# Web Services Security and Load Balancing in Grid Environment

Liang Fang, Aleksander Slominski, and Dennis Gannon

Computer Science Dept, Indiana University
Bloomington, IN, 47403 USA

{lifang, aslom, gannon}@cs.indiana.edu

## Abstract

*Web services security has some crucial problems to be solved in building Grid applications. In particular the current implementations of message-level Web services security are very slow and therefore extremely vulnerable to even the simplest types of Denial of Service (DoS) attacks. The more advanced is the security mechanisms, the worse the performance and scalability of the whole system. On the other hand, the marriage of Web services and Grid computing promises enormous computing resources for scientific and commercial Grid applications. The complexity of Web services security has come to the point that the processing steps can be broken into functional components that are distributed over the computational Grids. In this paper we introduce such an infrastructure that parcels out the message-level Web services security processing to load balanced computational Grid nodes. We designed it for scalability, dynamic configurability as well as DoS attack mitigation.*

## 1 Introduction

Derived from the notion of electric power grid, the computational grids, or in short, the Grids, comprises of a huge amount of autonomous computational and data resource domains, called *virtual organizations* (VO), distributed across the Internet. The Grid computing technologies make the resource sharing possible to users at each corner of the Grids. Under the direction of Global Grid Forum (GGF), Grid computing adopted Web services (WS) as its underlying implementation, because Web services' loose coupling and interoperability features perfectly meet the goals of the Grids. The merging of Grid computing and Web services technologies expedites the evolution on both sides. Grid services with sophisticated Web services technologies, such as Globus Toolkit [6] and UK eScience [7], arise to meet requirements of scientists, businessmen and government users.

Security is critical in both Grid computing and Web services. Grid security infrastructure (GSI) is the answer to the security issue in Grids. Fundamentally, GSI provides secure communication, mutual authentication, and delegation to Grid applications, by leveraging both transport level security and Web services security, which contains a collection of emerging specifications with the goals to address general distributed system security issues including authentication, authorization, privacy, confidentiality, integrity, non-repudiation, etc. Based on XML Signature and XML Encryption standards, Web services security specifications consist of WS Security, WS Trust, WS SecureConversation, etc. In contrast to transport level security (SSL and TLS), Web services security provides protection mechanisms to provide security at message level, which is agnostic to transport protocols and covers end-to-end communication routed across VO's, instead of the point-to-point fashion in TLS.

As Web services technologies become more sophisticated, they have their growing pains, especially in the aspects of performance and scalability when Web services security is present. The problem stems from XML manipulations. The generality and interoperability of XML-based Web services require a series of complicated operations in XML Signature and XML Encryption before XML messages are signed or encrypted. The breakdown results of Shirasuna et al.'s experiments indicate that among these operations, XML canonicalization in particular takes around $10^2 - 10^3$ milliseconds to process a SOAP envelope. Together with XML parsing and conversion efforts, message-level security has much more overhead than TLS-based solutions [10].

Web services in many cases solve computationally-intensive problems in a distributed way. When Web services security is applied, complex XML processing and security related operations become a bottleneck. It is especially em-

barrassing when the Web service runs on the Grids, with abundant computing resources idled at the same time.

Currently because of the lackluster performance, most of message-level security solutions still meander at the prototype stage. Unsurprisingly, scalability is even a trickier problem to be addressed. No mention the Denial of Service (DoS) attacks. With message-level security, Web services become so vulnerable to DoS attacks that it takes the attackers little effort to bring down an un-protected service.

In our Web services framework and its applications for scientific projects, such as Linked Environments for Atmospheric Discovery (LEAD), we encountered this dilemma: in the long term, message-level Web services security is advantageous over transport level security; in the short term, if no breakthrough could be made on the scalability bottleneck, WS security-based applications will not be able to meet the requirements of our applications. It is therefore imperative for us to investigate a set of viable solutions.

There are two threads of work in progress in response to the WS security performance and scalability problems. One thread is to optimize the workflow and implementation of the low-level XML operations, such as streaming validation for SOAP signatures [8]. However, while many groups are working toward this goal, the situation cannot be changed over night. We still foresee a long way to go before the optimized implementation of WS and XML security libraries are able to closely compete with TLS in their performance showdown.

