Adaptive Peer-to-Peer Web Clustering using Distributed Aspect Middleware (Damon) *

Rubén Mondéjar[1], Pedro García[1], Carles Pairot[1], and Antonio F. Gómez Skarmeta[2]

[1] Department of Computer Science and Mathematics, Universitat Rovira i Virgili
Avinguda dels Països Catalans 26, 43007 Tarragona, Spain
{ruben.mondejar, pedro.garcia, carles.pairot}@urv.cat
[2] Department of Computer Engineering, Universidad de Murcia
Apartado 4021, 30001 Murcia, Spain
skarmeta@fcu.um.es

Abstract. In this paper, we introduce the concept of adaptive peer-to-peer cluster and present our contributions on SNAP, a decentralized web deployment platform. In addition, we focus on the design and implementation of a load balancing facility by using the functionalities provided by our distributed AOP middleware (Damon). Using this approach, we are able to implement new mechanisms like decentralized session tracking and dynamic policies in a decoupled architecture. We believe that our model offers a novel approximation for modularizing decentralized crosscutting concerns in web environments.

1 Introduction

Nowadays, scalability and availability are two of WWW's main challenges. Therefore, servers may stop serving requests if their network bandwidth is exhausted or their computing capacity is overwhelmed. One way to deal with the scalability problem is to have several identical servers and give the user the option to select among them. This approach is simple, but it is not transparent to the client. An alternative is to rely on an architecture that distributes the incoming requests among these servers in an unobtrusive way. A successful solution to this problem comes in the form of *clustering* or *federation* of servers. Following a distributed pattern, servers are made redundant so as when one becomes unavailable, another one can take its place.

Many important websites operate in this way, but these replicated server alternatives are normally expensive to achieve and maintain. As a matter of fact, the actual trend is to head towards decentralized models. These models take advantage of the computing at the edge paradigm, where resources available from any computer in the network can be used and are normally made available to their members. However, such architecture also introduces new issues which have to be taken care of. Some of these issues include how to deal with constant node joins and leaves, network heterogeneity, and many others. Moreover, another important issue is the development complexity of new applications on top of this kind of networks.

* This work has been partially funded by the European Union under the 6th Framework Program, POPEYE IST-2006-034241.

For these reasons, we need a **middleware platform** that provides the necessary abstractions and mechanisms to construct distributed applications. The Java Enterprise Edition (JavaEE) (formerly known as J2EE) architecture is a worldwide successful middleware alternative for development of distributed applications. Nevertheless, it feels tied to the client-server model. In this setting, Aspect Oriented Programming (AOP) presents an interesting solution to modulate crosscutting concerns on JavaEE environments [1]. Thus, our aim is to modify any JavaEE server behaviour so as it is able to work on a peer-to-peer (p2p) web cluster.

Thereby, we present a solution based on *distributed AOP* [2, 3, 4]. Specifically, we have developed a new approach to support decentralized JavaEE crosscutting concerns that includes: **session failover** for stateful applications, a complete HTTP **load-balancing** technique which permits dynamic client redirection to other servers, and a **runtime policy** system that defines node selection and workload distribution patterns. The advantages of our solution are as follows: a complete abstraction and decoupled design, and a transparent and generic interception server side scheme which is valid for any JavaEE servlet container implementation.

2 Adaptive p2pWeb Clustering

WWW is the most used technology in the Internet. Wide-area application development usually targets web environments. However, clients suffer non-desirable errors like "page is currently not available" or "resource is not accessible" due to server problems. On the other hand, p2p computing provides and shares resources efficiently among all network *peers*. As a consequence, it seems natural to merge standardized WWW wide-area applications with the the benefits p2p has to offer. Until now, we have worked in this synergy of p2p and web technology with our p2pWeb model. In order to support web applications and services deployment and management, we have developed a p2pWeb platform called SNAP [5]. In this context, one of SNAP's current limitations is in providing stateful applications. Specifically, the problems this paper tries to resolve are the use of front-side load balancing and the lack of distributed session tracking. Moreover, due to the complexity of JavaEE architecture it is difficult to add new functionalities or behaviours in a transparent way.

AOP permits to elegantly intercept and modularize crosscutting concerns in runtime. In addition, **distributed AOP** offers many interesting features for our p2p web cluster including monitoring and adaptability capabilities. In this way and making good use of structured p2p substrates and dynamic aspect frameworks, we have designed Damon [4], a fully decentralized aspect middleware built on top of a structured p2p overlay. Therefore, this work represents our first approach merging both concepts: p2p clustering and distributed AOP. As seen in Figure 1, we present an adaptive p2pWeb architecture where Damon enables transparent and distributed interception over the SNAP platform.

