Xweb: A Framework for Application Network Deployment in a Programmable Internet Service Infrastructure

O. Ardaiz, L. Navarro
Computer Architecture Department
Polytechnic University of Catalunya
Barcelona 08034 Spain
{oardaiz, leandro}@ac.upc.es

Abstract

An application network consists of a number of application servers distributed throughout the Internet, connected and coordinated to provide services with low latency. Adding, removing and migrating servers, application networks adapt to demand variations. To create new servers anywhere in the Internet a programmable Internet service infrastructure is needed. In addition application network servers must be deployed co-ordinately. We propose a framework for application network deployment that implements such functionality.

1. Motivation

Internet services quality can be greatly improved if an application network provides them. Application networks are a set of coordinated application servers distributed throughout the Internet, thereby clients can access a nearby server that provides a low latency service. Application network provide client requests with good quality of service: a request from location X will be provided by nearer server A instead of farther server B; also server A load will not increase beyond a threshold, causing successive requests to be delayed, and some requests are redirected to a distant but less loaded server. Those servers are connected, setting up an application layer topology, and coordinated for request redirection, load balancing and replica consistency. Examples of application networks are content distribution networks, proxy-caching hierarchies, chat server networks or peer-to-peer networks as Gnutella [19].

Existing application networks are manually created and modified: new servers are manually installed and connections among servers are manually configured. However Internet services have very dynamic demands with temporal and spatial variations [18]; static application networks can not provide these demands with good service quality (unless they are overprovisioned to the worst case: the hot spot service demand f.e. Akamai network[1]). An application network that adapts to those variations will serve such dynamic demands with a good service quality. Adapting to demand variations involves adding and removing servers, migrating servers to locations where new demand arises, and deleting and creating connections among servers. To implement this functionality it is required a programmable Internet services infrastructure. A programmable infrastructure is composed of resources distributed throughout the Internet where service providers can remotely activate and stop application servers.

Moreover, to facilitate application networks adapt to demand variations maintaining good service quality while consuming few resources, application networks must be deployed, "to spread out or arrange for effective action" [11], instead of being uncoordinatedly activated. As a result of application network deployment, application servers are placed at appropriate locations, connected and coordinated to provide a good service quality.

1.1. Related Work

Provisioning variable service demands has been pursued by Radar [16] dynamic hosting service where static content is migrated towards hosts close to the demand. P2P and CDN networks that cache static content also move content close to clients on demand. Application service providers such as Ejasent [10] are proprietary frameworks for activating web service at increasing number of nodes as demand rises; but they only implement capabilities needed by their customers. As represented in figure 1, Xweb extends current static application networks towards an application network service that is dynamically deployed in a programmable infrastructure to adapt to demand variations.

Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (EUROMICRO-PDP'04)
1066-6192/04 $20.00 © 2004 **IEEE**

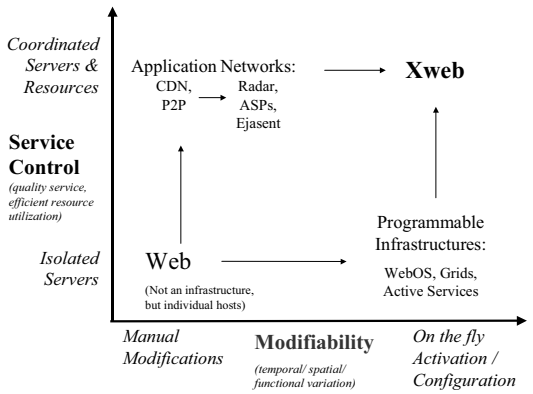


Figure 1. Xweb vs. other frameworks

On the other side open programmable Internet service infrastructures have been proposed to facilitate introduction of new services and promote third party service development. These infrastructures have been described as an Internet wide distributed operating system WebOS [23]; active services that provide per request on demand service activation, such as AS.1 framework [3]. Some gather processing resource for parallel computational intensive applications: Grids [12], some gather storage resources for static content dissemination and/or archival: Lbone [6], and some allocate generic resources for conferencing systems: Darwin [7]. However none of them has been designed with the goal of creating and modifying dynamically application networks. Xweb extends programmable infrastructures with deployment mechanisms (see figure 1), so that application network are dynamically activated, connected and coordinated. Vandal [13], Xbone [22] and Xbind [16] deploy virtual networks and services at layer three maintaining control of topology, routing and resource utilization, with various levels of programmability.