The other thread treats the performance issue at a high level with load balancing mechanisms used in distributed systems. While Web services are originally designed to provide distributed solutions, the complexity of Web services has come to the point that the processing of Web services needs to be granulated into functional components which are running in a distributed fashion, on the computational Grid. This is the intuition of our work, called DEN, introduced in this paper.

Backed by the abundance of computing-resource Grids, the goal of this work is to provide a scalable message-level security solution for practical Web service applications. The side benefits we derived from this work include reliability and dynamic configurability. Further development may make DEN a provisioning solution to general Web services and Grid services.

The rest of the paper is organized as follows: Section 2 covers the related work; Section 3 mathematically analyses the proposed model in Web services processing. The implementation work and applications are described in section 4. In section 5, with a set of performance data collected from the experiments, we evaluate this work in comparison to other approaches. We make a conclusion in section 6 in addition to some future directions.

## 2 Related Work

Because of the well-known performance issue of Web services, there have been quite a few techniques adopted during the development of real-life Web service applications. These techniques include client side and server side caching, XML compression [3], binary XML, XML stream processing [1, 8], etc, though most of them are only applicable under some specific circumstances.

For decades, load balancing has been researched and harnessed into different levels of distributed systems for scalability [2]. Similarly, enormous efforts have also been put into scalable Web servers since the prosperousness of WWW. The server-side load balancing approaches have two categories: hardware (or DNS-based), and software or (Web server-based). Both categories of approaches are popular in providing scalable Web contents.

DNS-based load balancing casts the logical server names to an IP address chosen from a cluster of them serving the same functions according to a specific algorithm. The most frequently used algorithms are round robin (RR) and server-state-based ones. Nowadays DNS-based load balancing usually has hardware support for acceleration.

Web server-level approaches are mostly software-based. There is a gateway that dispatches the requests to a Web farm. Different from DNS-based ones, the gateway dispatcher needs to rewrite the request packets by replacing the URL or IP addresses. Sometimes HTTP redirection is employed as an alternative.

However, though mostly in HTTP and supported by Web servers, Web services are not necessarily so. FTP, Simple Mail Transfer Protocol (SMTP) and Blocks Extensible Exchange Protocol (BEEP) are also seen to carry Web service traffic, which means that the existing Web server-based load balancing techniques cannot be put into all Web services.

Besides, both existing hardware and software load balancing solutions require replicated Web resources. Replication is not a concern when Web resources are static and Web services are stateless. After IBM and Microsoft introduced stateful Web services with their specifications and frameworks called Web Services Resource Framework (WSRF) and Web Services Enhancement (WSE) respectively, replicating a Web service could be very complicated with synchronization and consistency problems.

In retrospect of our problem defined in the first section, the bottleneck is in processing message-level security of SOAP messages instead of the Web service applications. If we could parallelize and pipeline the sophisticated processing steps while keep the application logic intact, that will address the problem with minimal efforts.

## 3 From WS-Dispatcher to DEN

### 3.1 WS-Dispatcher

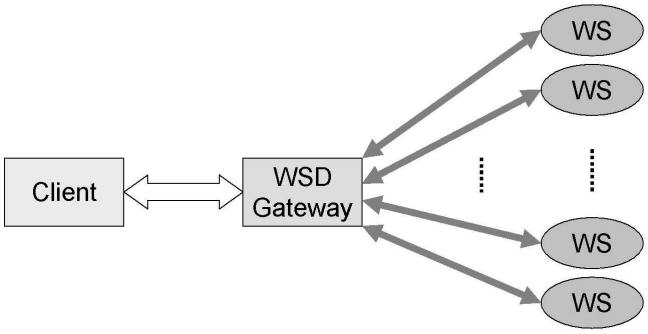


**Figure 1. A Simplified WS-Dispatcher**

WS-Dispatcher (WSD) was designed originally to deal with firewalls. Web services may choose their own port numbers for incoming requests; however firewalls only allow very limited well known port numbers to be open publicly. WSD serves as a gateway that listens on a well-known open port behind a firewall, such as port 80, for incoming requests, and forwards the requests to the corresponding Web service instances according to its routing table [11].