In this section, we explain the mechanisms to supply clustering issues, including distributed session failover, load balancing techniques, and runtime policies. In this line, we are focused on the inclusion of these features based on the separation of *De-*

centralized Crosscutting Concerns [4]. Thus, we also guarantee the necessary interdependence between the new *aspectized* mechanisms and the SNAP web server code. The real cohesion between SNAP and Damon is achieved with the corresponding pointcuts of the distributed aspects. Finally, these aspects are deployed on nodes that are running the SNAP applications that they aim to intercept.

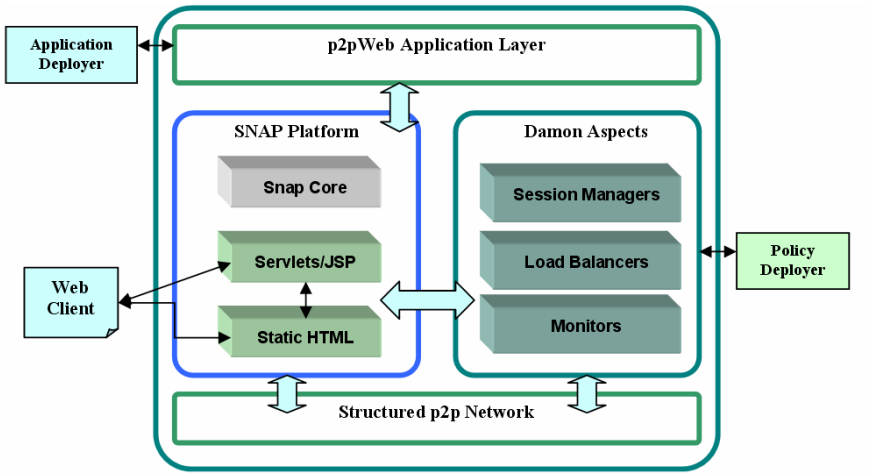

Fig. 1. p2pWeb architecture diagram.

2.1 Achieving Session Failover

In clustered environments, HTTP sessions from a web server are frequently replicated to other group servers. Session replication is expensive for an application server, because session request synchronization usually is a complex problem. Therefore, the initial issue we intend to solve is session tracking for stateful applications. Certainly, we need to use session migration when a node is shutdown (i.e. it crashes) or the load balancer decides to redirect the client to a different node.

Our solution is based on our Damon aspect framework, its persistence service and URL rewriting. For decentralized **session persistence** we use its ID to identify the session. This solution has a structural problem though, because session ID is only considered to be unique in the original host, but this is not applicable to whole network. Therefore, we need to change the session ID generator by means of intercepting the session creation code.

Once our session data is accessible throughout the network, we need a way to restore it whenever a new server becomes responsible for that client. The idea is to have meta-information that identifies the session directly embedded into the URL. This technique is known as **URL rewriting.** We mainly use URL rewriting to report the

client session ID to new server. URLs are modified before fetching the requested item, attaching the session ID like a usual request parameter. For instance:

"http://server1:8080/dcalendar/calendar.jsp?_JSESSIONID=08445a31a78661b5c746feff39a9db6e4e2cc5cf".

```
@Before("execution(* javax.servlet.http.HttpServlet.service(..)) AND args(req,res)")
public void retrieveSession(ServletRequest req, ServletResponse res) {

   HttpServletRequest httpReq =(HttpServletRequest)req;
   //restore session?
   if (req.getParameter("_JSESSIONID")!=null) {
      String jsessionid = req.getParameter("_JSESSIONID");
      PersistenceHandler dht = thisEndPoint.getPersistenceHandler(aspectClassName);
      Hashtable sessionData = (Hashtable) dht.get(jsessionid);
      if (sessionData!=null) restoreSessionData(sessionData, httpReq.getSession());
   }
}
```

Fig. 2. Retrieve Session Damon Pointcut.