2. Deployment Framework

Application networks must not be created activating application servers uncoordinatedly. Application networks must be deployed "to spread out or arrange for effective action" [11]. We have used this metaphor to model a deployment framework for computer based application network. In this deployment model there are application users that demand a service; a limited number of resources with which to construct that service. A deployment plan determines which resources have to be allocated, how to activate and coordinate server instances. This deployment plan is executed by allocating resources, distributing service code and installing, executing, connecting and coordinating servers that provide the application.

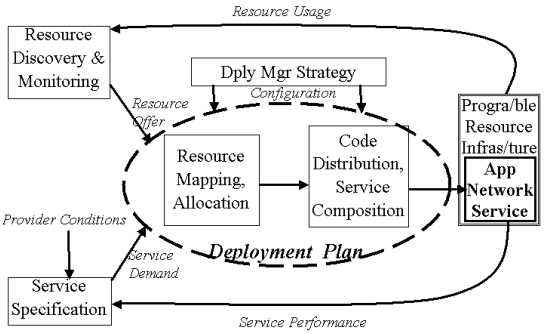


Figure 2. Framework building blocks

A deployment plan is designed to accomplish two different objectives: provide requested service level, and minimise resource utilization, while fulfilling any other imposed constraints. A deployment plan specifies application network deployment commands. The service plan contains all commands and information to activate an application network: resource mapping commands (where to allocate resources), resource allocation commands (which resources), and service composition commands (how to distribute code, how to bind server to resources, and how to connect and coordinate servers).

3. Xweb Building Blocks

To realize such model we propose to implement a framework with these building blocks (see figure 2): a programmable Internet service infrastructure, resource discovery and monitoring, service specifications, resource mapping, resource allocation, service composition. Resource discovery and service specifications provide input to create a deployment plan with a resource-mapping algorithm. Allocating resources, distributing code and composing an application network in the programmable infrastructure, carry out the deployment plan. Changes in demand or resource availability are feedback to resource discovery and service specification modules triggering a re-deployment operation.

3.1. Internet Service Programmable Infrastructure

An Internet service programmable infrastructure comprises a number of nodes that provide resources and an execution environment. The execution environment at each node must permit simultaneous execution of several servers from different application networks. Execution environments with these properties are multitasking operating systems, such as UNIX derived OS and virtual machines such as Java Virtual Machine.

3.2. Deployment Plan Input: Resource Availability and Service Specifications

Resource discovery and monitoring provides deployment managers with an up-to-date resource offer specifying which nodes support activation of new services, its resource type and quantity. Resources can be discovered and monitored proactively (managers query nodes) or indirectly (managers lookup resource availability in a directory f.e. Globus MDS [9]). Xweb employs a proactive mechanism called multicast expanded ring search used at the Xbone [21]. It sends queries at a multicast channel with increasing time-to-live packets; resources respond queries directly to managers in a unicast channel. This mechanism, sketched in Fig. 3, discovers large numbers of nodes in a short time with little network overhead.

Service specifications summarize service demand and include service-wide properties and resource requirements, which indicate how overall service is provided to users. Service specifications are represented by three different requirements:

Resource requirements, which can be normalized for one client demand, i.e. execution environment, storage size, per request network capacity, request duration, and service level.

Service demand is a table containing demand population and access frequency of demand regions, where regions can be IP subnets, AS areas, or geographical areas.

Service-wide constraints for application networks are: maximum network diameter (number of servers a request will be forwarded to before being provided by an origin server: high diameter application networks provide the least costly services because few requests reach to distant origin servers, low diameter networks provide lower response times since each request forwarding adds a delay), level of redundancy (number of different servers for each region: redundancy comes at a higher cost), a maximum number of server can be specified to avoid incurring in a large cost. Fig. 4 shows an XML specification.