It turns out this layer of indirection brings more benefits than expected. Illustrated in Figure 1, the WSD design is first of all a typical software-level dispatcher-based load balancing architecture. Moreover, WSD makes Web service naming easier. We do not necessarily wait until a Web service is instantiated for its possibly long weird URL. Instead, we may first advertise a WSD-based human-friendly URL name in Web services registries, and bind the service instance to the name in a routing table at run time.

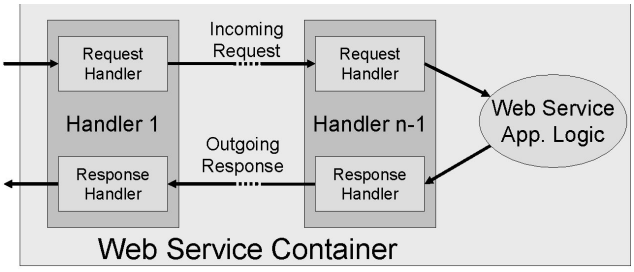


**Figure 2. Processing Stack of a Web Service**

Figure 2 shows a typical processing stack of a Web service. Server-side processing handlers could function for signature generation, signature verification, encryption, decryption, policy validation and so on. For each request $r$ to a Web service instance, there is a "processing vector":

$$\vec{h}(r) = \{\vec{h}_1(r), \vec{h}_2(r), ..., \vec{h}_n(r)\}, \quad (1)$$

while

$$\vec{h}_i(r) = \{h_{i-in}(r), h_{i-out}(r)\}.$$

$h_{i-in}$ means the processing direction is toward the application logic; while $h_{i-out}$ goes outward. Suppose $T$ is the time for a single computation node to process all the steps $\vec{h}(r)$:

$$T_0(r) = \sum \vec{h}(r) \quad (2)$$

Annotate the constant $w$ the overhead time for each request, during which WSD makes a routing decision and forwards the request. It also includes the packet transferring time. For Figure 1, the total cost of processing a request is

$$T(r) = \sum \vec{h}(r) + w. \quad (3)$$

Combine the equations 2 and 3. The average cost of requests for a load balancing tree with $m$ branches is

$$\hat{T} = \frac{T_0}{m} + w \quad (4)$$

Apparently, when there are more than one nodes, $\hat{T}$ beats $T_0$ as long as

$$w < \frac{m-1}{m} T_0, m > 1, m \in N.$$

It is not hard as currently $T_0$ is at least 100 milliseconds; while $w$ is usually at $ms$ to $10ms$ level. However, $w$ should never cost more than $T_0$, since $\lim_{m\to\infty} \frac{m-1}{m} T_0 = T_0$; otherwise $T$ will be even worse, though it may still scale.

### 3.2 DEN

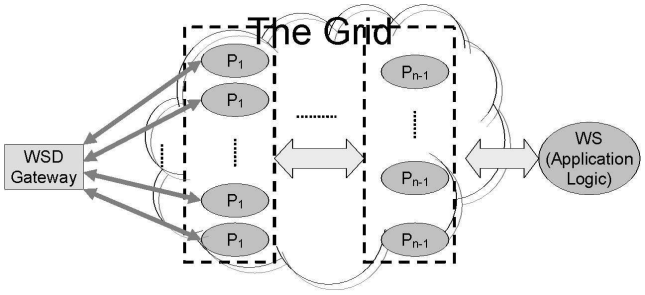


**Figure 3. DEN: WS Processing in Grids**

Similar to DNS-based or other software-level load balancing methods, with WS-Dispatcher, Web service providers have to maintain the replicas of their Web services, which have serious challenges in consistency and synchronization for the stateful ones.

DEN addresses the performance and scalability bottleneck directly by targeting at the Web services security processing steps without touching the application logic at all. It first granulates the processing vector and makes them each a separate processing node which is distributed across the Grids. The processing nodes with the same function are grouped as a functional processing farm. For instance, in DEN, we dedicate a cluster of Grid nodes to signature verification as a processing farm. In Figure 3, the processing workflow is pipelined. The ideal total cost of processing one request in a traditional sequential approach is:

$$\hat{T'} = \sum (\hat{h} + \vec{\Delta}) + w. \tag{5}$$

while

$$\hat{h}_i = \frac{h_i}{u_i}, \text{ and } \vec{\Delta} = \{\vec{\Delta}_1, \vec{\Delta}_2, ..., \vec{\Delta}_n\}.$$