Figure 2 shows the *retrieveSession* pointcut. Before requests are made, the retrieveSession pointcut is executed. This pointcut checks whether the session ID is among the request parameters in order to restore previous session information from the network. If it is found, such remote session data is loaded into the new local server session.

```
@Around("execution(* javax.servlet.http.HttpServlet.service(..)) AND args(req,res)")
public Object loadBalancer(JoinPoint joinPoint, ServletRequest req, ServletResponse res) {

   HttpServletRequest httpReq =(HttpServletRequest)req;
   HttpServletResponse httpRes =(HttpServletResponse)res;
      //redirect to other instance? Policy's decition (1)
      if (isTimeToRedirect(getRequestParam(req, "_REDIRECT_FROM"))) {
        String newHost = getNewHost(); //Policy decition (2)
        //Obtains request parameters and returns the new url destination
        String url = constructNewURL(httpReq)
        HttpSession session = httpReq.getSession(false);
        if (session!=null) { //Has session?
          ReflectionHandler rh = thisEndPoint.getReflectionHandler(aspectClassName);
          String sid = session.getId() + ":" + rh.getLocalInstance().getHostName();
          String jsessionid = generateSHA(sid);
          if (params.equals("")) url += "?_JSESSIONID="+jsessionid;
          else url += "&_JSESSIONID="+jsessionid;
          url += "&_REDIRECT_FROM="+ rh. rh.getLocalInstance().getHostName();
          session.invalidate();
        }
        httpRes.sendRedirect(url);
      }
      return joinPoint.proceed();
}
```

Fig. 3. Load-Balancer Damon Pointcut.

2.2 Load Balancing Technique

A common approach chosen in these cases is known as front-side load balancing. This technique is to perform load balancing only at the beginning of a session and thereafter the connection between client and server is fixed, without any interaction with a load balancing instance anymore. Regarding SNAP's initial load-balancer, it clearly follows a front-side based strategy using p2p locators [5]. For some applications, it can be adequate to bind clients to specific servers. However, such approxima-

tion has apparent limitations, such as what happens when the specific server the client is bound to stops working.

Trying to better improve SNAP's load balancing algorithm, we intend to dynamically map load-balancers to web applications. These are implemented using a Damon aspect (see Figure 3). By means of the session tracking aspect session data can be effectively restored. Again, the attached session ID in the URL identifies the client among the servers. The *loadBalancer* pointcut intercepts any client requests and determines whether they are to be served or redirected to any other running application server. It also provides two new extension methods: *isTimeToRedirect(String from)* and *getNewHost()*.

2.2.1 Load-Balancing Runtime Policies

In order to complete our system we provide policies designed to demonstrate the viability of our approach. Workload distribution in traditional web clusters is different from our decentralized system, because we do not have any centralized spot. Therefore existing centralized load balancing policies are to be modified to be efficient in our p2p system. In such case, we need to implement policies that perform well under heavy-loaded systems with highly variable workload.

There are many different algorithms which define the load distribution policy, among them are: random, Round-Robin, weight-based, least recently used (LRU), minimum load, threshold-based, local-space, last access time or other parameter-based policy. Following, we describe two examples of our implemented policies.

First, the Round-Robin is the most basic policy, although it is not the simplest, as it is the random one. We have implemented a decentralized version of the traditional Round-Robin policy where requests to each host are scattered throughout all hosts holding an application instance. Basically, it uses Damon reflection layer [4] to obtain the other instances, and chooses the next host after the previous one.

Secondly, we have also implemented the Least Recently Used (LRU) policy as well. In this policy, the host's stress index is calculated as the average of server requests per second. Since we need to perform communication among other instances, messaging methods are used to distribute current server stress information.

2.3 Validation

We observe that the cost of our new concerns locally is directly produced by the aspect engine of our Damon framework's implementation and it is similar to other evaluation results. We have as well conducted several experiments to measure the cost of our solution in a distributed scenario. In summary, we have mainly measured the system reaction to new policies activation and requests management.

The experiments were conducted on the PlanetLab [http://www.planet-lab.org] network, located in a wide variety of geographical locations, to measure the overhead of our system. Before each test, we estimated the average latency between hosts so as

to get a measure of how much overhead is incurred by the aspect activation and the following requests after the activation phase.

The values shown in Table 1 are the median of all the tests run. Each test was done using 500 random activations and advice calls for each pair of hosts. In conclusion, using the PlanetLab testbed, we verified the correct behaviour of the system and that Damon does not impose an excessive latency (the normalized incurred activation overhead is **3.27** and the one imposed by advice calls is **1.78**).

