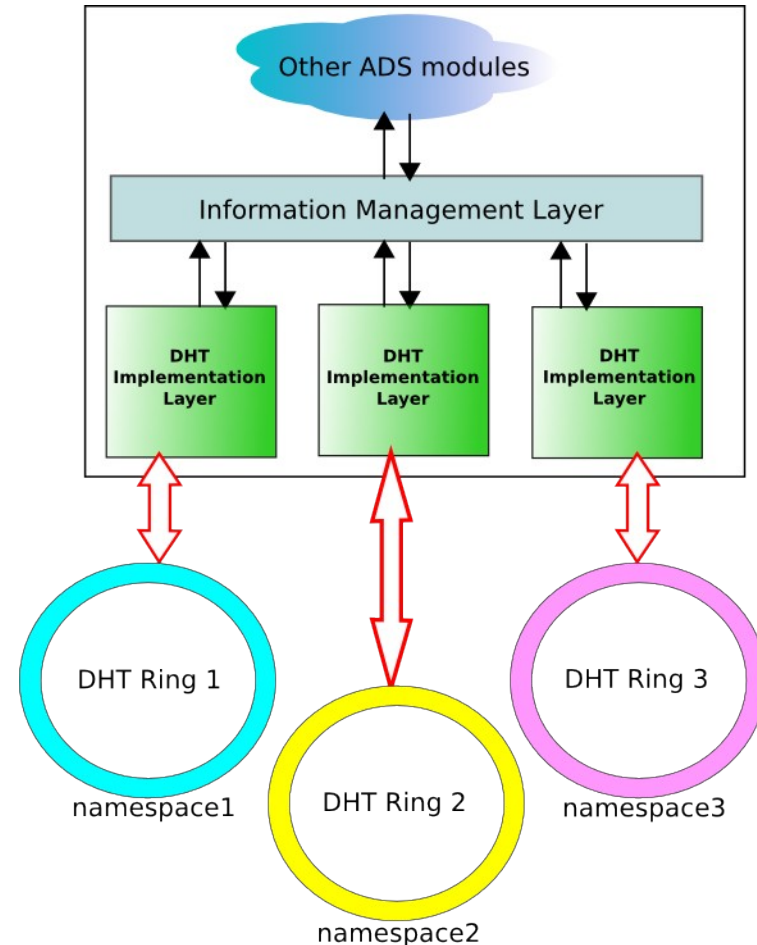
- **Service and Resource Directory Service**
 - Common service for all XtreemOS modules
- **Scalability to Grids**
 - 1K to 100K nodes = *can't be centralized!*
- **Reliability and Self-healing**
 - Stand a degree of resource failure with no/minimal loss, readjust after local faults
 - Should recover after a crash/reboot
 - Authorization/security support
 - Ease of management
- **P2P approach!!**





SRDS combines several DHTs

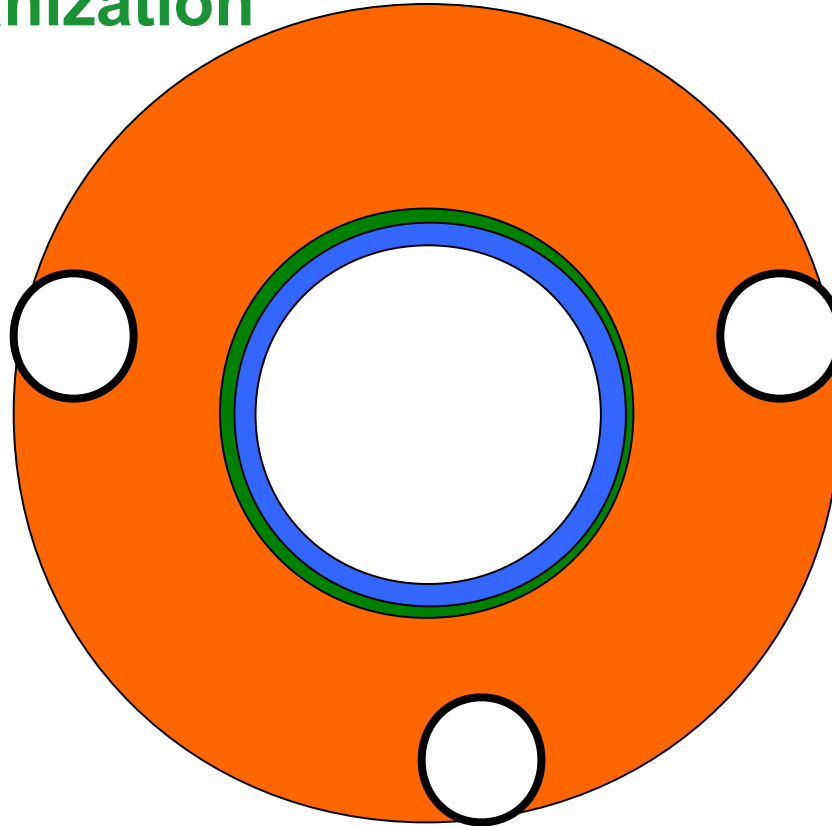
- **Provide information to several clients**
 - resource location and monitoring
 - application execution
- **One service integrates multiple P2P networks**
 - One instance of SRDS on each XtreemOS resource
- **Distributed Hash Table P2P networks**
 - OverlayWeaver
 - Scalaris
- **Hierarchically Structured P2P networks**
 - Resource Selection Service
- **Adaptive configuration**
- **Easy extendibility**