$u_i$ is the number of assigned processing nodes for the processing farm $i$; $\Delta$ is the overhead cost of a request for both inbound and outbound directions. In order to make $\hat{T'}$ be less than $\hat{T}$, $\Delta$ has to satisfy the following conditions:

$$\sum \Delta < T_0(\frac{1}{m} - \sum \frac{1}{u_i}) \tag{6}$$

It is impossible when $m = \sum u_i$, and only makes sense when we have a much large computation pool to keep $\frac{1}{m} - \sum \frac{1}{u_i} > 0$.

Furthermore, in a fully pipelined processing stack, because the bottleneck is $\max(\hat{h}(r))$, the actual cost is:

$$T_a = \sum (\max(\vec{h}) + \vec{\Delta}) + w. \tag{7}$$

To avoid any processing step to be the bottleneck:

$$\hat{h}_i = \hat{h}_j, i \neq j, i, j \in N \tag{8}$$

The equation 8 means the right proportional amount of nodes should be assigned to the farm against its computational cost. For example, if signature verification takes double the time of decryption, we should assign double nodes to signature verification to avoid it being the bottleneck.

However, pipelining is required only if there are dependencies between processing steps. Fortunately, many processing steps are not related to each other. For example, signature verification nodes do not rely on decryption nodes' results and vice versa. In these cases, their jobs can be executed in a parallel fashion. Given a parallelization ratio $\alpha_i$ of a specific processing farm $i$, we have:

$$\hat{T'_p} = \sum (\alpha\hat{h} + \vec{\Delta}) + w \tag{9}$$

The $\alpha$ makes the following condition much easier to be satisfied:

$$\sum \Delta < T_0(\frac{1}{m} - \sum \frac{\alpha_i}{u_i}) \tag{10}$$

Therefore, from the angle of performance, DEN is better than WS-Dispatcher, only when at least some of the processing jobs could be executed parallel.

Furthermore, with such a flexible architecture, the service provider could choose different WS security policies for different users under different circumstances, all on the fly. Correspondingly, different requests will have different routes through processing farms.

# 4 The Implementation

## 4.1 DEN and XSUL2

Evolving from SoapRMI and XSoap [12], XSUL is a lightweight Web services framework. The latest version of XSUL, XSUL2, leverages the design pattern named *Chain of Responsibility*, which decouples senders and receivers and allows a request go through a chain of handlers until it reaches the destination.

Based on XSUL2, DEN does one step further to tear off handlers from Web services by wrapping up these handlers as separate handler service nodes, which are dedicated for one processing task each. After the job is done, the processing service node forwards the request as well as the processed results to a node in the next farm or the ultimate service node through a routing handler.

From the model analysis in section 3, we understand that it is the parallelism that boosts the performance and scalability. Pipelining, however, scales in the sense of programming and development by decoupling the processing steps from the Web services' application logic. Considering many of these processing nodes are for general purposes like signature verification and message encryption, the developer may focus on the application logic first and integrate the corresponding processing farms seamlessly through configuring routing tables.

Between the processing farms are passed the message contexts, which store intermediate processing results for exchange. For instance, a policy validation handler may get the information in the message context on whether the current request has been verified by a signature handler and what the requester's principal is. The context is inserted in the SOAP message header section.

In the Grid environment, communication among the whole processing farms is secured through SSL/TLS channels. The routing handler applies Grid proxy certificates for secure communication. A proxy certificate is a short-lived certificate extension derived from a Grid user's Grid identity – an X.509 certificate. For secured environments like a cluster behind firewall, non-secured communication can be applied to save the extra cost.

