Moving Toward Distributed Web Services

April 8, 2004

1 Introduction

Web services are currently being hyped in the computer industry as the next major evolution for computing. So then what is a web service? The formal definition is "[w]eb services are a series of standards and evolving standards that are being designed and specified by the Worldwide Web Consortium (W3C) to foster cross-platform program-to-program communications"[3]. What this boils down to is that at it's heart, a web service is nothing more than a method to create platform independent, distributed, network-based applications. The truly interesting aspect of this goal for web services is that it will require a change in the basics of where an applications source is run, from a single location, to distributed around the Internet. Due to this change in focus, one can easily see that in order for a system created using this model to succeed, other systems beyond one's own must be operating reliably. This leads to the inevitable question, can developers trust that the web services that their applications relies upon shale be ready and available every time the application is run? The answer is a definite affirmative with the adaption of two currently existing networking technology coupled with standardization, the worry about the reliability of a web service could be reduced to the amount of faith a developer has in the network connection of the application's end user.

2 Unreliable Service

Reliability and interoperability are two related problems when used in the context of a reliable web service. Lack of reliability comes in the form of a service's server being over-loaded from the massive volume of messages that will be generated from a web service enabled applications, as well as the load on the servers themselves[3]. Interoperability comes into play since the services that the web service methods an application is calling could be changed, causing the application to no longer function[9]. Examples of these situations are easy to find. The easiest to find are servers that can not with stand massive amounts of messages transfered to it. One simply has to visit the web site slashdot, http://slashdot.org. On any given day there should be at-least one site linked to from this site that has become "slashdotted", a term meaning that the linked to server could not withstand a massive amount of traffic coming to the site and has disappeared from the face of the Internet. The second is almost as easy to find as one only needs to look through the API's published by Sun Microsystems for its Java programming language. Contained

in the APS's one will find numerous methods that are no longer implemented in the current version of the language, which causes applications written using those methods to fail.

While these are ready examples of what can indeed go wrong for web services in the future, we currently have time to address these issues before they become a problem for the application developer. The first thing that can be done is to use virtual server technics to provide a scalable solution to the problem of a massively loaded server. The second can be address by developing standards that web services must follow in order for them to be available for use in applications.

3 Technology Solution

In order to give application developers the confidence to use distributed web services in their applications two things must be addressed, fault tolerance at the server level and the web service level. In Section 3.1 we will show that server fault tolerance is a solved problem and there fore the majority of developer concerns have already been addressed, being that the service they are calling could disappear due to massive load. Then in Section 3.2 we will show emerging technology that can be applied to further increase the reliability of web services.

3.1 Server Fault Tolerance

The idea of using a virtual server to implement load balancing, for increased reliability, has been around for over a decade now. Indeed the diffusion of knowledge on how to implement these scheme has made its way down to the undergraduate level now, as shown in the posted lectures for the Rutgers University Linux Administration course. This should imply that anyone with a college degree in a technical field should be able to utilize these methods to alleviate some of the load on a server. Though the work was initially focused on web servers, there is nothing stopping them from working for web service servers. We selected the first implementation of a web server load balancing scheme since it was the initial work, as well as because implementing its solution is a trivial exercise. Also presented as a solution is Distributed Packet Rewriting (DPR) as it allows for the scalable balancing of requests without a central point of failure.

The initial work into this field was done at NCSA in 1994[4]. Their idea was to utilize the alternate server field of their DNS table entries so that identical, configuration wise, servers could serve as a scalable solution to the increased traffic to their web site. This very simple idea proved to be a workable solution to the problem. However, slight problems ensued from intermediate caching of DNS information, causing the load to not be evenly distributed across the virtual server. This was addressed by shortening the TTL field for the server in the DNS entry, which in turn decreased the effect of caching. This method could very quickly be applied to web services allowing the service provider to increase the amount of machines as demand for their service increases, simply by announcing via DNS that an alternate server exists.

DPR[2] is a more complicated system for load balancing, however it provides increased scalability as the central point of failure has been eliminated

by distributing the incoming requests between the servers themselves. By utilizing a reverse IP masquerade, DPR forwards packets directed to a free server from other servers in the cluster that may be experiencing too high of a load to deal with the request directly. This is an improvement over using DNS as the load balancer as the servers in this case can communicate to better balance the load, even if DNS route caching occurs. While this method does increase the service related load on the system due to its stateful nature, on average the servers should be able to deal with the slight increase in load due to DPR.

Many other methods for creating reliable servers exist, possessing varying degrees of complexity, and coming from many different vendors. What this means for web services is that providing the web service provider is serious about keeping their service running, an application developer should not have to worry about the web service disappearing due to popularity of the service. A perfect example of this is the search engine Google. The shock to the Internet would indeed be great the day that large numbers of people could not connect to the search service giant.

3.2 Service Fault Tolerance

Server reliability alone would probably be enough to allow worried developers to believe that the service they are calling will be there and work each time his application calls on the distributed service, yet still more reliability can be gained by improvements to how the web services are discovered and implemented. To gain improvements here a few things should happen. First a DNS-like registry should be created for the discovery of services, which are described in the registry in a standard and consistent way. Second the services that provide the same functionality should standardize on the method names and arguments that are provided by their services. The first can be solved with an evolving standard called Universal Description, Discovery, and Integration (UDDI), while the second requires a method to standardize the entries in UDDI.