VO-level organization

XOS node



XOS node

XOS node

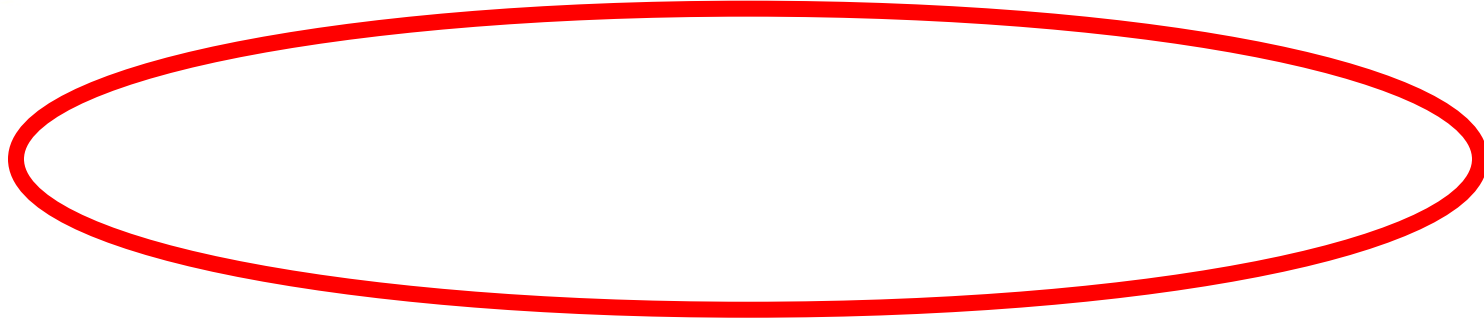




- **Infrastructure of XtreemOS**
 - exploits multiple P2P overlays
 - each resource and core node joins the overlays
- **How resources are discovered**
 - three-pass filtering
 - static checks (*few attributes*) performed by RSS overlay
 - *is node available?* - XACML filters exploited on leaf nodes
 - extensive and dynamic info is indexed within a DHT
- **DHTs also index**
 - heterogeneous/partially available data (e.g. JDS, ADS)