Table 1. Overhead observed of runtime aspect activation tests in milliseconds

Originator Host	*Destination Host*	*Latency*	*Activations*	*Advice calls*
planetlab2.urv.net	planetlab4.upc.es	10	97	35
planetlab-5.cs.princeton.edu	planetlab02.erin.utoronto.ca	73	214	103
planet1.scs.stanford.edu	bonnie.ibds.uka.de	180	449	260
planetlab02.dis.unina.it	planet1.manchester.ac.uk	45	192	122
planet1.cs.rochester.edu	planetlab-2.it.uu.se	108	409	220

3 Related Work

To the best of our knowledge, this work is the first approach in use distributed AOP in order to provide session tracking and load-balancing policies on a p2p web cluster. For this reason, we will basically focus this section on describing related work in more traditional solutions and in AOP approaches.

There exist different *non-AOP solutions* to introduce clustering in web environments. Regarding servlet filters and server mods, the Java Servlet specification version 2.3 introduces a new component type, called filter. A filter dynamically intercepts requests, before a servlet is reached. Responses are additionally captured after the servlet is left. There also exist a variety of server mods (like load-balancers) that directly depend on the server's implementation. Usually, these mods are difficult to bind to a specific server. Finally, WADI [http://wadi.codehaus.org] aims to solve problems about the state propagation in clustered web servers. Thus, WADI provides several services useful for clustering on JavaEE platforms. Nevertheless, its main drawback is that it needs wrapping extensions for each different server's implementation and forthcoming versions.

On the other hand, there are a number of non-distributed AOP solutions existent in the literature about concerns in JavaEE architectures [1, 6]. However, there are only a few distributed AOP solutions in complex systems and previous to Damon [4]. In [2], authors present a distributed AOP language called DjCutter. This language is the precursor of the remote pointcut concept. Remote pointcuts are similar to traditional remote method calls, which invoke the execution on a remote host. Nevertheless, these advices are executed in a unique and centralized host, thus making this solution inappropriate for dynamic systems. In this way, we can find more recent solutions like AWED [3] that solves many of these problems. AWED presents a more complete language for explicit aspect distribution. In our case, Damon is more like an abstracted and decoupled middleware that presents easier integration in this kind of scenarios.

4 Conclusions and Future Work

In this paper we have presented an adaptive p2p cluster using a distributed AOP approach. By recalling Figure 1, we observe that the novelty of this paper results in the two being merged into one cohesive solution. It is important to stress out that we could transparently swap SNAP p2p cluster by another web system thanks to uncohesion nature provided by Damon. Otherwise, we have designed the necessary mechanisms to allow stateful wide-area web applications: a **session failover**, an **HTTP load-balancing mechanism**, and a **runtime policy** system.

As a consequence, our solution aims to be as generic as possible thus supporting more dynamic environments. Our p2pWeb cluster, instead of being a traditional cluster with replicated servers; it is an effectively wide-area platform where each server holds different applications running on top of it. Moreover, we have designed our architecture using AOP which transparently intercepts the most significant servlet methods. By using such a solution we achieve more elegant, modular, and suitable mechanisms than traditional alternatives.

Finally, by means of our contributions, the client's experience and usability when browsing p2p web applications is improved, since all fault tolerance and load balancing algorithms run in the background transparently. As a consequence, the client remains unaware of server changes due to overwhelming or failures, as its session state propagates from one to another.

To conclude, we pretend to further develop our adaptive middleware in mobile scenarios (Mobile Ad-hoc Networks, MANETS) within the POPEYE IST project [http://www.ist-popeye.eu].

References

1. Mesbah, A., van Deursen, A.: Crosscutting Concerns in J2EE Applications. Seventh IEEE International Symposium on Web Site Evolution, 2005.
2. Nishizawa M., Chiba, S.: Remote Pointcut --- A Language Construct for Distributed AOP. Proc. of AOSD '04, Lancaster, UK. pp.7-16, 2004.
3. Benavides Navarro, L. D., Südholt, M., Vanderperren, W., De Fraine, B., Suvée, D.: Explicitly distributed AOP using AWED. In Proceedings of the 5th Int. ACM Conf. on AOSD'06, March 2006. ACM Press.
4. Mondejar, R., Garcia, P., Pairot, C., and Skarmeta, A. F.: Damon: a decentralized aspect middleware built on top of a peer-to-peer overlay network. In Proceedings of SEM'06 (Portland, Oregon, November 10 - 10, 2006).
5. Pairot, C., García P., Mondéjar, R.: Deploying Wide-Area Applications is a SNAP. IEEE Internet Computing Magazine. March/April 2007.
6. Han, M., Hofmeister, C.: Separation of Navigation Routing Code in J2EE Web Applications. Proceedings of ICWE'05, Sydney, Australia (2005).