The design is not limited to SOAP style Web services. It also supports generic HTTP commands like GET. XSUL2

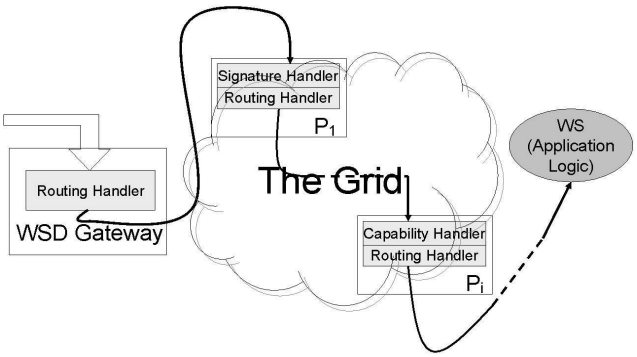


**Figure 4. DEN: The Workflow**

allows a client to "GET" a service's Web Services Definition Language (WSDL) document at runtime. We also plan to support WS-MetadataExchange as an HTTP independent way to discover service's WSDL. A WSDL document describes detailed information of a specific Web service instance such as the interface (also known as PortType) and the policy requirements including security. Previously, the WSDL content was fixed before the service was instantiated. With DEN, the corresponding handlers can rewrite WSDL documents by adding their specific requirements when WSDL documents are routed on the way being transferred to clients.

### 4.2 Scheduling and Routing Table

DEN has scheduling algorithms that make decisions when there are more than one choice in the routing table. A routing table is a hash table with registered service names as keys and endpoint vectors as values. Each endpoint vector contains at least one endpoint reference (EPR) bound to a specific service instance, either a processing node or a final Web service. Figure 4 shows how a request passes through the processing farms.

For the time being, there are two schedulers implemented – a round robin scheduler and a random scheduler. Taking the advantage of the Grids and Globus, a scheduler can instantiate a remote processing node by launching a GRAM job. GRAM stands for Globus Resource Allocation Manager, which is a running daemon that allows authorized jobs to be executed remotely on the Grids. More complicated scheduling algorithms will take the processing nodes' states into consideration. Adding or removing a processing node to or from a farm depends on the running nodes' statuses. When a processing node is launched, it automatically loads the routing table into memory. After that, every specific period of time the scheduler reloads the routing table file, which can be modified at run time and the change will be reflected as soon as the file is reloaded.

### 4.3 RPC vs. Asynchronous Mode

In WS-Dispatcher, both SOAP Remote Procedure Call (RPC) and asynchronous messaging styles are supported. Unlike RPC calls, asynchronous messaging calls do not wait for the processed results; therefore asynchronous Web services can accept more requests at peak time than the RPC implementations. Without blocking calls, system resources, such as network connections, open file handles, threads and processes, can be recycled faster. For long term jobs, say those that take more than one hour or even more than one month, the asynchronous style might be the only choice because usually client-server connections expire within minutes in RPC. Sometimes designers simulate one asynchronous call with multiple RPC calls as alternatives.

However, asynchronous messaging style takes more efforts in handling the messages with queues and the handling work can be more complicated with consistency problems to be taken care of. It does not change the system throughput but rather the response time by accepting the requests immediately, with the assumption that there will eventually be some off-peak time for the system to digest the queued requests. The length of queue should be estimated against the system load.

Despite that it is harder to program with, the messaging style is preferable to RPC nowadays in large Web service applications, due to its advantages explained. We too are heading in this direction in our system design: depending on the machine loads, after a threshold $\gamma$, the Web services will not return computing results immediately to the clients; instead, a receipt with a unique ID will be issued. Sometime later, the clients will be informed with notifications and pull the results from a specified message mailbox with that ID. We are even considering to apply the asynchronous style into DEN processing nodes, so that a pending request will not block a chain of processing nodes. The processing nodes can be reused sooner to accept more requests.

### 4.4 Cooperation with Other Load Balancing Options

DEN is a fine-grained Web services level load balancing solution. It also works with the other levels of solutions from DNS-level solutions, Web server-level solutions, to application level solutions. The WSD gateway can be replaced with other Web server-level solutions since their functions are similar. DNS-level load balancing may be necessary only if the WSD gateway is also overloaded, which infers to very heavy traffic, because WSD was designed to be very lightweight solely for routing purpose.

## 5 Use Cases and Experiment Evaluation

### 5.1 Use Cases

Linked Environments for Atmospheric Discovery (LEAD) project is a large meteorological effort that utilizes the Grids. A spectrum of Web services technologies are applied to LEAD, as well as WS security. Supported by TeraGrid, the largest scientific Grid in the world, LEAD is expected to accommodate thousands of users, which raises a high demand on the scalability requirement.