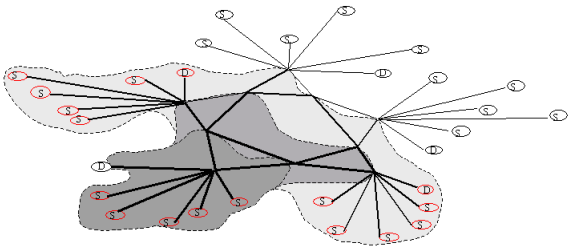


Figure 3. Multicast expanded ring search discovering increasingly further apart resources

```
<app_net_specifications id=www.mortero.com
Origin_Server="127.10.12.12:80/mortero.tar">
<Resource Requirements>
        <OSType>SunOS</OSType>
        <Storage Unit="Mbytes">200</Storage>
        <Service_Traffic Unit="Kbits">20<Service_Traffic>
        <Service_Time Unit="sec">10</Service_Time>
        <Client_Dist Unit="InetHops">2</Client_Dist>
        <Server_Load Unit="%">>20</ Server_Load>
</Resource Requirements>
<Service Demand>
          <RegionalDemand>
          <RegionId Unit="ASnum">1201</RegionId>
          <ClientsNum>1000</Traffic>
        <DemandRate Unit="req/sec">0.1</DemandRate>
          </RegionDemand>
</Service Demand>
<Service-wide Constraints>
<Max_Network_Diameter>1</Max_Network_Diameter>
        <Redundancy_Level>2</Redundancy_Level>
        <Max_num_servers>5</Max_num_servers>
</Service-wide Constraints>
</app_net_specifications>
```

Fig. 4. Application Network Specifications

3.3. Resource Mapping

Resource mapping functionality chooses most appropriate nodes where to deploy application networks. Resource availability determines sets of candidate nodes.

Service specifications reduce candidate sets and some combinations are discarded. Among remaining mappings, it is selected the one which optimises some value: either maximising overall service quality (service quality is measured by client-to-server distance and/or server load; distance is proportional to response time, Internet hops or path throughput; the higher client-to-server distance the worse service level can be expected) or/and minimising costs (cost is measured by resource consumption per server cost plus network traffic).

In the application networks literature there are several algorithms that optimise application networks, subject to different constraints. Some considers storage as the unique constrained resource and try to minimize overall client-to-server distance i.e. Dahlin cooperative replacement algorithm [13] or Stor-Serv Intelligent Storage Network [8]. Others model application networks as a load-balancing and minimize replication cost and distances, Radar [16].

We have designed a simple algorithm that we call "connected placement" application network mapping algorithm. If is based on the fact that application networks create most of its traffic from edge servers to client (caching can reduce outbound traffic by 40%, therefore it is a reasonable assumption), therefore it is more important to minimize client-to-server traffic. First it calculates positions of edge servers (servers nearer to clients) by a placement algorithm which minimizes overall client-to-server distances selecting only among those nodes with appropriate resources available. In a second phase those edge servers locations can be consider as client locations of second level servers, therefore its locations can be

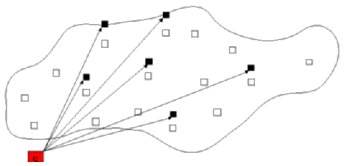
found executing the placement algorithm again. This is repeated until the maximum network diameter is reached, when servers are fully connected to each other setting up a mesh. We will evaluate it in a future work.

3.4. Resource Allocation

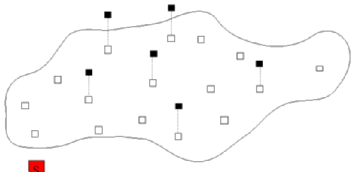
A local resource allocator at each resource node has to keep a list of allocated resources so that it does not allocate more resources than available. Besides it has to enforces strict partitioning of resources so that different application at the same node do not interfere with each other. Deployment managers can control resource allocations at resource agents with per node individual command messages. However it is a mechanism which suffers from scalability and resource contention problems. There exist more distributed alternatives such as "multicast injection", based on multicast, which provides distributed resource allocation with comparable efficiency, little resource contention and much less traffic generated at deployment managers locations, we have evaluated it in another paper [2].