3.2.1 UDDI

The UDDI project was started in September of 2000 with the goal of creating an e-business registry to enable company's to discover service providers online. This Universal Business Registry (UBR), can be compared to a telephone directory since the information is provided in three subsections being: company contact information, categorization of businesses by a standard taxonomy, and the technical information about each service listed. Operating the UBR, like operating DNS's root nodes fell to four company's, IBM, Microsoft, NTT COM, and SAP[7] [8]. Between then and now UDDI has evolved into part of the web service infrastructure, by allowing web services to automatically look-up the services it needs to complete its task. Since UDDI also allows for a large level of detail to be specified in the UBR, look-ups in the URB can specific enough to identify all services that match the interface requirements of the calling application[5]. When using the UBR as a DNS like service, applications that are written in a non-vendor specific way can use any service that provides the method it needs to complete its task.

A ready example is found in a stock ticker application. The application would search the UBR to find a service that provides looking up the current value of stocks. The registry would then reply with a web service that provides the service. Since many different sources could provide this information if one of them fails, the application could perform another look-up and find a server that is currently alive. This allows the user to never notice a disruption in service, which in turn allows the developer to treat the web services that the application utilizes as reliable.

3.2.2 Service Standardization

With the reliability provided by virtual servers and dynamic service look-up from the UDDI's UBR an application developer is provided with a very reliable system. By utilizing the methods aforementioned, worries about service availability from a technical stand point is all but solved. However, a final method is needed to fully insure continuous operation which is standardization. Standardization in this sense is needed on two fronts, first services listed in UDDI must describe themselves in a consistent manner relative to other services of the same type [1][6]. Second services should ensure that it's basic functions follow the same syntax as other services which provide the same functionality.

[1] proposes an idea for how to have a flexible standard for describing web services in a UBR. Their idea is to create a dynamic vocabulary which fits in well with the evolving nature of web services. In their model, an organization creates a vocabulary to describe the type of web service it provides. This body then will then control who can use this vocabulary in the UBR, in order to create a feeling of trust and consistency in the system. Anyone that then creates a similar service would then follow the established vocabulary to describe their new service and submit it to the original creator in order for the new service to be added to the UBR. This system should be sufficient providing that the controller of the vocabulary freely allows anyone to describe their service with their vocabulary and will only refuse publish a service that does not properly conform to the vocabulary.

This idea of a dynamic vocabulary also allows for web services to have overlaps which add reliability to end user applications. For example, if an application developer is creating a program that searches the web, it might first try to use Google's web service interface. However, if for some reason, Google is not available, the application could perform a look-up in the UBR and call the search method on a web service provided by Yahoo!. The problem now is no longer one of reliability, as the method still can be called and a web search is performed, but one of Quality of Service, which is outside the scope of this paper.

4 Conclusion

When dealing with distributed web services, a valid concern for developers is that an application could fail due to a lack of server reliability. We have shown, however, that by using standard fault tolerance technics, any single service could be made to with stand failures of individual servers using the technics described in [2] [4]. Reliability can further be enhanced by extending

UDDI to support the ideas presented in [1] combine with the basic service overlap we propose in 3.2.2. While this then allows developers to stop worrying about writing web service enabled applications due to reliability concerns, these methods do not address the topics of Quality of Service and the resulting complexity of the web services themselves and the applications utilizing them due to the increased control. In the first case it is probable that some reliability will have to be sacrificed in order to ensure the quality of a services, such as a search application that wants searches performed by Google and not Yahoo!. Should Google stop responding, the service would fail, however, if the developer was not as concerned about the QoS then the application would continue to function normally despite Google being down. The second is a much more difficult challenge as only time shale be able to provide the answer to this problem. The solutions provided in this paper at least require the knowledge equivalent of a four year bachelor's degree, though given that the adoption of web services will not be complete over night, the trickle down effect of advanced technics should filter down to require less skill to perform, allowing web services to truly be a developer friendly solution.

References

[1] Alan H. Karp and Kevin Smathers. Advertising and Discovering Business Services. Available at http://www.w3.org/2001/03/WSWS-popa/paper09.

[2] A. Bestavros, M. Crovella, J. Liu, and D. Martin. Distributed packet rewriting and its application to scalable server architecture s. Technical Report 1998-003, 1, 1998.

[3] Bloor Research - North America. Web Services Gotchas. Available at http://www.ibm.com/.

[4] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. *Computer Networks and ISDN Systems*, 27(2):155–164, 1994.

[5] P. Lacey. Uddi and dynamic web service discovery. *Dr. Dobb's Journal*, pages 30–35, 2004.

[6] Marwan Sabbouh, Stu Jolly, Dock Allen, Paul Silvey, Paul Denning. Interoperability. Available at http://www.w3.org/2001/03/WSWS-popa/paper08.

[7] The Stencil Group. Why UDDI Will Succeed, Quietly: Two Factors Push Web Services Forward, Apr. 2001. Available at http://www.stencilgroup.com/.

[8] The Stencil Group. The Evolution of UDDI, Jul. 2002. Available at http://www.stencilgroup.com/.

[9] Todd Karakashian. BEA Position Paper on the W3C Web Services Workshop, 2001. Available at http://www.w3.org/2001/03/WSWS-popa/paper48.