Powered by XSUL2 with the Apache XML security library [9] integrated, a secure Web service needs more than $100ms$ to return a processed result to each SOAP request. A single node can serve no more than a few hundreds of concurrent synchronous requests with the mainstream hardware. As the number of requests increases, the overloaded Web service becomes so slow that it will be incapable of responding before connections time out.

XPOLA, a fine-grained capability-based authorization framework for Web services [5], makes the situation even worse. Represented in Security Assertion Markup Language (SAML), each XPOLA capability is a set of access policies protected with the issuer's signature. Multiple signatures are required when a capability involves more than one parties. In addition to the signature for the whole message, an XPOLA-enabled Web service has to handle at least two signatures for each request. XPOLA's complexity was the last straw and thus motivated this load balancing work.

DEN distributes the signature verification and generation as well as the capability validation work to signature farms and capability farms in the Grids. Asynchronous messaging WSD further boosts the scalability by breaking the limitation of connection timeout. This make it possible to have a Web service with message-level security that scales well.

### 5.2 Evaluation

In our experiments, we deliberately choose slow machines to reflect the scalability results better in performance showdowns. Each processing node is deployed on one of the dedicated 16 SUN Ultra-5's with a 300MHz SPARC CPU and 256MB memory, connected with other nodes through GB networks. The codes are all in Java and running with SUN JRE1.4.2 Java virtual machine (JVM).

In order for the clients to make as many requests as possible to create the traffic for the server side, a client does nothing more than sending previously prepared SOAP messages in HTTP to the gateway node in either a concurrent or sequential way or both. All SOAP messages are 8363 bytes with a 7885-byte header containing a SAML assertion and a signature section. The clients reside on SUN Fire 280R machines with dual 1.2GHz SPARC CPU's and 4GB memory. Powerful clients and GB networks guarantee they affect the experiment results minimally. Actually since the experiments are under the same conditions, the amount of time the client side takes is constant.

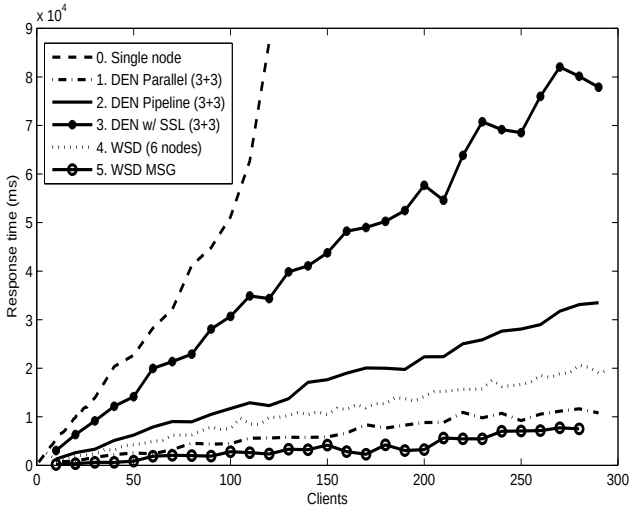


**Figure 5. Performance Showdown (Full)**

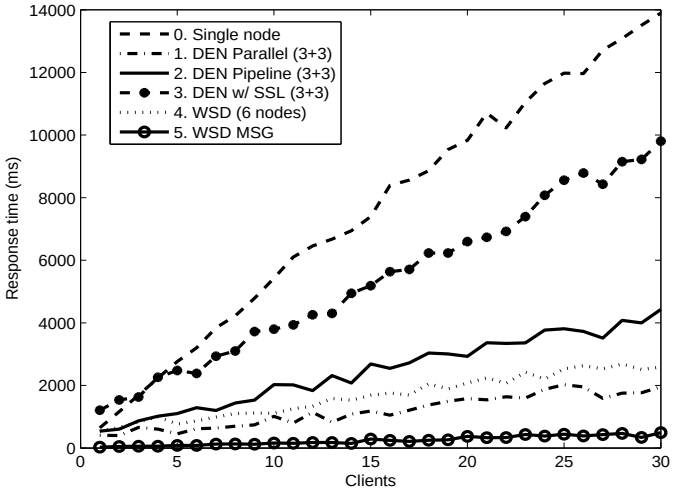


**Figure 6. Zoom-in (1-30 clients)**

At the server side, the traditional Web service processes the XPOLA-enabled SOAP messages in two steps with signature handlers and capability handlers. Signature handler verifies the signature against the whole SOAP message. Then capability handler checks the SAML assertion including the authorization policy and its own signature against the policy. Finally the signature handler signs the response messages and returns them to the clients. DEN wraps up

the handlers as processing nodes. Signature processing nodes are clustered as a signature farm; capability processing nodes form another farm similarly. Each processing node is started through GRAM and after that it acts as a daemon running solely on a host.