3.5. Application Network Composition

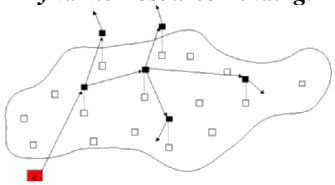
Application networks are composed of software entities executed at different nodes. They communicate and coordinate for effective service and efficient resources utilization. Mechanisms required to compose an application network are 1) distribute service code to nodes, 2) bind service code to local resources on those nodes, 3) coordinate all service instances.



Code distribution.



Dynamic Resource Binding.



Dynamic Servers Connection and Coordination

Figure 5. Composition Mechanisms

Application network deployment requires dynamic resource binding mechanisms, so that services bind and unbind from resources on the fly.

Composing an application network involves connecting and coordinating service instances. Communication is defined by connectivity among its members and coordination by rules governing coordination among them. Application layer network connections are TCP virtual circuits or UDP port pairs among servers. Rules governing coordination among service instance are very simple rules: "if cannot be processed here, forward to node x", "if coming from nodes x or y, forward to node z". Therefore it involves creating communication channels, and providing rules on how to cooperate. Again these mechanisms should be dynamic.

4. Xweb Implementation: Web Proxy Caching and Chat Server Application Network Deployment

Web proxy-caching application network provide a caching service for web clients. Web pages are cached at intermediate servers. Pages that cannot be provided by a server are forward to a parent proxy-cache [24]. Chat server application networks are a number of chat servers that are connected and coordinated to share chat channels [14].

4.1. Deployment and Security Architecture

The deployment model clearly has two differentiated roles: service provider and resource providers. Resource providers are programmable infrastructure nodes. They assume a passive role in that they provide resources and an execution environment. Service providers assume an active role in that they command deployment of the application network, controlling resource allocation and service composition. Therefore the system architecture is composed of resource agents at resource providers' nodes and deployment managers at service providers' nodes. Resource agent implements resource allocation, code distribution, resource binding and service composition. Deployment managers implement resource discovery, a service specification front-end, resource mapping, and deployment plan creation. Deployment managers command resource agents to allocate resources, obtain service code, bind service programs to allocated resources, and configure application network coordination.

Resource agents implement an access control mechanism controlled by resource providers. These give access to certain deployment manager to perform resource allocation and service composition operation on their

resources. Deployment manager communicate with resource agents through a secure SSL connection. In this way only authenticated deployment manager can perform operations over resources. Deployment managers only deploy application networks over authenticated nodes. A certification authority certifies both parties.

4.2. Xweb Programmable Infrastructure and Resources

The Xweb programmable infrastructure is made from Linux nodes and Java-Tomcat execution environments (therefore it is possible to deploy Linux service and Servlet based application networks). Each node makes public its resource availability: execution environment, maximum network capacity, storage capacity and adjacent network regions where service can be provided. In our prototype these values are provided by manual configuration, however they can be obtained dynamically. Our prototype consists of only three resource nodes, plus a deployment manager node. In such infrastructure it is possible to deploy several application networks made up of three servers.

4.3. Deployment Manager

Multicast Expanded-Ring Search Resource Discovery and Monitoring Xweb multicast expanded ring search is the Xbone [22] implementation. Resource agents listen to IP multicast address 224.192.0.1 waiting for resource discovery requests. Responses are only sent to trust managers in a unicast secure SSL connections containing resource properties. Deployment managers send resource discovery messages after receiving a deployment request from a user through its interface, and periodically send resource availability reports.

Specifications Input Deployment managers have a web interface so that several service providers can access from remote locations. Through this interface, service creators input service names, code location, start time, service input service specification plus application network duration and can adjust some deployment parameters.

Resource requirements supported are: execution environment (type of service), storage size, maximum distance to server, and network traffic generated; demand is specified as a list of AS regions; service wide constraints are maximum number of servers.