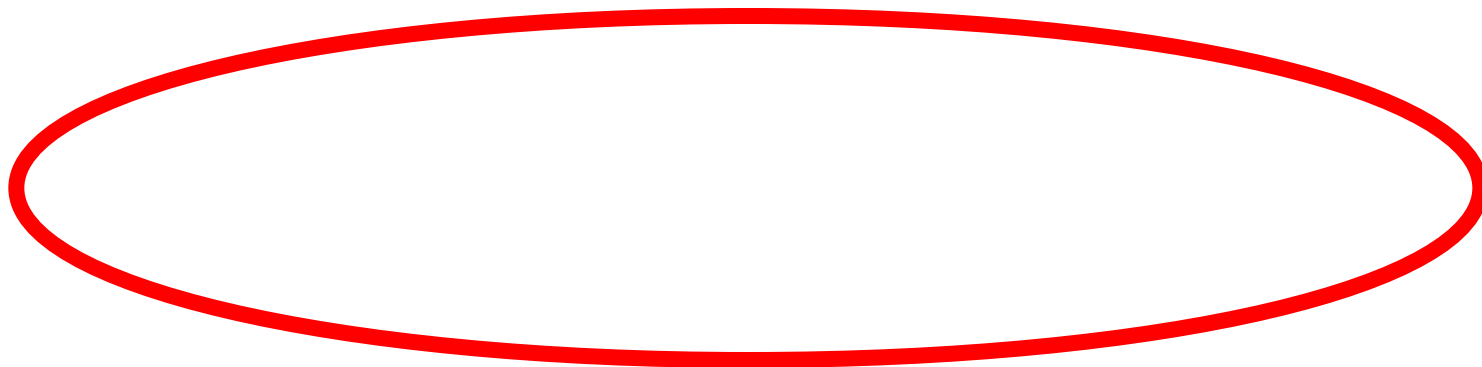


SRDS architecture



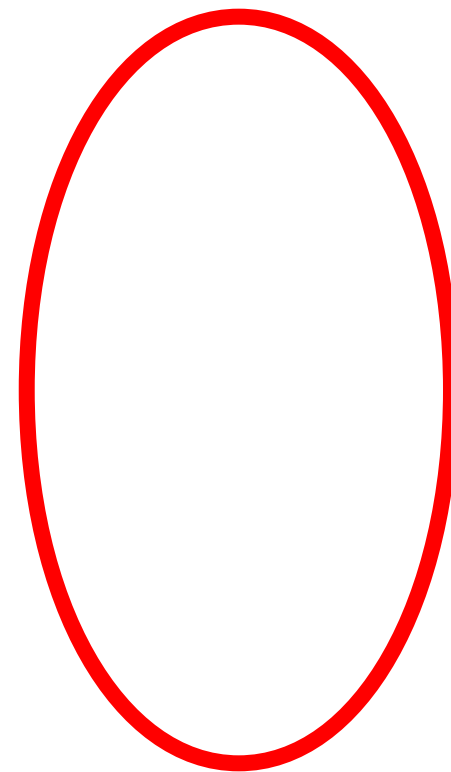


SRDS architecture



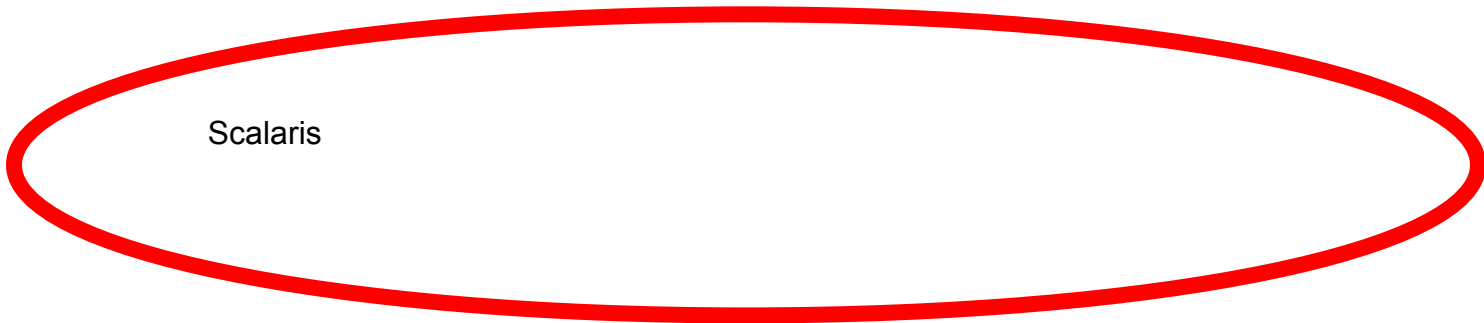


SRDS architecture





SRDS architecture





What is done under the hood

- **The SRDS/RSS combination handles resource location in XtreemOS**
- **SRDS also handles other information management tasks**
 - the user job information is on a DHT
 - can survive AEM shutdown/disconnection and allows user to reconnect to his jobs