Note that usually the performance result of the first request could be bad and unstable. It is mainly because of the internal mechanisms of JVM itself, such as class loading and JIT. Therefore in our experiments, each client repeats the request a specific number of times ranging from at least 2 to 30. The displayed final results are the average values of all the requests made. Even though, the results still cannot be theoretically smooth, as another unpredictable factor during the tests is the garbage collection of JVM, which could happen at any time.

In Figure 5 and 6, the response time of the single node RPC (No.0) is compared with 5 other load balancing options. Except the last one (No.5) in WSD messaging, the rest are done in RPC style, in which response time is identical to turnaround time. No.1 shows the performance of a parallel DEN, with a 3-node signature farm and a capability farm with the same number of nodes. According to Equation 8, the $1:1$ ratio is coincidental simply because capability handler also spend most of the time on signature, and thus needs almost the same amount of processing time as the signature handler. No.2 works with the same processing farms but in a pipelined way; No.3 is a pipelined DEN with SSL to secure the communication between the signature farm and the capability farm. For No.1–3 experiments on DEN, we do not need to redeploy or reboot any running nodes. No matter if they are processing nodes, gateway nodes, or the Web service, the change of DEN topology is done through changing hot-pluggable routing tables. The scheduling algorithm among all DEN-based experiments is round robin. The results from the basic WSD with 6 nodes is drawn as No.4, in which each node is a full Web service replica with all necessary security handlers, but the service state is not shared between replicas.

Theoretically, all RPC versions should be like the single node one that scales till the HTTP connection expires in about 1-2 minutes before returning results; in reality, they stop working earlier at about 300 connections due to various reasons including implementation issues and the limitation of operating systems and JVM. For example, there is usually a limit on the number of open file descriptors that correspond to the network connections. In spite, the figures still prove what we have proposed in the paper.

No.5, WSD messaging, has the best response time, as asynchronous messaging WSD simply acknowledges the acceptance of requests immediately as long as the processing queue is not full. Asynchronous communication thus achieves much better peak-time scalability than synchronous RPC. As expected, No.4 WSD has better turnaround time than pipelined DEN. The latter pays extra cost for routing and transferring packets over the wire. This extra cost is significant in No.3 with SSL. As shown in Figure 6, when the thread number is very small ($< 5$), No.3 is even slower than the single node. However, it scales along with the amount of requests. When No.0 crashes at about 120 clients, No.3 goes well beyond that point till about 360 clients. It is promising as the processing farms may require secure channels through SSL across virtual organizations in Grids. Another encouraging result is that parallel DEN scales better than pure WSD, which rewards the routing efforts of DEN. The non-secure DEN is ideal in a cluster environment behind firewall.

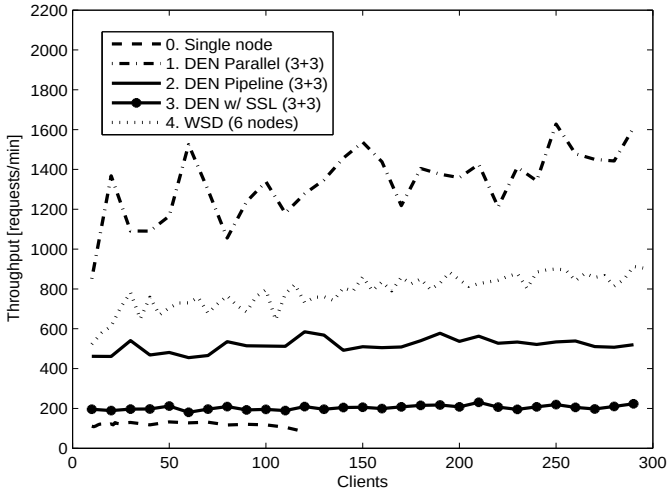


**Figure 7. Throughput Showdown**

The throughput experiments count how many requests could be made in a given period of time $T$. For every specific period of time $t$, a specific number, $n$, of concurrent requests are launched to the service. Figure 7 displays the throughput of each case except WSD messaging, because at the processing steps, it still needs to adopt any one of the 5 approaches. The figure shows the throughput is almost constant for all cases, and the most important of all, DEN and WSD scale.