Resource Mapping Algorithm The resource mapping algorithm implementation selects for each demand region a resource node with enough resources among those that are nearer to that region. This algorithm does not optimise number of resource locations, but it provides a good service since each demand region has one server that is nearby and has enough resources. Edge servers are connected in a tree whose root is the node not been selected for any region.

Deployment Process Control Deployment managers control if deployment process is being carried out as planned. They have to maintain a record of every operation started for future resource release or commit operations.

4.4. Resource Agents

Local Resource Allocator Resource agents keep a list of allocated and free local resources. It has to enforce strict partitioning of resources so that different application being provided from one node does not interfere with each other.

In our implementation there are two allocation tables for storage of resources and network bandwidth capacity. On receiving an allocation command, resource agents check if they have available resources to deploy a new service in its node. If so they proceed to decrease free resource table and increase allocated resources table in an atomic operation.

On Demand Code Distribution There exist many mechanisms for code distribution with different levels of security and performance. Existing programmable infrastructures provide mechanism for active code and executables distribution that are based on file transfer mechanisms such as FTP. As well any content distribution mechanism can be adapted such as multicast transport or HTTP-based.

Xweb implements a code distribution mechanism based on HTTP. Deployment managers send resource agents a code server location and code file name as a URL. Resource agents download code from such location through an HTTP GET transaction.

Dynamic Server Installation & Activation Applications are programmed so as to be bound to local resources forever at installation time and at service start time. They demand memory, CPU or network capacity to the local operating system. Without modifying service code it is required to halt and restart service programs to change its resource bindings. If application programs were programmed to bind or unbind to resources at an external signal, dynamic resource binding would be much easier.

Since we do not modify service programs, Xweb implements dynamic resource binding by halting and restarting services. Applications are installed using Linux RPM package tools. Server programs are stopped, read its new bindings from a configuration file, and restarted. Binding to a new resource allocation requires stopping and starting the service.

Dynamic Server Coordination & Communication Channel Creation It is required on-the-fly switching of communication channels, and on-the-fly reconfiguration of cooperation rules responding to deployment manager commands. However applications create TCP virtual circuits or UDP port pairs at start up.

Since we have not modified service programs, to change server connections applications need to be halted, read its new communication channel configuration, and restarted creating a new sockets connection. Again most applications can only be configured at start time on which will be their parents or peer servers. Therefore resource agents have to stop, reconfigure and restart servers to modify coordination rules. Fortunately a system signal sent to squid proxy-caching server makes it reread its configuration file.

4.5. Prototype Setup

The prototype programmable infrastructure consisted of 3 nodes. Every node had a Pentium IV processor with 10 Mbps LAN Internet connectivity and 1 Gbyte storage capacity. They were running Linux RedHat 7.1 therefore it could be deployed Linux applications. A resource agent was installed at each node. A fourth machine had a deployment manager installed. In the experiments it were deployed web proxy-caching hierarchy (Cache ALN), which used squid proxy-caching server [24], and chat server networks (Chat ALN), which used Internet Relay Chat server [14]. They required 1 Mbps and 100 Kbps network capacity, and 200 Mbytes and 1 Mbytes storage capacity respectively. Clients demand came in both cases

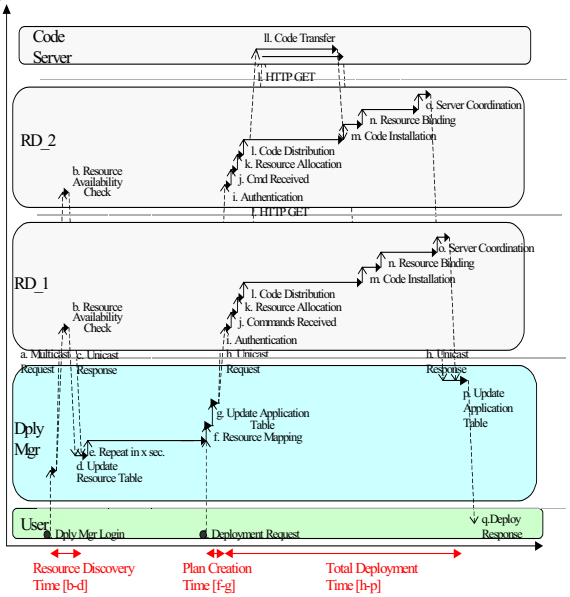


Fig. 6. Time plot of Deployment Actions

Table 1. Per node actions duration

from all available regions; therefore at least one server should be activated at each node.