- **Other functionalities are being developed**
 - Application- and User-oriented information services to be provided with configurable QoS
 - ask for redundancy, transactional behaviour, different query capabilities
 - features are linked to SRDS-provide *namespaces*
 - Application and users do not interfere w. each other
 - DHTs are selected and DHT keyspaces are partitioned transparently
 - Translation algorithms are employed by SRDS to provide complex functionalities



- **Scalaris is a DHT providing transactional capabilities**
 - augments the CHORD approach with replication (availability) and majority-based distributed transactions (data consistency)
 - ACID properties on a scalable structured overlay.
- **Used by SRDS for critical operations**
 - (e.g. allocation of Namespaces)
- **Used as foundation of Publish/Subscribe service**
 - Applications can subscribe to events and receive notifications





Resource Location : how do we use it?

- Resource location is handled SRDS and RSS by looking at the JSDL file submitted to the AEM
- Attributes are described according to the JSDL standard
- Resources are two-phase filtered by RSS and SRDS
- A JSDL extension allows to specify the tolerance a dynamic attribute must have with respect to its static value





down to XML

<Resources>

<CPUArchitecture>

<CPUArchitectureName>i386</CPUArchitectureName>

</CPUArchitecture>

<IndividualCPUCount>

<Range>

<LowerBound>0</LowerBound>

<UpperBound>16</UpperBound>

</Range>

</IndividualCPUCount>

<OperatingSystem>

<OperatingSystemName>Linux</OperatingSystemName>

</OperatingSystem>

<IndividualPhysicalMemory dynamic-epsilon="0.9">

<Range>

<LowerBound>0</LowerBound>

<UpperBound>20097152000</UpperBound>

</Range>

</IndividualPhysicalMemory>

<DiskSpace dynamic-epsilon="0.9">

<Range>

<LowerBound>0</LowerBound>

<UpperBound>4194304000000000</UpperBound>

</Range>

</DiskSpace>

<TotalResourceCount>

<Exact>100</Exact>

</TotalResourceCount>

</Resources>





- How to connect to the testbed

`ssh xuser@brunello.isti.cnr.it` (pws is xuser)

`ssh xos@131.254.201.60` (pwd is xtreamos)

`su -` (pwd is xtreamos)

- this is a core node; from here you can launch jobs





Listing resources through AEM

```
[root@xos-core ~]# xconsole_dixi
```

```
XtreemOS Console
```

```
$ xrs -a
```

```
Listing all resources:
```

```
Address = [://131.254.201.60:60000 (131.254.201.60) ]
```

```
Address = [://131.254.201.31:60000 (131.254.201.31) ]
```

```
Address = [://131.254.201.62:60000 (131.254.201.62) ]
```

```
Address = [://131.254.201.61:60000 (131.254.201.61) ]
```

```
Address = [://146.48.83.198:60000 (146.48.83.198) ]
```

```
$
```





Listing resources through AEM

```
[root@xos-core ~]# xconsole_dixi  
XtreemOS Console
```

```
$ xrs -a
```

**You will usually need a certificate for that,
when you are not root**

```
[root@xos-core ~]# xconsole_dixi -c user.crt  
XtreemOS Console
```

```
$ xrs -a
```





```
$ xrs -jsdl blank.jsdl
```

```
Listing resources matching JSDL query:
```

```
Address = [://131.254.201.31:60000]
```

```
Address = [://131.254.201.60:60000]
```

```
Address = [://131.254.201.61:60000]
```

```
Address = [://131.254.201.62:60000]
```

```
Address = [://146.48.83.198:60000]
```

```
$
```

Pretty print formatted version of the test jsdl files are in the *jsdl* directory, with their name ending in *_f.jsdl*





Listing resources through AEM

<code>AllResources.jsdl</code>	huge constraints
<code>Arch_PPC.jsdl</code>	PPC architecture (no resources)
<code>blank.jsdl</code>	
<code>blank_gnuplot.jsdl</code>	blank request running gnuplot
<code>cal.jsdl</code>	run cal
<code>dinamicity_0_5.jsdl</code>	use dinamicity
<code>From4CoresTo10.jsdl</code>	
<code>Until_2Cores.jsdl</code>	
<code>Until_2Cores_2GbRam.jsdl</code>	
<code>Until_4Cores.jsdl</code>	
<code>kmines.jsdl</code>	interactive job