## 6 Conclusions And Future Work

In this paper, DEN is shown to be able to address the WS security scalability issue, with the idea derived from pipelining and parallel computer hardware architecture and load balancing mechanisms in distributed systems. It is very natural approach to leverage the Grids as a virtual computer with immense computing power. More than that, it also provides desirable flexibility for developers building and de-

ploying these Web services.

We are planning the future work of DEN in several directions. The first one is to integrate asynchronous messaging mechanisms into DEN farms. We believe that it should be able to utilize the resources better. Some version of a distributed shared memory may be used to serve as the messaging bus among processing nodes. A more promising way to improve performance is to change processing nodes to communicate in binary streams instead of XML text messages, because efficiency is much more important than interoperability in DEN farms.

Because Web services security requires so much computing resources, it becomes a weak link that attracts DoS attacks. At the network level, DoS attacks and countermeasures have been discussed quite well. Being scalable is one of the passive countermeasures to DoS attacks. Backed with powerful Grids, DEN demands the attackers to acquire more resources to be influential, and massive attacks can be detected more easily by network level intrusion detection tools and the administrators. Other active countermeasures such as client puzzles [13, 4] are also applicable to Web services. They could be integrated into DEN as functional nodes in the future.

On the other hand, providing more adaptable routing algorithms is another potential direction. The ideal service will be able to automatically use to WS-Policy-compliant security policies by choosing processing routes on the fly. The requests to the same service from different users with different policies are to be routed through different processing nodes accordingly.

## 7 Acknowledgments

The authors would like to thank Wei Lu, Kenneth Chiu for their discussions and collaboration.

## References

[1] I. Avila-Campillo, T. J. Green, A. K. Gupta, M. Onizuka, D. Raven, and D. Suciu. XMLTK: An XML Toolkit for Scalable XML Stream Processing. In *PLANX*, 2002.

[2] V. Cardellini, M. Colajanni, and P.S. Yu. Dynamic Load Balancing on Web-Server Systems. *IEEE Internet Computing*, 1999.

[3] James Cheney. Compressing XML with multiplexed hierarchical models. In *the Proceedings of IEEE Data Compression Conference*, pages 163–172, Snowbird, Utah, 2001.

[4] Drew Dean and Adam Stubblefield. Using Client Puzzles to Protect TLS. In *10th Annual USENIX Security*, 2001.

[5] Liang Fang, Dennis Gannon, and Frank Siebenlist. XPOLA – An Extensible Capability-based Authorization Infrastructure for Grids. In *the 4th PKI R&D Workshop*, Gaithersburg, MD, April 2005.

[6] Ian Foster and Carl Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[7] T. Hey and A. E. Trefethen. The UK e-Science Core Programme and the Grid. *Future Generation Computing Systems*, 18:1017, 2002.

[8] W. Lu, K. Chiu, A. Slominski, and D. Gannon. A Streaming Validation Model for SOAP Digital Signature. In *HPDC-14*, 2005.

[9] The Apache XML Security Project. http://xml.apache.org/security.

[10] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance Comparison of Security Mechanisms for Grid Services. In *the 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, Nov. 8 2004.

[11] A. Slominski, A. di Costanzo, D. Gannon, and D. Caromel. Asynchronous Peer-to-Peer Web Services and Firewalls. In *the 7th International Workshop on Java for Parallel and Distributed Programming (IPDPS 2005)*, April 2005.

[12] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. Design of an XML based interoperable RMI system: SoapRMI C++/Java 1.1. In *the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, NV, June 2001.

[13] X. Wang and M. K. Reiter. Mitigating bandwidth-exhaustion attacks using congestion puzzles. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 257–267, New York, NY, USA, 2004. ACM Press.