5. Temporal Response Evaluation

We measured temporal response to a deployment request event because we are interested in finding how quickly application networks can be deployed, and how fast application networks adapt to demand and resource availability variations. Figure 6 shows Xweb temporal response, it consists of three main phases: resource discovery, deployment plan creation and deployment time. Resource discovery initiates as soon as an application network specification web form is post by a service provider, it ends when resource availability information is saved.

$$t_{Resource\_Discovery} = t_{Mcast\_Request} + t_{Resource\_Availability\_Check} + t_{Secure\_Unicast\_Response} + t_{Update\_Resource\_Table}$$

$t_{Mcast\_Request}$ is proportional to network diameter and message size. $t_{Resource\_Availability\_Check}$ depends on agent resource availability implementation: a probe, a notification or last read value. $t_{Update\_Resource\_Table}$ is proportional to file write-time.

Deployment plan creation time starts when deployment managers receive specifications, it ends when connections to resource agents start.

$$t_{Deployment\_Plan\_Creation} = t_{Resource\_Mapping} + t_{Update\_Application\_Table}$$

	Cache ALN	Chat ALN
$t_{Resource\_Availability\_Check}$	0.5 ms	
$t_{Secure\_Unicast\_Request}$	53 ms	≅
$t_{Resource\_Allocation}$	2.1 ms	≅
$t_{Code\_Distribution}$	10284 ms	1530 ms
$t_{Server\_Installation}$	1980 ms	508 ms
$t_{Resource\_Binding}$	4440 ms	0 ms
$t_{Server\_Activation}$	460 ms	253 ms
$t_{Server\_Coordination}$		
$t_{Node\_Deployment}$	17164 ms	2391 ms

Table 2. Deployment Actions Duration

	Cache ALN	Chat ALN
$t_{Update\_Resource\_Table}$	2.4 ms	
$t_{Resource\_Discovery}$	58.4 ms	
$t_{Resource\_Mapping}$	1.4 ms	≅
$t_{update\_Application\_Table}$	3.4 ms	≅
$t_{Deployment\_Plan\_Creation}$	4.8 ms	≅
$t_{update\_Application\_Table}$	3.5 ms	≅
$t_{Total\_Deployment}$	17175 ms	2402 ms

$t_{Resource\_Mapping}$ is the computation time required to calculate a resource mapping and a deployment plan. It will be proportional to number of resource nodes, demands areas and algorithm complexity. $T_{Update\_Application\_Table}$ is proportional to file write-time.

Deployment time starts when connections to resource agents start, and ends when latest resource agent confirms it has performed all indicated commands without errors and to update an application database.

$$t_{Total\_Deployment}=MAX[\ t_{Secure\_Unicast\_Request}+t_{Node\_Deployment\_Time} + t_{Secure\_Unicast\_Response}]\ (i) + t_{Update\_Application\_Table}$$

$$t_{Node\_Deployment}=t_{Resource\_Allocation} + t_{Code\_Distribution} + t_{Server\_Installation} + t_{Server\_Activation} + t_{Server\_Coordination}$$

$t_{Secure\_Unicast\_Request}$ is fairly long since secure SSL connections require several messages exchanges. $t_{Resource\_Allocation}$ depends on the resource allocator implementation. $t_{Code\_Distribution}$ is proportional to code and size, and code server distance. $t_{Server\_Installation}$ is proportional to number and size of application code files and file write time. $t_{Resource\_Binding}$ is proportional to number of required resources.

6. Results

6.1. Additional Support for Deployment

We have found that Internet services programmable infrastructures must provide additional support for application network deployment. Besides providing shared resources, an execution environment, and virtualisation support for simultaneous execution of multiple services, Internet services programmable infrastructures should provide:

Resource containers that allow strict resource partitioning among different applications. Which will likely compete for resources, using resources allocated to other applications or consume more resources than were allocated. Mechanisms such as resource container [5] or cluster reserves [4] are a possible solution.

Virtual resource binders and switching sockets should provide dynamic resource binding. So that applications bind dynamically to resources, instead of halting and restarting them, applications code could be modified to switch communication channels on receiving an external signal. Or unmodified applications could bind to a virtual resource binder at service start time, and virtual resource binders will bind and unbind to real resources at an external signal. A switching socket will permit to reconfigure dynamically communication channels among servers without application support. We are investigating Service sockets [20] that provide reconfigurable sockets.

A safe execution environments should not interfere with applications, altering maliciously or unintentionally its behaviour, configuration or data. It is a hard problem without an apparent good solution. Only encrypted programs seem not to be vulnerable to malicious execution environments [21].

6.2 Improving Temporal Response

$t_{Resource\_Discovery}$ is a very low value because there is low overhead due to transmission latencies (the experiment was carried on a local area network) and because $_{Resource\_Availability\_Check}$ is the lowest possible value (it was implemented as a function that return resource properties values obtained some time in the near past). $t_{Deployment\_Plan\_Creation}$ is also low because $t_{Resource\_Mapping}$ is low. Algorithm did not have to select among many mappings because specifications were chosen so that all available nodes were required. Scalability of both times will have to be measured through simulation.

$t_{Resource\_Allocation}$ is a low value because resource allocations are simply written to a locked file. $t_{Server\_Activation}$ and $t_{Server\_Coordination}$ are added because server coordination took place at server start up.

Total deployment has its largest overhead because $t_{Resource\_Binding}$ and $t_{Code\_Distribution}$ are very long. $t_{Resource\_Binding}$ for a proxy caching service involves partitioning storage resources in a directory hierarchy where files will be cached afterwards; for typical cache sizes, it is a long operation in general purpose file systems. $t_{Code\_Distribution}$ is the longest overhead; specially for squid proxy cache, squid code was 948Kbytes, and because code server was situated at an international site with average transfers rate of 80Kbytes. Possible mechanisms to diminish it are reusing installed code through caching, replicating code servers, or incrementally loadable application code. Service installation is the second longest action; it could diminish if application code could be incrementally installed. The third longest action, server activation, involves reading several configuration files, checking various system parameters and variables, and performing several operating system calls.

7. Conclusions

This work presents a framework for application network deployment in a programmable Internet service infrastructure that provides mechanisms so that application networks can be rapidly created and can adapt to demand variations. It consists of these building blocks: programmable infrastructure, resource discovery, service specification, deployment plan, resource allocation, and service composition. It has been implemented in a prototype called the Xweb. It consists of a programmable infrastructure made up of nodes that provides a programming environment where to servers can be dynamically activated. Each node has a resource agent

that implements resource allocation, code distribution, resource binding and service composition. Deployment managers implement resource discovery, a service specification front-end, resource mapping, and service plan. This prototype shows how to deploy web-caching hierarchies and chat server application networks.

Results from such experimental evaluation are two fold, first implementation efforts shows that programmable infrastructures should incorporate resource container for strict resource partitioning and virtual resource binders in the form of switching socket or similar for dynamic resource binding. In second place application networks have been deployed in a period of time on the order of seconds, which is better that what could be obtained previously by manual operations.

Acknowledgements

This work was inspired by a research stay in Information Science Institute University of Southern California in Xbone project directed by Joe Touch [22].

This work has been partially supported by the Spanish MCYT project TIC2002-04258-C03-01 “Global and Peer-to-Peer Computing for Cooperative Learning”.

References

[1] Akamai Inc. "FreeFlow", http://www.akamai.com, Dic 1999.

[2] Ardaiz O., Freitag F., Navarro L., “Multicast Injection for Application Network Deployment”, 26nd IEEE Conference on Local Computer Networks. Tampa, Florida, Nov 14-16,2001.

[3] Amir E., McCanne S., Katz R."An Active Service Framework and its Application to Real-time Multimedia Transcoding". Proc. SIGCOMM`98

[4] Aron M., Druschel P., Zwaenepoel W. “Cluster reserves: A mechanism for resource management in cluster-based network servers”, Proc. ACM SIGMETRICS 2000.

[5] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. “Resource containers: A new facility for resource management in server systems”. Proceedings of OSDI '99, pages 45-58, 1999.

[6] Bassi, A., Beck, M., Moore, T. and Plank, J. “The Logistical Backbone: Scalable Infrastructure for Global Data Grids” ,Asian Computing Science Conference 2002, Hanoi, Vietnam, December, 2002.

[7] Chandra P. et. Al., "Darwin: Customizable Resource Management for Value-Added Network Services", 6 th IEEE Intl. Conference on Network Protocols (ICNP'98), 1998.

[8] Chuang, J.. “stor-serv: Adding quality-of-service to network storage”. Wrksp on Inet Service Quality Economics, Cambridge MA, Dec. 1999.

[9] Czajkowski K., Fitzgerald S., Foster I., Kesselman C., ”Grid Information Services for Distributed Resource Sharing”. Proceedings of HPDC-10, IEEE Press, August 2001.

[10] Ejasent, www.ejasent.com, 2002.

[11] Encyclopaedia Britannica, www.britannica.com, 2002.

[12] Foster, I.; Kesselman, C.: “Globus: A Metacomputing Infrastructure Toolkit”, in Intl. Jrl. of Supercomputing Applications ,11/2, p. 115-129, 1997.

[13] Isaacs R.,. Leslie I., "Support for Resource-Assured and Dynamic Virtual Private Networks" IEEE Jrl. on Sel. Areas in Comm. Vol. n. 2001.

[14] Kalt C. Internet Relay Chat Protocol RFC 2810, Apr. 2000.

[15] Korupolu M.R., Dahlin M., "Coordinated Placement and Replacement for Large-Scale Distributed Caches", Proceeding of IEEE Workshop on Internet Applications, July 1999.

[16] Lazar A., Lim K. S., and Marconcini F., "Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture," IEEE JSAC, Special Issues on Distributed Multimedia Systems, Sept. 1996, pp. 1214—27.

[17] Rabinovich M., Aggarwal A., "RaDaR: A Scalable Architecture for a Global Web Hosting Service", WWW8 Conference, Toronto May 1999.

[18] Santos J.R., Dasgupta K., Janakiraman G.J. and Turner Y., “Understanding service demand for adaptive allocation of distributed resources”. In Proceedings of IEEE Global Internet Symposium (GLOBECOM '02), Taiwan, November 2002

[19] Ripeanu M. "Peer-to-Peer Architecture Case Study: Gnutella Network Analysis", 1st Intl. Conference in Peer-to-Peer Networks Aug. 2001, Linköpings Universitet, Sweden.

[20] Schmitt M:, Acharya A., Ibel M., Iancu C.,“Service Sockets: A Uniform User-Level Interface for Networking Applications”. Technical Report TRCS2001-08, Computer Science Department, UCSB, September 2001.

[21] Sander T., Tschudin C., ‘Protecting Mobile Agents Against Malicious Hosts’, Proceedings of Workshop on Mobile Agents and Security, number 1419 in LCNS, pages 44-60, 1997

[22] Touch J., Hotz S., "X-bone: a System for Automatic Network Overlay Deployment", Third Global Internet Mini Conference in conjunction with Globecom´98, Nov. 1998.

[23] Vahdat A.; Anderson T.; Dahlin M.; Culler D.; Belani E.: “WebOS: Operating System Services For Wide Area Applications”, in The Seventh IEEE Symposium on High Performance Distributed Computing. July 1998.

[24] Wessels D.,. Squid Internet Object Cache. http://www.squid-cache.org. January 1998.